The 12th Conference of PhD Students in Computer Science

Volume of short papers

CS^2

Organized by the Institute of Informatics of the University of Szeged

June 24 – June 26, 2020
Szeged, Hungary

Scientific Committee:

János Csirik (Co-Chair, SZTE)
Lajos Rónyai (Co-Chair, SZTAKI, BME)
András Benczúr (ELTE)
Tibor Csendes (SZTE)
László Cser (BCE)
Erzsébet Csuhaj-Varjú (ELTE)
József Dombi (SZTE)
István Fazekas (DE)
Zoltán Fülöp (SZTE)
Aurél Galántai (ÓE)
Zoltán Gingl (SZTE)
Tibor Gyimóthy (SZTE)
Katalin Hangos (PE)
Zoltán Horváth (ELTE)
Márk Jelasity (SZTE)
Tibor Jordan (ELTE)
Zoltán Kató (SZTE)
Zoltán Kása (Sapientia EMTE)
László Kóczy (SZE)
Andrea Kő (SZE)
János Levendovszki (BME)
Máté Matolcsi (BME)
Gyöngyvér Márton (Sapientia EMTE)
Branko Milosavljevic (UNS)
Valerie Novitzka (TUKE)
László Nyúl (SZTE)
Marius Otesteanu (UPT)
Zsolt Páles (DE)
Attila Pethő (DE)
Sándor Szabó (PTE)
Gábor Szederkényi (PPKE)
Jenő Szigeti (ME)
János Sztrik (DE)
János Tapolcai (BME)
János Végh (ME)

Organizing Committee:
Attila Kertész, Balázs Bánhelyi, Tamás Gergely, Judit Jász, Zoltán Kincses
Address of the Organizing Committee
c/o. Attila Kertész
University of Szeged, Institute of Informatics
H-6701 Szeged, P.O. Box 652, Hungary
Phone: +36 62 546 781, Fax: +36 62 546 397
E-mail: `cscs@inf.u-szeged.hu`
URL: `http://www.inf.u-szeged.hu/~cscs/`

Sponsors
Supported by the project "Integrated program for training new generation of scientists in the fields of computer science", No. EFOP-3.6.3-VEKOP-16-2017-00002. The project has been supported by the European Union and co-funded by the European Social Fund.
University of Szeged, Institute of Informatics

Preface

This conference is the 12th in a series. The organizers aimed to bring together PhD students working on any field of computer science and its applications to help them publishing one of their first papers, and provide an opportunity to hold a scientific talk. As far as we know, this is one of the few such conferences. The aims of the scientific meeting were determined on the council meeting of the Hungarian PhD Schools in Informatics: it should

- provide a forum for PhD students in computer science to discuss their ideas and research results;
- give a possibility to have constructive criticism before they present the results at professional conferences;
- promote the publication of their results in the form of fully refereed journal articles; and finally,
- promote hopefully fruitful research collaboration among the participants.

The papers emerging from the presented talks will be invited to be considered for full paper publication the Acta Cybernetica journal.

This year, due to the unfortunate international situation caused by the COVID-19 pandemic, the organizers had no option but to adapt, so they decided to convert the conference completely into a web-based event. Nevertheless, they still hope that the conference will be a valuable contribution to the research of the participants, who can perform virtual interactions to get feedback on their research progress.

Szeged, June 2020

Attila Kertész
Balázs Bánhelyi
Tamás Gergely
Judit Jász
Zoltán Kincses

Contents

Program

Wednesday, June 24

09:00 – 09:20	Opening
09:20 – 10:40	Talks – Graphs (4x20 min.)
10:40 – 11:00	Break
11:00 – 12:00	Talks – Machine Learning (3x20 min.)
12:00 – 13:00	Lunch Break
13:00 – 14:20	Talks – Security (4x20 min.)
14:20 – 14:30	Break
14:30 – 15:30	Talks – Program Analysis (3x20 min.)
15:30 – 15:40	Break
15:40 – 16:40	Talks – Healthcare (3x20 min.)

Thursday, June 25

09:20 – 10:40	Talks – Simulation (4x20 min.)
10:40 – 11:00	Break
11:00 – 12:00	Plenary Talk
12:00 – 13:00	Lunch Break
13:00 – 14:20	Talks – Privacy (4x20 min.)
14:20 – 14:30	Break
14:30 – 15:30	Talks – Computer Graphics I. (3x20 min.)
15:30 – 15:40	Break
15:40 – 16:40	Talks – Bugs (3x20 min.)

Friday, June 26

09:40 – 10:40	Talks – Computer Graphics II. (3x20 min.)
10:40 – 11:00	Break
11:00 – 12:00	Plenary Talk
12:00 – 13:00	Lunch Break
13:00 – 15:00	Talks – Distributed (6x20 min.)
15:00 – 15:10	Break
15:10	Closing

Detailed program

Wednesday, June 24

09:00	**Opening**
Session 1	**Graphs** - Session chair: Gábor Gosztolya
09:20	Orsolya Kardos and Tamás Vinkó: *Social network characteristics from the viewpoint of centrality measures*
09:40	Ahmad T. Anaqreh, Boglárka G.-Tóth and Tamás Vinkó: *Symbolic Regression for Approximating Graph Geodetic Number: A preliminary study*
10:00	Sándor Balázs Domonkos and Németh Tamás: *Use data mining methods in quality measurement in the education systems*
10:20	Viktor Homolya and Tamás Vinkó: *Side road to axioms for two-dimensional centralities: Weighted signed network reconstruction from hub and authority values*
10:40	Break
Session 2	**Machine Learning** - Session chair: Márk Jelasity
11:00	Bence Bogdándy and Zsolt Tóth: *Overview of Artificial Neural Network Abduction and Inversion Methods*
11:20	Dávid Papp: *Spectral Clustering based Active Zero-shot Learning*
11:40	Mohammed Mohammed Amin and István Megyeri: *Improving keyword spotting with limited training data using non-sequential data augmentation*
12:00	**Lunch Break**
Session 3	**Security** - Session chair: Ákos Kiss
13:00	Roland Nagy and Levente Buttyán: *Towards Rootkit Detection on Embedded IoT Devices*
13:20	Márton Juhász, Dorottya Papp and Levente Buttyán: *Towards Secure Remote Firmware Update on Embedded IoT Devices*
13:40	Mohammed B. M. Kamel, Péter Ligeti and Christoph Reich: *Private/Public Resource Discovery for IoT: A Two-Layer Decentralized Model*
14:00	László Viktor Jánoky, János Levendevoszky and Péter Ekler: *A Novel JWT Revocation Algorithm*
14:20	Break
Session 4	**Program Analysis** - Session chair: István Siket
14:30	Tamás Aladics, Judit Jász, and Rudolf Ferenc: *Feature Extraction from JavaScript*
14:50	Artúr Poór: *Static Analysis Framework for Scala*
15:10	Jenifer Tabita Ciuciu-Kiss, István Bozó, and Melinda Tóth: *Towards Version Controlling in RefactorErl*
15:30	Break
Session 5	**Healthcare** - Session chair: Richárd Farkas
15:40	José Vicente Egas-López and Gábor Gosztolya: *Using the Fisher Vector Approach for Cold Identification*
16:00	Hayder K. Fatlawi and Attila Kiss: *Activity Recognition Model for Patients Data Stream using Adaptive Random Forest and Deep Learning Techniques*
16:20	Zoltán Richárd Jánki and Vilmos Bilicki: *Crosslayer Cache for Telemedicine*

Thursday, June 25

Session 6	**Simulation** - Session chair: Zoltán Gingl
09:20	Hamza Baniata:
	Fog-enhanced Blockchain Simulation
09:40	Dilshad Hassan Sallo and Gábor Kecskeméti:
	Parallel Simulation for The Event System of DISSECT-CF
10:00	Péter Hudoba and Attila Kovács:
	Toolset for supporting the number system research
10:20	András Márkus:
	Task allocation possibilities in simulated Fog environments
10:40	Break
11:00	**Plenary Talk**
	Tibor Gyimóthy:
	Software Maintenance and Evolution of Large Systems
12:00	**Lunch Break**
Session 7	**Privacy** - Session chair: András London
13:00	Ádám Fodor, László Kopácsi, Zoltán Á. Milacski and András Lőrincz:
	Speech de-identification with deep neural networks
13:20	Zoltán Szabó and Vilmos Bilicki:
	EHR Data Protection with Filtering of Sensitive Information in Native Cloud Systems
13:40	István Fábián and Gábor György Gulyás:
	On the Privacy Risks of Large-Scale Processing of Face Imprints
14:00	Yuping Yan and Péter Ligeti:
	A Combination of Attribute-Based Credentials with Attribute-Based Encryption for a Privacy-friendly Authentication
14:20	Break
Session 8	**Computer Graphics I.** - Session chair: Péter Balázs
14:30	Gábor Karai and Péter Kardos:
	Distance-based Skeletonization on the BCC Grid
14:50	Papp György, Hoffmann Miklós and Papp Ildikó:
	Embedding QR code onto triangulated meshes
15:10	Csaba Bálint:
	Iterative Operations on Footpoint Mappings
15:30	Break
Session 9	**Bugs** - Session chair: Árpád Beszédes
15:40	Balázs Szűcs and Áron Ballagi:
	An Industrial Application of Autoencoders for Force-Displacement Measurement Monitoring
16:00	Gabriella Tóth and Máté Tejfel:
	Component-based error detection of P4 programs
16:20	Attila Szatmári:
	An Evaluation of Bug Taxonomy and Fault Localization Algorithms in JavaScript Programs

Friday, June 26

Session 10	**Computer Graphics II.** - Session chair: Gábor Németh
09:40	Tekla Tóth and Levente Hajder: *Minimal solution for ellipse estimation from sphere projection using three contour points*
10:00	Róbert Bán and Gábor Valasek: *Geometric Distance Fields of Plane Curves*
10:20	Mátyás Kiglics and Csaba Bálint: *Quadric tracing: A geometric method for accelerated sphere tracing of implicit surfaces*
10:40	Break
11:00	**Plenary Talk** Gábor Tardos: *Fingerprinting Digital Documents*
12:00	**Lunch Break**
Session 11	**Distibuted** - Session chair: Tamás Vinkó
13:00	A. Al-Haboobi and Gábor Kecskeméti: *Reducing Execution Time of An Existing Lambda based Scientific Workflow System*
13:20	Ebenezer Komla Gavua and Gábor Kecskeméti: *Improving MapReduce Speculative Executions with Globlal Snapshots*
13:40	Biswajeeban Mishra and Biswaranjan Mishra: *Evaluating and Analyzing MQTT Brokers with Stress-testing*
14:00	Gábor Székely, Gergő Ládi, Tamás Holczer and Levente Buttyán: *Towards reverse engineering protocol state machines*
14:20	Dániel Pásztor, Péter Ekler and János Levendovszky: *Energy-efficient routing in Wireless Sensor Networks*
14:40	Dániel Balázs Rátai, Zoltán Horváth, Zoltán Porkoláb and Melinda Tóth: *Traquest model - a novel model for ACID concurrent computations*
15:00	Break
15:10	**Closing**

PLENARY TALKS

Software Maintenance and Evolution of Large Systems

Tibor Gyimóthy
Institute of Informatics,
University of Szeged, Hungary

Managing large software systems is a difficult task, mainly due to the constant evolution of these systems. The software developers have to manage the continuous software changes with little downtime and cost. In this talk some methods are presented which can assist the software maintenance and evolution process. Software quality models based on product metrics can be used to identify bad smells and error-prone constructions in software code. Static slicing methods are able to identify the data and control flow dependences of software elements. Dynamic slicing approaches can be used in the optimization of testing a program debugging. Finally, we give some remarks on the applications of machine learning methods in software maintenance.

Fingerprinting Digital Documents

Gábor Tardos
Alfréd Rényi Institute of Mathematics
Hungarian Academy of Sciences, Hungary

There are several approaches to make spreading illegal copies of sensitive or protected digital documents harder. The fingerprinting approach does not prevent legitimate users from spreading the document, but makes anonymity impossible for them: the identity of the user who leaked the document can be identified. This approach works in cases when the number of legitimate users is limited and is based in hiding unique identifiers (fingerprint codes) in the copy of every user.

A natural attack against fingerprinting is based of several legitimate users colluding. If they compare their copies of the documents they find where they differ – this is the embedded fingerprint code – so they can remove this code and publish the document in a way that is not traceable back to them. Surprisingly, however, clever design of the fingerprint code can make fingerprinting resilient against such collusion attacks.

Reducing Execution Time of An Existing Lambda based Scientific Workflow System

A. Al-Haboobi and Gábor Kecskeméti

Abstract: Recent developments in the field of cloud computing have led to emerging an innovative service called Function as a Service (FaaS). Cloud functions have received increased attention across a number of disciplines in recent years. Researchers have executed scientific applications such as workflows using function platforms. Functions could reduce the execution time and cost for scientific workflows due to their features: auto-scaling and fine-grained pricing schema. In this paper, we developed an improved system based on the DEWE v3 system that can process large-scale workflows with three different execution modes: (virtual machines, cloud functions, and a hybrid mode). We can improve DEWE v3 by modifying its scheduling algorithm in the cloud function environment. The improved algorithm can schedule a group of jobs with precedence requirements to run in a single function invocation for speeding up the execution time. We evaluated the improved system with large-scale Montage workflows by comparing its result with the original DEWE v3. The experimental results show that the improved system can minimize the execution time and cost in contrast to DEWE v3 in most cases.

Keywords: Scientific workflows, Cloud functions, Serverless architectures.

Introduction

A scientific workflow consists of a large number of dependent jobs with complex precedence constrains between them e.g., Montage [1]. These applications need large resources for processing such as cloud computing that is increasingly being used. Moreover, they require Workflow Management Systems (WMS) such as Pegasus [2] for handling the jobs which follow a specific order of processing to achieve a desired goal. Workflows can be executed on traditional cloud platforms such as Infrastructure as a Service (IaaS) on a number of Virtual Machines (VMs). Function as a Service (FaaS) is a commercial cloud platform for running distributed applications e.g. scientific workflows in cloud. It can be used instead of IaaS for some applications due to its ability for providing auto-scaling and fine-grained pricing schema. For example, the billing interval of cloud functions is 100 ms while for VMs is over second. Determining the number of VMs on cloud is a challenge for executing a workflow because there are different numbers of released jobs in each phase. Therefore, in order to reduce the total execution time of a workflow, we can add more VMs for processing the jobs in a certain phase that may lead to occur a resource under-utilization issue for the same workflow during other phases. However, a cloud function allows to run hundreds of invocations in parallel where the auto-scaling occurs in Faas automatically while in IaaS needs to handle it. We need a scheduling algorithm that dispatches jobs to Lambda with precedence requirements to be run in a single Lambda invocation. Therefore, it can reduce the duplicated data transfer that occurs between the Lambda storage service and its invocation execution environment which is leading to additional communication cost.

We improved the DEWE v3 [3] system because it has demonstrated the ability of cloud functions such as Lambda to execute large-scale montage workflows. We have changed the scheduling algorithm of DWED v3 for reducing the duplicated data transfer. Moreover, it occurs during the execution between the Lambda storage service and its invocation execution environment leading to additional communication cost. In addition, it has achieved by scheduling jobs with precedence requirements to be executed in a single Lambda invocation. As a result, the improved scheduling algorithm can reduce the execution time of scientific workflows on the Lambda platform.

We have evaluated the improved system with large-scale a 6.0-degree Montage workflow that contains a total of 8,586 tasks with a total size of 38GB. We have tested our improved system by comparing its execution time with DEWE v3 on the Lambda function. The experimental results show that the improved system can outperform DEWE v3 in most cases.

Related Works

A number of studies [3,6] have investigated whether large-scale scientific workflows can be executed on a function platform due to its limits such as memory and temporary storage. [4] presented a prototype for executing workflows on cloud functions and the system evaluated with 0.25- and 0.4-degree Montage workflows. However, they tested with small-scale Montage workflows and they have a deficiency with large-scale workflows. In [3] the authors developed a DEWE v3 system that is a based on function platform which is able to process large-scale scientific workflows using AWS Lambda and Google Cloud Functions (GCF). It can process scientific workflows in three different execution modes: virtual machines, cloud functions, and hybrid mode (virtual machines and cloud functions). It was evaluated with a small- and large-scale Montage workflow. DEWE v3 uses a scheduling algorithm that dispatches jobs to Lambda without considering jobs with precedence requirements to be run in a single Lambda invocation. Therefore, it can reduce the duplicated data transfer that can occur between the Lambda storage service and its invocation execution environment which is leading to additional communication cost.

Improving An Existing Workflow Management System

Our improved approach is based on the DEWE v3 system and involves modifying the scheduling algorithm for the cloud function mode. This can speed up the workflow execution time by processing a set of tasks with precedence requirements in a single Lambda invocation. As a result, it can reduce the duplicated data transfer between the Lambda storage and the function invocation execution environment. Algorithm 1 shows the pseudo-code of the improved algorithm for scheduling a workflow. The modified scheduling algorithm changed the dispatching behaviour of DEWE v3 by releasing parent tasks with its children tasks in order to the same Kinesis shard. Where each shard will assure the ordering of tasks in processing which parent task must be executed before its children tasks. The improved algorithm will create a number of partition keys equal to the number of Kinesis shards. We will explain the steps of the improved scheduling algorithm. Step 1: It reads the XML file of workflow definition. Step 3: At the beginning of the scheduling, all the parent tasks will schedule across the Kinesis shards without their children tasks to run on Lambda. Step 4: Then, each parent task completed the execution, then in Step 5: its children will remove it as a parent. Where each child task will become a parent task if it has no other parents. For example, Task 1 is the parent of Task 2 and it completed the execution and Task 2 become a parent. Next, in Step 6 to Step 7 where each parent task will schedule and then in Step 8 to Step 12 each child task will also schedule if it has no other parents. For example, Task 2 will schedule with its children Tasks 4 and 5 on the same Kinesis shard. The parent task with its children tasks will run in a single Lambda function invocation. Lambda will pull tasks from Kinesis shard and execute them sequentially based on their order. As a result, this will lead for reducing the duplicated data transfer between the Lambda storage and the function invocation execution environment. However, a particular task might fail to execute in Lambda due to different reasons such as out of disk space error,

Algorithm 1 The improved scheduling algorithm.

```
Read the workflow definition (dag.xml)
$Ti \leftarrow task, Tj \leftarrow free - task$ , $KSi \leftarrow Kinesis - shard, Li \leftarrow Lambda - fumction$
At the beginning scheduling all tasks that are eligible to run Tj
while (Tj has completed processing) do
    Remove Tj as a parent from its children Ti
    while (Ti has no more parent) do
        schedule Ti to KSi to run on Li
        while (Ti has children) do
            if (A child has only one parent Ti) then
                Remove Ti as a parent from its child
                Schedule the child of Ti to KSi to run on Li
            end if
        end while
    end while
end while
```

exceeding the maximum execution time limit and out of memory error. In this case, Lambda will retry to execute the task until it succeeds.

Evaluation

In this section, we evaluated the improved system by comparing it results with the DEWE v3 system on AWS Lambda. The both systems tested with Montage workflows as a test case with different workflow degrees: 2.0-degree, 4.0-degree and 6.0-degree. We selected Montage that is a real-world astronomy application. It is used for different benchmarks and performance evaluation [5]. We tested the both systems with the Montage 6-degree workflow with job dependencies on Lambda. It is a very large-scale scientific workflow which contains a total of 8,586 tasks, 1,444 input data files and 22,850 intermediate files, with a total size of 38GB. It is a very compute- and data-intensive scientific workflow. It contains short jobs such as mProjectPP, mDiffFit and mBackground while the long jobs are six: mConcatFit, mBgModel, mAdd, mImgtbl, mShrink and mJPEG. All the short and long jobs are executed on Lambda except the mAdd jobs which are executed on the virtual machine t2.xlarge. This occurs because the size of the input/output files exceeds the storage space in Lambda. The total execution time of both systems with Montage workflow 6-degree as shown in Fig. 1. In this experiment, the both systems evaluated on the same platform configuration as follows: The Lambda Memory size is 3008 MB. The Lambda execution duration is 900 seconds. The batch size of the Lambda function is 20. The Kinesis shard number is 30. The virtual machine is t2.xlarge that executed mAdd jobs with its features: 16 GiB of memory and 4 vCPUs.

Conclusion

In this paper, we have improved the scheduling algorithm of DEWE v3 system for the Lambda function execution environment. We modified the dispatching behaviour of workflow by releasing a parent task with its children tasks to the same Kinesis shard. Then, the grouped tasks with precedence requirements can be executed in a single Lambda invocation. As a result, the duplicated data transfer can be reduced between the Lambda storage service and its invocation execution environment. Therefore, speed-up can be achieved for the execution

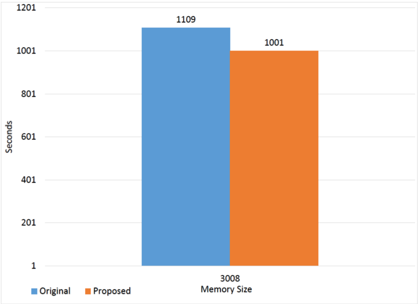


Figure 1: The execution time of the both systems with a 6.0-degree Montage workflow with job dependencies running on 3GB Lambda memory size.

time of scientific workflows. We evaluated the improved system with large scale scientific workflow without job dependencies which are Montage workflow degrees: 2.0, 4.0 and 6.0. Moreover, We also tested it with a 6.0-degree Montage workflow with job dependencies that is a very compute- and data-intensive scientific workflow. The experimental results of both systems show that the improved system can achieve better results than the DEWE v3 system in most cases. In the future work, we will extend the improved system to run on heterogeneous memory sizes of cloud functions to reduce the execution time and cost. Moreover, we will extend a Workflow Management System (WMS) tool for the DISSECT-CF simulator in order to be able for simulating the execution of scientific workflows.

Acknowledgements

This research was supported by the Hungarian Scientific Research Fund under the grant number OTKA FK 131793.

References

[1] Jacob, J.C., Katz, D.S., Berriman, G.B., Good, J., Laity, A.C., Deelman, E., Kesselman, C., Singh, G., Su, M.H., Prince, T.A. and Williams, R., 2010. Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking. arXiv preprint arXiv:1005.4454.

[2] Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Su, M.H., Vahi, K. and Livny, M., 2004, January. Pegasus: Mapping scientific workflows onto the grid. In European Across Grids Conference (pp. 11-20). Springer, Berlin, Heidelberg.

[3] Jiang, Q., Lee, Y.C. and Zomaya, A.Y., 2017, November. Serverless execution of scientific workflows. In International Conference on Service-Oriented Computing (pp. 706-721). Springer, Cham.

[4] Malawski, M., 2016, November. Towards Serverless Execution of Scientific Workflows-HyperFlow Case Study. In Works@ Sc (pp. 25-33).

[5] Juve, G. and Deelman, E., 2008, December. Resource provisioning options for large-scale scientific workflows. In 2008 IEEE Fourth International Conference on eScience (pp. 608-613). IEEE.

[6] Pawlik, M., Figiela, K. and Malawski, M., 2019. Performance considerations on execution of large scale workflow applications on cloud functions. arXiv preprint arXiv:1909.03555.

Speech de-identification with deep neural networks

Ádám Fodor, László Kopácsi, Zoltán Á. Milacski and András Lőrincz

Abstract: Cloud-based speech services are powerful practical tools but the privacy of the speakers raises important legal concerns when exposed to the Internet. We propose a deep neural network solution that removes personal characteristics from human speech by converting it to the voice of a Text-to-Speech (TTS) system before sending the utterance to the cloud. The network learns to transcode sequences of vocoder parameters, delta and delta-delta features of human speech to those of the TTS engine. We evaluated several TTS systems, vocoders and audio alignment techniques. We measured the performance of our method by (i) comparing the result of speech recognition on the de-identified utterances with the original texts, (ii) computing the Mel-Cepstral Distortion of the aligned TTS and the transcoded sequences, and (iii) questioning human participants in A-not-B, 2AFC and 6AFC tasks. Our approach achieves the level required by diverse applications.

Keywords: speech processing, voice conversion, deep neural network, text-to-speech, speaker privacy

Introduction

Cloud-based speech services have improved recently due to the large amount of voice data that is exploited by deep learning technology, giving rise to superhuman performance in several tasks. Consequently, it seems wise to use such utilities in practice.

Unfortunately, many speech applications involve legal concerns regarding privacy. Several tasks have been formulated to eliminate personal information from samples without spoiling the linguistic content before uploading. Voice conversion (VC) operates by altering certain features of human speech [8]. Voice transformation (VT) converts the signal as if it was uttered by a target speaker [5]. De-identification is the process that intends to remove any personal information from the data that could be associated with identity. VC and VT may be applied to solve de-identification, but the papers in the literature suffer from several flaws: the VC algorithm in [4] is approximately invertible and relies on a good voice transformer, while VT [5, 7] requires data from pairs of speakers and is unable to anonymize the target speaker.

Our contributions are as follows. For de-identification, we propose to transform utterances to a generic voice of a Text-to-Speech (TTS) engine, by taking advantage of utterance-text sample pairs. We use an end-to-end trainable Deep Neural Network (DNN) to learn the many-to-one VT task. We suggest to learn the mapping at vocoder level. We show that the trained network gives rise to tolerable distortions at utterance level by conducting two experiments: comparing the outputs of Google's Automatic Speech Recognition (ASR) system for the original TTS output and the de-identified utterance and measuring the Mel-Cepstral Distortion (MCD). To confirm de-identification success, we further performed four perceptual listening studies with human subjects (A-not-B test: distinguishing transformed utterances of different speakers, 2-Alternative Forced-Choice (2AFC) test: classifying utterances from female/male and child/adult speakers, and 6-Alternative Forced-Choice (6AFC) test: estimating the number of speakers). Our proposal is irreversible and it requires only speech-transcript sample pairs for training, which are readily accessible in the literature. We argue that our method performs favorably compared to several baseline methods.

Proposed method

We describe the features set, the pre-processing steps and the DNN architectures in detail here. The de-identification pipeline can be seen in Fig. 1.

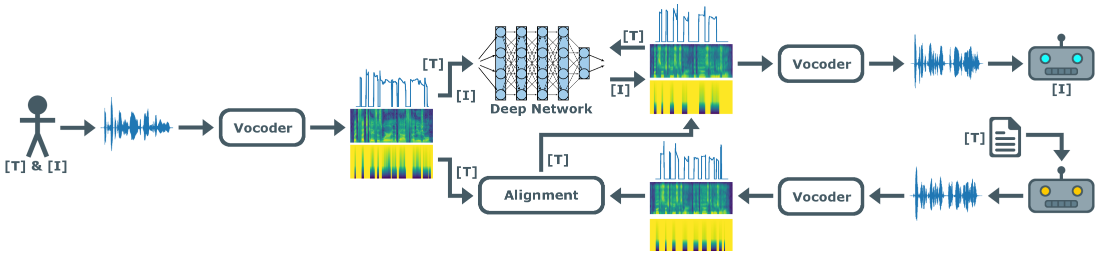


Figure 1: Schematic diagram of our proposed method. Training [T]: vocoded human voice is input to a Deep Neural Network (DNN) that is trained to approximate the aligned TTS output. Inference [I]: vocoded human voice is de-identified by the DNN and transformed back to utterances by the vocoder.

We used the WORLD [6] vocoder for the estimation of the fundamental frequency ($F0$), aperiodicity and spectral envelope. Mel-Generalized Cepstrum (MGC) and band aperiodicity (BAP) were calculated from spectral envelope and aperiodicity, respectively. Linear interpolation of $\log F0$ was calculated. We applied a thresholded binary voiced/unvoiced (V/UV) mask. Dynamic features were determined using MGC and BAP.

The Festival Speech Synthesis System [1] was used to generate TTS audio samples from the corresponding transcripts. The choice of Festival was motivated by comparing several TTS systems. The TTS generated sound files were aligned to match with the corresponding sound files produced by the speaker. We used Dynamic Time Warping (DTW) for the alignments.

For feature transformation, various deep learning architectures were applied and compared: (1) we experimented with an architecture, which we refer to as Dense, having four 1,024 unit dense layers. (2) we used a Convolutional Neural Network (ConvNet) with two 1D convolutional layers of 512 units and kernel width 7 and stride 1, and two 1,024 unit dense layers. (3) tried a model, which we call C-BLSTM, having three batch normalized 256 unit 1D convolutional layers with kernel width 3, one 128 unit BLSTM layer and two 512 unit dense layers was also tested, where the first dense layer was batch normalized. Finally, two state-of-the-art architectures based on (4) Residual Networks (ResNet) and (5) Wav2Letter [3] were also evaluated.

Within all networks, we used ReLU activation functions and dropout layers with probability between 0.2 and 0.3, in addition to adding a final dense output layer on top with linear activation.

Results

We implemented the neural networks in Python using Keras and applied PyWorld and SPTK for feature extraction. We used early stopping with patience 10 and Adam optimizer with its default parameters. We trained the models on TIMIT [10], and CSLU Kids' Speech Version 1.1 [9] data sets. In the latter case a subset was selected from the scripted collection based on the quality of DTW transformed TTS utterances.

To show that the trained networks have acceptable noise levels at utterance level, we conducted two quantitative experiments. In the first experiment we were concerned with the ASR accuracy. Here we measured the intelligibility of the trained networks with the Google Cloud Speech-to-Text system. The precision values are given in Table 1. In the second experiment, we evaluated MCD between de-identified Dense network outputs and the aligned TTS signals by averaging over all 1,680 test sample pairs of the TIMIT corpora using the Dense architecture. The mean and standard deviation values are presented in Fig. 2 for our method (last column) together with values available in the literature. Our method significantly outperformed its baselines.

Table 1: Details of the objective evaluations on the TIMIT database. The average and the standard deviation of the Letter Accuracy Rate (LAR) measured with Google Cloud Speech-to-Text system is presented for the proposed architectures and input features. Notation: "il$F0$" means interpolated $\log F0$.

Method	Architecture	ASR (LAR) precision using the following features:		
		$\log F0$ + MGC + BAP	il$F0$ + MGC + BAP	il$F0$ + MGC + BAP + deltas
Dense	5 dense	0.77 ±0.17	**0.85 ±0.15**	**0.84 ±0.16**
ConvNet	2 conv + 3 dense	0.76 ±0.17	0.82 ±0.17	0.82 ±0.15
C-BLSTM	3 conv + blstm + 3 dense	0.77 ±0.14	0.79 ±0.15	0.79 ±0.15
ResNet	4 residual + 3 dense	**0.79 ±0.15**	0.82 ±0.16	0.82 ±0.14
Wav2Letter	9 conv + 2 dense	0.76 ±0.16	0.79 ±0.17	0.77 ±0.16

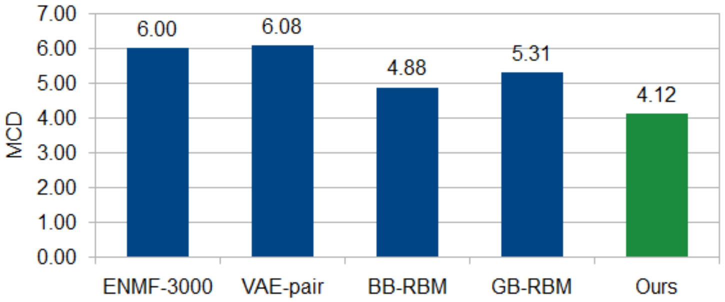


Figure 2: Mean Mel-Cepstral Distortion (MCD) values of various schemes: Exemplar-based Nonnegative Matrix Factorizations (ENMF) [2] using 3000 randomly selected source-target pair frames, VAE-pair [2], Bernoulli-Bernoulli RBM (BB-RBM) [7], Gaussian-Bernoulli RBM (GB-RBM) [7] and our proposed method.

To confirm that our Dense network can properly de-identify human speech, we conducted four qualitative experiments with human participants in an isolated environment using TIMIT and CSLU Kids' Speech. Given randomly selected samples listeners had to decide (1) which samples were produced by kids?, (2) whether a pair of utterances was produced by the same speaker or not, (3) what was the gender of the original speaker, and finally (4) they had to guess the number of speakers within two groups.

In all four tests, the results were convincing, subjects performed like *random choice*, see Table 2. Participants were unable to sense any of the relevant aspects of the speakers. In the first 6AFC test, none of the subjects inferred accurately. In the second 6AFC test, 4 subjects out of 22 predicted correctly, which matches random guessing within tolerance.

Table 2: Results of the perceptual listening experiments. We report the average and the standard deviation of the identification accuracy.

Task	# of subj.	# of samp.	Accuracy mean ±std	Random choice
Child/Adult	14	20	0.52 ±0.15	0.5
A-not-B	22	20	0.56 ±0.15	0.5
Female/Male		15	0.51 ±0.15	0.5
# of Speakers		6	0	0.16
		6	0.18	0.16

Summary

We presented a deep neural network based speech de-identification method that can map vocoder features of human speech to those of a generic TTS engine with little or minimal loss of sound quality using the TIMIT data set. The novelty of our scheme is that de-identification is based on speech-text sample pairs, which are widely available in the speech processing community. In the resulting signal, the identity of the speaker is concealed, as confirmed by our perceptual listening experiments.

The dynamics of the original speaker is inherited due to the application of DTW, A limitation of the technique. We hypothesize that this problem may be alleviated by applying DTW in the loss function of the deep network, a potential future work. We observed degraded performance on the CSLU Kid's Speech corpus calling for additional research in augmenting speech data, in improving the quality of the DTW transformation and in applying more sophisticated DNN architectures. The observed limitation for children's speech may also mean that larger databases are needed for high quality de-identification for certain accents.

Acknowledgements

The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-16-2017-00002, EFOP-3.6.3-VEKOP-16-2017-00001). AL was supported by the National Research, Development and Innovation Fund of Hungary via the Thematic Excellence Programme funding scheme under Project no. ED_18-1-2019-0030 titled Application-specific highly reliable IT solutions.

References

[1] A. Black. The Festival Speech Synthesis System: System Documentation (1.1. 1). *Tech. Rep. HCRC/TR-83*, 1997.

[2] C.-C. Hsu, H.-T. Hwang, et al. Voice conversion from non-parallel corpora using variational auto-encoder. In *APSIPA, Asia-Pacific*, pages 1–6. IEEE, 2016.

[3] V. Liptchinsky, G. Synnaeve, et al. Letter-based speech recognition with Gated ConvNets. *CoRR*, abs/1712.09444, 2017.

[4] C. Magariños, P. Lopez-Otero, et al. Reversible speaker de-identification using pre-trained transformation functions. *Comp. Speech and Lang.*, 46:36–52, 2017.

[5] S. H. Mohammadi and A. Kain. Voice conversion using deep neural networks with speaker-independent pre-training. *Proc. IEEE SLT Workshop*, pages 19–23, 2014.

[6] M. Morise, F. Yokomori, et al. WORLD: a vocoder-based high-quality speech synthesis system for real-time applications. *IEICE Trans. Info. Sys.*, 99(7):1877–1884, 2016.

[7] T. Nakashika, T. Takiguchi, et al. Voice conversion based on speaker-dependent restricted. (6):1403–1410, 2014.

[8] J. Qian, H. Du, et al. VoiceMask: Anonymize and sanitize voice input on mobile devices. *arXiv:1711.11460*, 2017.

[9] K. Shobaki, J.-P. Hosom, et al. The OGI's Kid's Speech corpus and recognizers. *Proc. 6th ICSLP*, 2000.

[10] V. Zue, S. Seneff, et al. Speech database development at MIT: TIMIT and beyond. *Speech Comm.*, 9:351–356, 1990.

Symbolic Regression for Approximating Graph Geodetic Number
A preliminary study

Ahmad T. Anaqreh, Boglárka G.-Tóth and Tamás Vinkó

Abstract: Graph properties are certain attributes that make the structure of the graph understandable easily. Occasionally, standard methods cannot work properly for calculating graph properties due to the huge computational complexity needed to obtain their exact values, especially for real-world graphs. In contrast, heuristics and metaheuristics are alternatives proved their ability to provide sufficient solutions in a reasonable time. In this work, symbolic regression with one of its tools called Cartesian Genetic Programming has been used to derive formulas capable to approximate a specific graph property called geodetic number for random and real-world graphs based on different training data as inputs.

Keywords: Symbolic Regression, Cartesian Genetic Programming, Graph Geodetic Number

Introduction

Topological representation is the simplest way to represent graphs, where the graph is set of vertices and edges. On the other hand, spectral representation (e.g., adjacency matrix, Laplacian matrix) can significantly help to describe the structural and functional behavior of the graph. Adjacency matrix is a $(0, 1)$-matrix which indicates whether the nodes are adjacent or not. Well-known algorithms like Dijkstra's and Floyd-Warshall algorithm use the adjacency matrix to calculate the shortest paths for a given graph. Is also known that the diameter for a given graph is small if the absolute value of the second eigenvalue of the adjacency matrix is small [1]. Laplacian matrix is a square matrix which can be used to calculate the number of spanning trees for a given graph. The eigenvalues of the Laplacian matrix are non-negative, less than or equal to the number of vertices, and less than or equal to twice the maximum vertex degree [2]. Considering these important relations between the graph properties, eigenvalues of spectral matrices and more parameters (to be discussed in the forthcoming sections), which can be calculated easily even for complex graphs, symbolic regression is one of the good choices to verify the connection between graph parameters and properties and use such parameters for approximating network properties.

Preliminaries

Symbolic Regression

Symbolic regression is a mathematical model attempt to find a simple formula such that its output fits a given output in term of accuracy based on a set of inputs. The inputs are set of predefined parameters and constants. The model combines these parameters and constants by a set of given arithmetic operators (e.g., $+, -, \times, \div$, etc.) to build-up a formula. Even though there are some algorithms in the literature using symbolic regression apart from genetic programming [3], essentially, genetic programming is considered as one of the most popular approaches applied by symbolic regression [9].

One of the most famous Genetic programming tools is Cartesian Genetic Programming (CGP) developed by Julian Miller [4]. CGP is an evolutionary algorithm which applies iterations until getting a final solution. The solution is gained either by reaching the maximum number of iterations (number of generations) or a fitness threshold has reached (target fitness). CGP begins by creating some initial solutions, then the best solution will be chosen by evaluating these solutions based on the fitness function. Then, these solutions will be used to create the next generation in the algorithm. Thus next generation solutions will be a mixture of chosen solutions from the previous generation, where the new generation solutions should not be perfectly the same of the previous ones. This can be done by mutation. Mutation used to change small parts of the new solutions and usually mutation for CGP occurs probabilistically. The mutation rate is the probability of applying the mutation on a specific new solution. Eventually, the algorithm must terminate. As we mentioned before there are two cases in which this occurs: the algorithm has reached the maximum number of generations, or the algorithm has reached the target fitness. At this point, a final solution is selected and returned. Using CGP in our work refers to the fact that CGP uses only the inputs actually contribute to the final computation, namely, some inputs can be totally disregarded, that leads to efficient and simple formulas.

Network Properties

A simple connected graph is denoted by $G = (V, E)$, where V is the set of vertices and E is the set of edges.

Geodetic number Geodetic number is the minimal-cardinality set of vertices, such that all shortest paths between its elements cover every vertex of the graph[6]. Assume that $n = |V|$ and $m = |E|$. Given $i, j \in V$, the set $I[i, j]$ contains all $k \in V$ which lies on any shortest paths (*geodetics*) between i and j. The union of all $I[i, j]$ for all $i, j \in S \subseteq V$ is denoted by $I[S]$, which is called as *geodetic closure* of $S \subseteq V$. Formally

$$I[S] := \{x \in V : \exists i, j \in S, x \in I[i, j]\}.$$

The *geodetic set* is a set S for which $V = I[S]$. The *geodetic number* of G is

$$g(G) := \min\{|S| : S \subseteq V \text{ and } I[S] = V\}.$$

Degree-one node The degree of a node is the number of edges linked the node to other nodes in the graph. If there is one edge connected the node, this node is called *degree-one node*. It is easy to see that degree-one nodes are always part of the geodetic set [8].

Simplicial node Vertex v called *simplicial node* if its neighbors form a clique (complete graph), namely, every two neighbors are adjacent. Ahangar et al. [7] prove that if G is a nontrivial connected graph and v is either a cut-vertex or a simplicial vertex of G, then v belongs to every geodetic set of G.

Betweenness centrality Betweenness centrality (BC) for a specific node v is the proportion of all shortest paths pass through this node. The nodes with high BC called central nodes, these nodes might have more impact in the graph comparing to the nodes with low BC. It is shown in [8] if G is a star graph with n nodes the $g(G) = n - 1$ where the central node with the highest BC, that all the shortest paths passing through, will never be in the geodetic set. Moreover, in the tree graph G with k leaves the $g(G) = k$, that mean the leaves with low BC are geodetic nodes while the root and the parents with higher BC are not part of the geodetic set.

Methodology

Although there are not many papers proposing the idea of using symbolic regression for approximating graph properties, the work by Martens et al. [5] was a good starting point for

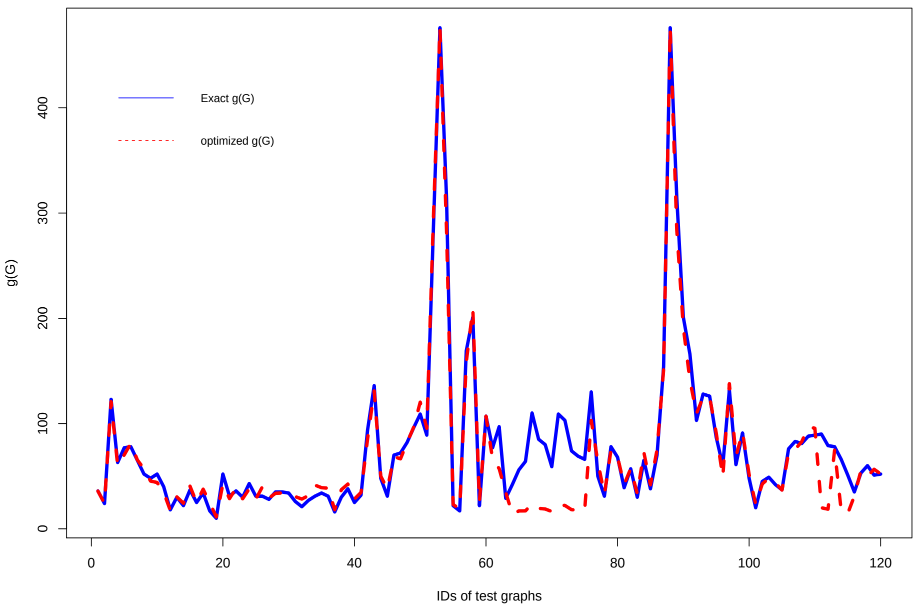


Figure 1: Exact $g(G)$ and optimized $g(G)$ for real-world test graphs

us. They made their experiments on real-world networks. They used CGP to optimize the diameter and isoperimetric number. In our case, we aim at obtaining results for the geodetic number using random and real-world graphs. We have used CGP-Library, which is a cross-platform Cartesian Genetic Programming implementation developed by Andrew Turner[1]. To be able to use CGP all we need is training data. Each training data will contain instances and each instance contains two parts: graph parameters and chosen constants as inputs, the graph property exact value as output. Thus, CGP will attempt to join the parameters and constants by using arithmetic operators to achieve output close to predefined output. The arithmetic operators we have used in all the cases were: $+, -, \times, \div, \sqrt{\cdot}, x^2, x^3$. The parameters we have used in this work: eigenvalues of the adjacency matrix and Laplacian matrix, number of degree-one nodes, number of simplicial nodes, etc.

Numerical experiments

As we mentioned in the previous sections, the experiments we have applied is to investigate the geodetic number property for random graphs and real-world graphs. To obtain the formulas for either random graphs or real-world graphs, we have run the CGP 100 times. From these 100 formulas we choose the best formulas according to its output's absolute deviation from the exact value (best formula gave the least deviation).

For instance the best formula gave best approximation for the geodetic number on real-world graphs is:

$$deg1 + sim + \sqrt{M} - 2, \tag{1}$$

where $deg1$ is the number of degree-one nodes, sim is the number of simplicial nodes, and M is the number of edges.

Figure 1 shows a comparison between optimized $g(G)$ and exact $g(G)$ for 120 graphs. We created these as sub-graphs of different sizes (with $14 \leq N \leq 140$) from known test graphs

[1] http://www.cgplibrary.co.uk/files2/About-txt.html

taken from the literature[2]. It is obvious how the optimized values are very close to the exact values although there are two gaps in the figure indicating that for some graphs the optimized value is less than the exact value. For these graphs, the number of simplicial nodes was zero. Since the formula is the summation of the number of simplicial nodes, the number of degree-one nodes, and the number of edges, if one of these values is zero that will cause these gaps.

Conclusion

Our work demonstrates that symbolic regression is applicable to derive formulas based on given inputs for the graph geodetic number. The formulas gave acceptable results compared to the exact values. Here, we have proposed a general formula to compute the geodetic number for real-world graphs. Thus, the geodetic number can be obtained in a reasonable computational time even for graphs with thousands of nodes and edges, while obtaining the exact geodetic number is an NP-hard problem.

Acknowledgements

The project was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

References

[1] F. R. K. Chung: Diameters and Eigenvalues. Journal Of The American Mathematical Society **2**(2), 187–196 (1989).

[2] William N. Anderson Jr. and Thomas D. Morley: Eigenvalues of the Laplacian of a Graph. Linear and Multilinear Algebra **18**(2), 141–145 (1985).

[3] Trent McConaghy1: FFX: Fast, Scalable, Deterministic Symbolic Regression Technology. Genetic Programming Theory and Practice IX,235-260 (2011).

[4] Julian F. Miller: Cartesian Genetic Programming, Springer (2011).

[5] Marcus Martens, Fernando Kuipers, and Piet Van Mieghem: Symbolic Regression on Network Properties. Genetic Programming, 131–146 (2017).

[6] Hansen, P. and van Omme, N.: On pitfalls in computing the geodetic number of a graph. Optimization Letters **1**(3), 299–307 (2007).

[7] HA Ahangar, F Fujie-Okamoto, V Samodivkin: On the forcing connected geodetic number and the connected geodetic number of a graph. Ars Combinatoria **126**, 323-335 (2016).

[8] Frank Harary, Emmanuel Loukakis, Constantine Tsouros: The geodetic number of a graph. Mathematical and Computer Modeling **17**(11), 89–95 (1993).

[9] Koza, John R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA (1992).

[2] the original graphs can be found in the repository http://networkrepository.com

Task allocation possibilities in simulated Fog environments

András Márkus

Abstract: Nowadays billions of smart devices utilise the network and storage capacity of complex systems. These devices having various sensors are usually managed in Internet of Things systems, which are often supported by Fog Computing or Cloud Computing services. Within the Fog paradigm, nodes are installed close to the users (i.e their devices) to achieve faster and more reliable communications and sensor data management than former cloud solutions. In this paper, we present an extension of a simulation tool called DISSECT-CF-Fog, which provides a more detailed fog model by enhanced location awareness and multi-layer fog node management. We also exemplify the utilisation of these fog properties by developing and validating different task allocation strategies in simulated IoT-Fog-Cloud environments. We also show an evaluation of a scenario comparing four different approaches for task allocation in these systems.

Keywords: Fog Computing, Internet of Things, Simulation, Task allocation

Introduction

According to the paradigm of the Internet of Things (IoT), sensors, actuators and smart devices are connected through the Internet. Based on [1], the number of smart devices will overstep 75 billions all over the world by 2025. IoT is often encoupled with Cloud Computing, because the huge amount of sensed data require storing and processing for further analysis with cloud resources. In the past years a new paradigm called Fog Computing appeared (grown out of Cloud Computing), where the generated sensor data are stored and processed on so-called fog nodes, which are located geographically closer to the end users for minimising latency and ensuring privacy of the data [2]. These fog nodes usually have less computing power and limited functionality as well (due to their constrained nature). Different type of applications, such as real time or forecasting systems, can utilise fog architectures, which typically have more layers of fog or cloud nodes with different resource constraints and characteristics. These nodes can process bag of task applications composed of the generated data of sensors, hence one of the open issues of Fog Computing is how the operating costs and the node response times can be minimised by an appropriate allocation of the tasks.

The investigation of real-world fog systems and topologies are rarely feasible, thus different simulation environments are utilised by the researchers for such purposes. In this paper we chose the DISSECT-CF-Fog simulator[1] to be extended with a more realistic fog model, and for further analysing task allocation possibilities. The rest of this paper we give a brief introduction to related works in Section , in Section we present our simulator extension, the proposed strategies and their evaluation. Finally, Section concludes our work.

Related work

The literature of Fog, Cloud and IoT systems contains numerous articles with diverse approaches aiming at task and virtual machine (VM) placement for IoT applications. Mann et al. [3] compared seven different VM placement algorithms for clouds. They executed their experiments in the well-known CloudSim simulator, relying on cloud properties such as CPU speed, resource capacity, utilisation ratio and energy consumption of the available nodes.

[1] The DISSECT-CF-Fog simulator is available at: https://github.com/andrasmarkus/dissect-cf/

Vasconcelos et al. [4] presented a different approach, where the Fog topology is modelled as a weighted graph, and they considered resource cost, bandwidth, and latency of the smart devices. Resource availability and the topology of the network are also considered during their validations performed in the Cooja simulator.

Xia et al. [5] proposed four heuristics for application placement in SimGrid with the following parameters: required CPU, RAM, bandwidth, and latency of the fog nodes. Three allocation policies were presented by Bittencourt et al. [6], which were evaluated in the one of the most commonly used Fog simulators called iFogSim. The proposed strategies took into account CPU capacity, the arrival time and the delay-priority of the tasks. Skarlat et al. [7] presented a framework, where fog nodes are utilised when possible. Their optimisation solution considers the resource capacity of fog nodes, the resource demand of a service, the link delay between nodes, and different properties of the application (e.g. deployment and response time).

Based on these works we can state that the evaluation of task scheduling algorithms have been performed in different simulation environments, and most approaches consider almost the same characteristics of the Fog and IoT systems. Beside the former commonly used cloud properties, location and mobility-related properties are used in fogs, but more heterogeneous and multi-layered Fog-Cloud systems are rarely analysed.

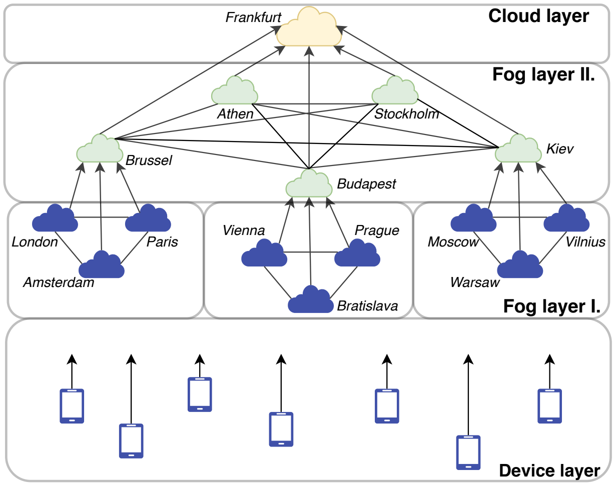


Figure 1: The considered Fog topology in the evaluation

The proposed simulator extension and its evaluation

An algorithm for the allocation of a task of an IoT application in our proposed model can consider the energy consumption of the execution environment (i.e. fog or cloud node), the load of the network used for communication and data transfers, and the transfer, storage and execution costs. When a selected fog node is overloaded, the execution of the appropriate task will be delayed, which has a negative effect on the makespan of it application. To overcome this problem, we also introduce the possibility of multi-layer fog node management by enabling task offloading from (possibly overloaded) nodes to others. A typical fog topology can contain numerous nodes, some of them are grouped as a Fog cluster, which restricts the access and visibility of other nodes. These nodes can be ordered into layers, where higher level fog layers usually contain stronger physical resources. We implemented this envisioned model in the DISSECT-CF-Fog simulator. We developed a solution that is able to handle fog properties dynamically, hence performing task allocation and reallocation during the experiments. In this way, different approaches can be created and implemented by extending the new, *Application-Strategy* abstract class.

We defined four strategies to validate the usability of our proposal. The *Random* strategy is the default, which always chooses one from the connected nodes randomly. The *Pushing Up* strategy always chooses the parent node (i.e. a node from a higher layer), if available. The disadvantage of these strategies is the disability to consider increased network traffic and costs of the operation. The third strategy called *Holding Down* considers data protection, because the application keeps the data as close to the end-user as possible. In this way the network traffic is minimal, but the execution time of the application can increase dramatically (due to the overload of the lowest layer). Finally, the *Load Balancing* strategy ranks the parent nodes (if available) and all neighbour nodes (from its own cluster) by network latency and the ratio of the available CPU capacity and the total CPU capacity. The algorithm picks the node with the highest rank (i.e. closest and least loaded).

We evaluated these proposed strategies in a European-wide weather forecasting scenario with a fog topology having 10000 weather stations. Each station is equipped with five sensors (e.g. measuring weather conditions (e.g. temperature, humidity) with 50 bytes of sensor data). The time interval between two measurements was set to one minute, and the whole simulation period took one day. The topology contains two Fog layers with 14 different Fog nodes organised into clusters, and one cloud layer having a single cloud node. Each node has at lest 40 CPU cores and 44 GB RAMs, and they are mapped to different cities, with exact latency values defined between them using WonderNetwork[2]. The defined topology can be seen in Figure 1, which also depicts the Fog clusters and layers with different colour. The arrows represent routes responsible for communication between the layers, while the (undirected) edges are for the message exchanges inside a cluster. The network capability of the smart devices (or stations) are modelled with a 4G network with an average 50 ms of latency. The fog nodes are modelled with real VM specifications according to the Amazon Web Services (AWS): the lowest Fog layer has VMs with 2 CPU cores, 4 GB RAMs and 0.051$ hourly price, the top fog layer has VMs with 4 CPU cores, 8 GB RAMs and 0.101$ hourly price, finally the cloud layer has VMs with 8 CPU cores, 12 GB RAMs and 0.204$ hourly price. Each VM can process only one task (represented by 250 Kilobytes of data) at the same time.

Table 1: Evaluation results of the proposed strategies

Strategies	Random	Pushing Up	Holding Down	Load Balancing
Num. of VM	78	73	60	78
Network utilisation (sec.)	54	40	0	43
Data transferred (MB)	1076	279	0	346
Timeout (min.)	42.04	4.88	175.16	4.66
Fog Layer I. cost ($)	66.41	66.40	72.05	66.40
Fog Layer II. cost ($)	21.71	22.02	0	22.01
Cloud cost ($)	13.10	0.74	0	0.81
Sum. of cost ($)	101.22	89.16	72.05	89.22

Table 1 summarises the average results after executing the scenario with all strategies five times. For the comparison of the strategies, we measured how many VMs were required for processing the tasks (and their data) during the operating hours. The Network utilisation metric reflects the network load, and it represents the time taken to transfer the sensor data from the source to the actual processing node, while the Data transferred metric represents its size. The Timeout value means the time taken to finish data processing after the last sensor

[2] WonderNetwork website is available at: https://wondernetwork.com/pings. Accessed in January, 2020.

measurement was performed. According to the used AWS pricing models, we could calculate the exact costs of the usage of the resources, separating each layer.

Concerning the results, the *Random* strategy occasionally used the network unnecessarily (1074 MB for 54 seconds), and the task were often moved to the cloud layer resulting in the highest cost (101.22 $). The *Holding Down* strategy used the least VMs (60), however its timeout value extremely increased (175.16 minutes), pointing out the need of the upper layers. Nevertheless, it is obviously the cheapest solution with 72.05 $. The *Pushing Up* approach secured us average cost, but its timeout is the best so far. Finally, the *Load Balancing* algorithm showed slightly better task allocation for almost the same price, with the timeout value of 4.66 minutes.

Conclusion

In this paper we presented a simulation solution for investigating task allocation problems in Fog environments. We developed an extension to DISSECT-CF-Fog with a revised fog model, and proposed four strategies to exemplify its utilisation. We also compared the performance of these strategies. In our future work we plan to extend our strategies with more sophisticated methods, and perform evaluations of larger-scale IoT systems.

Acknowledgements

This research was supported by the Hungarian Scientific Research Fund under the grant number OTKA FK 131793, and by the Hungarian Government under the grant number EFOP-3.6.1-16-2016-00008.

References

[1] Taylor, Robin and Baron, David and Schmidt, Daniel. (2015). The world in 2025 - predictions for the next ten years. 192-195. 10.1109/IMPACT.2015.7365193.

[2] Mahmud, Md and Buyya, Rajkumar. (2016). Fog Computing: A Taxonomy, Survey and Future Directions. 10.1007/978-981-10-5861-5_5.

[3] Mann, Zoltan and Szabo, Mate. (2017). Which is the best algorithm for virtual machine placement optimization?. Concurrency and Computation: Practice and Experience. e4083. 10.1002/cpe.4083.

[4] Vasconcelos, Danilo and Andrade, R and Severino, Valdenir and De, J and Souza,. (2019). Cloud, Fog, or Mist in IoT? That Is the Question. ACM Transactions on Internet Technology. 19. 25. 10.1145/3309709. 10.1145/3309709.

[5] Xia, Ye and Etchevers, Xavier and Letondeur, Loic and Lebre, Adrien and Coupaye, Thierry and Desprez, Frederic. (2018). Combining Heuristics to Optimize and Scale the Placement of IoT Applications in the Fog. 153-163. 10.1109/UCC.2018.00024.

[6] Bittencourt, Luiz Fernando and Diaz-Montes, Javier and Buyya, Rajkumar and Rana, Omer and Parashar, Manish. (2017). Mobility-Aware Application Scheduling in Fog Computing. IEEE Cloud Computing. 4. 26-35. 10.1109/MCC.2017.27.

[7] Skarlat, Olena and Nardelli, Matteo and Schulte, Stefan and Borkowski, Michael and Leitner, Philipp. (2017). Optimized IoT Service Placement in the Fog. Service Oriented Computing and Applications. 11. 427-443. 10.1007/s11761-017-0219-8.

Static Analysis Framework for Scala

Artúr Poór

Abstract: Static program analysis has many application areas: from bug finding and code verification to checking safety preconditions of transformations. Many transformations target improving performance through parallelization, and they need elaborate static analysis to ensure that the transformation does not change the program semantics.

This paper presents a static analysis library for Scala, a hybrid object-oriented and functional programming language. The library is IDE-independent and represents the result of an ongoing research project, features control flow analysis and data flow analyses.

Introduction

Automatic discovery of program parts that have unexploited parallelism involves thorough static analysis. In order to claim two statements are safe to execute in parallel, it is necessary to verify that no dependency exists between the statements. If it turns out that one statement is dependent on the other, their relative order must not be changed to preserve the result of the computation.

Two kinds of dependency between statements are distinguished: data and control dependency. When statement S_2 *data depends* on statement S_1, it means one of the following: S_1 writes memory region that S_2 subsequently reads (true dependency), S_2 reads memory region that S_1 subsequently writes (anti-dependency), or S_2 and S_1 both writes the same memory region (output dependency). When S_2 is *control dependent* on S_1 then, intuitively, the execution of S_1 directly governs whether S_2 will be executed.

Determining whether two statements are in dependency relationship requires different kinds of static analysis. Control flow analysis and data dependence analysis are common, and further analyses are determined by the nature of the parallelization strategy. In case of loop parallelization, for example, Artigas et. al. [1] utilized side effect analysis to create regions of exception-free statements, and Kennedy et. al. [3] analyzed loop-carried dependencies between loop iterations. Discovering divide and conquer algorithms [6] may need analysis of pointer bounds so that a procedure calls access non-overlapping ranges of memory. Such analyses either do not depend on control flow analysis (flow-insensitive) or they do (flow-sensitive). Existence of the latter justifies the implementation of control flow analysis for Scala.

The main contribution of this paper therefore is a static analysis library for Scala, which features control flow analysis and presents detailed control flow graph of Scala programs. The control flow graph is input of flow sensitive analyses. An example of a flow sensitive data flow analysis is live variables analysis. We hope the library will be the foundation of a semi-automatic parallelization tool for Scala in the future, although static analysis for code maintenance can also be implemented on top of the library. For example, live variables analysis has been implemented, which can be used to detect dead code. The library can also construct call graph of methods.

Representation of Scala Programs

Intermediate representation is a data structure for representing source code with the goal of analysis, transformation or compilation. The library builds an abstract syntax tree from Scala source code as an intermediate representation. The process involves three phases. First, the source code is fed into `scalac`, the Scala compiler, which performs lexical, syntactic and static semantic (e.g. name resolution and type inference) analyses. The output of this phase is an AST, which is annotated with semantic information (e.g. symbols and types), although the compiler removes syntactic sugars and comments, and inserts extra code during the process. Second, the very same code is also fed into a parser which yields a faithful representation of the source code without types and other semantic information. Finally, the two ASTs are merged so that a faithful, type-annotated AST is created. We refer to this as resugared AST in this paper.

In order to see why the three steps are necessary, consider the following Scala code from `scalafix` library:

```
object Patch {
 /** Combine a sequence of patches into a single patch. */
 def fromIterable(seq: Iterable[Patch]): Patch =
  seq.foldLeft(empty)(_ + _)
  ...
}
```

The compiler will discard the comment, fills in the implicit type parameter of @foldLeft@. The compiler will also desugar the placeholder syntax into anonymous function with fresh argument identifiers:

```
def fromIterable(seq: Iterable[Patch]): Patch =
 seq.foldLeft[Patch](empty)(((x$0: Patch, x$1: Patch) => x$0.+(x$1)))
```

This clearly complicates presenting the code for users after code transformation (e.g. inline method). While comments are not essential in compiling the above code, they should be preserved when the code is reconstructed from the intermediate representation.

After each of the two front ends constructed ASTs from the same Scala code, they are traversed simultaneously in depth-first order from top-level declarations to expressions. For each node in the untyped AST, a matching node is searched in the typed AST to gather static semantic information. The matching criterion is that the two nodes must belong to the same position in the same source file, that is, their lexical token sequence must be identical modulo the peculiar handling of position information in the two front ends in some cases.

Terminal and non-terminal nodes of the resugared AST fall under one of three categories: declaration, definition or expression, represented by @Decl@, @Defn@ and @Expr@, respectively. The ambiguity introduced by the placeholder syntax in the above code example can be resolved with a dedicated syntactic category for infix applications (@AppInfix@), which diverges from ordinary anonymous functions (@Lambda@). The design of class hierarchy follows that of the parser library of our choice, `scala.meta`. Main difference is the addition of type information for expressions and symbols for definitions and declarations.

Control Flow Analysis

Flow sensitive data flow analyses rely on preliminary control flow analysis which approximates the flow of control in the program. The result of the analysis is a control flow graph.

Nodes of a graph are basic blocks, sequences of code which are executed without interruption. If the control reaches a basic block, then all the code in that block will be executed. Edges in a graph represent flow of control between basic blocks. Label on an edge indicates whether the control is passed conditionally (e.g. in loops) or unconditionally (e.g. method invocations and exceptions).

Basic blocks are referred by labels (@BLabel@). That level of indirection allows us to represent cycles in the graph. Such cycles are introduced by loops and recursive methods. A basic block can be either open or closed (this design decision is inspired by Hoopl [5], a static analysis framework) on either end. The distinction between open and closed ends allows us to construct blocks in a type safe manner. A block is only considered complete when both ends are closed. Only complete blocks can be added to a control flow graph. An open block of type

@Block[A,C,O]@ for some type @A@ consists of one @BFirst@, which holds the block label, followed by zero or more @BMiddle@s, concatenated together using @BCat@s. A @BMiddle@ is a pointer to an expression in an AST which implicitly passes control to the next expression in the block. When the control flow analysis reaches an explicit control passing expression, it completes the block using a closing @BLast@, which becomes of type @Block[A,C,C]@, and starts a new block. Type safety ensures that control transfer cannot occur in the middle of a block as a @BMiddle@.

```scala
sealed abstract class O // Open
sealed abstract class C // Closed

sealed abstract class Block[A[_,_],E,X]  // Entry, eXit
  extends NonLocal[E,X] { ... }
case class BFirst[A[E,X]<:NonLocal[E,X]](element: A[C,O])
  extends Block[A,C,O]  { ... }
case class BMiddle[A[E,X]](element: A[O,O])
  extends Block[A,O,O]  { ... }
case class BLast[A[E,X]<:NonLocal[E,X]](element: A[O,C])
  extends Block[A,O,C]  { ... }
case class BCat[A[_,_],E,X](b1: Block[A,E,O], b2: Block[A,O,X])
  extends Block[A,E,X]  { ... }
```

The analysis[1] is still in early phases. Only a subset of Scala is supported currently, which includes sequences, `if` expressions, `while` loops, recursive functions.

Related Work

A refactoring tool for Scala has been developed and published as master's thesis [8] by Mirko Stocker. The tool implements several useful refactorings: rename, extract method, extract local, inline local and organize imports. The tool has been integrated into Scala IDE for Eclipse, which seems to be discontinued since 2017. The tool also performs static analysis on Scala programs to support the refactorings. Similarly to our approach, this work also uses the Scala compiler to construct an AST. The supported refactorings do not require control flow analysis, and the tool does not implement it.

IntelliJ IDEA is one of the few IDEs which support Scala via plug-in. The Scala plug-in implements an incremental compiler front-end for type checking code and providing auto-completion for developers. The list of available refactoring includes rename, move class, extract trait, extract method, introduce variable, introduce field, introduce parameter. The plug-in also features control flow analysis, which covers more features of Scala, in order to highlight unreachable code.

Ilya Segey et. al. reported a survey [7] of automatic refactorings for Scala and the challenges posed by language features and static semantics of Scala. The refactorings are compared to Java analogues, highlighting the differences and pitfalls. Many transformations and refactorings are implemented in IntelliJ IDEA, including add missing imports, remove unused imports, introduce and inline variables and extract method. The report also discusses a few more useful refactorings specific to Scala.

Static analysis engine Insane [4] is also capable of constructing control flow and call graphs of Scala programs. Furthermore, it analyses class hierarchy and pointers. The project seems to be discontinued since 2013, and we failed to compile the source code due to `jar` dependencies that are no longer available.

LambdaFicator [2] is a Java tool for migrating pre-8 Java code to use some of the new functionality that Java 8 introduced. LambdaFicator automates two refactorings. First, it

[1] Implementation is available at https://github.com/poor-a/parscala

transforms anonymous inner classes into lambda expressions. Second, `for` loops over Java collections are converted into application of higher order functions, which use lambda expressions.

Conclusion

Getting started with tool-supported semi-automatic refactorings is far from trivial for a language such as Scala. Safe refactoring require precise static analysis so that the refactoring does not introduce changes in the program semantics. Building the static analysis framework for Scala revealed challenges in designing the intermediate representation.

This paper presented the static analysis framework for Scala along with design of intermediate representation. The focus was on control flow analysis, which enables us to perform many interesting flow sensitive data flow analysis in the future.

References

[1] Pedro V. Artigas, Manish Gupta, Samuel P. Midkiff, and José E. Moreira. Automatic loop transformations and parallelization for java. In *Proceedings of the 14th International Conference on Supercomputing*, ICS '00, pages 1–10, New York, NY, USA, 2000. Association for Computing Machinery.

[2] Alex Gyori, Lyle Franklin, Danny Dig, and Jan Lahoda. Crossing the gap from imperative to functional programming through refactoring. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 543–553, New York, NY, USA, 2013. Association for Computing Machinery.

[3] Ken Kennedy, Kathryn S. McKinley, and Chau-Wen Tseng. Analysis and transformation in the parascope editor. In *Proceedings of the 5th International Conference on Supercomputing*, ICS '91, pages 433–447, New York, NY, USA, 1991. Association for Computing Machinery.

[4] Etienne Kneuss. Interprocedural static analysis engine for Scala, 2013.

[5] Norman Ramsey, João Dias, and Simon Peyton Jones. Hoopl: A modular, reusable library for dataflow analysis and transformation. *SIGPLAN Notices*, 45(11):121–134, September 2010.

[6] Radu Rugina and Martin Rinard. Automatic parallelization of divide and conquer algorithms. *SIGPLAN Notices*, 34(8):72–83, May 1999.

[7] Ilya Sergey, Dave Clarke, and Alexander Podkhalyuzin. Automatic refactorings for scala programs. Technical report, Department of Computer Science, KU Leuven; Leuven, Belgium, March 2010.

[8] Mirko Stocker. Scala refactoring. Master's thesis, HSR Hochschule für Technik Rapperswil, 2010.

An Evaluation on Bug Taxonomy and Fault Localization Algorithms in JavaScript Programs

Attila Szatmári

Abstract: Due to the asynchronous and loosely typed nature of JavaScript, developers are prone to making mistakes. Finding an error in complex software programs is an expensive task. Fault localization (FL) can reduce the time spent on finding bugs. However, it raises the question: *"Which bugs are easier to find for these algorithms, and why?"* Therefore, I performed an empirical study on JavaScript programs to evaluate the relationship between bug taxonomy and fault localization algorithms' efficiency. Results show that bugs that occurred due to **incomplete feature implementation** and **incorrect feature implementations** are significantly different. I further investigated these taxonomy types to better understand which bug features make the localization easier. Results also show that bugs occurred due to incorrect input validation are easier to find.

Keywords: BugsJS, JavaScript, Fault localization, Bug Taxonomy

Introduction

Used by many developers, JavaScript gained popularity over the years. Due to its popularity, frameworks built on JavaScript are changing rapidly. Considering this and the language's asynchronous nature developers are more likely to make mistakes.

Various techniques have been proposed for testing applications written in this language, however, there is a lack of bug benchmarks proposed for JavaScript. In my work, I used the relatively new bug benchmark called BugsJS [1] to compare how well Fault Localization algorithms perform.

Debugging in complex software programs is an expensive task, still, it is one of the most crucial parts of software development. Spectrum-Based Fault Localization (SBFL) can reduce the time spent on finding bugs by sorting code elements in a suspiciousness order, thus leading developers into the right direction.

I was interested in whether the features of bugs affect the efficiency of SBFL algorithms, and if they do which are these. This leads us to the research question:

> **RQ**: Are there bug features on which SBFL algorithms perform significantly better or worse than on others?

In this paper, I will separate cases where the code element has a rank less or equal to three (top-3), less or equal to five (top-5), less or equal to ten (top-10), and when it is over ten (other). This is commonly referred to as top-N. Results show us that bugs existing due to incorrect input validation are more likely to be successfully localised than other types, while incorrect file path ranked higher in the other (ranks over 10) ranges.

In the next section, I briefly explain how Fault Localization works and what BugsJS and Bug Taxonomy are. In Section Threats to Validity, I draw the conclusion based on the results introduced in Section Results, thus answering the research question.

Background

Spectrum-Based Fault Localization

There are various studies on Fault Localization techniques. [2, 3, 4] In my work I focused on one of the most popular FL techniques, called Spectrum-Based Fault Localization. It takes information about the program execution i.e. program's spectra. The coverage matrix represents the spectrum, whose rows demonstrate the test cases and columns show the code elements (in our case functions). An element of the matrix is 1, if the function is covered by test case, otherwise it is 0. In another matrix the test results are stored, where 0 means the test case passed and 1 when it failed.

Using these matrices, four basic statistical numbers are calculated for each ϕ function:

1. ϕ_{ep}: number of passed tests covering ϕ
2. ϕ_{ef}: number of failed tests covering ϕ
3. ϕ_{np}: number of passed tests not covering ϕ
4. ϕ_{nf}: number of failed tests not covering ϕ

Using these four numbers, Tarantula [5], Ochiai [6] and DStar [7] provide a ranked list of functions as an output. Whichever function ranked the highest is the most suspicious of containing a bug.

BugsJS

BugsJS is a JavaScript bug benchmark, which includes 453 bugs from 10 open-Source Node.js server-side projects. Originally, the "per-test" measurement was extremely slow. It was re-instrumenting the code in every test run. Therefore I used the hook function of Mocha to run the tests without re-instrumenting the code. Due to this, I was able to use 7 out of 10 projects. The reason for this is, there is a conflict between the Node.js versions of the new instrumentation and the projects.

Bug Taxonomy

The authors of BugsJS made a classification on JavaScript bugs based on their features. In Table 1, I listed these categories and the ranks for each formulae proposed in Section . Then I summed the number of bugs in every subcategory and main category.

Table 1: Number of labels for each metric

		TOP-3 (#)			TOP-5 (#)			TOP-10 (#)			OTHER (#)			# OF BUGS IN SUBCATEGORY	# OF BUGS IN MAIN CATEGORY
Main category	**Sub category**	**Tarantula**	**Ochiai**	**DStar**	**Tarantula**	**Ochiai**	**DStar**	**Tarantula**	**Ochiai**	**DStar**	**Tarantula**	**Ochiai**	**DStar**		
incorrect feature implementation	configuration processing	11	11	11	14	14	14	16	17	17	3	2	2	19	182
	incorrect data processing	29	29	29	33	35	36	37	39	39	12	10	10	49	
	incorrect handling of regex expressions	3	3	3	3	3	3	4	4	4	1	1	1	5	
	incorrect filepath	4	4	2	5	5	3	6	7	5	1	0	2	7	
	incorrect input validation	52	54	55	65	66	66	74	74	74	4	4	4	78	
	incorrect output	9	9	9	9	10	11	13	14	14	2	1	1	15	
	performance	1	1	1	2	2	2	2	2	2	0	0	0	2	
perfective maintenance		4	4	4	5	5	5	5	5	5	0	0	0	5	5
generic	missing type conversion	1	1	1	1	1	1	1	1	1	0	0	0	1	18
	variable initialization	6	6	6	6	6	6	6	6	6	1	1	1	7	
	data processing	3	3	3	3	3	3	4	4	4	0	0	0	4	
	return statement	2	2	2	2	2	2	2	2	2	0	0	0	2	
	typo	2	2	2	3	3	3	3	3	3	0	0	0	3	
	loop statement	1	1	1	1	1	1	1	1	1	0	0	0	1	
incomplete feature implementation	configuration processing	11	11	11	14	14	14	16	17	17	3	2	2	19	155
	error handling	5	5	5	8	8	8	9	9	9	0	0	0	9	
	incomplete data processing	8	8	7	9	9	8	12	12	11	2	2	3	14	
	incomplete output message	1	1	1	1	1	1	2	1	1	0	1	1	2	
	missing input validation	64	68	70	85	87	87	99	99	99	12	12	12	111	

Results

In this section, I will present the results and show which taxonomy types are significantly different in terms of ranking.

First, I investigated the main features of JavaScript bugs. Table 1 shows the numbers counted of taxonomy labels for each metric. Bugs labelled with *"perfective maintenance" or "generic"* occurred fewer than bugs that happened due to *"incomplete-" or "incorrect feature implementations"*. As a result, I investigated the subcategories of bugs labelled with the latter.

In the first main category, which is the *"incorrect feature implementation"* there are 8 sub categories. To verify if there is a statistical significant difference among the number of ranking associated with these subcategories, I used Fisher's exact test. It is a statistical significance test, which is one of the non-parametric methods and is used in the analysis of contingency tables.

The null hypothesis is that the ratio of belonging to a top-N range is not higher for one category than the others. When that value is less than the chosen significance level, i.e. 0.05, then the null hypothesis is rejected. Therefore, the ratio of belonging to a range is different for a given label than the others. In almost all cases, the same taxonomy labels were significantly different for every formulae, whenever it is not, I will list the name of the formulae as well. Running Fisher's exact test, the following labels were significantly different in terms of ranking:

- **In top-3**: "incorrect input validation" and "incorrect filepath" with DStar.
- **In top-5**: "incorrect input validation" with all, and "incorrect filepath" with DStar.
- **In top-10**: "incorrect input validation" with DStar and Tarantula, "incorrect filepath" with DStar, "incorrect data processing" with Ochiai and Tarantula.
- **Other**: "incorrect data processing" with Ochiai and Tarantula, "incorrect input validation" with DStar and Tarantula, "incorrect filepath" with DStar.

When function level coverage is used, then FL algorithms can localize faulty code elements in the top-1 category, however I did not include it in the paper. The reason for that is: none of the bug taxonomy categories were significantly different in top-1. None of the subcategories of *"Incomplete feature implementation"* were significantly different.

This test only determines if there is a difference in probability, however it does not show its direction. For better demonstration I counted the number of labels occurring in the non-overlapping rank-ranges as well.

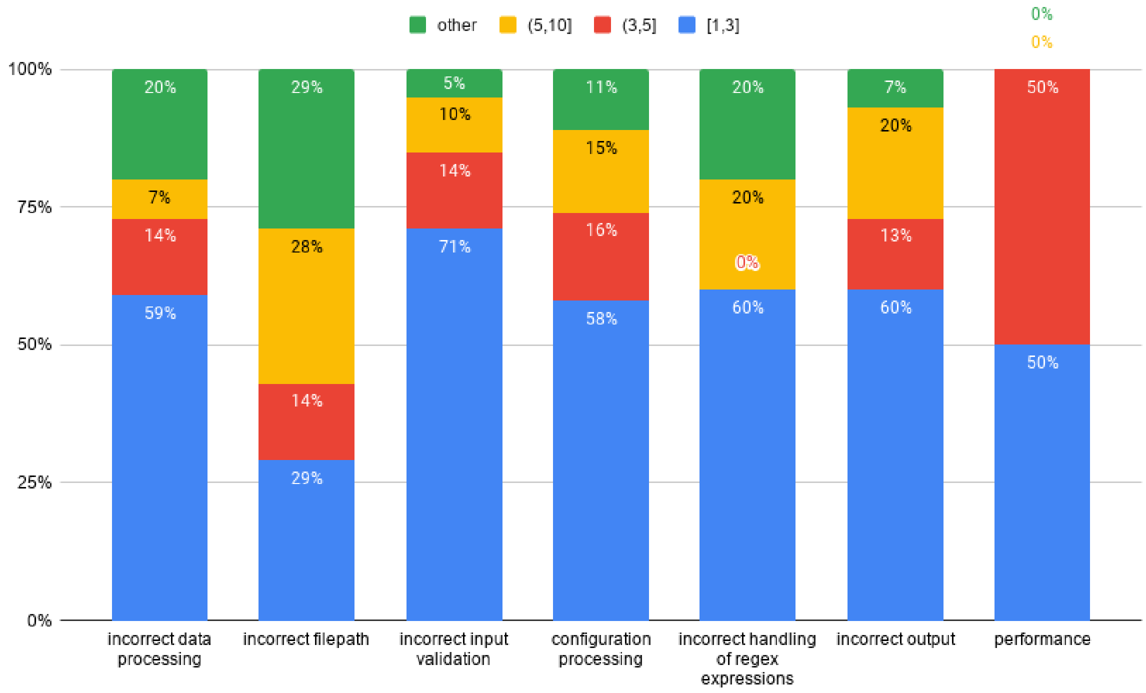


Figure 1: DStar interval statistics.

Looking at Figure 1, we can determine whether bugs with certain taxonomy labels are easier to find for the FL algorithm that implements the DStar formulae.

DStar ranked buggy methods, whose bugs occurred due to *"Incorrect input validation"*, in the top-10 category more often than buggy methods with different taxonomy labels. Taking the results from Fisher's exact test into consideration, we can say that such buggy methods are more successfully localised by DStar.

On the other hand, DStar ranked less the buggy methods with the type *"Incorrect filepath"* in the top-10 category, and more in the other section. We can see, it is significantly worse than other types, i.e. it is harder for DStar to localise bugs with this taxonomy label.

Conclusion

First I investigated the main taxonomy categories. Among the four categories, there are two that occurred fewer than the other two. Therefore, I investigated the remaining two, i.e. *"Incorrect feature implementation"* and *"Incomplete feature implementation"*.

Among the subcategories of the first there were two significantly different categories, however, none of the latter's subcategories were significantly different.

"Incorrect input validation" is significantly easier and *"Incorrect filepath"* is significantly harder to find for DStar. Therefore, answering my research question:

> **RQ**: There is one bug feature that is easier to find for DStar. Buggy methods that happen because *"Incorrect input validation"* are more successfully localised in the top-10 range. There are two bug features that are harder to find for SBFL algorithms. *"Incorrect filepath"* is less likely to be successfully localised in top-3, top-5 and top-10 ranges. *"Incorrect data processing"* bugs ranked lower in the top-10 ranges.

These results can help researchers implement new, more robust FL algorithms that take bug taxonomies into consideration.

Threats to Validity

While buggy methods that happened due to *"incorrect outputs"* seem just as good (i.e. easier to find for DStar) as *"Incorrect input validation"*, we can see there were only 15 bugs with such label. Therefore, statistically it could not be significantly different. If there were more bugs, labelled with *"incorrect outputs"* it would probably be significantly different.

Bugs in the benchmark were labelled with bug taxonomy types by the authors, however all authors participated in this manual activity to counter for personal bias.

References

[1] Péter Gyimesi, Béla Vancsics, Andrea Stocco, Davood Mazinanian, Árpád Beszédes, Rudolf Ferenc, and Ali Mesbah. BugsJS: a benchmark of JavaScript bugs. In *Proceedings of the 12th IEEE Conference on Software Testing, Verification and Validation (ICST'19)*, pages 90–101, April 2019.

[2] Priya Parmar and Miral Patel. Software fault localization: A survey. *International Journal of Computer Applications*, 154(9), 2016.

[3] Higor Amario de Souza, Marcos Lordello Chaim, and Fabio Kon. Spectrum-based software fault localization: A survey of techniques, advances, and challenges. *ArXiv*, abs/1607.04347, 2016.

[4] Spencer Pearson, José Campos, René Just, Gordon Fraser, Rui Abreu, Michael D Ernst, Deric Pang, and Benjamin Keller. Evaluating and improving fault localization. In *Proceedings of the 39th International Conference on Software Engineering*, pages 609–620. IEEE Press, 2017.

[5] James A. Jones and Mary Jean Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, ASE '05, pages 273–282, New York, NY, USA, 2005. ACM.

[6] Rui Abreu, Peter Zoeteweij, Rob Golsteijn, and Arjan J.C. van Gemund. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software*, 82(11):1780 – 1792, 2009. SI: TAIC PART 2007 and MUTATION 2007.

[7] W. E. Wong, V. Debroy, R. Gao, and Y. Li. The DStar method for effective software fault localization. *IEEE Transactions on Reliability*, 63(1):290–308, 2014.

An Industrial Application of Autoencoders for Force-Displacement Measurement Monitoring

Balázs Szűcs and Áron Ballagi

Abstract: The applications of artificial intelligence and neural networks in the industrial process monitoring and supervision are on the rise. One potential use case of these technologies are the anomaly detection in processes and measurements, without the need of pre-programming well defined patterns and supervision functions, thus unexpected events can be detected dynamically. In this paper we present a novel, neural network based method for the monitoring of press-in and joining processes. The new method, in contrast with the classical approaches, which are using envelope test or window functions, the autoencoder based approach is capable to detect unexpected events and anomalies, which are cannot be pre-programmed. By applying the above mentioned method, a higher level of quality assurance can be achieved. We present the new method through the example of force-displacement monitoring of mounting a sealing ring.

Keywords: machine learning, anomaly detection, neural networks, autoencoder, measurement, monitoring, industry 4.0

Introduction

In the series production, enormous amount of data generated by the in- and post-process measuring systems every minute. Due to the volume of the data the process monitoring is at a very high degree of automation, but there are anomalies which the monitoring systems cannot be prepared beforehand. Joining and assembly processes like pressing, riveting, clinching and caulking are often monitored with force-displacement measurements. These processes are often characterized in that certain force and/or way points have to be reached or have to be avoided to achieve a functionally reliable joining. A typical way of these kind of process monitoring is to monitor the force-displacement curves with envelope or window functions. These functions are programmed based on empirical values from well prepared manufacturing tests. A problem with this kind of process monitoring is that it lacks the capability to deal with unexpected anomalies outside the window functions. Measurements can still be classified as OK if the curves do not violate the predefined rules of the windows, but there are clear and visible anomalies outside the supervised sections. In Figure 1 the OK-NOK measurement curves can be seen. The first window monitors the sliding of the sealing ring, the second the first contact of tone wheel chamfer, and the third monitors the end position.

These windows and window rules are predefined, the curve must enter and enter from each window from a specific direction. As we have seen, the blue curve does not violate the rules of the windows, although the process is clearly faulty. Our goal is to detect all of the anomalies without the need of defining new windows.

Autoencoders

Autoencoders (Figure 2) are a special type of artificial neural networks, which are used to efficient data coding, dimensionality reduction and denoising. These neural networks has an input layer, an output layer and one or more hidden layers connecting them. Through the learning process, the data from the input is sampled and encoded by the hidden layers of the network, thus learning a higher level representation of the input data. The output layer has

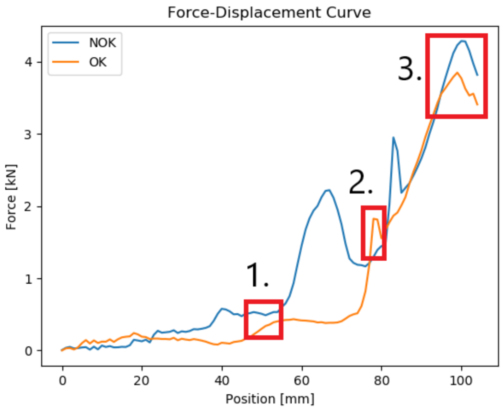


Figure 1: Force-Displacement Curves: OK *(blue)*, NOK *(orange)*, (1. Sealing ring sliding: enter left, exit right; 2. First contact part chamfer: enter bottom, exit right;, 3. End position: enter bottom)

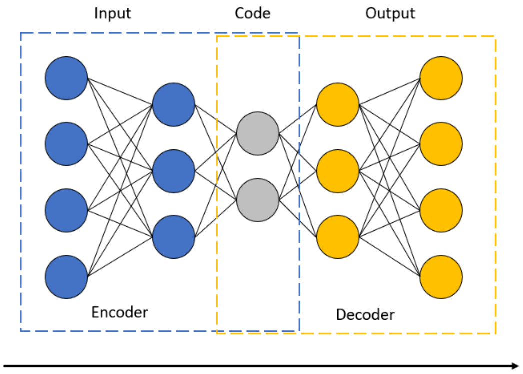


Figure 2: Basic autoencoder architecture

the same number of neurons as the input layer, the goal of this arrangement is to reconstruct the input data on the output of the network. The model uses backpropagation to minimize the difference between the input and the output (Eq.: 1 - 3). These behaviour makes these models efficient at denoising and dimension reduction tasks.

$$\phi : \mathcal{X} \rightarrow \mathcal{F} \tag{1}$$

$$\psi : \mathcal{F} \rightarrow \mathcal{X} \tag{2}$$

$$\phi, \psi = \underset{\phi,\psi}{\arg\min} \, \| X - (\psi \circ \phi) X \|^2 \tag{3}$$

The encoder function ϕ (Eq.: 1), maps the input data $\mathcal{X}$, to the latent space (hidden layer) $\mathcal{F}$, which is present at the code layer (bottleneck) of the network. The decoder function ψ (Eq.: 2), the opposite of (Eq.: 1) maps the latent space $\mathcal{F}$ at the bottleneck to the output. The output $\mathcal{X}$ is the same as the input function. Thus, the network trying to recreate the original input data after some generalized non-linear compression. The aim of the autoencoder is to select our encoder and decoder functions in such a way that we require the minimal information (minimizing the reconstruction error) to encode the input data such that it be can regenerated on the output of the network.

Classification of force-displacement curves with autoencoders

Our method is illustrated by the process of pressing a sealing ring onto a crankshaft. The sealing ring prevents the oil leakage from a combustion engine. The pressing process is supervised with a conventional window function monitoring. The nature of the process and the measurement curves are well known. The window functions are placed in a well-defined position of the pressing process, such as the positioning and sliding of the sealing rings, the point of first contact between the sealing ring and the crankshaft, and the end point of the pressing (Figure 2). The anomalies during the pressing process or the faults on the surface of the crankshaft can damage the sealing ring, thus the sealing ring cannot seal effectively. These faults are monitored with the window functions, but the faults outside the windows cannot be detected. To detect the unforeseen faults in the process we used the denoising behaviour of the autoencoder.

The data

The input data contains 105 samples per measurement, each sample corresponds to a specific force value in a specific position. The values of the samples varies from 0 to 4 kN. To accelerate the convergence of the model, thus shortening learning process, we scaled the input data to the range 0 to 1. We used 26365 measurements to train the model.

The model

We used a baseline autoencoder architecture. In the encoder part of the network the input layer consists 105 nodes, the first hidden layer 50 nodes, and the second hidden layer 20 neuron. The decoder part of the model is the reverse of the encoder part, with 20, 50 and 105 neurons. We used rectified linear activation function, for optimization we used the Adam optimizer and the loss function of the model was mean squared error. We trained the model with the batches of 32 samples, which only consists OK measurements. After 15 epochs the model learned to reconstruct the input data. The model was built with the frameworks TensorFlow and scikit-learn in Python language.

Results

We trained the model only with OK measurements, that means the model reconstructs the input data on the output with relative small error. When the model receives a measurement containing anomaly on the input, the reconstruction error is one or two orders of magnitude higher than in the case of OK measurements.

We classified the measurements by the error rate. The threshold value of reconstruction error was chosen to be 0.1 based on the descriptive statistics of the reconstruction error of 200 true positive OK and NOK samples. With this method we achieved a full curve length monitoring, which can detect anomalies outside of window functions. The classification can be done by a threshold value or a simple machine learning algorithm too. Physical experiments showed correlation between the degree of damage on the parts and degree of anomalies in the pressing curves, but further researches are required to create an adequate physical model.

Conclusion

The above introduced method presents a simple, yet powerful process monitoring technique. The autoencoder based approach is capable to detect unforeseen faults and anomalies in processes, where the nature of the measurement curves is well known. With this procedure the

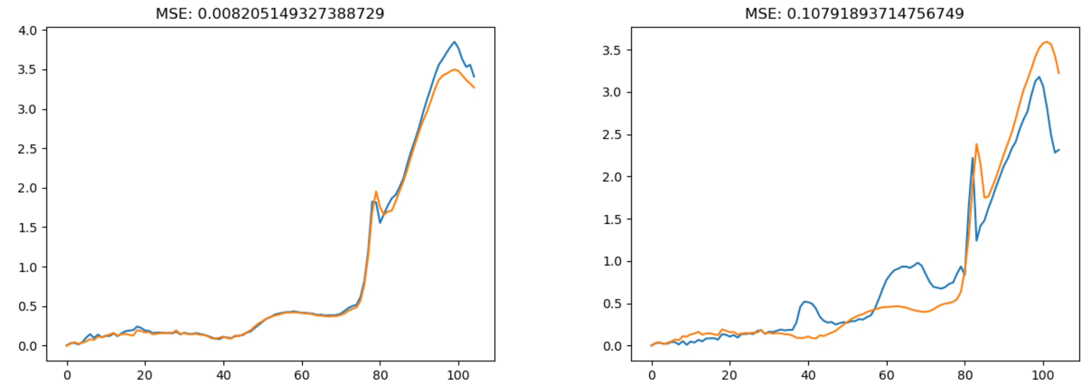


Figure 3: Reconstruction of measurements, blue: original curve, orange: reconstructed curve. The reconstruction error is relative small in the OK case *(left)* and one or two orders of magnitude higher in the NOK case *(right)*

pre-programming of window functions is avoidable, thus a flexible monitoring system can be made. These algorithms are reliable, but they can only classify the measurements after the process is done. Further research is recommended on other neural network architectures, like recurrent neural nets to detect the anomalies in-process.

References

[1] Kramer, Mark A. (1991), Nonlinear principal component analysis using autoassociative neural networks, *AIChE Journal. 37 (2): 233-243. doi:10.1002/aic.690370209.*

[2] Vincent, Pascal; Larochelle, Hugo (2010), Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion, *Journal of Machine Learning Research. 11: 3371-3408.*

[3] Frey, Brendan; Makhzani, Alireza (2013), k-Sparse Autoencoders, *arXiv:1312.5663.*

[4] Gondara, Lovedeep (2016), Medical Image Denoising Using Convolutional Denoising Autoencoders, *IEEE 16th International Conference on Data Mining Workshops (ICDMW). Barcelona, Spain: IEEE: 241-246. doi:10.1109/ICDMW.2016.0041. ISBN 9781509059102.*

[5] Pedregosa et al. (2011), Scikit-learn: Machine Learning in Python, *JMLR 12, pp. 2825-2830*

[6] Martín et al., (2015) TensorFlow: Large-scale machine learning on heterogeneous systems, *Software available from tensorflow.org*

Overview of Artificial Neural Network Abduction and Inversion Methods

Bence Bogdándy and Zsolt Tóth

Abstract: Defining the intent and the reasoning behind the decisions of machine learning models is a very important step towards general machine intelligence. Since the rise of modern machine learning and big data, this subject has not received much spotlight. Induction, deduction and abduction reasoning are the three forms of logical inferences. Inductive reasoning can be used to generate theories or rules based on previous premises and consequences. Deductive reasoning is used for finding the consequences of premises based on known theory. Abductive reasoning generates and evaluates the possible premises based on an existing theories and conclusions.

Artificial neural networks are capable function approximators and can be used in order to implement induction and deduction processes. These networks can be used for modeling complex systems using only the input and output of a system without mathematical analysis. Neural Network abduction aims to find one, or all possible explanations, for certain outcomes of the function. Inversion of artificial neural networks allows us to recommend input settings for systems after their behavior was modeled with neural networks. The connections and similarities between artificial neural network abduction and invertion has not been explored before.

Keywords: artificial neural network, machine learning, inversion, abduction

Methods

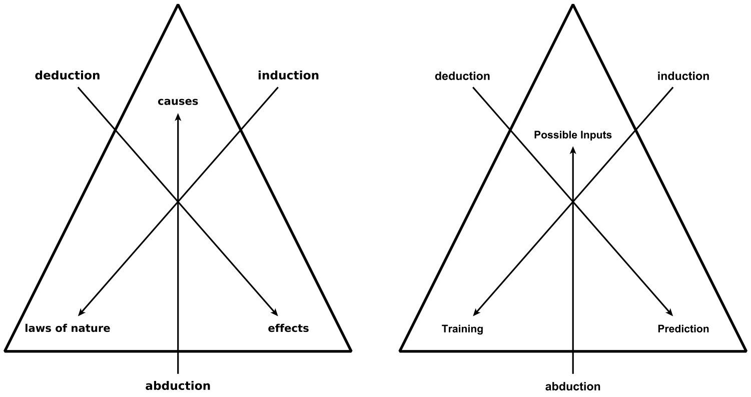
Logical Inference

Logical Inference was originally described as the combination of induction and deduction. Over centuries of scientific research, abductive reasoning has been introduced into modern logic. Inductive reasoning is the process of collecting all the available premises of a given conclusion. In contrast, deductive reasoning uses set conclusions and hypotheses. In the event that previously unknown premises are introduced, deduction can be used to connect premises to certain events. The triangle of the logical inferences can be seen on Figure 1a.

Abductive reasoning is the newest, third process in the triangle of mathematical logic inferences. Abductive reasoning is non-monotonic, which provides an interesting challenge during the examination of the inferences it proposes. In this process, the conclusions and the premises are known and the task is to explain the prior hypotheses tied to certain conclusions. As a consequence, the set of available abductive inferences provide no definitive answer to what prior events caused certain outcomes. With the prior explanation of the logical inferences, the task is to find one, or all elements in the set of abductive infereces, in order to provide an explanation for consequences.

Artificial Neural Networks

Artificial Neural Networks aim to recreate the processes of the human brain in order to solve well specified problems. Training of neural networks is a supervised learning process where the weights of each neuron are calculated iteratively in order to minimize the difference of actual and expected outputs. A trained neural network is used for prediction which is a function that takes inputs and the trained network connections in order to approximate



(a) Mathematical inferences

(b) Artificial neural network functions

outputs. Training process can be done by multiple different optimization methods, but in most cases, variants of the back-propagation algorithm are used.

Artificial Neural Network and Logical Inferences

The logical inference triangle contained inductive, deductive and abductive reasoning.

The translation between artificial neural network functions and logical inferences can be examined on Figure 1b. Artificial neural network functions achieve the same goals as the forms of reasoning in a different context.

The induction process of a neural network can be interpreted as the artificial neural network training. Instead creating hypotheses from premises and conclusions, neural network training approximates a function which creates weighted connections between given inputs and outputs. Prediction can be interpreted as the deductive reasoning in logical inferences, as it creates conclusions based on given hypotheses and premises. After these considerations, it would be logical to assume that abduction would be a natural component of neural networks.

Abduction creates premises, based on existing conclusions and hypotheses. Many possible premises can exist for a combination of conclusion, hypotheses and different inputs can produce the same outputs in an artificial neural network.

Results

Neural-Symbolic Learning Systems

Garcez et. al [3] described multiple different solutions to the problem of artificial neural network abduction. The described solutions are collectively called Neural-Symbolic Learning Systems [2]. The first approach uses connectionist modal logicand is based on the works of Gabbay and Woods [1]. The approach creates alternate paths to the input node by separating different explanations for the input. Connectionist modal logic introduces possibility and necessity to traditional logic. Abductive logic programming [5] approach uses a set of integrity criteria for a set of abductible inputs. The implementations of these abductive logic programs create abductive neural networks. The networks use special neural structure and counters in order to explore possible explanations for an output.

Artificial Neural Network Invertion

Artificial neural network inversion can be described as the inversion of the approximated function. Inversion of artificial neural networks aims to find a single or multiple input values where the neural network yields the desired output. During inversion, the output and the weights are fixedand the input values are calculated.

Jensen et. al [4] presented an overview of some of the available inversion methodsand corresponding real-life applications. Two major categories of artificial neural network inversion methods are distinguished which are *Single-element search* and *Multi-element search*.

Single-element search solutions use discrete optimization algorithms and can only find a single possible input combination for a given output. The *multi-element search* search the input hyperspace for possible input vectors and are based on *heuristic*and *metaheuristic* optimization algorithms.

William-Linder-Kindermann

The *Williams-Linder-Kindermann* [7, 6] single-element inversion is an algebraic optimization method which produces one possible combination of inputs for a specific output. The algorithm uses a modified back-propagation optimization to determine an input for a given output. The *WLK* algorithm is based on a gradient descent which is shown by Equation 1 where i_k^t denotes the kth neuron in layer t. The step size is represented by η and the direction is defined by the fraction which represents the error.

$$i_k^{t+1} = i_k^t - \eta \frac{\partial E}{\partial i_k^t} \tag{1}$$

The error $\partial E / \partial i_k^t$ is propagated back to the input layer. Equation 2 presents the formula of the error calculation. φ' is the derivative of the activation function.

$$\delta_j = \begin{cases} \varphi'(o_j)(oj - t_j) & j \in O \\ \varphi'(o_j) \sum_{m \in H,O} \delta_j w_{j,m} & j \in H, I \end{cases} \tag{2}$$

Genetic Algorithm Inversion

Multi-element search methods use heuristic and metaheuristic algorithms in order to iteratively search for possible input values. Inputs can be represented as a point in an n-dimension function, which are transformed by the neural network in order to return an m dimensional output. In order to find the possible inputs for a given output, evolutionary algorithms have to find boundaries that are assigned to a given output in the n-dimensional space. A solution is proposed by Reed, Russell D and Marks, Robert J [8]. The fitness function aims to evenly distribute the generated entities along a manifold. A repelling force between entities calculated based on the relative distance between the point and the manifold, with respect to the distance to other points as well.

Overview

The abductive exploration of neural networks has been an actively researched topic since the end of the 1980's. A number of implementation has been created for the abduction of artificial neural networks. While not explicitly called abduction, neural network inversion provides an implementation of abduction.

The mentioned papers on abduction and inversion provide a comprehensive look at how this topic evolved over time. To our best knowledge, the connections between abductive reasoning and neural network inversion have not been discussed before.

Discussion

In this paper, the definitions of logical inferences were overviewed, with special attention given to abductive reasoning and artificial neural network invertion. Furthermore, the connections between the types of logical reasoning and artificial neural network processes were examined. Abduction provides a method of finding potential premises leading to certain consequences. Neural network abduction can be achieved using a multitude of different methods. In the core of all methods is the desire to predict possible premises for given consequences and neural network structure. Abductive reasoning is a logical inference and therefore the most obvious way would be to try to solve the problem with the tools of mathematical logic. Garcez [2] showed that existing trained neural networks can be converted into abductive networks using two techniques. Neural network inversion has been a different scientific topic. Inversion methods are created in order to find the inverse of the approximated function. The inverse function is capable of delivering potential inputs of the function. This is achieved by using heuristic inversion methods on a trained neural network and an output. Two methods of inversion were examined in this paper. The first method, called the *WLK*, is capable of finding a single combination of input values. The second method uses a metaheuristic genetic algorithm. This method is volatile, as every run of the genetic algorithm can result in a different set of input values, given its metaheuristic nature. In this paper, the connections between the two topics have been discussed. Examples and different abduction, and invertion were examined, in order to find similarities between the methods. This paper provides a basic overview of methods in both topics and shows that their functionalities can be interchanged.

References

[1] D Gabbay and J Woods. The reach of abduction: Insight and trial, 2005.

[2] Artur S d'Avila Garcez, Krysia B Broda, and Dov M Gabbay. *Neural-symbolic learning systems: foundations and applications*. Springer Science & Business Media, 2012.

[3] Artur S d'Avila Garcez, Dov M Gabbay, Oliver Ray, and John Woods. Abductive reasoning in neural-symbolic systems. *Topoi*, 26(1):37–49, 2007.

[4] Craig A Jensen, Russell D Reed, Robert J Marks, Mohamed A El-Sharkawi, Jae-Byung Jung, Robert T Miyamoto, Gregory M Anderson, and Christian J Eggen. Inversion of feedforward neural networks: Algorithms and applications. *Proceedings of the IEEE*, 87(9):1536–1549, 1999.

[5] Antonis C Kakas, Robert A. Kowalski, and Francesca Toni. Abductive logic programming. *Journal of logic and computation*, 2(6):719–770, 1992.

[6] Joerg Kindermann and Alexander Linden. Inversion of neural networks by gradient descent. *Parallel computing*, 14(3):277–286, 1990.

[7] Alexander Linden and J Kindermann. Inversion of multilayer nets. In *Proc. Int. Joint Conf. Neural Networks*, volume 2, pages 425–430, 1989.

[8] Russell D Reed and Robert J Marks. An evolutionary algorithm for function inversion and boundary marking. In *Proceedings of 1995 IEEE International Conference on Evolutionary Computation*, volume 2, pages 794–797. IEEE, 1995.

Evaluating and Analyzing MQTT Brokers with Stress-testing

Biswajeeban Mishra and Biswaranjan Mishra

Abstract: MQTT (MQ Telemetry Transport) is a simple, lightweight, open-source and widely used publish/subscribe type communication protocol for IoT systems. No matter which radio technology is used to deploy an IoT/Machine-to-Machine (M2M) network, all independent data generating end devices (sensors and actuators) must make their data available through the Internet for further processing, and send control information back. For this, they heavily rely on the special messaging protocols like MQTT designed for M2M communication within IoT applications. This study aims to evaluate the performance of several MQTT Broker implementations by putting them under stress-test. The evaluation of the servers is made in a realistic test scenario, and the comparison of the results is presented by different metrics (CPU, latency, message rates). We also provide a detailed discussion of the applied test conditions (QoS level, message throughput per client and message payload size, etc.). Our results showed that Mosquitto is the most efficient, optimized broker implementation, and Bevywise's MQTT Route is the second with respect to message processing capabilities under full CPU load in all QoS categories.

Keywords: Internet of Things, MQTT, MQTT Brokers, Performance Evaluation, Stress-testing

Introduction

Internet of Things devices are growing rapidly and increasingly becoming parts of our lives as the costs of sensors and actuators are on a continuous decline. Today, the footprint of IoT is significantly visible everywhere. It is rare to find any industry that does not get revolutionized with the rise of IoT. IoT Networks use several radio technologies like WLAN, WPAN, etc. to communicate at the lower level. No matter which radio technology is used to create an M2M network, the end device or machine must make their data available to the Internet [1]. The performance of M2M communication heavily depends on the underlying special Messaging protocols designed for M2M communication within IoT applications. There are many M2M data transfer protocols are available for IoT systems-MQTT, CoAP, AMQP, and HTTP, to name but a few. Amongst these M2M protocols, characteristics like lightweight, open, simple and easy to deploy, make MQTT an ideal protocol for communication in constraint environments like IoT [2].

MQTT was created in 1999, and became an OASIS standard in 2014. It works on the top of TCP/IP and uses 1883 and 8883 ports for unencrypted and encrypted communication respectively. Its minimalistic designing principles surrounding less 'network bandwidth' and 'device resource requirements' makes MQTT able to transmit telemetry information between constrained devices (devices having limited processing capabilities and memory) over low-bandwidth, high-latency or unreliable networks. The MQTT protocol has two kinds of network entities: a message broker(server) and client(a publisher or subscriber). MQTT based IoT devices/applications(publishers) send or publish messages on a topic head to a server called an MQTT broker; then the broker delivers the messages to those clients (subscribers) that have previously subscribed to that topic. There are many MQTT server/broker options available from different vendors. MQTT provides three Quality of service levels for delivering messages to an MQTT Broker and any client (ranging from 0 to 2). At QoS 0, a message will not be acknowledged by the receiver or stored and delivered by the sender. This is often called "fire and forget." It is the minimal level and guarantees the best delivery effort. At QoS 1, acknowledgment is assured. Data loss may occur. At least once delivery is guaranteed. At QoS 2, exactly once delivery of a message is guaranteed.

The aim of this paper is to evaluate the performance of several MQTT Broker implementations put under stress. The evaluation of the servers is made in a realistic test scenario, and the comparison of the results is presented by different metrics (CPU, latency, and message rates). We also provide a detailed discussion of the applied test conditions (QoS level, message throughput per client and message payload size, etc.). The remainder of this paper is organized as follows: Section 2 presents the test topology and details of the MQTT brokers put into stress, Section 3 details the test scenario, and Section 4 summarizes test results. Finally, Section 5 concludes the paper highlighting the main points of our future work.

Test Environment and Test Topology

In this experiment we fire messages at very fast rates from a publisher machine to MQTT server using a publishing script based on Paho Python MQTT library. The subscriber machine runs ’mosquitto_sub’ command line subscribers. ’mosquitto_sub’ is an MQTT client for subscribing to topics and prints the messages that it receives. During this test we redirect mosquitto_sub output to /dev/null to ensure resources are not consumed for printing MQTT messages on the terminal. Also, we have configured each subscriber to subscribe to all published topics to make sure a higher server load at reasonable message rates. Figure 1 presents the test topology.

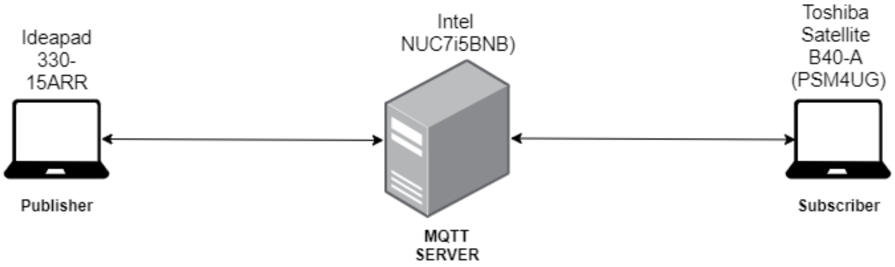


Figure 1: Test Topology

In our local test environment, we have used an Ideapad 330-15ARR, as a publisher machine running publisher threads. With 8 hardware threads, it is capable of firing messages at higher rates. At high message publishing rate with multiple publishers overall CPU usage stays below 70% during the entire course of the experiment on this hardware. MQTT broker was run on an Intel NUC(Intel Corporation, NUC7i5BNB) and subscribers run on an Intel Core 2 Duo machine(Toshiba, Satellite B40-A)Throughout the test. The CPU usage on the subscriber side doesn’t exceed 80%. No swap usage observed on the publisher, broker, and subscriber machines during the tests. Gigabit Ethernet (local network) was used to ensure there is no network bottleneck. See table 1 for hardware details of test environment.

Table 1: Hardware details of the test environment

HW Details	Publisher	MQTT Broker	Subscriber
CPU:	64 bit AMD Ryzen 5 2500U @3600 MHz	64 bit An Intel(R) Core(TM) i5-7260U CPU @ 2.20GHz	Intel(R) Pentium(R) CPU 2020M @ 2.40GHz
Memory:	8GB, SODIMM DDR4 Synchronous Unbuffered (Unregistered) 2400 MHz (0.4 ns)	8GB, SODIMM DDR4 Synchronous Unbuffered (Unregistered) 2400 MHz (0.4 ns)	2GB, SODIMM DDR3 Synchronous 1600 MHz (0.6 ns)

Test Scenario

This section sheds light on our test conditions and the number of brokers evaluated. Following test conditions are remain constant for all the brokers throughout the experiment - (a) Number of topics: 3 (via 3 publishers threads), (b) Number of publishers: 3, (c) Number of subscribers:15 (subscribing to all 3 topics), (d) Payload:64 bytes, (e) Topic names used to publish large number of messages: topic/0', 'topic/1', 'topic/2', (f) Topic used to calculate latency'topic/latency'. Latency is defined as the message transmission time from publisher to subscriber. We measured message delivery time by publishing an MQTT message on a topic different from the topics used to fire a large number of messages to stress the server. This is to ensure that the processing time of a currently queued messages on broker on a specific topic head doesn't affect the processing time of another message published on another topic head. A good implementation should efficiently handle processing of all message irrespective of rate of messages/topic. Moreover, this is close to the real world scenario of a broker experiencing extreme load conditions while a client is trying to publish a message on a random topic. In MQTT, every message gets transmitted as s single telemetry parameter.Hence in this experiment small payload size is chosen not to overload server's memory. during the entire test the message payload size is fixed and set to 64 bytes. All the brokers(servers) were put under stress-test with default configuration settings. The evaluated brokers are Mosquitto 1.4.15[2], ActiveMQ 5.15.8[4], HiveMQ Community Edition 2020.2[5], and Bevywise MQTT Route 2.0[3].

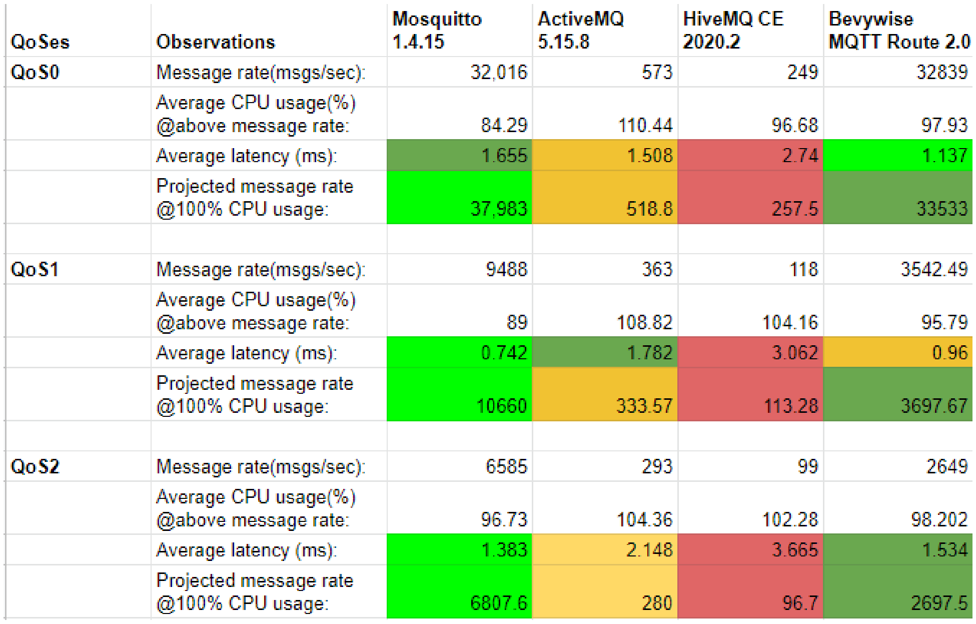
Test Results: A comparative Analysis

This section presents a comparative analysis of the performance of MQTT brokers based on the stress-test results. In our test, we found Mosquitto being a single-threaded implementation beats all other brokers by a huge margin in message processing rate across all QoS categories. It is the most efficient, optimized implementation with the least latency in QoS1 and QoS2 category among all brokers we have tested so far. Bevywise MQTT Route occupies the second position(after mosquitto) with respect to message processing capabilities @ 100% CPU load in all QOS categories. We also observed that it has lower latency/message delivery time compared to ActiveMQ and HiveMQ across all QOSes. It has shown better latency(less round the trip time) than Mosquitto in QOS 0. In this experiment, we have tried to limit the CPU usage around 100% for the process group to have a fair comparison with other brokers. The figure 2 presents a detailed comparative analysis of test-results.

The message-rate mentioned in the above figure indicates the number of messages/second required to push the broker to approximate 100% total CPU usage. It is to be noted that some broker implementations like HiveMQ CE, ActiveMQ can scale up automatically to utilize available resources on the system. These brokers create multiple threads or sub-processes to handle the higher message load. The CPU utilization data presented in this table is for the process group(consisting of all sub-processes/threads) of MQTT Server. So, CPU utilization percent for these brokers can go up to 400% on a machine with 4 cores. However, we are not considering that in our experiment as other brokers in the test are single-threaded and don't scale up automatically.

Conclusion

M2M protocols are the backbone of communication in IoT systems. There are many M2M communication protocols are available such as MQTT, CoAP, AMQP, and HTTP. In this paper, we investigated and analyzed the performance of MQTT brokers in terms of projected message rate @100% CPU usage and the average time taken for message transmission by putting them



QoSes	Observations	Mosquitto 1.4.15	ActiveMQ 5.15.8	HiveMQ CE 2020.2	Bevywise MQTT Route 2.0
QoS0	Message rate(msgs/sec):	32,016	573	249	32839
	Average CPU usage(%) @above message rate:	84.29	110.44	96.68	97.93
	Average latency (ms):	1.655	1.508	2.74	1.137
	Projected message rate @100% CPU usage:	37,983	518.8	257.5	33533
QoS1	Message rate(msgs/sec):	9488	363	118	3542.49
	Average CPU usage(%) @above message rate:	89	108.82	104.16	95.79
	Average latency (ms):	0.742	1.782	3.062	0.96
	Projected message rate @100% CPU usage:	10660	333.57	113.28	3697.67
QoS2	Message rate(msgs/sec):	6585	293	99	2649
	Average CPU usage(%) @above message rate:	96.73	104.36	102.28	98.202
	Average latency (ms):	1.383	2.148	3.665	1.534
	Projected message rate @100% CPU usage:	6807.6	280	96.7	2697.5

Figure 2: A comparative presentation of test results

under stress-test. Our results showed, Mosquitto being a single-threaded implementation beats all other brokers by a huge margin in message processing rate across all QoS categories. It is found to be the most efficient, optimized MQTT broker implementation among all the brokers we put into test. Bevywise MQTT Route occupies the second position(after Mosquitto) with respect to message processing capabilities @~100\% CPU load in all QOS levels, and broker implementations like HiveMQ CE, ActiveMQ can scale up automatically to utilize available resources on the system. In the future, it would be interesting to observe the performance of MQTT brokers in a cloud-deployed test environment. We plan to widen the scope of this research by putting more brokers under stress-test and create a generic, easily configurable message blaster for MQTT brokers.

References

[1] Nitin Naik, Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. IEEE International Systems Engineering Symposium, Vienna, pp. 1-7, 2017.

[2] MQTT Version 5.OASIS Standard, https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html. Accessed in March, 2020.

[3] Bevywise MQTT Route Developer's Guide, https://www.bevywise.com/mqtt-broker/developer-guide. Accessed in March, 2020.

[4] ActiveMQ 5 Documentation, https://activemq.apache.org/components/classic/documentation/. Accessed in March, 2020.

[5] HiveMQ Editions, https://www.hivemq.com/developers/community/. Accessed in March, 2020.

[6] Mosquitto man page, https://mosquitto.org/man/mosquitto-8.html. Accessed in March, 2020.

Iterative Operations on Footpoint Mappings

Csaba Bálint

Abstract: Surfaces defined by foot mappings $\mathbb{R}^3 \to \mathbb{R}^3$ are similar to signed distance functions [3] (SDFs) regarding surface construction and direct visualization methods. For the latter, the minimum and maximum operations approximate the SDF of the union and intersection of the argument distance functions. We present a fast iteration for the intersection operation on foot mappings to obtain precise footpoint for the resulting surface.

Keywords: Computer Graphics, Constructive Solid Geometry, Signed Distance Function

Introduction

From any sample point $\boldsymbol{p} \in \mathbb{R}^3$ a signed distance function (SDF) $f : \mathbb{R}^3 \to \mathbb{R}$ is a continuous function that evaluates to the signed Euclidean distance measured from the surface. That is

$$\big|f(\boldsymbol{p})\big| = d(\boldsymbol{p}, \{f = 0\}) \qquad (\forall \boldsymbol{p} \in \mathbb{R}^3),$$

where $d(\boldsymbol{p}, A)$ is the point-to-set distance, and $\{f = 0\}$ is the zero level-set. The sign encodes whether $\boldsymbol{p}$ is inside $\{f \leq 0\}$ or outside $\{f > 0\}$ which allows set-operations to be defined on SDFs. Let f, g SDF, then according to [3],

$$\begin{aligned} d\big(\boldsymbol{p}, \{f \leq 0\} \cup \{g \leq 0\}\big) \ &\geq \ \big|\min\{f(\boldsymbol{p}), g(\boldsymbol{p})\}\big| \qquad (\forall \boldsymbol{p} \in \mathbb{R}^3)\ , \\ d\big(\boldsymbol{p}, \{f \leq 0\} \cap \{g \leq 0\}\big) \ &\geq \ \big|\max\{f(\boldsymbol{p}), g(\boldsymbol{p})\}\big| \qquad (\forall \boldsymbol{p} \in \mathbb{R}^3)\ . \end{aligned} \tag{1}$$

The $\min(f, g) = \boldsymbol{p} \mapsto \min\{f(\boldsymbol{p}), g(\boldsymbol{p})\}$ function estimates the SDF of the union of $\{f \leq 0\}$ and $\{g \leq 0\}$ objects extremely well. For example, on the outside of both objects, the estimation is precise. For this reason, many SDF representations use $\min$ and $\max$ operations to combine primitive geometries into complex scenes [2]. However, the approximation is imprecise on the union for the $\min$ operation, and only exact within the intersection for the $\max$ intersection SDF approximation.

The precision can be quantified for any SDF by comparing the real distance-to-surface value to that of the function:

$$q_f(\boldsymbol{p}) := \frac{\big|f(\boldsymbol{p})\big|}{d(\boldsymbol{p}, \{f = 0\})} \qquad (\forall \boldsymbol{p} \in \mathbb{R}^3) \tag{2}$$

Signed distance function estimates (SDFEs) are defined using the above local precision. If there exists a $c > 0$ global precision such that $0 < c \leq q_f(\boldsymbol{p}) \leq 1$, then $f : \mathbb{R}^3 \to \mathbb{R}$ is an SDFE.

Distance representations can be directly ray-traced via various sphere tracing algorithms [3, 5, 3, 7]. SDFE precision measures the slowdown of the sphere tracing algorithm; however, computing $q_{\max(f,g)}(\boldsymbol{p})$ for the intersection operation can be expensive.

Methods based on distance transform rely on a discretization of the distance function and spreading the distance values by approximating the distance to the surface from the neighboring values [6, 7]. However, the local computations introduce errors that can result in even worse precision for set operations then the $\min$ and $\max$ SDF operations. To increase precision and reduce memory usage, we devised iterative algorithms that compute the distance to the intersection set from any point $\boldsymbol{p} \in \mathbb{R}^3$ to the intersection object. On the other hand, the presented algorithms require a different representation of the surfaces.

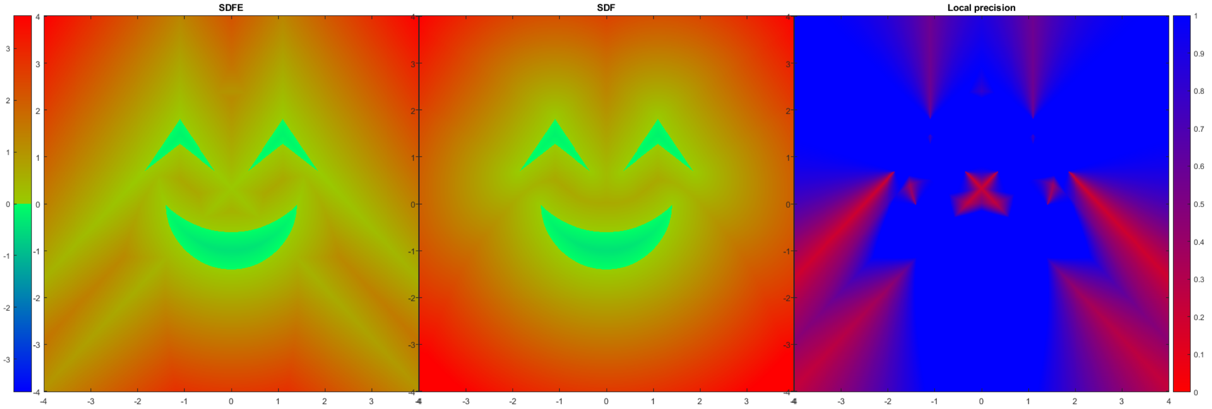


Figure 1: Left: 2D SDFE obtained through min and max set operations using transformations of a half-plane (line) and a circle primitive. Local precision is the ratio of the SDFE (left) and the exact SDF (middle) is displayed on the right signaling the slowdown of sphere tracing. Our footpoint iteration produced the middle image.

Foot mapping

The function $\boldsymbol{f} : \mathbb{R}^3 \to \mathbb{R}^3$ is a foot mapping if

1. $\mathbb{R}^3 \ni \boldsymbol{p} \mapsto \left\|\boldsymbol{f}(\boldsymbol{p})\right\|$ is a distance function
2. $\boldsymbol{f}(\boldsymbol{p} + \boldsymbol{f}(\boldsymbol{p})) = \boldsymbol{0}$ for all $\boldsymbol{p} \in \mathbb{R}^3$

This means that $\boldsymbol{f}$ returns a vector pointing to one of the closest points on the surface it defines. Thus $\boldsymbol{p}+\boldsymbol{f}(\boldsymbol{p})$ is the footpoint. Similarly to SDFs, the footpoint representation necessitates solid geometry information for the set-operations to be defined. Let us assume we can decide if $\boldsymbol{p}$ is inside $\boldsymbol{p} \in F \subset \mathbb{R}^3$ closed set or outside $\boldsymbol{p} \in \mathbb{R}^3 \setminus F$, where the boundary set is $\partial F = \{\boldsymbol{x} \in \mathbb{R}^3 \mid \|\boldsymbol{f}(\boldsymbol{x})\| = 0\} \subset F$.

Let $F \subset \mathbb{R}^3$ and $G \subset \mathbb{R}^3$ be two objects with the foot mapping $\boldsymbol{f} : \mathbb{R}^3 \to \mathbb{R}^3$ and $\boldsymbol{g} : \mathbb{R}^3 \to \mathbb{R}^3$, respectively. Our task is to produce a foot mapping $\boldsymbol{h} : \mathbb{R}^3 \to \mathbb{R}^3$ with $H = F \cup G$ or $H = F \cap G$ similar to (1). This paper only describes the intersection since the complement geometry has the same foot mapping and $H = F \cup G = \mathbb{R}^3 \setminus \big((\mathbb{R}^3 \setminus F) \cap (\mathbb{R}^3 \setminus G)\big)$.

Footpoint Intersection Iteration

If $\boldsymbol{p} \in F \cap G$ then, the intersection approximation is precise in (1), so

$$\boldsymbol{h}(\boldsymbol{p}) = \begin{cases} \boldsymbol{f}(\boldsymbol{p}) & \text{if } \|\boldsymbol{f}(\boldsymbol{p})\| \leq \|\boldsymbol{g}(\boldsymbol{p})\| \\ \boldsymbol{g}(\boldsymbol{p}) & \text{otherwise} \end{cases} \qquad (\boldsymbol{p} \in F \cap G)\ . \tag{3}$$

If the closest point to F from $\boldsymbol{p} \notin F \cap G$ is inside the G set, then that point is the closest point to $\boldsymbol{p}$ in the $F \cap G$ intersection. Thus,

$$\boldsymbol{h}(\boldsymbol{p}) = \begin{cases} \boldsymbol{f}(\boldsymbol{p}) & \text{if } \boldsymbol{p} + \boldsymbol{f}(\boldsymbol{p}) \in G \\ \boldsymbol{g}(\boldsymbol{p}) & \text{if } \boldsymbol{p} + \boldsymbol{g}(\boldsymbol{p}) \in F \end{cases} \qquad \big(\boldsymbol{p} \in \mathbb{R}^3 \setminus (F \cap G)\big)\ . \tag{4}$$

However, this still leaves some $\boldsymbol{h}(\boldsymbol{p})$ values for us to define via iterative algorithms. The idea of this naive midpoint approach is to step closer to the intersection and re-evaluate $\boldsymbol{h}$:

$$\boldsymbol{h}(\boldsymbol{p}) := \frac{\boldsymbol{f}(\boldsymbol{p}) + \boldsymbol{g}(\boldsymbol{p})}{2} + \boldsymbol{h}\left(\boldsymbol{p} + \frac{\boldsymbol{f}(\boldsymbol{p}) + \boldsymbol{g}(\boldsymbol{p})}{2}\right) \qquad \text{if not (3) or (4).} \tag{5}$$

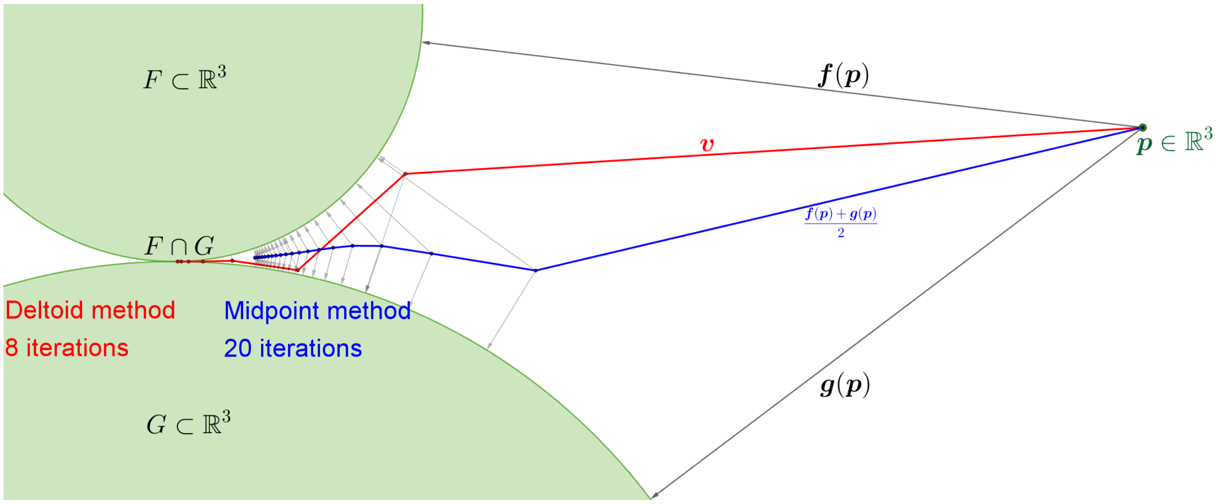

$F \subset \mathbb{R}^3$
$\boldsymbol{f}(\boldsymbol{p})$
$\boldsymbol{v}$
$\boldsymbol{p} \in \mathbb{R}^3$
$\frac{\boldsymbol{f}(\boldsymbol{p})+\boldsymbol{g}(\boldsymbol{p})}{2}$
$F \cap G$
Deltoid method
8 iterations
Midpoint method
20 iterations
$\boldsymbol{g}(\boldsymbol{p})$
$G \subset \mathbb{R}^3$

Figure 2: Comparison of the midpoint and deltoid footpoint iterations in a 2D scene where F and G objects are touching circles. The deltoid method converges much faster to the $F \cap G$ single-point intersection.

One can stop evaluating the recursion when one of (3) or (4) yield a value or after a predefined number of iterations. Figure 2 illustrates the convergence.

Deltoid iteration

Since the foot vectors are perpendicular to the surface, we are looking for a vector $\boldsymbol{v} = \alpha \cdot \boldsymbol{a} + \beta \cdot \boldsymbol{b}$ that are perpendicular to both $\boldsymbol{a} := \boldsymbol{f}(\boldsymbol{p})$ and $\boldsymbol{b} := \boldsymbol{g}(\boldsymbol{p})$ vectors. Denoting the dot product as $xy = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$ yields:

$$\begin{matrix} \langle \boldsymbol{v}-\boldsymbol{a},\boldsymbol{a}\rangle = 0 \\ \langle \boldsymbol{v}-\boldsymbol{b},\boldsymbol{b}\rangle = 0 \end{matrix} \quad \iff \quad \begin{bmatrix}\boldsymbol{a}^\top \\ \boldsymbol{b}^\top\end{bmatrix} \cdot \boldsymbol{v} = \begin{bmatrix}\boldsymbol{a}^\top \\ \boldsymbol{b}^\top\end{bmatrix} \cdot \begin{bmatrix}\boldsymbol{a} & \boldsymbol{b}\end{bmatrix} \cdot \begin{bmatrix}\alpha \\ \beta\end{bmatrix} = \begin{bmatrix}aa \\ bb\end{bmatrix} .$$

Solving the equation for α and β assuming $d := aa \cdot bb - ab \cdot ab \neq 0$ gives:

$$\begin{bmatrix}\alpha \\ \beta\end{bmatrix} = \begin{bmatrix}aa & ab \\ ba & bb\end{bmatrix}^{-1} \cdot \begin{bmatrix}aa \\ bb\end{bmatrix} = \frac{1}{d}\begin{bmatrix}bb & -ab \\ -ab & aa\end{bmatrix} \cdot \begin{bmatrix}aa \\ bb\end{bmatrix} = \frac{1}{d}\begin{bmatrix}bb \cdot (aa - ab) \\ aa \cdot (bb - ab)\end{bmatrix} . \tag{6}$$

For the deltoid footpoint iteration, we substitude $\boldsymbol{v} = \alpha \cdot \boldsymbol{a} + \beta \cdot \boldsymbol{b}$ from (6) into (5), so $\boldsymbol{h}(\boldsymbol{p}) := \boldsymbol{v} + \boldsymbol{h}(\boldsymbol{p} + \boldsymbol{v})$ unless (3) or (4) provide a foot vector.

Conclusion

Computing the SDF in Figure 1 with the midpoint approach about ten times slower compared to the deltoid method whilst achieving similar accuracy. Note that the iterations had to be nested to produce the CSG tree of set-operations. Figure 2 demonstrates the superior convergence of the deltoid algorithm. However, convergence and performance evaluation of the mentioned algorithms are out of the scope of this paper. Further algorithm variants and empirical results with various applications will be submitted in a subsequent paper.

Acknowledgements

EFOP-3.6.3-VEKOP-16-2017-00001: Talent Management in Autonomous Vehicle Control Technologies — The Project is supported by the Hungarian Government and co-financed by the European Social Fund. Supported by the ÚNKP-19-3 New National Excellence Program of the Ministry for Innovation and Technology. National Research, Development and Innovation Office HUNGARY Új Nemzeti Kiválóság Program

References

[1] John C. Hart. *Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. The Visual Computer*, 12:527–545, 1994.

[2] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design (3rd Ed.): A Practical Guide*. Academic Press Professional, Inc., San Diego, CA, USA, 1993.

[3] Benjamin Keinert, Henry Schäfer, Johann Korndörfer, Urs Ganse, and Marc Stamminger. *Enhanced Sphere Tracing*. In Smart Tools and Apps for Graphics, Giachetti A., (Ed.) The Eurographics Association, 2014.

[4] Csaba Bálint, Gábor Valasek. *Accelerating Sphere Tracing*. EG 2018 - Short Papers, Diamanti O., Vaxman A. (Ed.) The Eurographics Association, 2018.

[5] Róbert Bán, Csaba Bálint, Gábor Valasek. *Area Lights in Signed Distance Function Scenes*. EG 2019 - Short Papers, Cignoni P., Miguel E. (Ed.) The Eurographics Association, 2019.

[6] Ricardo Fabbri, Luciano da F. Costa, Julio Torelli, Odemir Bruno. *2D Euclidean distance transform algorithms: A comparative survey*. ACM Computing Surveys, 2008.

[7] Gábor Valasek. *Generating Distance Fields from Parametric Plane Curves*. 10th International Conference on Applied Informatics. Annales Mathematicae et Informaticae. 48. pp. 83-91.

Traquest model - a novel model for ACID concurrent computations

Dániel Balázs Rátai, Zoltán Horváth, Zoltán Porkoláb and Melinda Tóth

Abstract: Atomicity, consistency, isolation and durability are essential properties of many distributed systems. They are often abbreviated as the ACID properties. Ensuring ACID comes with a price: it requires extra computing and network capacity to ensure that the atomic operations are done perfectly, or they are rolled back.

When we have more strict requirements on performance, we need to give up the ACID properties entirely or settle for eventual consistency. Thanks to the ambiguity of the order of the events, such algorithms can get very complicated since they have to be prepared for any possible contingencies. Traquest model is an attempt for creating a general concurrency model that can bring the ACID properties without sacrificing a too significant amount of performance.

Keywords: ACID, concurrency, consistency, atomicity, concurrency model, fault tolerance

Introduction

The word Traquest comes from the words Request and Transaction. The core of the idea comes from the microservices architecture [8]. In the case of the microservices when we send a request, it can initiate some modifications in the global state in a transactional way. Microservices are mostly based on the request-response model. When the Request returns with no errors, that means the modifications in the global state are done, and the transaction is over. While if the response is an error, that means there were no modifications in the global state at all. Requests can make other requests, so like that more complex transactions can be assembled.

In such a request-response model, the ACID properties come at a high price. The service which calculates the response has to be sure that any write operations arising must be synchronized, committed and persisted before it can reply with a response. Of course, on the other hand, if ACID is not a requirement, the service can be very fast. In this case, the service can just read and write the state of the local server and answer to the client immediately (and perhaps synchronize the writes later if eventual consistency is a requirement).

However, it is not just the performance that can cause a problem. It is very tough to ensure atomicity itself when we nest the services. Suppose we have a service A calling two other services B and C. Suppose that B responses properly but C responses with an error. In this case, we can assume that C has rolled back properly, but B should be rolled back as well. In the request-response model, there is no mechanism to roll back a request after it has been responded. Therefore it is hard to chain more services properly when atomicity is a requirement. We can be sure to have a proper response if the happy path happens, but if there is an error arising at some of the chained requests our system can get easily stuck into an invalid intermediate state.

It seems there is a seemingly unsolvable dilemma between ACID properties and efficiency. The proposed Traquest model is an attempt to resolve this dilemma and therefore improve the efficiency of the ACID systems. The Traquest model is something similar to the request-response model. We can send requests to a Traquest, and the Traquest replies with an answer, but here the answer is not a simple response message, but rather an established parent-child connection between the two Traquests with a temporary response, a so-called Trasponse. When a Traquest gets a request it can immediately carry out read and write operations on the local server, and it can immediately reply with a Trasponse, however, of course, that still might take time to synchronize the effects of the operations with other servers. Therefore Trasponse is only a temporary response. One might think about a Trasponse like it would say*"I received and processed your request, here is my immediate temporary answer which is very probably correct, but I* *still might have to discuss it with the other servers if it is absolutely sure, I will let you know later if you can rely on this response with absolute certainty, meanwhile if you have any new information that might change the response I gave, please let me know so I can undo this."*

In Section 2, we describe the main concepts behind the Traquest model. Section 3 presents some related work. Finally, Section 4 concludes the paper.

The Traquest model structure

The request-response model is used on a local level as well and not only between different computing nodes. Asynchronous callback functions can behave equivalently. We can send the request content and the callback function as an argument, and the callback function can contain the response in an argument. This mechanism is often used to wrap network-based request-responses, but for local asynchronous operations as well.

However, callbacks can get complicated, when they are heavily used, and we want to handle the exceptional scenarios. To this end in computer science, *Future*, *Promise*, *Delay*, and *Deferred* refer to constructs used for synchronizing program execution in some concurrent programming languages. They describe an object that acts as a proxy for a result that is initially unknown, usually because the computation of its value is not yet complete. The term *Promise* was proposed in 1976 by Daniel P. Friedman and David Wise [9] and Peter Hibbard called it *Eventual* [5]. A somewhat similar concept *Future* was introduced in 1977 in a paper by Henry Baker and Carl Hewitt [3].

Traquests behave most similarly to *Promises*; therefore, we use them as a baseline for the explanation. Traquests, just like Promises, are placeholders for a temporarily unknown value. Promises just like Traquests can be chained together and depend on each other. However, in case of the Promises, once a Promise returns with a response, this response is final, it cannot be modified afterwards. On the other hand, Traquests can be strongly bonded together to form a tree structure, a so-called Traquest tree. A Traquest tree creates the transaction, and if any Traquest fails in the Traquest tree, all the Traquests are failing. When the Traquests are failing, they are not just returning an error, but they are ensuring that if there was any modification created by them, it will be appropriately rolled back so that the global state of the system will not be affected by half done transactions. To be able to achieve this Traquests are containing some additional mechanisms.

Promises contain two parts. One part is responsible for executing the required asynchronous algorithm, and another part is responsible for handling the response coming back. The response can run into two different branches depending on whether the executed asynchronous algorithm has succeeded or failed. Traquests has all of those parts, but they have some others as well to be able to cooperate with the whole Traquest tree.

Promises have a **Request** (the inner part containing the asynchronous algorithm which shall be executed) and a **Response** (the outer part, the placeholder object which will contain the response value of the asynchronous algorithm in the future) mechanism. The Promise has a **Resolve branch** to be called with the proper response value by the asynchronous algorithm when it has successfully finished and a **Reject branch** to be called in case of failure. The response has a **Then branch**to be called in case of success with the proper response value, and a **Catch branch** for failure handling.

Traquests also have **Request** and **Response** mechanism, but an additional **Binding** mechanism is used as well. The Binding mechanism is key for the Traquest model to be able to bind Traquests together into a tree structure. Like that, a Traquest tree can act as a single entity, and it can form a complete atomic transaction.

The **Request mechanism** has the following branches. The **Response branch** is the same as the resolve branch at the Promises. This should be executed, when the asynchronous algorithm successfully finishes with the proper value. The **Mistake branch** is slightly different from the reject branch of the promises. A mistake is called when a temporary failure happens. If there is a chance that the failure has happened only because the wrong order (e.g. division by zero) of the asynchronous operations, then a mistake should be triggered. Mistakes can be undone later, and the Traquest might rerun in proper order. The **Terminate branch** is used in case of final failures. This function terminates the whole Traquest tree and results in a full rollback on all the Traquests in the Traquest tree.

The **Then branch** of the **Response mechanism** is the same as the *Then* branch of the Promises. The **Catch branch** is similar to the *Catch* branch of the promises, but it is used explicitly for the mistakes. It can also avoid spreading up the mistake to parent Traquests or let it spread further. The **Finally branch** is executed no matter if the Traquest was properly committing or it was terminated.

The **Binding mechanism** of the Traquests has the following concepts:

Parent-child binding – When a Traquest is created the reference to the parent Traquest should be defined. If it is not defined, that means the created Traquest will be the root of the Traquest tree.

Undo – A mistake happens when an exception occurs because the Traquests are executed out of order. However, it can happen that the Traquest has already responded with a seemingly correct response, and an out of order conflict turns out only later. In this case, an undoing mechanism can be executed, which rolls back the necessary Traquests on the affected branch of the Traquest tree and re-executes them.

Rollback – A callback is provided for the case when the Traquest needs to revert the changes it made so far. If the Traquest did not create any changes directly to the global state just by calling other Traquests this part can be omitted because the rollbacks spread automatically on the Traquest tree.

Committing – It is a mechanism used when all the Traquest in the tree has returned, and a final commit can be initiated. This happens completely hidden and automatically when all the Traquests in the tree has returned.

Data protectors

So far, we were discussing that the Traquests are forming a tree structure. However, a tree structure by itself would never result in any conflicts which would be the core of the Traquest model to be able to handle them effectively. Conflicts are happening when two different processes are trying to read or write the same part of the global state. To this end, Data protectors were constructed. Data protectors are entities responsible for managing a given segment of the global state. They protect the given global state particle from conflicting reads and writes.

The goal of each branch of the Traquest tree is to interact somehow with the global state. Therefore, each branch, at some point ends up in a Data protector. When a Traquest reads or writes to a Data protector, the Data protector generates new Traquest containing the read or write operation. This new Traquest can be bounded to the original Traquest as a child; therefore, it becomes part of the Traquest tree.

When more Traquests are using the same Data protector, the Data protector can use the logical timestamps of the Traquests to decide which read or write operation should be answered first. If a Traquest with an earlier timestamp comes after a Traquest with a later timestamp has been already responded to, the Data protector can call the undo mechanism of the already responded Traquest and serve the newly requesting Traquest. Therefore, the Data protector can easily resolve any conflicts.

Furthermore, because all the conflicts are recognized and resolved at the Data protectors most of the conflict resolving features of the Traquests are used only by the Data protectors themselves. This way the increased complexity of the Traquests can be mostly hidden from the developers, and they do not need to care with the failure handling parts at all, except taking care about the *Finally branch* of the root Traquest. As a result, using Traquests can be as straightforward as using Promises or even more.

Consistency and Fault tolerance

Traquests interact with each-other using serializable data constructs only. Therefore, Traquests can be located on different computing notes as well, and they still can interact. Fault tolerance requires the replication of the different particles of the global state to several computing nodes. Traquests are perfect for creating replicas of a desired global state particle and managing them in a consistent way. It is enough to only add new Traquest tree branches to each write operations that replicates the operation on different computing nodes. Thanks to the atomic property of the Traquest tree, the state will always remain consistent. For the read operations, we do not need such replication since the writes are already ensuring the consistency.

Pipelining

Traquest can offer pipelining similarly to Proimises[1]. Traquests first can accomplish all the tasks they can do locally, collect what they cannot and send them in one single network message to another computing node. This way, the number of effectively necessary network messages can be highly reduced. This pipelining mechanism can be done automatically and hidden from the developers.

The Traquest tree branches created purely for replication does not need to contain any meaningful responses; therefore, their *Then* branch can be omitted as well. These Traquests are the so-called Tail Traquests. Tail Traquests combined with pipelining can result in a very efficient lazy synchronization, which means, that a sophisticated algorithm can be executed in a Traquest tree and the synchronization mechanism can happen in the background without blocking or slowing down the execution. The only restriction is that the synchronizations have to be finished before the whole Traquest tree commits. Therefore, pipelining can be very efficient, since all the synchronization steps can be awaited and executed in one round message over the network.

Related work

The "Layers" architectural pattern has been described in various publications [2], and it is the most widely used pattern in case of the enterprise web applications. When the Business layer executes the desired algorithm, it continuously has to access the Data access layer to read the global state and to write back the changed state. When the algorithm requires only a few iterations depending on each-other with the Data access layer, this causes no problem. On the other hand, when there are several depending steps, each read and write requires a roundtrip on the network. However, many databases (e.g. most of the SQL databases) can easily handle atomic transactions, the number of the necessary roundtrips implicates a massive limitation in the overall performance. This is a very strict limitation that occurs in any architecture where we separate the location when we execute the business logic from the location where we store the global state.

The Traquest model is able not only to ensure atomicity, but it is also a promising way to ensure consistency. Therefore, hereby we take the most relevant consistency protocols [7] under investigation respective to the Traquest model.

Primary-Based protocols provide proper consistency for an arbitrary type of data. To keep the data consistent, they have to synchronize each write at least with the primary server. For instance, in such case the algorithm has to be blocked until a read it depends on gets a confirmation from the primary server. This requires many iterations of roundtrip messages; therefore, Primary-Based Protocol implies a strict limitation in the performance.

Quorum-Based Protocols have very similar limitations to the Primary-Based Protocols. Each read and write operation has to be confirmed by other computing nodes before the executed algorithm can rely on the operation and step forward. The only exception is the Read-One, Write-All scheme. In this case, it is enough only to read the local state of the data; however, it requires even more messages to write synchronizations. This scenario can be only suitable in case of very read-heavy applications.

Conclusion

Providing ACID properties can be crucial for many applications, but it requires a massive compromise in the performance. The Traquest model is a proposed potential solution for this problem. By creating temporary responses and building up a mechanism for rolling back the conflicting parts of a running distributed algorithm it can ensure atomicity in a very fast way. This way, the Traquest model can provide lazy conflict resolving for atomicity and lazy replication for consistency and fault tolerance. Combined with pipelining in an optimistic concurrency case, the Traquest model require magnitudes fewer network messages than any of the investigated state of the art solutions.

Acknowledgements

The research was supported by the ÚNKP-19-3 New National Excellence Program of the Ministry for Innovation and Technology and by the project no. ED_18-1-2019-0030 (Application-specific highly reliable IT solutions) has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme funding scheme.

References

[1] Mark S. Miller; Darius Bacon. Promise pipelining at erights.org, 2010.

[2] Regine; Rohnert Hans; Sommerlad Peter; Stal Michael Buschmann, Frank; Meunier. *Pattern-Oriented Software Architecture, Volume 1, A System of Patterns.* Wiley, 8 1996.

[3] Henry Baker; Carl Hewitt. The incremental garbage collection of processes. In *SIGPLAN Notices 12, 8*, pages 55–59. ACM, 8 1977.

[4] Peter; Steiger Richard Hewitt, Carl; Bishop. A universal modular actor formalism for artificial intelligence. *IJCAI*, 1973.

[5] Peter Hibbard. Parallel processing facilities. In Stephen A. Schuman, editor, *New Directions in Algorithmic Languages*. IRIA, 1976.

[6] Typesafe Inc. Transactors, 2013.

[7] Andrew S. Tanenbaum; Maarten Van Steen. *Distributed Systems - Principles and Paradigms*. Prentice Hall, 2 edition, 2007.

[8] Justus Bogner; Alfred Zimmermann; Stefan Wagner. Analyzing the relevance of soa patterns for microservice-based systems. In *10th Central European Workshop on Services and their Composition*, 2018.

[9] Daniel Friedman; David Wise. The impact of applicative programming on multiprocessing. In *International Conference on Parallel Processing*, pages 263–272, 1976.

Energy-efficient routing in Wireless Sensor Networks

Dániel Pásztor, Péter Ekler and János Levendovszky

Abstract: Efficient data collection is the core concept of implementing Industry4.0 on IoT platforms. This requires energy aware communication protocols for Wireless Sensor Networks (WSNs) where different functions, like sensing and processing on the IoT nodes must be supported by local battery power. Thus, energy aware network protocols, such as routing, became one of fundamental challenges in IoT data collection schemes. In our research, we have developed novel routing algorithms which guarantee minimum energy consumption data transfer which is achieved subject to pre-defined reliability constraints. We assume that data is transmitted in the form of packets and the routing algorithm identifies the paths over which the packets can reach the Base Station (BS) with minimum transmission energy, while the probability of successful packet transmission still exceeds a pre-defined reliability parameter. In this way, the longevity and the information throughput of the network is maximized and the low energy transmissions will considerably extend the lifetime of the IoT nodes. In this paper we propose a solution that maximizes the lifetime of the nodes.

Keywords: IoT, WSN, energy-efficient, routing, WiFi

Introduction

Industry 4.0 is part of the fourth industrial revolution. One of the main focus of i4.0 is digital data acquisition and analysis of complex manufacturing processes, which requires a number of different sensors and communication equipment to measure and transmit the information about the process.

Since in many cases, connecting each sensor to a wired network would prove to be physically infeasible, wireless IoT devices can provide an efficient solution for controlling the sensor and transmit the collected data via a wireless network. This also gives flexibility, i.e. additional sensors can be easily added to or removed from the network as needed. This combination of the sensor and an IoT device with wireless transceiver will be called a *node* in the forthcoming discussion.

Unfortunately, wireless devices need to be powered by typically through a built-in batteries which needs to be recharged periodically. Under these circumstances, energy efficiency becomes a driving force when developing IoT communication protocols.

To save on transmission energy , it can often be disadvantageous for a particular device to send its message directly to the base station due to the energy consumption needed for reliable large distance communication. Instead, it may be useful to implement multi-hop packet transfers from the sender node to the BS via some relay nodes. In this paper we develop novel routing algorithms for packet transfer that ensures extended lifetime of the network.

The rest of the paper is organised as follows. In Section 2 the related work is summarised. In Section 3 the model is defined. Section 4 introduces the two-hop and multi-hop algorithms with numerical performance evaluation. Section 5 concludes the paper and proposes further research directions.

Related Work

In the literature, several different algorithms have been proposed for wireless efficient communication in wireless sensor networks. LEACH[1] assigns nodes to be cluster heads periodically whose responsibility are collecting the messages in their region. After compressing the received packets into a single message, every cluster head transmits it's message to the base station.

PEGASIS[2] creates a chain between the nodes close to each other. At every round, the measured values are aggregated and sent towards one particular node through the chain, which in turn transmits to the base station. This node changes every round.

The key difference between previous work and our research is that with our model, the lifetime of the network can be extended while high probability of successful packet delivery can be ensured.

Model

To investigate the different algorithms for routing, we first introduce the Rayleigh-fading model, which gives us a connection between the transmission energy and the probability of successfully packet transfer

$$g_{ij} = -d_{ij}^{\alpha} \frac{\theta \sigma_Z^2}{\ln P_{ij}} \tag{1}$$

where g_{ij} is the energy used in the transfer, d is the distance between the communicating nodes, α is the spatial dimension used in our models, θ and σ are parameters of the environment and communication, $\ln P_{ij}$ is the probability of successfully receiving the message between nodes i and j. This can be simplified to the following equation:

$$g_{ij} \ln P_{ij} = \omega_{ij} \tag{2}$$

where ω_{ij} is a constant dependent on the distance between the nodes and the parameters of the environment.

Let our wireless sensor network consist of N stationary nodes and a base station collecting the messages sent by the nodes. We place the nodes and the base station at random places for every simulation in a unit square. Each node starts with a given energy E, and transmit messages in a random order, given the constraint that the base station must receive it with a probability P_s. At any given time, only one message can be transmitted. We run the simulation until a node's energy level drops to zero, becoming a dead node. An example of this can be seen in figure 1.

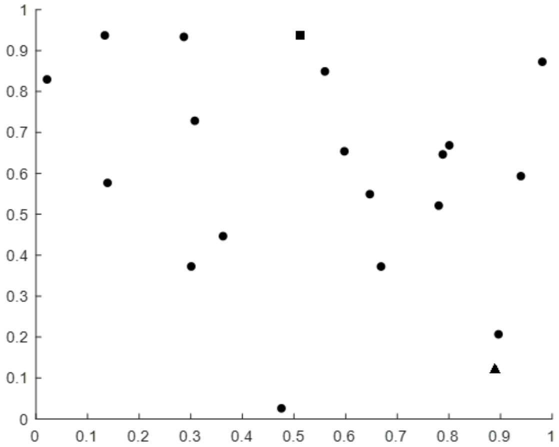


Figure 1: An example of one WSN. The square is the base station, while the triangle is a node currently sending a message.

Our proposed routing algorithms work on the principle that nodes with higher energy levels should participate more frequently in message routing.In order to achieve this, instead of minimizing the sum of the energies used in the transfer of a given message, we maximize the minimum remaining energy level after transferring a packet. This can be accomplished if and only if for every node the energy levels reach a common energy level after the transfer.

Proposed algorithms

Based on this observation, the objective of our algorithms is to bring the energy level of the nodes involved in a packet transfer to the highest common energy level while still satisfying the reliability constraint (guaranteeing that the packet will reach the BS with a pre-defined probability). We propose the following routing algorithms:

- *Direct sending*, i.e. the source node sends the packet directly to the base station without using an intermediate node. This is the simplest algorithm which serve as a baseline algorithm.

- *K-hop* algorithm, in which case at most k-1 intermediate node form the path for packet transfer.

- *Multi-hop* algorithm is a special case of k-hop, where we do not set the number of nodes prior to the routing algorithm (i.e. any number of intermediate nodes can be used in a path).

It can easily be demonstrated that calculating the optimal solution for two-hop routing results in a quadratic equation to be solved for every possible intermediary node. In contrast, calculating the solution for k-hop and multi-hop routing requires significantly more calculations. First we have to calculate the unique solution for a given permutation, which requires finding the root of an (k-1)-degree complete polynomial which satisfies the constraints. After that we must check every permutation of nodes for the optimal solution, making the complexity at least $O \mid V \mid$! Because of this, we opted to use an approximation.

Instead of solving the optimal common energy level problem, let us first find for a given energy distribution which guarantees that a packet can be send form the source node to the BS with the highest probability. Formally, this can be written as follows, using equation 3:

$$\max_{g} \sum_{j=0}^{m} \frac{\omega_{j,j+1}}{g_{j,j+1}} \tag{3}$$

Since ω must be a negative number, we can see that for a given path, the maximum transmission probability can be reached if $g_{l_j l_{j+1}} = c_j(k)$, meaning that every node along the path is using their remaining energy to send the message. Since we know the energy level of every node before the transmission occurs, we can calculate $\gamma_{j,j+1} \equiv \frac{\omega_{j,j+1}}{g_{j,j+1}}$, making the problem:

$$\max \sum_{j=0}^{m} \gamma_{j,j+1} = \min \sum_{j=0}^{m} -\gamma_{j,j+1} \tag{4}$$

which makes this problem equivalent to finding the shortest path in a graph with the edges having weight $-\gamma_{j,j+1}$. This can easily be solved using the Bellman-Ford algorithm for directed graphs, which has a worst case complexity of $O(\mid V \mid\mid E \mid) = O(\mid V \mid^3)$.

With this solution, we can approximate the optimal common energy level for multi-hop routing. Instead of every node sending with it's remaining energy, let us choose a common energy level c, which every node participating in the transmission must reach. This gives us the energy for every node with which they can participate in the transmission: $g_{j,j+1} = c_j(k) - c$, from which the previously presented approach gives us the maximum transmission probability.

Looking at the relation between the chosen common energy level and the maximum transmission probability, we can intuitively see that if we lower the energy level, the transmission probability rises since nodes can use more energy in the transmission. Because of this, we can use binary search over the interval $(0, c_s(k))$ for the common energy level where the maximum transmission probability reaches the given success probability. This gives us an approximate solution with complexity $O(\mid V \mid^3 \ln \frac{c_{max}}{\delta c})$, where c_{max} is the starting energy level of the nodes and δc is the maximum absolute error between the energy levels of the optimal and approximated solution.

The proposed algorithms were implemented in MATLAB, and simulation were run with it. We have changed the placement of the nodes and the order of the messages being sent between simulations, and measured the number of messages being sent before the first node run out of energy. The results can be seen in table 1.

Table 1: Number of messages before first dead node.

	Minimum			Average			Maximum		
Node count	Direct	2-hop	Multi	Direct	2-hop	Multi	Direct	2-hop	Multi
10	26	76	75	112.2	160.32	152.09	289	416	398
20	78	220	136	212.39	350.79	281.53	536	761	608
50	156	590	314	428.7	987.3	656.3	1392	1896	1443
100	308	1256	650	760.2	1980.2	1178.9	1673	3622	2287

It can be seen that under every circumstance, two-hop routing performed better than either direct or multi-hop routing. Comparing to direct routing, two-hop can make use of an intermediary node, so nodes farther away or with lower energy are able to conserve energy. This is in contrast to the results of multi-hop routing, where the use of more intermediary nodes leads to shorter lifetime. Examining the energy levels after each message, we concluded that while the remaining energy levels are indeed higher compared to the two-hop algorithm, the use of multiple nodes result in an overall higher energy usage which depletes the network faster.

Conclusion and future works

In this paper we have developed different routing algorithms for energy aware IoT data communication. As the performance analysis have revealed the 2-hop routing performed the best for every case. In the future, we would like to examine the k-hop algorithm for k larger than 2. Another future development may relate to the network topology. I these results were measured with randomly placed nodes. In the future, we would like to consider the special network topologies including indoor transmission, as well as different packet sending frequencies , when optimising the routing algorithm. The model developed can be further expanded by introducing barriers between nodes (such as buildings). We plan to apply the findings of our research in the wireless sensor network deployed at ZalaZone (being a test environment for future cars).

Acknowledgement

This work was supported by the BME-Artificial Intelligence FIKP grant of EMMI (BME FIKP-MI/SC) and by the János Bolyai Research Fellowship of the Hungarian Academy of Sciences.

References

[1] W. R. Heinzelman, A. Chandrakasan and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, Maui, HI, USA, 2000, pp. 10 pp. vol.2-.

[2] S. Lindsey and C. S. Raghavendra, "PEGASIS: Power-efficient gathering in sensor information systems," Proceedings, IEEE Aerospace Conference, Big Sky, MT, USA, 2002, pp. 3-3.

Spectral Clustering based Active Zero-shot Learning

Dávid Papp

Abstract: Supervised machine learning tasks often require a large number of labeled training data to set up a model, and then prediciton - for example the classification - is made based on this model. Nowadays tremendous amount of data is available on the web or in data warehouses, although only a portion of those data is annotated and the labeling process can be tedious, expensive and time consuming. Active learning tries to overcome this problem by reducing the labeling cost through allowing the learning system to iteratively select the data from which it learns. In special case of active learning, the process starts from zero-shot scenario, where the labeled training dataset is empty, and therefore only unsupervised methods can be performed. In this paper I propose a query strategy to estimate the informativeness value of the unlabeled data in the special zero-shot situation. The approach uses ClusterGAN (Clustering using Generative Adversarial Networks) integrated in the spectral clustering algorithm and then it selects an unlabeled instance depending on the cluster membership probabilities. The results are compared with various active learning query startegies on the MNIST dataset.

Keywords: active learning, zero-shot, query strategy, clustering, spectral clustering, generative adversarial network

Introduction and related work

The main goal of classification applications is to make predictions with high accuracy. A crucial part of this process is the model creation, which is based on the labeled training data (where the labels are the ground truth categories); hence the gathering of labeled data is also an important component of supervised machine learning. One can collect large amount of inexpensive unlabeled data through real-world applications [1], however labels for this data can be expensive, time-consuming or difficult to obtain. For example accurate labeling of speech utterances is extremely time consuming and requires trained linguists [2], on the other hand annotating gene and disease mentions for biomedical information extraction usually requires PhD-level biologists [3]. Consequently in these cases it is recommended to limit the number of labeled data that used for training, while attempting to achieve high accuracy. Let $U = \{u_i\}, i = 1...m$ denote the total amount of (unlabeled) data available for training; the goal is to select only a subset of this data and assign labels to them, thereby creating the $L = \{l_j\}, j = 1...n$ labeled dataset. The easiest technique is to randomly select L, this method is called passive learning, although it could lead to sub-optimal samples due to the randomness. A more sophisticated approach would be to consider the informativeness of the unlabeled data and then select the most informative ones. This approach is called active learning [4], where the learning system is allowed to iteratively select unlabeled instances and ask for their label. The key idea is that carefully picked, informative data allow the learning algorithm to perform better with less training. A decisive part of an active learning system is how it estimates the informativeness of unlabeled instances; these procedures are called query startegies. There are some traditional query startegy frameworks in the literature, e.g. uncertainty sampling [5], query-by-committee (QBC) [6], expected model change [7], expected error reduction [8], or density-weighted method [9].

Usually active learning query strategies assume that the selection process already started and train a classification model based on L. In special active zero-shot learning situation, the procedure starts with empty L, and therefore only unsupervised techniques (e.g. clustering) can be used. The aim of active zero-shot learning [10, 11, 12] is to find a small number of informative seen classes to facilitate unseen class predictions. The setting of this task contains seen and unseen categories, however in this paper I propose a query strategy, named Spectral Clustering Based Sampling (SCBS), to solve problems with only unseen classes.

The proposed query strategy utilizes ClusterGAN (Clustering using Generative Adversarial Networks, [13]) integrated in Spectral Clustering [14] framework to form the clusters in zero-shot condition; then the algorithm queries an unlabeled instance based on the cluster membership probabilities. The next section delineate the proposed approach, and after that the experimental evaluation is presented.

Spectral Clustering Based Sampling

Given a set of data points $x_1, ..., x_m$ with pairwise similarities s_{ij} (or distances) a similarity graph G can be built to model local neighborhood relationship between the data points. Based on the constructed G graph, a similarity matrix S can be derived, where the value of an element s_{ij} corresponds to the weight of the edge between x_i and x_j in G (if those points are not connected by an edge in G, then $s_{ij} = 0$). Let D be a diagonal degree matrix with $D_{ii} = \sum_j s_{ij}$.

The fundamental step of spectral clustering is the calculation of graph Laplacian matrix from the matrices S and D [15]. For example the unnormalized graph Laplacian matrix can be computed as expressed in Eq. 1, and I used this in the SCBS algorithm. Another two popular Laplacians are the symmetric normalized and left normalized [16].

$$L = D - S \tag{1}$$

Let matrix V be defined as the matrix containing the first k eigenvectors $v_1, ..., v_k$ of L as columns. At this point I applied ClusterGAN [13] on the rows of V to form $C_1, ..., C_k$ clusters. One advantage of using ClusterGAN clustering is that it provides decision vectors $d_1, ..., d_m$ for each data where the elements are the cluster membership probabilities.

I perform this algorithm on the initial unlabeled data set U, then query instances based on the informativeness values calculated from $C_1, ..., C_k$ and $d_1, ..., d_m$. I developed two variants of SCBS, the Global SCBS (G-SCBS) and Local SCBS (L-SCBS); both of them essentially operates the same way, however the former minimizes the informativeness metric over each element of U, while the latter examines only a reduced unlabeled set U_{C_j}, which contains the elements of a single cluster at a time. Furthermore, two different techniques were used to determine the unlabeled instance to query, the first one minimizes the uncertainty of the most probable cluster, and the second one minimizes the information entropy over all possible cluster assignments; as can be seen in Eq. 2 and Eq. 3, respectively.

$$u^* = \underset{u}{\operatorname{argmin}} \left(1 - d^*\right), \tag{2}$$

$$u^* = \underset{u}{\operatorname{argmin}} \left(-\sum_{j=1}^{k} d_{ij} \times \log d_{ij}\right), \tag{3}$$

where d_{ij} denotes the probability that unlabeled instance u_i belongs to cluster C_j, and d^* represents the probability of the most probable cluster.

Experimental evaluation

In this section I present the experiments that were conducted on the MNIST database of handwritten digits, which consist of 60,000 train and 10,000 test images. 10 different subsets were randomly selected from this dataset, each of them contained 500 images. During the experiments, the following 4 SCBS method variants were tested:

- G-SCBS using Eq. 2
- G-SCBS using Eq. 3
- L-SCBS using Eq. 2
- L-SCBS using Eq. 3

I tested several additional methods that are already proposed in the literature; the Random [17], the Centroid [18], the Border-based [19] and the Hybrid [19] active learning query strategies. The results of these competitor methods are compared to the results of the proposed SCBS based techniques.

The tests were performed in the special zero-shot situation, so at the start of active learning process U contained the total 500 images of the test datasets and L was empty. Consequently, the first steps are imporant to set up an adequate initial labeled image collection. At the testing of each dataset I investigated the first 100 active learning iterations, and evaluated the accuracy from the results at each iteration. The accuracy (ACC) is the ratio of the correct decisions and all decisions, as can be seen in Eq. 4. The different types of decisions come from the confusion matrix: True Positive, False Positive, True Negative and False Negative. Note that since at zero-shot active learning there is not enough labeled items to perform supervised learning (i.e. classification), the elements of the confusion matrix are derived from the clustering results.

$$ACC = \frac{TP + TN}{TP + FP + TN + FN} \tag{4}$$

Acknowledgements

The research was supported by the ÚNKP-19-3 New National Excellence Program of the Ministry of Human Capacities.

References

[1] Panda, N., Goh, K. S., & Chang, E. Y. (2006). Active learning in very large databases, *Multimedia Tools and Applications*, 31(3), 249-267.

[2] X. Zhu. Semi-Supervised Learning with Graphs. PhD thesis, *Carnegie Mellon University*, 2005.

[3] Chowdhury, M., & Faisal, M. (2010). Disease mention recognition with specific features, *In Proceedings of the 2010 workshop on biomedical natural language processing*, pp. 83-90.

[4] Settles, B. (2009). Active learning literature survey, *University of Wisconsin-Madison Department of Computer Sciences*.

[5] Lewis D. and Gale. W. (1994). A sequential algorithm for training text classifiers, *In Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 3-12.

[6] Tsai, Y.L., Tsai, R.T.H., Chueh, C.H., Chang, S.C. (2014). Cross-domain opinion word identification with query-by-committee active learning, *In: Cheng, S.M., Day, M.Y. (eds.) TAAI 2014. LNCS*, vol. 8916, pp. 334-343.

[7] Cai, W., Zhang, Y., Zhou, J. (2013). Maximizing expected model change for active learning in regression, *In: IEEE 13th International Conference on Data Mining*, pp. 51-60.

[8] Mac Aodha, O., Campbell, N., Kautz, J., Brostow, G. (2014). Hierarchical sub-query evaluation for active learning on a graph, *In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 564-571.

[9] Settles B. and Craven. M. (2008). An analysis of active learning strategies for sequence labeling tasks, *In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1069-1078.

[10] Xie, S., Wang, S., & Yu, P. S. (2016). Active zero-shot learning, *In Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pp. 1889-1892.

[11] Xie, S., & Philip, S. Y. (2017). Active zero-shot learning: a novel approach to extreme multi-labeled classification, *International Journal of Data Science and Analytics*, 3(3), pp. 151-160.

[12] Gavves, E., Mensink, T., Tommasi, T., Snoek, C. G., & Tuytelaars, T. (2015). Active transfer learning with zero-shot priors: Reusing past datasets for future tasks, *In Proceedings of the IEEE International Conference on Computer Vision*, pp. 2731-2739.

[13] Mukherjee, S., Asnani, H., Lin, E., & Kannan, S. (2019). Clustergan: Latent space clustering in generative adversarial networks, *In Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, pp. 4610-4617.

[14] Von Luxburg, U. (2007). A tutorial on spectral clustering, *Statistics and computing*, 17(4), 395-416.

[15] HU, P. (2012). Spectral Clustering Survey

[16] Chung, F. R., & Graham, F. C. (1997). Spectral graph theory (No. 92), *American Mathematical Soc*.

[17] Kang, J., Ryu, K. R., & Kwon, H. C. (2004). Using cluster-based sampling to select initial training set for active learning in text classification, *In Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 384-388.

[18] Hu, R., Mac Namee, B., & Delany, S. J. (2010). Off to a good start: Using clustering to select the initial training set in active learning, *In Twenty-Third International Florida Artificial Intelligence Research Society Conference (FLAIRS 2010)*, pp. 26-31.

[19] Yuan, W., Han, Y., Guan, D., Lee, S., & Lee, Y. K. (2011). Initial training data selection for active learning, *In Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication*, p. 5.

Parallel Simulation for The Event System of DISSECT-CF

Dilshad Hassan Sallo and Gábor Kecskeméti

Abstract: Discrete Event Simulation (DES) frameworks gained significant popularity to support and evaluate cloud computing environments, by providing decision-making for complex scenarios as well as saving time and effort. The majority of these frameworks lack of parallel execution. DISSECT-CF is one of the frameworks that introduced an improvement in performance of Infrastructure as a Service (IaaS) simulation. Although DISSECT-CF execution time is faster than the majority, it still executes sequentially. This paper introduces parallel execution to the most abstract subsystem in DISSECT-CF (event system). The new subsystem detects when multiple events occur at a specific time and then multi-threads these events. The number of independent frequent events, plays a crucial role to invoke the new subsystem and increase the performance. Achieving a high degree of repeated events leads to better performance. We focused on time management scenarios as a part of simulation to show the leverage of parallelism. We also focused on events that not having an influence on the future. The results show that parallel version scales proportionally with the number of cores and it reaches five times faster than sequential version.

Keywords: Cloud computing, parallel simulation, DISSECT-CF

Introduction

The ubiquity of emerging advanced technologies such as fog computing, edge computing and Internet of things (IoT), has urged to develop DES frameworks being capable of predicting and evaluate the behaviour of these [2]. Moreover, these provide more flexibility than the real systems, by providing a reproducible environment for repeated evaluation of various scenarios and algorithms with minimal costs[4]. Despite DES simulators may support several levels of parallel and distributed computing [2], they are mostly written following a sequential execution model. Introducing the capability to support parallel execution and scaling is crucially to these kinds of simulators. DISSECT-CF [1] is one of the frameworks that able to simulate internal components and processes of cloud infrastructures. Its extensibility was demonstrated towards IoT and fog computing use cases. Although the execution time of DISSECT-CF is significantly faster than even the most prominent simulator in the field "CloudSim" [4], this performance advantage is still not enough for the most demanding current research use cases (e.g., simulating millions of IoT devices and their continuum with clouds). Its sequential execution is a significant bottleneck, thus parallel execution is needed for scaling its performance efficiently to meet the newest challenges in the field.

This paper proposes parallel execution to the event system layer of DISSECT-CF, as this layer is the most heavily used one during the simulations, and its time management features are generally used by all other components. Our extension focuses on events that should execute independently, but practically simultaneously executed and parallelises their evaluation. This can be fulfilled by explicitly separated recurrent events to load balancing and distributing them over available cores, leading to exploiting more cores per single machine. We have evaluated our proposed extension via several test scenarios. In these, we prepared situations which have different amounts of independent but simultaneously occurring events in the system. These scenarios were carefully designed to cover a different aspect of implementing most methods of Event subsystem classes, while they also represent a wide variety of potential use cases expected in future simulations. The advantage over performance can be remarked when many recurrent events need to be composed to achieve parallelism. Our findings show that our parallel time management layer leverage from multicore execution, especially so when the sequential workload is split into batches of over 32 events, leading to faster execution considerably.

Related work

In this section, the most relevant works to the suggested approach are briefly presented with highlighting their notable drawbacks. The authors [2] have conducted a survey over 33 simulators that support DES, including prominent framework CloudSim [6]. They provide different significant features to support IaaS cloud computing. However, the gap was found in the majority represented by they were built sequentially with lack of parallel execution. The authors [3] extend the CloudSim simulator to dubbed "Cloud2Sim" that support concurrent and distributed simulations. However, it exploits external features such as Hazelcast, Infinispan and Hibernate to provide concurrent execution. The authors [4] provide a parallel DES framework for simulating computing tasks, but it classifies discrete events to various groups before simulating. The authors [5] suggested initial step in the parallel and distributed execution in a real system, which could not adapt to current environments. However, introducing parallel execution to simulators leading to control simulation easily as well as repeating tests free of cost. After presenting several previous works that are related to the subject of the proposed research, and showing the lack of sufficient parallel execution. This paper provides parallel execution to the event system of DISSECT-CF simulator to speed up execution of simulations.

Methodology

DISSECT-CF simulator introduced substantial features to IaaS and allowing easy extensibility to support other concepts such as IoT. Despite DISSECT-CF reduces the execution time of simulation, it executes sequentially. It could face challenges to simulate modern technology at the same time such as IoT connected devices, which are rapidly increasing and could reached billion devices. DISSECT-CF simulator consists of five major subsystems that mostly implement different tasks in an independent manner. The lowest layer (Event system) was built sequentially to execute on a single core CPU. Based on existing application programming interface (API) of DISSECT-CF, recurring events can execute in parallel manner when they happen at a specific time. In each time, the degree of parallelism can be varied depending on how frequent each event occurs. When all subscribed events happen at a specific time, the degree of parallelism is 100%, if half of them happened at one time, the degree is 50% and so on. Increasing these recurrent events with a high degree at any specific time, leads to insufficiently manage by single CPU and may delay the execution.

Inner class called `Parallel` was designed inside `Timed` class to implement parallelism using ForkJoinPool technique. This class designed to fit with existing methods of `Timed` class, which benefit from any improvement. Executing `Parallel` class depends on the size of recurrent events list, if they exceed determined size, then the `Parallel` class will invoke to execute repeated events simultaneously. It runs recurring events after dividing them into equal lists for balancing load over the processors. With respect to the number of cores, there are three factors that influence the performance of parallel execution. Firstly, the size of a list allocated to recurrent events. This determines when `Parallel` class will invoke to execute these events simultaneously. When the size of the list is small, associating and managing threads over the list is time consuming. Secondly, the degree of parallelism. It is important to observe to which degree usually recurrent events occur at a specific time. This shows the advantage of parallel version over sequential version regarding performance and scaling. Finally, the size of the simulation has a significant role to show the benefit of the parallel version. When it becomes large, lead to exploit all the cores.

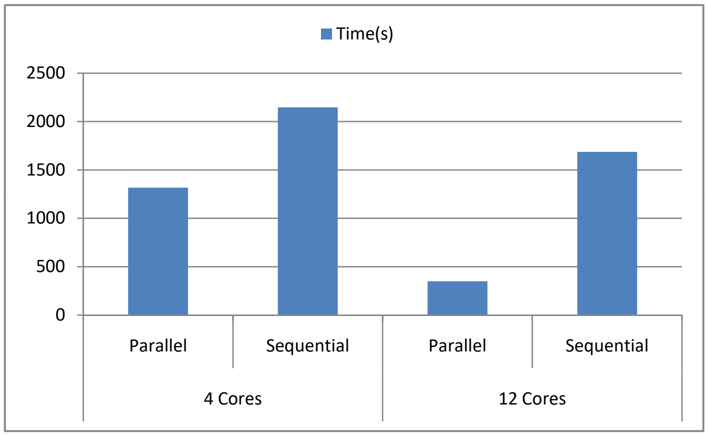


Figure 1: Result in time(s) for Parallel and sequential using 4 cores and 12 cores

Evaluation

Two laptops (Intel (R) Core (TM) i7-4600U CPU @ 2.10GHz (4 CPUs), 2.7GHz) and (Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz (12 CPUs), 2.2GHz) were used for evaluation of Parallel execution of DISSECT-CF. Several scenarios with different purposes designed carefully to test the performance of the parallel version. These focused on time management and ensure all events happen properly at desired time. Our scenarios create 100,000 recurring event objects executing with different frequencies to test parallel version in various conditions. These capable to control the degree of parallelism as well as the size of the simulation. The source code of scenarios is available on GitHub [1]. As the invocation of `Parallel` class depends on a size of a list that contains repeated events at a specific time, several tests with different sizes (8,12,32,64) have been done to specify the suitable size to store recurring events. The result shows, better performance can be obtained when the size of the list for recurrent events equal or exceeds 32.

Parallel version scales better with the number of cores and exploit entirely the power of multi-cores. Figure 1 shows the execution of both parallel and sequential versions on 4 and 12 cores for 100,000 recurrent event objects. When the number of cores is 4, parallel version run 1.6 faster than sequential, with increasing number of cores to reach 12, it runs approximately five times faster than sequential. The degree of parallelism has a significant effect over the performance of the parallel version. The advantage of parallel to reduce execution time is a little at 25%, comparing to 100%. Figure 2 shows the execution time of parallel and sequential for 100,000 recurring event objects with five different degrees.

Conclusion

DISSECT-CF is one of the robust simulators that brought extraordinary features to improve the performance of IaaS simulation. It is built to accompany the latest technology with easily extensibility. In terms of execution time, DISSECT-CF is fast and reliable but still runs sequentially, and it is not leveraging of power of multiprocessor to speed up the execution of simulations. It also does not also scale with the number of cores. The parallel version was built to handle this issue and being useful when there is necessity to apply that, represent occurring multiple events at a specific time. It scales with the number of cores and leads to reduce the time of execution significantly.

[1] The code is available at following website: https://github.com/dilshadsallo/dissect-cf-examples

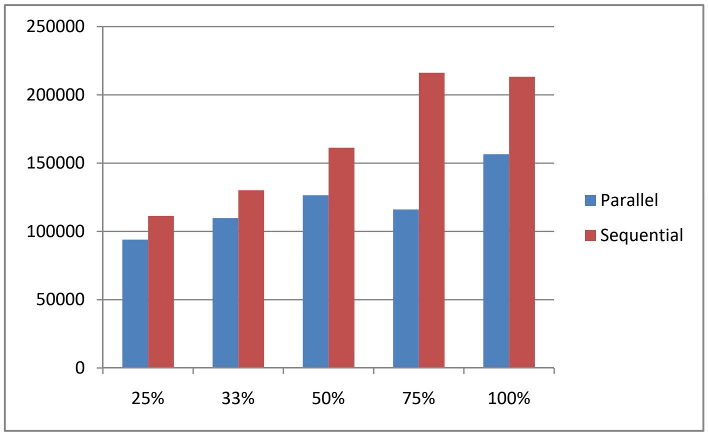


Figure 2: Execution time(s) of parallel and sequetial for five different degree of parallelism

Acknowledgements

This research was supported by the Hungarian Scientific Research Fund under the grant number OTKA FK 131793.

References

[1] Kecskemeti, G., 2015. DISSECT-CF: a simulator to foster energy-aware scheduling in infrastructure clouds. *Simulation Modelling Practice and Theory*, 58, pp.188-218.

[2] Byrne, J., Svorobej, S., Giannoutakis, K.M., Tzovaras, D., Byrne, P.J., Ostberg, P.O., Gourinovitch, A. and Lynn, T., 2017, April. A review of cloud computing simulation platforms and related environments. *In International Conference on Cloud Computing and Services Science* (Vol. 2, pp. 679-691). SCITEPRESS.

[3] Kathiravelu, P. and Veiga, L., 2014, September. Concurrent and distributed cloudsim simulations. *In 2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems* (pp. 490-493). IEEE.

[4] Liu, J., Zhou, Y., Zhang, D., Fang, Y., Han, W. and Zhang, Y., 2014, December. Muclouds: Parallel simulator for large-scale cloud computing systems. *In 2014 IEEE 11th Intl Conf on Ubiquitous Intelligence and Computing and 2014 IEEE 11th Intl Conf on Autonomic and Trusted Computing and 2014 IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops* (pp. 80-87). IEEE.

[5] Fujimoto, R.M., Malik, A.W. and Park, A., 2010. Parallel and distributed simulation in the cloud. *SCS M&S Magazine*, 3, pp.1-10.

[6] Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A. and Buyya, R., 2011. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1), pp.23-50.

Improving MapReduce Speculative Executions with Global Snapshots

Ebenezer Komla Gavua and Gábor Kecskeméti

Abstract: Hadoop is a MapReduce implementation for distributed storage and computation. However, this implementation has issues managing poor performing jobs. This challenge, called speculative execution, is mostly handled by running backup tasks. The main contribution of this paper is a proposed the application of consistent global snapshots and stable property to resolve this challenge. This involves the capturing of snapshots of all data I/O into mappers and reducers before and after data executions. The snapshots are then compared to determine the poor performing tasks. These tasks are quickly divided and redistributed amongst the inactive mappers and reducers based on an algorithm on data complexity. As a future work, we are considering executing these algorithms to evaluate their performance by testing it with heterogeneous data sets.

Keywords: MapReduce, Global Snapshots, Speculative Executions.

Introduction

Distributed global snapshots have been widely used as a technique to achieve reliability and fault recovery in distributed and cloud systems. During a distributed computing session, a snapshot can be taken, to capture and preserve the session's instantaneous execution state. This is done to insure against failure at a later time [1, 2]. The concepts of Global Snapshot and stable property are designed to monitor an asynchronous network algorithm while it runs. By repeatedly computing and evaluating the global snapshot, the stable property can be detected[3].

The Apache Hadoop software environment provides a popular implementation for distributed data storage and MapReduce computing [5]. However, the handling of poor performing jobs remains a challenge. Apache Hadoop does not fix or diagnose slow task but rather launches another equivalent tasks as backup when it detects a task running slower than expected. This process is called Speculative Execution in Hadoop[6]. MapReduce runs a speculative copy of a slow task (also called a *backup task*) on another machine to finish the computation faster.

The main contribution of this work is to propose a technique involving Consistent Global Snapshot and stable property concepts of distributed computing as means to diagnose detect slow tasks. This will enable solutions to be provisions as early as possible to improve general system performance. In order to achieve this, consistent global snapshots of all data inputs and outputs (I/O) into mappers before and after processing are captured. This process is also repeated for reducers before and after processing and during the intermediate session. The captured snapshots are then compared to determine the poor performing tasks. This involves monitoring the jobs which take a longer time to exhibit a stable property (i.e.those at a quiescent state). These tasks are quickly divided and redistributed amongst the inactive mappers and reducers based on an algorithm on data complexity for re-execution.

The remainder of this paper is structured as follows. In Section 2, we reviewed research works in relation to Speculative Executions. In Section 3, we present our methodology for detecting poor performaning jobs in MapReduce and proposed improvements. Finally, Section 4 concludes the paper with recommendations for future work.

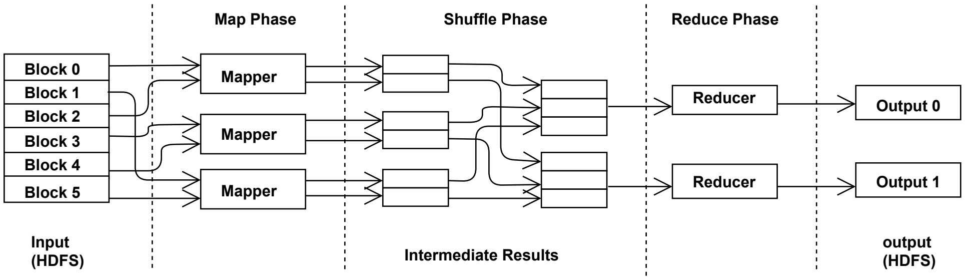


Figure 1: MapReduce Structure

Related Works

Research works have been conducted over the years to mitigate poor performing jobs in Hadoop MapReduce, as this challenge contributes to the system's over all throughput. Wang et al [7] proposed an extended speculative execution strategy called Partial Speculative Execution (PSE). They analyzed checkpoint information of original tasks. Speculative tasks start from the checkpoint instead of starting from scratch.

Zaharia et al designed a simple, robust scheduling algorithm, LATE [4], which uses estimated finish times to speculatively execute the tasks that hurt the response time the most. Wang et al [8] developed a new speculation scheme ESPLASH which can efficiently and quickly identify the stragglers, submit the speculative tasks to the most appropriate nodes and avoid resource waste on the unnecessary speculative execution. .

The above researches have addressed the challenge of managing poor performing jobs to some extent. However, they did not employ consistent global snapshots in their algorithms.

Methodology

This section is divided into three sections. The first section briefly explains the MapReduce programming model. The second section explains speculative execution in MapReduce and the third section discusses the application of global snapshots and stable property to MapReduce.

MapReduce Programming Model

To utilize MapReduce, a programmer must express their computations as jobs. The job inputs must be specified to yield key-value pairs. Job processing consists of two stages: firstly, a user-defined map function is applied to each data input record to generate a list of intermediate key-value pairs. Secondly, a user-defined reduce function is applied once on each distinct key in the map output and passed on the list of intermediate values associated with that key. These functions are parallelized by the MapReduce framework to ensure fault tolerance as shown in figure 1. The process of MapReduce operation is shown as :

$\text{map}(\text{key}_{\text{in}},\text{value}_{\text{in}}) \rightarrow \text{list}(\text{key}_{\text{out}}, \text{value}_{\text{intermediate}})$

$\text{reduce}(\text{key}_{\text{out}} , \text{list}(\text{value}_{\text{intermediate}})) \rightarrow (\text{value}_{\text{out}})$

Speculative Executions

Speculative Executions is Hadoop MapReduce way of tackling poor performing jobs. In the MapReduce framework, speculative tasks start from the same workload as their original tasks. Firstly, a speculative map task enters the input data same as the original map tasks which increases I/O cost since the input data always comes from Hadoop Distributed File System (HDFS). Secondly, speculative reduce tasks are recopied from the intermediate data

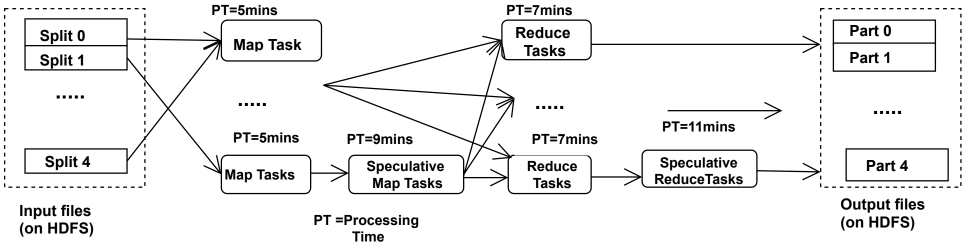


Figure 2: Speculative Execulation

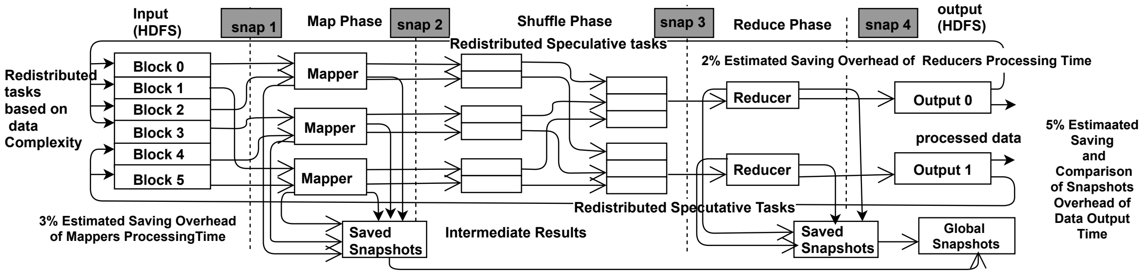


Figure 3: Structure of MapReduce with Global Snapshots

from almost all map tasks. Thirdly, all speculative tasks must be recomputed from all disks and remote mappers since they are partially processed data. The extra data entry, recopying and recomputing sessions make speculative execution inefficient. This is displayed in figure 2.

Application of Consistent Global Snapshots on MapReduce

The algorithm proposed to be implemented on MapReduce is the ChandyLamport global snapshot algorithm[9]. This snapshot algorithm is a monitoring algorithm designed to work as a send/ receive network algorithm. The snapshot algorithm initially sends a snap input through the system about to utilize the MapReduce programming model. This is to capture the state of the system before processing commences.

Once the initial checks are completed, the snapshot algorithm is sent ahead of every data input at all stages of the MapReduce Function with a $marker$ message. The $marker$ message is utilized to indicate the change in state of the system before and after data processing.

The snapshot algorithm then takes snapshots during data input before a map phase commences and data output (intermediate data) after the map phase. Snapshots are also taken during the data input before the reduce phase and data output after the reduce phase.

Once a snapshot is taken at a particular time, the captured data is reported (saved) for comparisons later. The various captured locally captured snapshots at each level of data I/O constitute a global snapshot of a particular execution.

At the end of a process execution, global snapshots are compared. The processes which take a longer time to reach a quiescent state are the identified as the poor performing tasks. The monitoring algorithm must be executed consistent and at a high rate in order not to delay the general system performance. The points of capturing of local snapshots is shown in figure 3.

The data output from the poor performing jobs, are divided into two parts based on data complexity by an algorithm. The less complex data are processed first, to be followed by the more complex ones. The jobs tasks are then redistributed amongst the inactive mappers and reducers based on the data complexity. These are done at a high execution rate to complete the entire data processing cycle.

Conclusion

Hadoop MapReduce executes poor performing task speculatively to ensure that the entire data input meets expected target. However, in most cases, these poor performing task delays the general system performance.

In this paper, we proposed global snapshots as a technique for improving Speculative Execution in MapReduce. A monitoring algorithm will be utilized to take snapshots of data inputs and outputs at key stages during the MapReduce programme execution. This will enable the poor performing tasks to be quickly identified and redistributed amongst the inactive mappers and reducers by an algorithm. The tasks will be processed based on data complexity until all the tasks are thoroughly executed at a high execution rate.

As a future work, we are considering executing these algorithms to evaluate its performance by testing it with heterogeneous data sets. Moreover, the experimental data gathered will be compared with existing experimented data to assess the efficiency of this proposed technique.

Acknowledgements

This research was supported by the Hungarian Scientific Research Fund under the grant number OTKA FK 131793.

References

[1] Ajay D Kshemkalyani. Fast and message-efficient global snapshot algorithms for large-scale distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 21(9):1281–1289, 2010.

[2] Yonghwan Kim, Tadashi Araragi, Junya Nakamura, and Toshimitsu Masuzawa. A concurrent partial snapshot algorithm for large-scale and dynamic distributed systems. *IEICE TRANSACTIONS on Information and Systems*, 97(1):65–76, 2014.

[3] Ardalan Kangarlou, Dongyan Xu, Paul Ruth, and Patrick Eugster. Taking snapshots of virtual networked environments. In *Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing (VTDC'07)*, pages 1–8. IEEE, 2007.

[4] Matei Zaharia, Andy Konwinski, Anthony D Joseph, Randy H Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *Osdi*, volume 8, page 7, 2008.

[5] Apache Hadoop. Apache hadoop project, 2011.

[6] Huanle Xu and Wing Cheong Lau. Speculative execution for a single job in a mapreduce-like system. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 586–593. IEEE, 2014.

[7] Yaoguang Wang, Weiming Lu, Renjie Lou, and Baogang Wei. Improving mapreduce performance with partial speculative execution. *Journal of grid computing*, 13(4):587–604, 2015.

[8] Jiayin Wang, Teng Wang, Zhengyu Yang, Ningfang Mi, and Bo Sheng. esplash: Efficient speculation in large scale heterogeneous computing systems. In *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8. IEEE, 2016.

[9] K Mani Chandy and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems (TOCS)*, 3(1):63–75, 1985.

Distance-based Skeletonization on the BCC Grid

Gábor Karai and Péter Kardos

Abstract: In this paper we study Strand's algorithm that computes the surface skeleton of 3D objects sampled on the BCC grid based on a hybrid strategy [6]. Furthermore, we present two improved versions of this algorithm, which have less time complexity, and are less sensitive to the visiting order of border points. One of them is also capable of extracting curve skeletons. All investigated methods can be extended to arbitrary distance metrics.

Keywords: BCC grid, distance transform, topology preservation, thinning

Introduction and Notations

A skeleton is a region-based shape descriptor that summarizes the general form of objects. The two most important requirements that should be met by skeletons is to preserve the geometrical and topological object features. In 3D the *curve skeleton* or *centerline* is represented by 1D line segments, and the *surface skeleton* or *medial surface* may contain 2D surface patches, too. Distance-based skeletonization techniques focus on the detection of ridges in the *distance map* of boundary points. In 3D discrete spaces, the set of *centres of maximal balls (CMB's)* is generated for this purpose [1]. This set depends on the chosen distance metric. *Chamfer metrics* give good approximations to the Euclidean distance by assigning a weight to each grid point in a small neighborhood. The distance-based approach can fulfill the geometrical requirement but it fails to preserve topology, as the set of CMB's is usually disconnected. Another strategy for skeletonization, called as *thinning*, is based on the iterative peeling of the object boundary. Thinning algorithms preserve topology, if proper deletion rules are given (see [3]). To ensure both above mentioned criteria, distance-based methods are often combined with thinning. These approaches can be further classified as follows [1]:

- *Anchor-based thinning*: The detected CMB's or local maxima (anchor points) are assumed to be skeleton points during the subsequent thinning method.
- *Distance-ordered thinning*: In each iteration step of the thinning phase, only border voxels with the same distance value are considered.
- Combinations of the previous two strategies. We refer to them as *hybrid algorithms*.

Most 3D skeletonization algorithms work on digital pictures sampled on the cubic grid. An alternative structure, the *body-centered cubic grid* tessellates the space into truncated octahedrons, which results in a less ambigous connectivity structure compared to the cubic grid [7]. The BCC grid is defined as the following subset of $\mathbb{Z}^3$:

$$\mathbb{B} = \{(x, y, z) \in \mathbb{Z}^3 \mid x \equiv y \equiv z \text{ (mod 2)}\}.$$

In the voxel representation of the BCC grid, any two neighboring elements share a face. The set $N_{14}(p)$ contains the 14 face-neighbors of point p, including p, depicted in Fig. 1, and we say that any point in $N_{14}(p)$ is *14-adjacent* to p. Furthermore, we divide $N_{14}(p)$ to the subsets $N_6(p)$ and $N_8(p)$ as it is shown in Fig. 1, and let $N_i^*(p) = N_i(p) \setminus \{p\}$ $(i \in \{6, 8, 14\})$.

Let $\langle a, b\rangle$ denote the general *Chamfer mask* for weighted distance transform, where a and b are the weights assigned to all points in $N_8^*(p)$ and $N_6^*(p)$, respectively. Many possible weight combinations are examined in [2]. Note that $\langle 1, 1\rangle$ and $\langle 1, 2\rangle$ masks are *neighbor-based distance*s, called d_{14} and d_8, respectively.

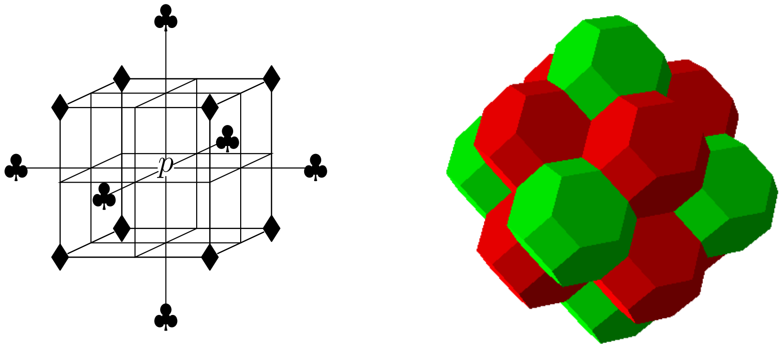


Figure 1: Elements of $N_{14}(p)$ in $\mathbb{B}$ (a). Voxel representation of these points (b). The set N_6 of the central point $p \in \mathbb{B}$ contains p and the six points marked "♣" (green voxels in (b)). The set N_8 contains p and the eight points marked "♦" (red voxels in (b)).

The $(14, 14)$ *binary digital picture* is a quadruple $\mathcal{P} = (\mathbb{B}, 14, 14, B)$ [4]. A *black component* or an *object* is a 14-connected set of points, called as *black point*s in B, while a *white component* is a 14-connected set of points, called as *white point*s in $\mathbb{B} \setminus B$. Here we assume that a picture contains finitely many black points. A black point is called a *border point* in a picture if it is 14-adjacent to at least one white point. Let $CB(p)$ and $CW(p)$ be the number of black components and the number of white components in $N^*_{14}(p)$, respectively.

A *simple point* is a black point whose deletion is a topology preserving reduction [4]. Strand and Brunner proposed the following characterization of simple points:

Theorem. [5] *Let* p *be a border point in a* $(\mathbb{B}, 14, 14, B)$ *picture. Then* p *is a simple point if and only if* $CB(p) = CW(p) = 1$.

Strand's Algorithm and Its Improved Variants

Strand proposed the first skeletonization algorithm for binary objects sampled on the BCC grid [6]. His hybrid method assumes the d_{14} distance metric and it can extract only surface skeletons by preserving non-simple points and surface edge points. A point $p \in B$ is considered as a *surface edge point*, if there are two points $q, r \in N^*_{14}(p) \cap B$ such that $N_{14}(q) \cap N_{14}(r) = \{p\}$ and there is no point $s \in N^*_{14}(p) \cap B$ such that $N_{14}(p) \cap N_{14}(s) \subseteq B$. The algorithm consists of the following three steps:

1. Compute the d_{14} distance map of the input picture $\mathcal{P} = (\mathbb{B}, 14, 14, B)$, and detect the set H of CMB's. Furthermore, let $S = \{p \in B \mid N_{14}(p) \cap H = \emptyset\}$.

2. Thin the object "roughly" by sequentially deleting simple points of set S in ascending order of their distance value. All points in S are examined exactly once.

3. Do a final sequential thinning on the remaining 3–4 voxel thick object by preserving the surface edge points such that the remaining border points are visited in descending order of their distance value. This step is executed until there are no more deletable points.

The sequential thinning in step 2 and 3 suffer from the disadvantage of being sensitive to the visiting order of border points with the same distance value. As a result the final skeleton usually has some "false" skeleton points (see Fig. 3b). Another drawback is the computational complexity: if I denotes the number of grid points, then Strand's algorithm has a runtime of $O(I^{4/3})$ in the case of optimal implementation as it highly depends on the input object's thickness.

To construct linear time algorithms, we merge the 2nd and 3rd step and simplify the organization of thinning iterations. Therefore, these methods consist of a distance transformation step and a thinning step. The set S from Strand's algorithm is not used but the deletion rule is also based on the preservation of non-simple points and surface edge points. We introduce parameter t_{max} which gives an upper limit to the visiting number of each object point during the thinning phase (i.e., the iteration step will be repeated at most t_{max} times). Therefore, if t_{max} is a positive integer, then the runtime complexity is $O(t_{max} \cdot I)$. Alternatively, setting t_{max} to ∞ means that the number of visiting numbers is not limited. This parameter is applied in both of our improved algorithms. The motivation is to reduce the sensitivity to the visiting order of border points.

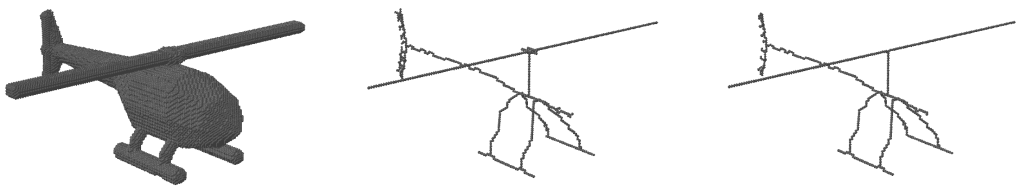
Our first improved variant, called **Mod-1**, is an anchor-based thinning method that considers local maxima instead of CMB's to be anchor points because many CMB's are proved to be "false" skeleton points on weighted (except d_{14} and d_8) distance maps. Point p is a local maximum, if and only if there is no point $q \in N^*_{14}(p)$ such that $DT(q) > DT(p)$, where $DT(p)$ denotes distance value of p. During the thinning phase of **Mod-1** the anchor points are recognized as skeleton points.

Our second improved variant, called **Mod-2**, is a distance-ordered thinning method that omits the detection of anchor points. The border points are visited in ascending order of their distance value during the thinning phase.

To extract curve skeletons, we set t_{max} to ∞, and instead of surface edge points we retain either of two types of curve endpoints **C1** and **C2** in algorithm **Mod-2**. Point $p \in B$ is a **C1** *endpoint*, if and only if $|N^*_{14}(p) \cap B| = 1$, i.e., p has only one black neighbor in $N^*_{14}(p)$. Let u be this black neighbor. p is a **C2** *endpoint*, if and only if p is a **C1** endpoint and u is a line point. Point u is a *line point* if and only if $|N^*_{14}(u) \cap B| = CB(u) = 2$. We consider a border point deletable if it is simple but not an endpoint. The corresponding algorithms are called **Mod-C1** and **Mod-C2**, respectively. These variants terminate only if there are no more deletable points during an iteration step of their thinning phase.

Results and Conclusions

The proposed algorithms **Mod-1** and **Mod-2** have linear runtime complexity and are less sensitive to the visiting order of border points compared to Strand's method. According to our experience, it is sufficient to set t_{max} to at most 3 to produce "clear" surface skeletons. All examined algorithms preserve topology due to the fact that all deletable points are simple. All methods can be extended to arbitrary distance metrics including any $\langle a, b \rangle$ weighted distance or even the Euclidean distance. Note that optimal parameters (e.g., distance metric, thinning strategy) depend on the application. Some examples are illustrated in Fig. 2 and Fig. 3 where the indicated types of algorithms and distance metrics were used. Numbers in brackets show the number of object or skeleton points.

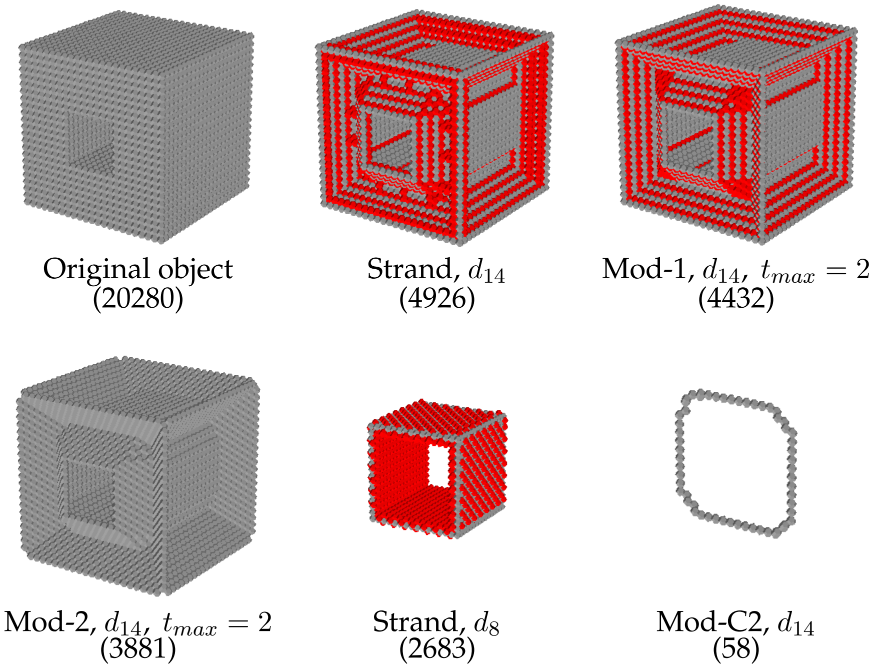


Original object
(94031)

Mod-C1, $\langle 6, 7 \rangle$
(644)

Mod-C2, $\langle 6, 7 \rangle$
(579)

Figure 2: Extracted C1- and C2-centerline of a helicopter.



Original object (20280)

Strand, d_{14} (4926)

Mod-1, d_{14}, $t_{max} = 2$ (4432)

Mod-2, d_{14}, $t_{max} = 2$ (3881)

Strand, d_8 (2683)

Mod-C2, d_{14} (58)

Figure 3: Extracted surface (b–e) and curve skeletons (f) of a holey cube with various parameters. On (b), (c) and (e) anchor voxels are gray and further skeleton voxels are red. The resulting C1- and C2-centerline (f) coincide with each other because no C2 endpoint was detected.

Acknowledgements

This research was supported by the project "Integrated program for training new generation of scientists in the fields of computer science", no EFOP-3.6.3-VEKOP-16-2017-00002. The project has been supported by the European Union and co-funded by the European Social Fund.

References

[1] G. Borgefors, I. Nyström, G. Sanniti di Baja: *Discrete Skeletons from Distance Transforms in 2D and 3D* In: Siddiqi K., Pizer S.M. (eds) Medial Representations. CIV 37. Springer, Dordrecht (2008)

[2] C. Fouard, R. Strand, G. Borgefors: *Weighted distance transforms generalized to modules and their computation on point lattices*, PR 40(9), 2453–2474 (2007)

[3] T. Y. Kong: *On topology preservation in 2-d and 3-d thinning*, International Journal of Pattern Recognition and Artificial Intelligence 9, 813–844 (1995)

[4] T. Y. Kong, A. Rosenfeld: *Digital topology: Introduction and survey*, CVGIP 48, 357–393 (1989)

[5] R. Strand, D. Brunner: *Simple Points on the Body-Centered Cubic Grid*, Technical Report 42, Centre for Image Analysis, Uppsala University, Uppsala, Sweden (2006)

[6] R. Strand: *Surface skeletons in grids with non-cubic voxels*, Proceedings of the 17th International Conference on Pattern Recognition, ICPR 2004, Vol. 1, 548–551 (2004)

[7] R. Strand: *The face-centered cubic grid and the body-centered cubic grid – a literature survey*. Internrapport, Centrum für Bildanalys, Uppsala University, Centre for Image Analysis (2005)

Towards Reverse Engineering Protocol State Machines

Gábor Székely, Gergő Ládi, Tamás Holczer and Levente Buttyán

Abstract: In this work, we are addressing the problem of inferring the state machine of an unknown protocol. Our method is based on prior work on inferring Mealy machines. We require access to and interaction with a system that runs the unknown protocol, and we serve a state-of-the-art Mealy machine inference algorithm with appropriate input obtained from the system at hand. We implemented our method and illustrate its operation on a simple example protocol.

Keywords: Automated Protocol Reverse Engineering, State Machines, Mealy Machines

Introduction

Many systems use closed protocols whose specification is not made publicly available. Examples include industrial control systems and in-vehicle embedded networks. Often, it would be very beneficial to understand those closed protocols. For instance, network anomaly detection tools cannot monitor industrial control systems without understanding their protocols, hence, cannot detect potential cyber attacks on them.

Protocol reverse engineering is the activity of uncovering the specification of an unknown protocol. This can be a tedious work, so automation is required to make it practical. The goal of automated protocol reverse engineering methods can be two-fold: determining the format of messages used by the protocol and recovering the state machine of the protocol. In another paper [2], we studied the problem of determining the message formats of unknown binary protocols, and developed a tool which can take captured network traffic containing messages of the protocol and output the identified message types and the semantics of message fields for the different message types. In this work, we are addressing the problem of inferring the state machine of an unknown protocol, and we assume that message types have already been identified (e.g., by using our tool mentioned above).

Our method is based on prior work on inferring Mealy machines, and in particular, on the work of Shahbaz and Groz [3]. We use their Mealy machine inference algorithm and extend it with elements that make it possible to use their conceptual results in practice to reverse engineer the state machine of real-world protocols in an automated way. Our method requires access to and interaction with a system that runs the unknown protocol, and it basically consists in serving the Mealy machine inference algorithm of Shahbaz and Groz with appropriate input obtained from the system at hand.

Inferring Mealy Machines

We use Mealy machines to represent the state machine of a protocol, as they can be used in a more straightforward manner to model the behavior of protocols using requests and responses (which is quite typical in practice) than finite state machines or Moore machines can. Mealy machines differ from simple finite state machines in that for every state transition that is triggered by an input, an output is defined. The set I of possible inputs is called the input alphabet and the set O of possible outputs is called the output alphabet.

Angluin described an algorithm in [1] that can be used to infer minimal finite state machines, and this algorithm can be adapted for inferring Mealy machines too. In this work, we adopt the techniques of Shahbaz and Groz described in [3], and in the sequel, we refer to the Mealy machine inferring algorithm described in [3] as ${L_M}^+$.

Since we use the ${L_M}^+$ algorithm as a black box, a high level overview of its operation is sufficient for our purposes here. The ${L_M}^+$ algorithm is executed by a *learner* and it requires a *teacher*. The teacher knows the Mealy machine to be inferred, and the task of the learner is to infer that machine. The teacher can answer two types of queries for the learner: first, for a certain sequence of input characters, the teacher returns the output of the machine to be inferred (input query); second, the teacher can determine whether a certain Mealy machine conjectured by the learner is the same as the one to be inferred (equivalence query). If the conjectured machine differs from the real one, then the teacher returns a counterexample: a sequence of input characters for which the real and the conjectured machines produce a different output.

Applying and extending the ${L_M}^+$ algorithm

A Mealy machine can be used to model the state machine of a client-server protocol in a fairly straightforward manner: the input alphabet of the machine can contain the possible messages that the client may send to the server (i.e., the requests) and the output alphabet can contain the possible messages that the server may send to the client (i.e., responses, acknowledgements, errors, *etc.*).

Clearly, including all possible individual messages in the input and output alphabets can easily lead to problems: a huge resulting Mealy machine and a very long running time of the ${L_M}^+$ algorithm. For instance, if a message contains a 4-byte timestamp, then the alphabet would contain at least 2^{32} elements to represent all possible messages containing different timestamp values. To bring the size of the alphabets in a manageable range, we represent message types by the elements of the input and output alphabets instead of individual messages. A message type models a group of messages that have the same format but that may differ in the specific values in the fields of the given message type.

In order to work with messages types, we use two helper functions: a message classifier and a message generator. The message classifier function takes a particular message as input and returns its message type. The message generator function takes a message type as input and generates a valid message that has the specified message type. While the message classifier function should be deterministic, the message generator can be non-deterministic: the values of the message fields can be randomly generated as long as the message remains well-formed (i.e., consistent with its type). An additional function, a message updater is also useful, which takes as input the set M of all previously sent messages and a particular message m of this set, and returns a new message m' of the same type as m, such that the values of certain fields in m' are the mutations of the corresponding values in m, and M does not include m' yet. The updater function takes into account the semantics of certain fields: for instance, constant fields and identifiers are not mutated, a counter is mutated by incrementing it, *etc.* Such an awareness of semantics improves the efficiency and accuracy of our algorithm.

Recall that we aim at reverse engineering the protocol state machine of a system under test (SUT). To achieve this goal, we use the ${L_M}^+$ algorithm as the learner that infers the unknown Mealy machine representing the protocol, and we need to provide the teacher that answers the queries of the learner. We construct the teacher by using the above defined helper functions, and by sending messages to and observing responses of the actual implementation of the protocol provided by the SUT.

This works in the following way: We start by using the message generator helper function to produce messages for every message type. We use these pre-generated messages to avoid a deterministic protocol appearing to be non-deterministic due to freshly generated random values used in the same message at different stages of our algorithm. Then, we run the ${L_M}^+$ learner and we respond to its queries. Input queries are answered by first resetting the SUT, then sending the pre-generated messages (after running the message updater helper function on them) corresponding to the learner's input query to the SUT, and finally running the message classifier helper function on the SUT's responses to get the message types that the learner can understand. Equivalence queries are answered by generating random input queries, running them against both the SUT and the conjectured machine, and comparing their outputs. The number of queries needed to decide about equivalence with a given confidence level has been studied in [1] and we follow those guidelines. Once the ${L_M}^+$ learner conjectures a Mealy machine that is deemed correct, our algorithm terminates with that machine as the output.

However, the above described version of our algorithm may return an incomplete protocol state machine, because it may happen that only a single message of a given type is generated, which always triggers the same type of response, while it may be possible that other variants of the same message would result in a different response. Consider, for example, a request for reading the content of some memory address; the response can be the data found at the specified address or an error if the address was invalid. The extended version of our algorithm attempts to find these additional behaviors by finding messages of the same type that trigger different responses of the SUT. This is done by generating multiple messages of each type of the input alphabet I, and running the simple version of our algorithm with them. The resulting Mealy machine is analyzed and messages that have the same type but do not produce different behavior are removed (we call this step *deduplication*). Then, new messages are generated and added to the set of possible inputs, and the simple algorithm is executed again. This is repeated until a certain amount of runs in a row do not generate new messages that induce different behavior.

Implementation and evaluation on a simple protocol

We implemented the ${L_M}^+$ algorithm and our algorithm in Python, using the NetworkX[1] package for representing Mealy machines. We designed the implementation to be modular, such that the different components are well separated and easy to replace. This is important for future improvements and to be able to easily plug the functions that are different for each protocol (e.g., the message generator, the message classifier, and the part of the teacher that handles communication with the SUT).

For testing and illustration purposes, we constructed and used a simple protocol, which is illustrated in Figure 1. The protocol has 3 states: the starting state *DISCONNECTED*, the state *BASE*, where only *get* is a valid input, and the state *WRITE*, where both *get* and *write* are valid inputs. The difference between *get* and *bad_get* is that the parameter (target address) of the *get* message is valid, while it is invalid in a *bad_get*, and likewise with *write* and *bad_write*. Messages *get* and *bad_get* are of the same type, and similarly, messages *write* and *bad_write* have the same type. These two message types are used by the message generator to generate messages with random addresses.

Figures 2, 3, and 4 show the inferred Mealy machine in different rounds after deduplication. The request messages are postfixed with a number; this is a counter showing how many times the message generator was called when the given message was generated. The starting state is *, and every other state is labeled by the messages that can be used in sequence to reach that state. In the first round, the algorithm generates two instances of each message type, however, each instance of the same message type produced the same behavior, therefore, only one of them were kept (see Figure 2). In the second run, a new instance was generated from each message type, but these did not show new behavior either, so they were discarded too. In the third round, a variant of the *write* message type was generated that resulted in an *ok* response, as opposed to the *error* response triggered by the other variant of the *write* message, so this was

[1] https://networkx.github.io/

kept (see Figure 3). Finally, in the fourth round, the algorithm also finds a *get* message variant that results in different response observed so far, hence it is retained (see Figure 4). After 5 rounds with no new behavior found, the algorithm stopped. As we can see in Figure 4, in our example, the Mealy machine inferred is identical to the state machine of the example protocol (except for the names of the states and message variants, of course).

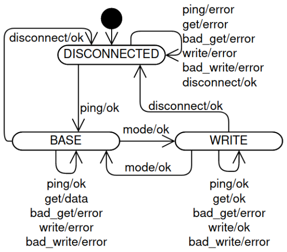


Figure 1: Protocol Mealy machine

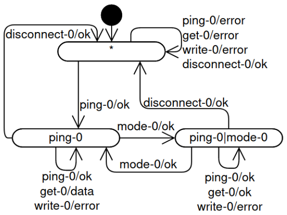


Figure 2: First round output

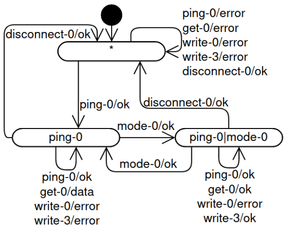


Figure 3: Third round output

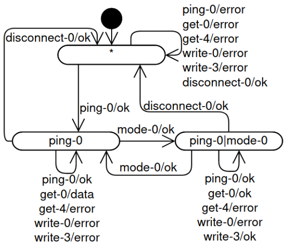


Figure 4: Fourth round output

Acknowledgment

The research presented in this paper has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013), the Hungarian National Research, Development and Innovation Fund (NKFIH, project no. 2017-1.3.1-VKE-2017-00029), and the IAEA (CRP-J02008, contract no. 20629).

References

[1] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[2] Gergő Ládi, Levente Buttyán, and Tamás Holczer. Message format and field semantics inference for binary protocols using recorded network traffic. In *IEEE Conference on Software, Telecommunications and Computer Networks (SoftCom)*, September 2018.

[3] Muzammil Shahbaz and Roland Groz. Inferring mealy machines. In *International Symposium on Formal Methods*, pages 207–222. Springer, 2009.

Component-based error detection of P4 programs

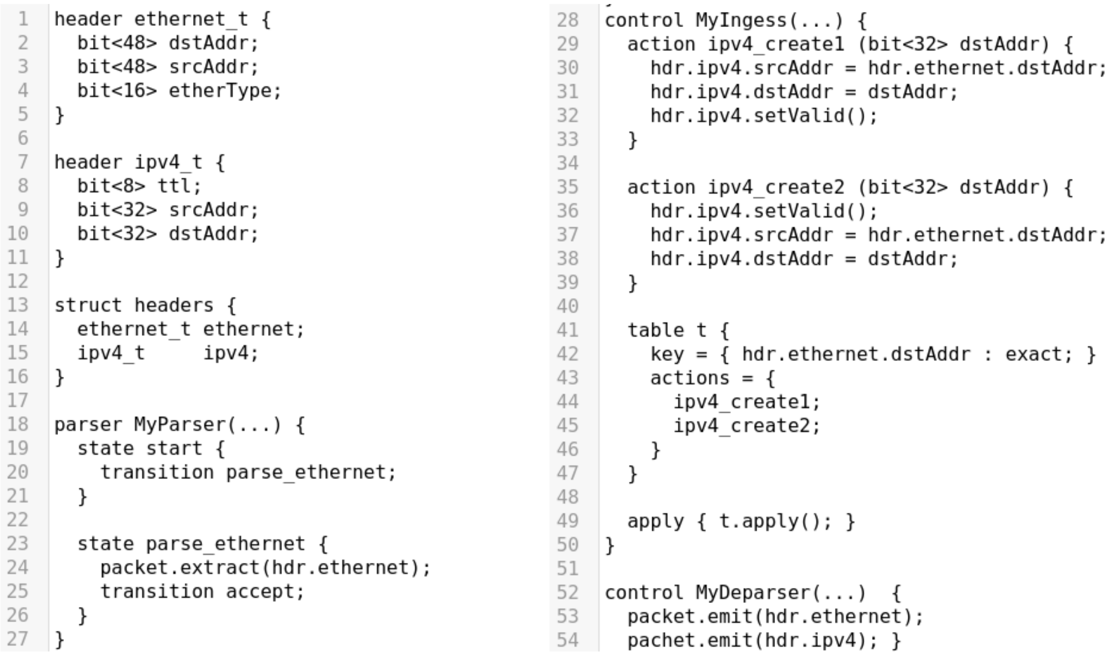
Gabriella Tóth and Máté Tejfel

Abstract: P4 is a domain-specific language to develop the packet processing of network devices. These programs can easily hide errors, therefore we give a solution to analyze them and detect predefined errors in them. This paper shows the idea, which works with the P4 code as a set of components and processes them one by one, while calculating their pre- and post-conditions. This method does not only detect errors between the components and their connections, but it is capable to reveal errors, which are hidden in the middle of a component. The paper introduces the method and shows its calculation in an example.

Keywords: P4, error detection, component

Introduction

We introduce a component-based formal method to detect errors in P4 programs. The antecedent of the method is an error detection based on a rule system [4]. With this idea, we approach the detection from backwards and process the code from the smallest units to the biggest ones. This solution can not only check the error possibilities but give additional information about the code for the developer.



```
 1  header ethernet_t {
 2    bit<48> dstAddr;
 3    bit<48> srcAddr;
 4    bit<16> etherType;
 5  }
 6
 7  header ipv4_t {
 8    bit<8> ttl;
 9    bit<32> srcAddr;
10    bit<32> dstAddr;
11  }
12
13  struct headers {
14    ethernet_t ethernet;
15    ipv4_t     ipv4;
16  }
17
18  parser MyParser(...) {
19    state start {
20      transition parse_ethernet;
21    }
22
23    state parse_ethernet {
24      packet.extract(hdr.ethernet);
25      transition accept;
26    }
27  }
28  control MyIngess(...) {
29    action ipv4_create1 (bit<32> dstAddr) {
30      hdr.ipv4.srcAddr = hdr.ethernet.dstAddr;
31      hdr.ipv4.dstAddr = dstAddr;
32      hdr.ipv4.setValid();
33    }
34
35    action ipv4_create2 (bit<32> dstAddr) {
36      hdr.ipv4.setValid();
37      hdr.ipv4.srcAddr = hdr.ethernet.dstAddr;
38      hdr.ipv4.dstAddr = dstAddr;
39    }
40
41    table t {
42      key = { hdr.ethernet.dstAddr : exact; }
43      actions = {
44        ipv4_create1;
45        ipv4_create2;
46      }
47    }
48
49    apply { t.apply(); }
50  }
51
52  control MyDeparser(...)  {
53    packet.emit(hdr.ethernet);
54    pachet.emit(hdr.ipv4); }
```

Figure 1: Example P4 code

P4 programs We work with the P4 language, which is a domain-specific programming language to develop the packet processing of network devices. When a packet arrives at a device as a bitstream, the P4 program gets that bitstream as the input and starts to work with it. In Figure 1 there is an example program. P4 programs work with the header information of a network packet. The developers can define what kind of header information they work with (rows 1-16.). P4 programs have three main processing parts: parser, modifier and deparser. The parser (rows 18-27.) gets the input and extracts the header information from the packet. The modifier part (rows 28-50.) modifies the header information – based on match-action tables, the content of which comes from an external controller – and after the modification, the deparser (rows 52-54.) creates the new packet with the calculated header information to forward it to the network.

Method of detection

Motivation We would like to create a tool, based on our results, to help the P4 developers to write correct programs. In this first version of the method, we concentrate only those errors to detect, which are caused by using invalid headers and uninitialized fields. Using them can cause undefined behaviour by unknown values. However, we still work with the subset of the P4 language, but we plan to extend the rule system to work with a bigger language and detect more error cases. Now we are also working on a representation of this method to be able to compare it with other error detectors [1, 3, 2].

This method based on the precondition and postcondition of the program units. As we see in Figure 1, there are different programming structures in the P4: actions, tables, control functions. It works with them one by one, and calculates condition pairs for each of them, starting from the smallest ones – like actions – finishing with the biggest ones – like control functions.

The phases of the formal method The input of the method is the source code that will be checked. The whole process contains three main phases: the Condition Calculator, the Main Condition Calculator and the Final Checker. The Condition Calculator has two parts: the call graph, and the subcondition calculator. The Condition Calculator check all of the components – actions, tables and control functions – in the P4 source and calculate pre- and post-condition pairs for them, while checking their correctness. The Main Condition Calculator works with the parser – gets the main precondition from it – and the deparser – gets the main postcondition from it. The Final Checker works with the calculated pieces of condition and checks that if every main precondition matches with the needed preconditions. If the matching is not correct then it means that there are some errors in the code. However, if the matching is right in the preconditions then it checks the same with the postconditions.

Condition Calculator The graph contains the calling relationship between the control functions, tables and actions. Based on the graph it will make an order of these components to process them with the condition calculator. The condition calculator goes through all of them and creates their pre- and post-condition, which describes the correct working of the program.

This phase processes the modifier part of the P4 code, and calculate pre- and post-conditions for the different components. The result of this phase shows the claim – what kind of header information it needs to work well – and the offering – what kind of header information it will create after the processing – of the components.

Call Graph The call graph handles the modifier part of the input. Its vertices are the components of this part – the control functions, match-action tables and actions – and the directed edges describes a calling relation between them.

After the processing of the code, it will create a list of the components. This list starts with those components, which do not depend on any other components, – they have no outgoing edges – and the last two will be the ingress and the egress.

Subcondition calculator The list of the components gives a hint for the subcondition calculator in which order it should process the components, therefore whenever a command of a component's calling is processed, the condition of that component will have been calculated. During the processing of the components, it uses a rule system, which will calculate a $condition$ $state$. The type of the $condition\ state$ – Figure 2 is a set of pairs, where a pair shows a name of the component and its list of pre- and post-condition as pairs. Every condition contains a $valid$ and an $invalid$ container, which contains a list of headers' and fields' name. During the method, we use the $valid$ word as a synonym of the property of initialized fields to be easier.

```
ConditionState ∈ {(name : [{pre : {valid : [ids], invalid : [ids]}, post : {valid : [ids], invalid : [ids]}})]}
```

Figure 2: Condition State Describer

The rule system uses an operator $\vdash$, the type of which can be seen in Figure 3. It works with a known $condition\ state$, a name of a component – the one, that is processed when it uses this rule – and the program code. The rules are based on the structure of the program, therefore it has a deterministic usage.

```
⊢: (ConditionState × Name × Program) → ConditionState
```

Figure 3: Type of $\vdash$

The rules of the system are rewriting rules – we have the expression in the bottom and we rewrite it to the top of the rule –, which define the behaviour of the $\vdash$ operator. In Figure 4, there are the rules for the assignment, sequence, table, skip and the calling of a table. In the rules, we use some notations: $(A \parallel B)$ shows a rewriting from A to B. The $notUsed$ is a function which gives a set of headers' and fields' name, which have not been used during the calculation – the calculation has no condition about them. The $Used$ is a really similar function, but here there are the names of the used units. The result of $Valid()$ function is $true$, when the unit is $valid$ in the postcondition. And $\times$ means it checks if the preconditions match, then it merge them, then checks if the postconditions match, then it also merge them, therefore we get a new condition pair.

$$\frac{H \cup H.n[pre \to Valid\{expr.notUsed()\}, post \to Valid\{h, h.f\}]}{H, n \vdash h.f = expr}\ V(expr)$$

$$\frac{H \cup (H.n \parallel H.n[pre \to Valid\{k.notUsed()\}] \times \{H.a_1, \ldots, H.a_n\})}{H, n \vdash keys : \{k\} actions : \{a_1, \ldots, a_n\}}\ V(k)$$

$$\frac{H \cup [H.n \parallel H.table\_name \times H.n]}{H, n \vdash table\_name.apply()}$$

$$\frac{H}{H, n \vdash Skip} \qquad\qquad \frac{(H, n \vdash S_1), n \vdash S_2}{H, n \vdash S_1;\ S_2}$$

where $V(e) = \forall x \in e.Used() : x.Valid()$

Figure 4: A subset of the rule system

Example

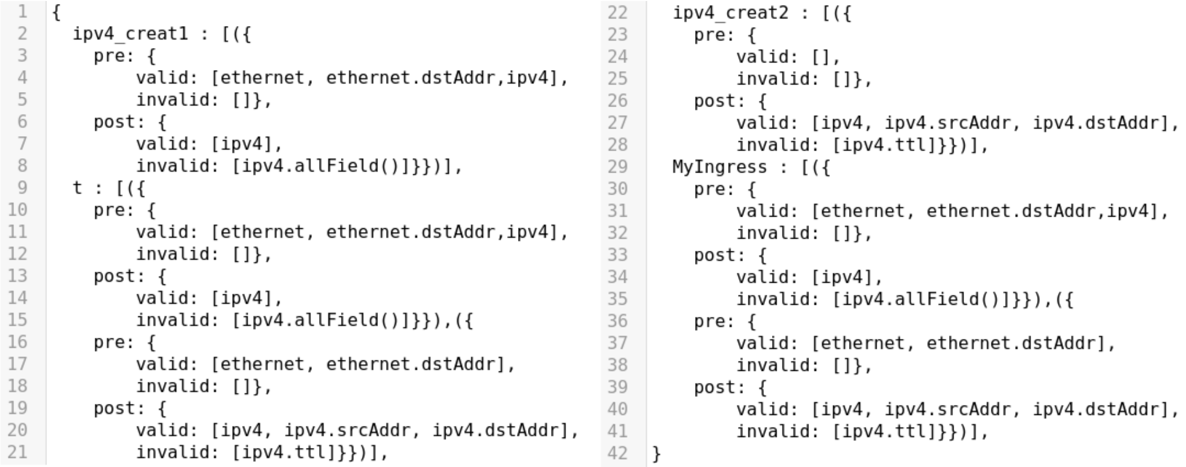
Figure 1 is an example P4 program which will be used to show an example usage of the method.

Calculate conditions First it creates the call graph. There is only an ingress control flow, which calls a table, which can call one of the two actions. Based on the graph it can create the following list to define the order of the processing:$[ipv4\_create1, ipv4\_create2, t, MyIngress]$.

Based on the order of the list, it starts the calculation from the following expression:

$$(((\text{Empty}, \text{"ipv4\_create1"} \vdash \text{ipv4\_create1}), \text{"ipv4\_create2"} \vdash \text{ipv4\_create2}), \text{"t"} \vdash \text{t}), \text{"MyIngress"} \vdash \text{MyIngress}$$

After the calculation of conditions we will get pairs of pre- and post-conditions of the different components – Figure 5.



```
{
  ipv4_creat1 : [({
    pre: {
        valid: [ethernet, ethernet.dstAddr,ipv4],
        invalid: []},
    post: {
        valid: [ipv4],
        invalid: [ipv4.allField()]}})],
  t : [({
    pre: {
        valid: [ethernet, ethernet.dstAddr,ipv4],
        invalid: []},
    post: {
        valid: [ipv4],
        invalid: [ipv4.allField()]}}),({
    pre: {
        valid: [ethernet, ethernet.dstAddr],
        invalid: []},
    post: {
        valid: [ipv4, ipv4.srcAddr, ipv4.dstAddr],
        invalid: [ipv4.ttl]}})],
  ipv4_creat2 : [({
    pre: {
        valid: [],
        invalid: []},
    post: {
        valid: [ipv4, ipv4.srcAddr, ipv4.dstAddr],
        invalid: [ipv4.ttl]}})],
  MyIngress : [({
    pre: {
        valid: [ethernet, ethernet.dstAddr,ipv4],
        invalid: []},
    post: {
        valid: [ipv4],
        invalid: [ipv4.allField()]}}),({
    pre: {
        valid: [ethernet, ethernet.dstAddr],
        invalid: []},
    post: {
        valid: [ipv4, ipv4.srcAddr, ipv4.dstAddr],
        invalid: [ipv4.ttl]}})],
}
```

Figure 5: Result of calculation of conditions

Calculate the main conditions The main conditions are calculated from the parser and deparser. It works almost the same way as in our previous paper [4], therefore we do not go into details, but we got the result in another format.

```
Pre: [{
  Valid: [ethernet, ethernet.allFields()], InValid: [ipv4, ipv4.allFields()]}]

Post: [{
  Valid: [ethernet, ethernet.allFields(), ipv4, ipv4.allFields()]},{Valid: [drop
    ]}]
```

Final Checker In the end, the main conditions will be matched – in this case only – with the conditions of the *ingress*. We can see, the first precondition of the *ingress* does not match with the main condition because it would need a *qmathitvalid ipv4* header, but after the parsing phase it is not *valid*, therefore it shows an error here. The second precondition of the *ingress* would be good, but its postcondition is not matching with any of the main postconditions. The first main postcondition could be good, but it needs a *valid ipv4* header with *valid* fields, and after the program execution, the *ipv4.ttl* field is not valid. It means there is another error in that point. Our next job to define what kind of error does the method find.

Acknowledgements

This research was supported by the project “Integrated program for training new generation of scientists in the fields of computer science”, no EFOP-3.6.3-VEKOP-16-2017-00002.

References

[1] Lucas Freire, Miguel Neves, Lucas Leal, Kirill Levchenko, Alberto Schaeffer-Filho, and Marinho Barcellos. Uncovering Bugs in P4 Programs with Assertion-based Verification. In *Proceedings of the Symposium on SDN Research*, SOSR '18, pages 4:1–4:7, New York, NY, USA, 2018. ACM.

[2] Jed Liu, William Hallahan, Cole Schlesinger, Milad Sharif, Jeongkeun Lee, Robert Soulé, Han Wang, Călin Caşcaval, Nick McKeown, and Nate Foster. P4v: Practical Verification for Programmable Data Planes. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 490–503, New York, NY, USA, 2018. ACM.

[3] Radu Stoenescu, Dragos Dumitrescu, Matei Popovici, Lorina Negreanu, and Costin Raiciu. Debugging P4 Programs with Vera. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 518–532, New York, NY, USA, 2018. ACM.

[4] Gabriella Tóth and Máté Tejfel. A formal method to detect possible p4 specific errors. In *Position Papers of the 2019 Federated Conference on Computer Science and Information Systems*, volume 19 of *Annals of Computer Science and Information Systems*, pages 49–56. PTI, 2019.

Embedding QR code onto triangulated meshes

György Papp, Miklós Hoffmann and Ildikó Papp

Abstract: The QR code is widely used to provide easily accessible information to anyone with a smartphone, and since these devices play a significant role in our everyday life, a QR code can be read by most people. However, significant deformation can make it challenging to decode a printed QR code. In order to avoid deformation, the printed label should be affixed on a developable surface patch of a 3D model. In the work of Kikuchi et al. [3] and Peng et al. [5], the QR code is embedded onto surfaces to give more freedom in providing additional information. In the first work, the QR code is engraved onto B-spline CAD models, while in the second one, it is embedded code onto general meshes and optimizes its dark modules and minimizes their carving depth to increase the engraved QR code readability. In this paper, we introduce a method to embed a QR code onto any area of general meshes without using subdivision on the mesh. Besides, we propose a method to determine the largest size of the QR code automatically for a user-selected position. Also, our method can find better direction to carve the QR code and achieve a larger size for a user given point in even on highly curved areas. Additionally, we introduce a solution to speed up finding the carving depth for embedding the QR code. Our proposed method uses the Horizon-Based Ambient Occlusion for calculating the shadow in the engraved areas with the help of the GPU.

Keywords: QR code, 3D printing, triangulated mesh, embedding, ambient occlusion

Introduction

Providing information for items or parts is globally used for describing how to use or assemble them. This supplementary information can be enclosed in the form of a printed document, which can contain QR codes besides text. Moreover, the QR code even can be printed on a label and affixed onto the surface. However, the placement of the label is limited for developable surfaces to avoid significant deformation of the QR code, which could negatively affect its readability. This restriction can be avoided by using a 3D printer for embedding the QR code. The method encloses the additional information by engraving the dark modules of the QR code onto any surface, not only developable ones. Kikuchi et al. [3] embed QR codes onto CAD models, which are constructed by using B-spline surfaces, and Peng et al. [5] increase the readability of the QR codes, which are engraved onto general meshes, by modifying its modules and optimizing its carving depth. However, these solutions require modification on the surface before the QR code can be embedded. In this paper, we propose a method that can engrave QR codes onto any triangulated mesh without making any preprocessing step before the embedding. Besides, our engraving solution can determine the largest size for a QR code automatically at a given point on the mesh. Moreover, we propose to use ambient occlusion techniques to speed up the process of finding the optimal carving depth of the QR code with the help of the GPU.

Previous work

Embedding information in the form of a QR code into objects can serve different purposes. In the work of Wei et al. [6], the QR code is embedded by using selective laser melting (SLM) technology into metallic components as a safety feature to fight against illegal counterfeits. Another one is to track components through the additive manufacturing chain and prevent 3D printing attacks by embedding obfuscated QR codes [2]. Besides these, objects also can be tagged with a QR code to describe and provide useful information about them. Aircode [4] embeds the user-defined information by placing structured air pockets under the surface of the item.

Kikuchi et al. [3] proposed a technique to embed QR codes onto CAD models, which are constructed from B-spline surfaces. The black modules of the QR code are carved onto the surface to shadow them and create the required contrast between the white and dark regions of the QR code for decoding. In their work, a method is presented to find the optimal carving depth for the QR code. In the work of Peng et al. [5], a method has been proposed to improve the readability of the embedded QR code. Firstly, the modules of the QR code are modified to decrease the number of dark modules that are connected. Then, the carving depth is minimized for the QR code with an iterative method. It defines different carving depth for small areas in the modules.

The gaming industry uses different global illumination methods to achieve a more realistic look for the game. Since the high frame rate is essential, the Screen Space Ambient Occlusion (SSAO) techniques are widely used for quickly producing a visually convincing approximation of realistic lighting for a given camera view. One of the many available techniques is the Horizon-Based Ambient Occlusion [1] (HBAO), which is used in our solution.

Embedding QR code

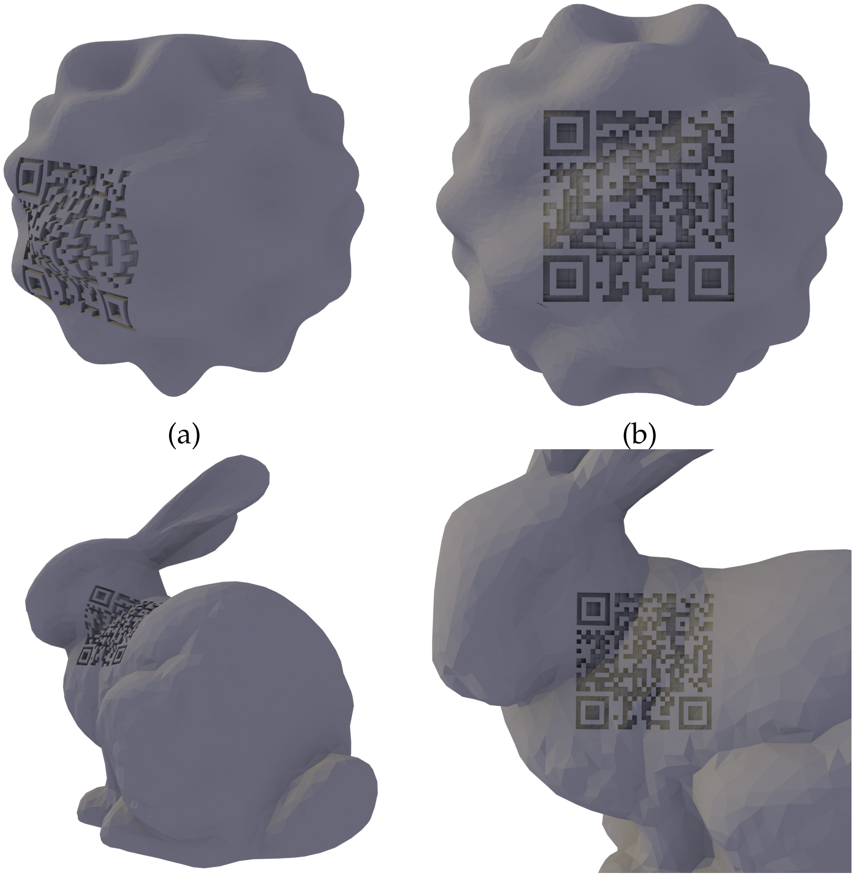
Similar to the work of Peng et al. work, we also use triangulated meshes for engraving QR codes. However, our goal was to preserve the original mesh without performing any preprocessing step on it. In Peng's work, the selected region for the QR code is being remeshed to achieve a dense triangulation. Also, the proposed method of Kikuchi et al. for B-spline surfaces has a preprocessing step to generate and insert the required knot lines and values. Our proposed method embeds the QR code in the following steps.

- Finds the direction and position to project the QR code
- Removes affected triangles and triangulate the modules of the QR code
- Find their carving the depth and connects the embedded QR code with the mesh

Finding a plane for the QR code, which defines the position and the direction to project the QR code is an iterative process. An initial plane h is centered at the given point by using an initial size and orientation with the normal vector of the selected point. Then, its corners are projected onto the surface by using the central projection. The center of the projection C is positioned on the normal vector of plane h, and its distance from h is defined by a free parameter g. As a result, a truncated pyramid is formed from the plane and the projected points of the corners. It contains the vertices, which normal vectors are used to update plane h, by replacing its normal vector with the average of the vertices' normal vector in each iteration. The next iteration uses the updated plane as an initial plane.

Our results showed that after the first iteration, the normal vector of plane h is good enough to stop the iteration. Also, the other reason to stop is that in the following steps, only small corrections can be made on the normal vector, which does not have significant effects later in the projecting and engraving process. During the iteration, our method can maximize the size of the QR code for a given viewing direction by using the contour of triangles that are visible from it. This filtering criteria of the triangles can easily be changed, for example, to avoid highly curved areas.

The modules of the QR code are projected by using the same central projection and plane h. Then, our method triangulates and engraves the dark modules while it finds their carving depth. As the last step, a connection between the mesh and the triangulated QR code is formed by using constrained Delaunay triangulation because a hole is created by removing triangles that contained a projected QR code. Figure 1. shows the resulting embedded QR code of our method on two different models.



(a)

(b)

(c)

(d)

Figure 1: Embedding a QR code onto a sphere with a bumpy surface and the Stanford bunny. Our method can engrave QR codes onto highly curved areas whilst preserving readability. The models from the images can be viewed on the following website `https://arato.inf.unideb.hu/papp.gyorgy/qr_embedding/`.

Carving depth with ambient occlusion

The process of calculating the craving depth for the QR code is time-consuming, in both the work of Peng et al. and Kikuchi et al. Since the embedded QR code is read from the direction of the projection we propose to use a Screen Space Ambient Occlusion technique in order to speed up this process by harnessing the computation capacity of the GPU. Our solution is also an iterative process like the ones from Kikuchi's and Peng's work. However, our method calculates the carving depth for each module, not for the whole QR code or small parts of it. We use the Horizon-Base Ambient Occlusion to render the QR code. The viewpoint with the direction is given by the central projection from the embedding process. However, in the case of using parallel projection, its direction also can be used with our method. The resulting image of HBAO is used to calculate the average shadow for each module. The carving depth is increased for each module until its shadow is under a threshold value, similar to the one in Kikuchi's work. The HBAO technique only approximates real shadow. However, the different carving depth of the modules correctly can be predicted with its result. Also, it can produce a result in 5–10 seconds instead of minutes.

Conclusion

There are numerous ways to provide information for components or objects. One of them is to encode the information into a QR code and then embed it into an item. In this paper, a solution was proposed on how to engrave QR codes onto the surface of triangulated meshes. Our method preserves the original triangulated mesh and only changes the area of the QR code and its adjacent triangles. Also, larges size can be achieved for a given viewpoint by using our solution than using Principal Component Analysis for projecting and embedding the QR code. The rest of the mesh and the embedded QR code are connected with the help of the Constrained Delauney triangulation method. Further, we propose to use the central projection for embedding the QR code and avoid the appearance of deformation in cases of highly curved areas.

Additionally, we propose to use a Screen Space Ambient Occlusion technique to speed up the process of finding the carving depth for the modules of the QR code. The HBAO technique was used in our solution to utilize the computation capacity of the GPU and find the carving depth in seconds. The appearance of the ray-tracing capable GPUs can provide an excellent future work opportunity in terms of implementing a faster carving depth search algorithm with more accurately calculated shadows.

Acknowledgements

This work was supported by the construction EFOP-3.6.3-VEKOP-16-2017-00002. The project was supported by the European Union, co-financed by the European Social Fund.

References

[1] Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 Talks*, SIGGRAPH '08, New York, NY, USA, 2008. Association for Computing Machinery.

[2] Fei Chen, Yuxi Luo, Nektarios Georgios Tsoutsos, Michail Maniatakos, Khaled Shahin, and Nikhil Gupta. Embedding tracking codes in additive manufactured parts for product authentication. *Advanced Engineering Materials*, 21(4):1800495, 2019. doi: 10.1002/adem.201800495; 06.

[3] Ryosuke Kikuchi, Sora Yoshikawa, Pradeep Kumar Jayaraman, Jianmin Zheng, and Takashi Maekawa. Embedding qr codes onto b-spline surfaces for 3d printing. *Computer-Aided Design*, 102:215–223, 2018. ID: 271502.

[4] Dingzeyu Li, Avinash S. Nair, Shree K. Nayar, and Changxi Zheng. Aircode: Unobtrusive physical tags for digital fabrication. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, pages 449–460, New York, NY, USA, 2017. ACM.

[5] Hao Peng, Lin Lu, Lin Liu, Andrei Sharf, and Baoquan Chen. Fabricating qr codes on 3d objects using self-shadows. *Computer-Aided Design*, 114:91–100, 2019.

[6] Chao Wei, Zhe Sun, Yihe Huang, and Lin Li. Embedding anti-counterfeiting features in metallic components via multiple material additive manufacturing. *Additive Manufacturing*, 24:1–12, 2018. ID: 306190.

Fog-enhanced Blockchain Simulation

Hamza Baniata

Abstract: Blockchain (BC) technology was proven as enhancement factor for security, decentralization, and reliability, leading to be successfully implemented in crypto-currency industries. However, it is not standardized yet, so it is still in the research phase for other applications. Fog computing (FC) is one of the recently emerged paradigms that needs to be improved to serve Internet of Things (IoT) environments of the future. To evaluate a proposed integration of Fog-Blockchain protocol/method, a suitable simulation environment is needed for the validation before the implementation. Such environment may preserve high cost and great efforts. Most of the available simulation environments provide Fog simulation, or Blockchain simulation, but not both. In this paper, we present the results of the first phase of our research, leading to a Fog-Blockchain simulator, whose main goal is to ease the experimentation/validation of Fog-blockchain protocols/methods. We propose that different consensus algorithms, different placement options of the BC in the FC architecture, and different uses of the BC in the simulation shall be implemented. We validate our results by applying a case study, showing the pattern of average time consumption of Blocks validation in a single BC node.

Keywords: Fog Computing, Blockchain, Simulation.

Introduction

Fog Computing (FC) is a geographically distributed computing architecture, in which various heterogeneous devices at the edge of network are ubiquitously connected, to collaboratively provide elastic computation, communication and storage services [1]. Blockchain (BC) is a distributed ledger technology in the form of a distributed transactional database, secured by cryptography, and governed by a consensus mechanism [2], where participants that do not fully trust each other agree on the content of the ledger by running a consensus algorithm [3].

Participants or *nodes* of a BC network are mostly divided into two groups: *Miners* and *Non-Miners* [4]. Miners are the ones eligible to collect new valid transactions that are waiting to be confirmed, combine them in one block, and distribute it to the network. That is, if the Block was validated by two conditions, ability/wealthiness of all transactions generators to perform the transactions, and a correct puzzle solution. The puzzle condition applies in most proof-based consensus algorithms [5], such as the Proof of Work (PoW), Proof of Schedule (PoSch), and Proof of Stack (PoS) algorithms. Different algorithms set different threshold difficulty of the puzzle, according to the application used. For example, Bitcoin's algorithm sets the difficulty of block generation to wherein one block is generated every 10 min [6].

Generally speaking, a proof-based BC system should define the following steps: Transaction Generation, Transaction Distribution, Transaction Validation, Transaction Pooling, Block Mining, and Block Confirmation. *'A Verification process'* indicates the recognition of a transaction generator by other network entities through the linkage with his/her public/private keys.

In this paper, we propose a simulator specifically designed for mimicking the deployment of Proof-based BC in a FC architecture. The concepts of Validation, Verification, Confirmation, Proof-based Consensus, and Mining, are all available in the current version of the simulator, while the re-configuration of the fog properties will be available in the second phase of our research. Meanwhile, analysis of the mimicry can be easily obtained.

The remainder of this paper is organized as follows. Section presents short state-of-the-art review regarding some integration approaches of BC and FC. Section provides an overview of our research work, and introduces the simulator components, code, and framework. Section presents our validation of the code. Finally, Section concludes.

Related work

Cisco started its own virtual Fog Data Services platform early in 2019 [7], with at least 1.00 GHz computing power, 4-GB vRAM, and 23 GB storage. The main techniques to design hardware platforms able to cope with IoT requirements, such as power consumption and management, IO architecture, and security were surveyed in [8].

The trust in Blockchain is typically gained by the majority consensus of a piece of information validity, and a user verify-ability [9]. However, the proposed architecture in [10] limits the necessity of verification by all nodes of the network. That is, public keys and private keys -online and offline versions- are only registered and verified by few previously-verified and trusted neighbours instead of having to verify it by all network nodes.

Authors of [11] proposed a reputation system for fog nodes, using Blockchain Ethereum smart contracts. The system suggests that IoT devices rate fog nodes according to specific modifiable criteria. Accordingly, fog nodes obtain trustworthiness value that would indicate how reliable they are. IoT devices' credibility is also computed, according to specific contributions.

Our proposed Blockchain-Fog simulation environment

Searching in research databases, we found that a BC can be deployed in any layer of the FC architecture (i.e. End-user layer, Fog layer, Cloud layer). BC can be deployed for one or more of the following purposes: Data management, Payment/Trading, Identity/Authentication management, and Reputation management. Consequently, a BC-FC simulation environment should allow users to choose: the consensus algorithm they want to implement, the FC layer wherein they need to deploy the BC, and the purpose/protocol of the implemented BC. Meanwhile, the processing steps defining the BC, along with the Fog properties, must be clear and configurable. The simulation environment should also be able to provide analysis and reports during, and after the experiment, and a user-friendly Graphical User Interface (GUI).

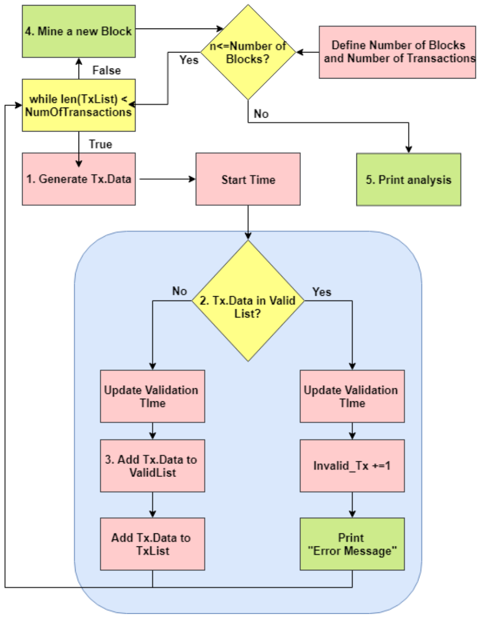


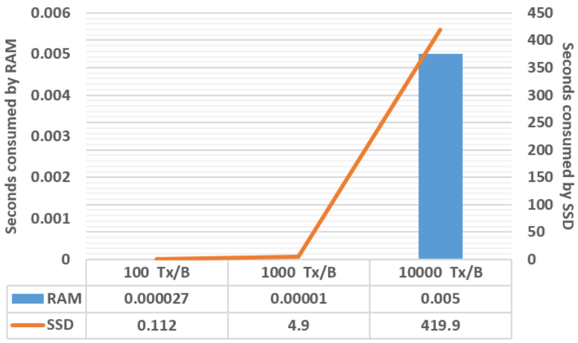
Figure 1: Workflow of our current BC simulator

To accomplish our goals, we decided to conduct our work in two phases: The first phase includes implementing a BC simulation that satisfies the processing steps that define a BC, except for the 'distribution step'. After experimenting/validating the BC simulation, we survey all available research papers/projects that integrates BC in a FC environment. Such survey may direct the focus of our research into major properties, recommendations, and challenges of a BC-FC integration. In the second phase, we investigate BC and FC simulation environments, which highlights the properties, services, abilities, and weaknesses of those environments. We then compare our observations about the BC-FC integration approaches, with available simulation environments, leading to set strong knowledge foundations to serve in our second phase. Accordingly, we implement the 'Distribution step' in a way that satisfies BC-FC integration recommendations, and solves the challenges presented about current simulation environments.

So far, we have finished the first phase, by implementing a BC, using Python programming language, that satisfies all BC steps, except for the distribution step. Our code is provided publicly at GetHub.com[1]. The simulation of the first phase was successfully used to validate our proposed PF-BVM mechanism published in an international conference [12]. We also conducted a comprehensive survey concerning BC-FC integration approaches, and submitted it to an internationally recognized journal. Our observations and results will be publicly available as soon as the survey is accepted for publication.

As presented in Figure 1, our implementation randomly generates numbers representing the transactions (Txs). Transactions can also be generated by the user, as an input. Once a Tx is generated (Step1), it gets checked whether it is in a list of the valid Txs (Step2). If the data already exists in the list, an error message is printed on the screen, and the program moves to the next randomly generated transaction. Otherwise, the program adds it to the list (Step3). Once the number of Tx/B -or gas limit- is reached the block is mined (Step4), and added to the chain (Step5).

Case Study



	100 Tx/B	1000 Tx/B	10000 Tx/B
RAM	0.000027	0.00001	0.005
SSD	0.112	4.9	419.9

Figure 2: Comparison of average Block validation time for BC is saved in RAM vs. in Hard Disk

The evaluation experiment in this paper aims to show the validity of our code, and some of the features it provides.

Some BC simulation environments like BlockSim [13], PeerSim [14], and iFogsim [15], simulates the validation step with a delay without actually performing the validation, hence all transactions are considered valid. We conducted a comparison experiment in which we compute the *real* average time needed to validate a whole block, for different gas limit values, while the BC is held in the RAM of the computer, and in the secondary storage unit of the computer. The first code keeps the BC in the RAM, while the second code keeps the BC on hard disk. We performed the experiment using an Intel i5-8265U CPU, backed up by 12 GB of DDR4 SDRAM, 45 GB vRAM, 500 GB SSD, Apache HTTP Server 2.4.41, and a MySQL database on the hard disk. We tested three scenarios of gas limit, 100Tx/B, 1000Tx/B, and 10000Tx/B, where 10 blocks are validated, mined, and confirmed. The results of our experiment is presented in Figure 2.

According to this experiment, we showed, using our simulation code, that more transactions saved in the ValidList, and more Tx/B rate, lead to exponential increase of average block validation time. We also showed how saving the BC on RAM decreases the time consumption compared to the case when it is saved on the hard disk. However, the exponential pattern remains in both cases despite the difference of range in the time consumption metric.

Conclusions and Future Work

In this paper we introduced our work on a Blockchain-Fog simulation environment, whose main goal is to ease the experimentation and validation of new BC-FC integration protocols/

[1] https://github.com/HamzaBaniata/BlockChainValidation

methods. We divided our work on the simulation into two phases, the first includes the implementation of a BC simulation on a single node, and surveying the state-of-the-art regarding BC-FC integration systems. We validated our work in the first phase using a case study that provided usable analysis of the average validation time of a block on a single node. The second phase, which is our future work, includes developing a BC-FC simulation, that allows users to define and configure the simulation regarding the role of the BC, the placement of the BC in FC, and the deployed consensus algorithm.

References

[1] Shanhe Yi, Zijiang Hao, Zhengrui Qin, and Qun Li. Fog computing: Platform and applications. In *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)*, pages 73–78. IEEE, 2015.

[2] Roman Beck, Michel Avital, Matti Rossi, and Jason Bennett Thatcher. Blockchain technology in business and information systems research, 2017.

[3] Carlos Faria. Blocksim: Blockchain simulator. 2018.

[4] Andreas M Antonopoulos. *Mastering Bitcoin: Programming the open blockchain*. " O'Reilly Media, Inc.", 2017.

[5] Giang-Truong Nguyen and Kyungbaek Kim. A survey about consensus algorithms used in blockchain. *Journal of Information processing systems*, 14(1), 2018.

[6] Yusuke Aoki, Kai Otsuki, Takeshi Kaneko, Ryohei Banno, and Kazuyuki Shudo. Simblock: a blockchain network simulator. *arXiv preprint arXiv:1901.09777*, 2019.

[7] Cisco. Cisco fog data services. https://www.cisco.com/c/en/us/products/cloud-systems-management/fog-data-services/index.html?dtid=osscdc000283, 2019. [Online; accessed 09-November-2019].

[8] Maurizio Capra, Riccardo Peloso, Guido Masera, Massimo Ruo Roch, and Maurizio Martina. Edge computing: A survey on the hardware requirements in the internet of things world. *Future Internet*, 11(4):100, 2019.

[9] Steem. Steem an incentivized, blockchain-based, public content platform. https://steem.io/SteemWhitePaper.pdf, 2017. [Online; accessed 21-November-2019].

[10] Louise Axon. Privacy-awareness in blockchain-based pki. *Cdt technical paper series*, 2015.

[11] Mazin Debe, Khaled Salah, Muhammad Habib Ur Rehman, and Davor Svetinovic. Iot public fog nodes reputation system: A decentralized solution using ethereum blockchain. *IEEE Access*, 7:178082–178093, 2019.

[12] Hamza Baniata and Attila Kertesz. Pf-bvm: A privacy-aware fog-enhanced blockchain validation mechanism. In *Proceedings of the 10th International Conference on Cloud Computing and Services Science*, volume 1, pages 430–439, 05 2020.

[13] Maher Alharby and Aad van Moorsel. Blocksim: A simulation framework for blockchain systems. *ACM SIGMETRICS Performance Evaluation Review*, 46(3):135–138, 2019.

[14] Alberto Montresor and Márk Jelasity. Peersim: A scalable p2p simulator. In *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*, pages 99–100. IEEE, 2009.

[15] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296, 2017.

Activity Recognition Model for Patients Data Stream using Adaptive Random Forest and Deep Learning Techniques

Hayder K. Fatlawi, Attila Kiss

Abstract: Precise detection of the current activity status for chronic diseases patients could play a significant role for protect their lives against sudden decline in health. Combining the information form various data resources present a reasonable challenge.On the other hand, stream classification techniques have a privilege of low computational time but they need a feedback for adapting the classifier. This work proposes a model for providing efficient automatic feedback for adaptive random forest classifier using deep learning classifying of video stream from surveillance systems.

Keywords: Random Forest ;Deep Learning ; Data Stream ;Activity Recognition

Introduction

Recognition of human activity using machine learning presents important tool for avoiding mortality or risky injuries as a result of the fall, especially for patients with chronic diseases and the elderly [1] [2] [3]. Many of data resources can provide a real time stream data for patients such as smart phone, smart watch, wearable sensors, and surveillance systems. Each one of those resources has its own data features and combining the information form them present a reasonable challenge. Although the low computational time of stream classification techniques but they need a feedback for adapting the classifier.

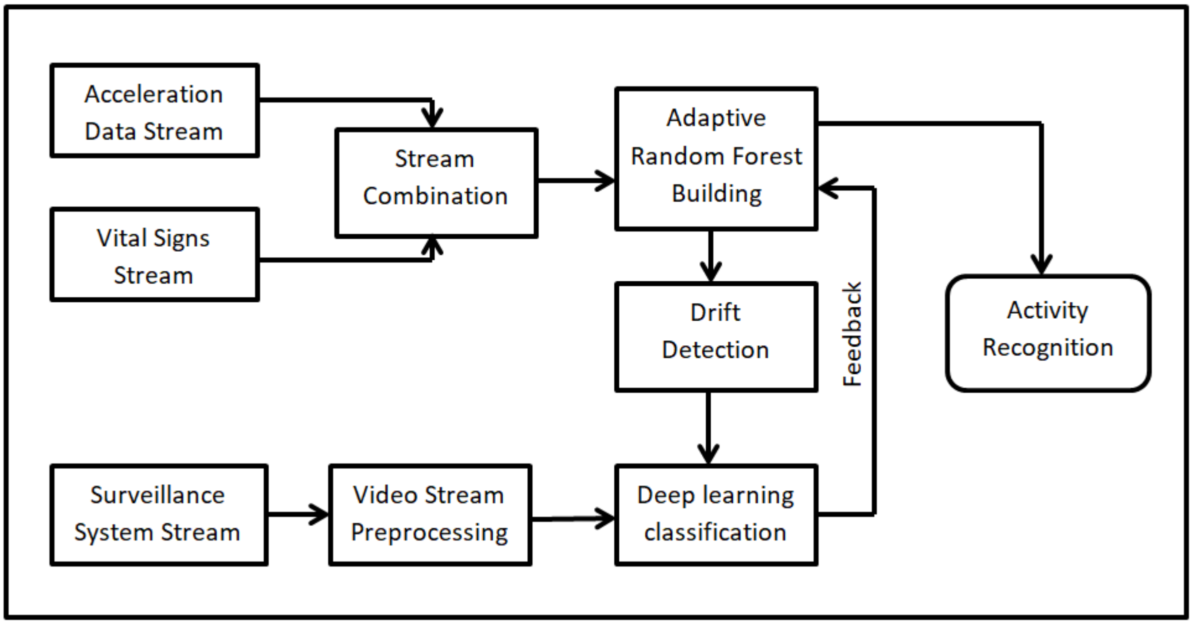
In case of using stream mining with sensors data of smart devices, there is a necessity to provide the feedback in an efficient way without depending on the patient. The recognition of activity using the data of surveillance systems could provide a feedback for the quality of the decision that made by stream mining classifier. Deep learning present an efficient technique for classifying the video stream. Convolutions Neural Networks (CNN) is one of deep learning techniques, it has a limitation which is handling only the spatial information in which video frames that extracted from videos are considered as a static images [4]. Three-dimensional convolutional neural networks with a linear classifier has a better performance in dealing with video recognition tasks [5].

The field of human activity recognition has increasing research interest recently. In [6] a multi-fused features system based on sequences of depth map using temporal motion data and joints of human skeleton. Code vectors of those features was trained using hidden Markov model(HMM), the experimental results on three datasets showed better performance of the proposed system. Temporal patterns based algorithm was proposed by [7] to represent activities for auto recognition. High accuracy results was obtained using the proposed algorithm on real dataset.

Wearable sensor based human activity recognition system was proposed by [8] to identify ballet movements for investigating a dancer's pain. The research tried also to determine the effect of location and number of sensors on classification accuracy. In [9] Multi hidden Markov models method was proposed for activity segmentation and recognition of stream sensor data without using sliding window methods. The researchers applied the proposed method on two smart home datasets. Our model combines the strength of both adaptive random forest and deep learning to generate a integrated model for recognize activity based on multi types of stream data.

Methodology

Three types of data stream are utilized in the proposed model, the first two types i.e acceleration data and vital signs are combined. The resulted data stream is considered an input for Adaptive Random Forest (ARF) algorithm. If drift is detected, deep learning classification will used based on the third type of data i.e surveillance system stream, and the result of this classification is used as a feedback for the decision of ARF. Figure 1 illustrates the steps of the proposed model.



Acceleration Data Stream
Vital Signs Stream
Stream Combination
Adaptive Random Forest Building
Drift Detection
Feedback
Activity Recognition
Surveillance System Stream
Video Stream Preprocessing
Deep learning classification

Figure 1: The proposed Model for Real-Time Patients Monitoring

The combination of sensors data requires synchronizing the samples of the same time slice, the size of stream sample is a user defined parameter. ARF used very fast decision tree (VFDT) as a base classifier which is a variation from typical decision tree designed for adapting with distribution changes. The ensemble of ARF contains a specific number from base classifiers i.e VFDT which are created using random sub-sampling with replacement from data stream. For detecting drift change ARF uses adaptive sliding window (ADWIN) which tracking the average of bits in the stream and change the length of windows if a change in average value happen.

Three-dimensional architecture for convolutional neural networks which proposed by [9] is used for analysis the video stream. Many of preprocessing steps are applied for the video frames starting from the moment of drift change detection and step backward for a specific interval time. The architecture of CNN contains four convolution layers with 3 x 3 x 3 kernels, two max-pooling layers, with two fully connected layers, and ending with one softmax output layer that dedicate if the fall activity state is yes or no. This simple design aims to minimize the response time, the result is sent as a feedback for ARF to modify its classifiers based on this more reliable decision. Algorithm 1 describe all the steps of the proposed model.

Discussion and Expected Results

Real time response from the recognition system with low time latency is very important for treatment process. Surveillance video system stream requires many of preprocessing steps

Algorithm 1 Algorithm of Real-Time Patients Monitoring

```
Result: Activity Class
initialization
S1 acceleration data stream , S2 vital signs stream, S3 surveillance systems video stream.
for each time t do
    Read S1,S2.
    Combine the two streams SC= S1 & S2
end for
for each SC in time t do
    Apply Adaptive Random Forest on SC and get classification result CA.
    if DiftDetection()=True then
        Apply preprocessing on S3 in time t.
        Training Deep learning network on S3 and get classification result CB.
        if CA = CB then
            Activity Class = CA
        else
            Reset Window size of Adaptive Random Forest
        end if
    else
        Activity Class = CA
    end if
end for
```

which delayed the response. So, the key in our proposed model is utilizing sensors data stream which has a small size and requires few preprocessing operations, and trigger the deep learning only if a change in this data will detected.

The fast decision of fall detection will be made by ARF which can be send to the responsible for the patient, meanwhile CNN will confirm the decision to avoid false detection and this confirmation will be used to adopt ARF. Based on fast response from ARF and high quality of CNN, we expect high accuracy and fast performance from the proposed model.

Conclusions

Machine learning can be an effective tool to preserve the lives of patients using activity recognition. The combination of adaptive random forest and convolutional neural networks classification techniques can provide a more reliable system with the presence of multi-sources of stream data. Expected high accuracy and fast performance from the proposed model were considered based on a fast response from ARF and the high quality of deep learning.

Acknowledgment

The project was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

References

[1] Baños, Oresti and Damas, Miguel and Pomares, Héctor and Rojas, Ignacio. Activity recognition based on a multi-sensor meta-classifier, *International work-conference on artificial neural networks,Springer*,pp.208-215 2013.

[2] Chikhaoui, Belkacem and Ye, Bing and Mihailidis, Alex. Ensemble learning-based algorithms for aggressive and agitated behavior recognition, *Ubiquitous Computing and Ambient Intelligence,Springer*. pp. 9-20, 2016.

[3] Al-Fatlawi, Ali H and Fatlawi, Hayder K and Ling, Sai Ho. Recognition physical activities with optimal number of wearable sensors using data mining algorithms and deep belief network, *39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), IEEE*, pp. 871-2874, 2017.

[4] Jalal, Ahmad and Kim, Yeon-Ho and Kim, Yong-Joong and Kamal, Shaharyar and Kim, Daijin. Robust human activity recognition from depth video using spatiotemporal multi-fused features, *Pattern recognition, Elsevier*, Volume 61, pp. 295-308, 2017.

[5] Liu, Ye and Nie, Liqiang and Liu, Li and Rosenblum, David S. From action to activity: sensor-based activity recognition,*Neurocomputing, Elsevier*, Volume 181, pp. 108-115, 2016.

[6] Hendry, Danica and Chai, Kevin and Campbell, Amity and Hopper, Luke and O'Sullivan, Peter and Straker, Leon, Development of a Human Activity Recognition System for Ballet Tasks. *Sports Medicine-Open,Springer*, Volume 6,Issue 1, pp. 10, 2020.

[7] Chua, Sook-Ling and Foo, Lee Kien and Juboor, Saed Sa'deh Suleiman. Towards real-time recognition of activities in smart homes, *International Journal of Advanced Intelligence Paradigms, Inderscience Publishers (IEL)*, Volume 15,Issue 2, pp. 146-164, 2020.

[8] Fan, Yin and Lu, Xiangju and Li, Dian and Liu, Yuanliu. Video-based emotion recognition using CNN-RNN and C3D hybrid networks,*Proceedings of the 18th ACM International Conference on Multimodal Interaction*, pp. 445-450, 2016.

[9] Tran, Du and Bourdev, Lubomir and Fergus, Rob and Torresani, Lorenzo and Paluri, Manohar. Learning spatiotemporal features with 3d convolutional networks. *Proceedings of the IEEE international conference on computer vision*, pp. 4489-4497, 2015.

On the Privacy Risks of Large-Scale Processing of Face Imprints

István Fábián and Gábor György Gulyás

Abstract: As technology advances, the number of applications relying on face recognition is on the rise. While facial recognition technologies have many benefits, it's important to use them in a responsible manner in order to avoid privacy risks. In this paper we analyze the privacy risks of the processing of face imprints generated by face recognition technologies. We characterize the risks of re-identification attacks against facial imprint databases in multiple scenarios regarding different attacker strength. Our findings show that even if a large number of subjects are surveilled and the attacker can only inefficiently benefit from using the embeddings, the risk of re-identification is still concerning.

Keywords: facial recognition, privacy, risk, identification, face embedding, deep metrics

Introduction

With the trend of technology getting cheaper and the advance of smart technologies, security and surveillance cameras were getting more and more wide spread recently. According to recent news, Chongqing, a single Chinese city alone exhibit 2.58 million surveillance cameras alone [5]. While this is not constrained to distant countries such as China, as London also has 627,707 of such cameras [5]. As the technology is cheap, these devices enable face recognition (hereafter also referred to as FR) technologies at scale. However, we can be certain that using FR widely will have a significant impact on the society as a whole, on the everyday life at companies, and on personal lives as well.

In this paper we consider risks posed by FR. In particular, we look at the case of the large-scale storing and processing of face imprints generated by FR technologies. This technology uses the photo or a video frame containing a person's face to extract an imprint from it. The imprint, or how the literature calls it, the embedding, describes the face based on its unique characteristics. Therefore it can be used for identification. When generated by deep learning techniques, the embedding is usually hard for a human to interpret, as usually it is a vector of real values. Length of the vector may differ, OpenCV [7] and DLIB [4] provide embeddings at the length of 128, while others provide longer, e.g., 512 float values [2].

Identification (i.e. the recognition) works by comparing multiple embedding vectors to each other and calculating similarity between them (e.g. via Euclidean distance). At the end, pairwise similarities of the embeddings determine whether the two faces are of the same person. It's presumed that the lower the distance, the higher the similarity, and the similarity of embeddings is proportional to the similarity of the faces. Usually if the distance is below a certain threshold, the embeddings are considered to belong to the same person.

The system setup is the following: cameras observe some areas (for example at a company, or in a public space) and extract facial embeddings of people passing by. Either the cameras themselves are capable of doing the extraction, or they transfer their footage to a capable server device that would do so. Depending on the use case (tracking, authentication, identification, etc.), either embeddings are stored in a database to be used later, or are compared to other embeddings that are already stored in the database.

The reason why storing this may be concerning is that embeddings are biometric data (they are based on features of the human body that the person cannot change) and unlike other biometric data, such as fingerprints, facial images can be easily captured without a person's knowledge and consent, and can be captured in large-scales [1]. Therefore, in this paper, we look at risks related to the processing of embeddings, more specifically we analyze the privacy risk of re-identification by using face imprints.

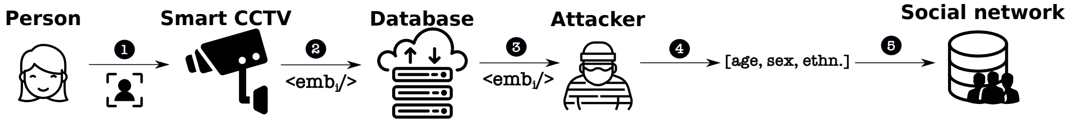


Figure 1: The considered attack when a malicious third party reconstructs demographic data from embeddings and re-identifies the embedding by looking up potential data subjects on social networking sites.

Attacker models

Unique pieces of information can be combined indirectly with other data sources in order to put back the names over de-identified data (i.e. where all directly identifying attributes are removed). We call such procedures re-identification attacks. Consider an example where a company publishes a database with information about its employees, de-identified by removing explicitly identifying fields (names, email, etc.) and replacing them with unique random IDs. While this database alone might be considered de-identified, an attacker may link records from this company-related dataset by using the demographic data to the corresponding records in a medical dataset.

In our work we consider re-identification attacks related to embeddings. There are several ways how an attacker can be successful at re-identification by using face embeddings:

- By matching embeddings, e.g. the attacker has a photo, extracts an embedding and looks for a match. As mentioned in the Introduction: if the distance between the two embeddings is below a threshold then the embeddings belong to the same person with some probability.
- If direct search of embeddings is not possible, the attacker could reconstruct the face from the embedding in the database [6], and run a visual search in a face database (e.g. photos on a social network).
- Assuming that embeddings may contain demographic data about the data subject, the attacker can try reconstructing such data from the embedding itself (for example using a machine learning model trained for this task) and looking that up in another database. As reconstruction of faces is possible, this should be possible too, however, the accuracy of such attacks are still questionable (we leave the exploration of this as future work).

In our work, we consider the third class of attacks. Latanya Sweeney showed in 2000 that the zip code, sexuality and date of birth combined together provide a unique identifier for 87% of the population based on US census data. [8]

Therefore, if an attacker accurately predicts such demographic data from embeddings, and knows further pieces of background information such as place of work, she may learn the identity of the anonymized person by looking him on social network sites (e.g. on LinkedIn).

Let us explain this attack as follows (see Figure 1). In the 1st step a camera takes the photo of a person and extracts a face embedding, and in the 2nd step this embedding is stored in a database. In the 3rd step the attacker gets the database of embeddings, where she predicts the related demographic data (age, sexuality, ethnicity) in the 4th step. The final 5th step is when the attacker uses this demographic data to re-identify the person on a social network site.

Risk level estimation

We demonstrate the previously discussed attack. We consider different scenarios, based on the number of people in the database and the accuracy of the algorithm that's trying to predict demographic data from the embeddings. To determine the feasibility and threat level of the attack, we ran simulations on the UCI Machine Learning Repository's Adult Dataset [3]. This dataset contains demographic information (including age, sex and race) for more than 30,000 records of people. Furthermore, these records are not of individuals, but of types of individuals, where the "finalweight" column describes the number of people represented by the given record. Our aim with the simulations is to examine what level of re-identification is theoretically possible for a database of 10, 50, 100 and 300 people randomly sampled from [3], if the accuracy for predicting all age, race and sex vary (60%, 75% and 90%). We assumed a machine learning model that can predict age in 10 year intervals.

As per our attacker model, the database sizes were chosen to be reasonable assumptions for the number of employees of a small or medium sized company. To construct the smaller original databases of size 10, 50, 100 and 300 for the simulation, we randomly sampled the required number of entries from [3] using the values in the *'fnlwgt'* column as weights, which indicate the number of people believed to be represented by a given entry. After sampling, according to the prediction accuracy percentage (90%, 75%, or 60%) the corresponding number of rows were left untouched, while the remaining rows' age attributes were randomly permuted to simulate the inaccurate predictions. This random permutation was then repeated with the same prediction accuracy percentage for the other two attributes, too (sex and ethnicity). This way we ended up with three predicted databases for each smaller original database, where all three attributes were predicted with either 90%, 75%, or 60% accuracy. As the last step, for each database we counted what percentage of data subjects were correctly predicted to fall in an equivalence class of size 1, 2-5, 6-10, 11-20 and 20+. We then repeated this procedure 100 times and averaged out the results.

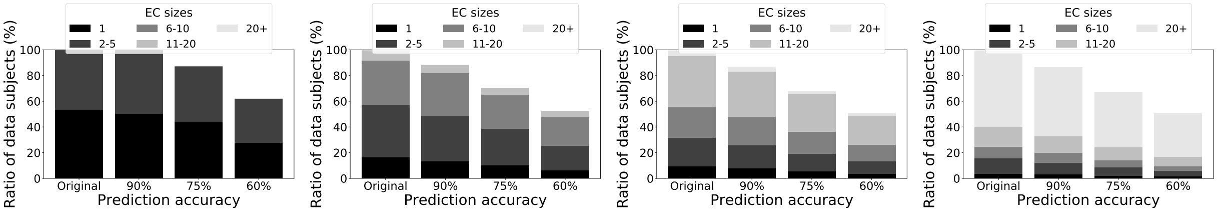
Figures 2a-d show our findings. We can observe that there are many records in unique or small equivalence classes both in smaller and larger databases, which pose high privacy risks. The attacker is the most successful at re-identification in the case of the database of 10 with the highest 90% prediction accuracy, when 50.1% of people fall in a unique equivalence class, and all the others fall in an equivalence class of size 2 to 5. If the accuracy is decreased to 60%, still 27.7% falls in a unique equivalence class, and 33.9% falls in an equivalence class of size 2 to 5 (see Figure 2a). Regarding the largest database of 300, 3.75% of people are in a unique equivalence class, and 11.79% are in an equivalence class of size 2 to 5. Even in the worst case scenario for the attacker, which is 60% accuracy for a database of 300, the rate of people in unique equivalence classes doesn't fall below 1.38%, and nor does the rate of people in an equivalence class of size 2 to 5 fall below 4.64% (see Figure 2d). Thus, it's worth noting that while a decrease in accuracy results in a decrease in re-identification probability, risks are not diminished drastically. As a result, due to the privacy risk presented, the actual achievable prediction accuracy must be examined in future work.

In summary, as expected, the smaller the database size, the higher the re-identification risk is, because smaller sized databases have a higher chance of being reconstructed in such a way that people are correctly mapped to an equivalence class of size 1 to 5. Indeed, the higher the prediction accuracy, the higher the re-identification risk is, because the higher percentage of people are predicted to be in the correct equivalence class.

Conclusion

In this paper we have presented the potential privacy risks in relation to the processing of face embeddings. We have discussed why face embeddings must be considered sensitive biometric data and we looked at various attacker models that could pose a threat to data subjects' privacy via re-identification.

In particular, we analyzed the risks of re-identification by reverse-engineering demographic data (age, sexuality, ethnicity) from embeddings stored in a database. We found that the smaller the database and the higher the accuracy of prediction, the higher the re-identification risks are.



(a) $D = 10$ (b) $D = 50$ (c) $D = 100$ (d) $D = 300$

Figure 2: The ratio of equivalence classes (EC) in the predicted database (D) for various database sizes and prediction accuracies.

In the case of a 10 person database and 90% accuracy, 50.1% of people belonged to a unique equivalence class, while this number decreased to 27.7% at 60% prediction accuracy. The risks are also not negligible even for larger databases, because for a database of 300 we showed that at 90% accuracy 3.75% of people are in a unique equivalence class, and 11.79% are in an equivalence class of size 2 to 5.

In the future, we aim to train a machine learning model to find out the actual achievable prediction levels for these demographic data. We are also working on ways to modify the embeddings in such a way to make these predictions harder, without compromising the usability of the embeddings.

Acknowledgments

The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Infocommunications). Project no. FIEK_16-1-2016-0007 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Centre for Higher Education and Industrial Cooperation - Research infrastructure development (FIEK_16) funding scheme.

Icons made by Pixel perfect, fjstudio, Freepik, Pause08, surang from www.flaticon.com.

References

[1] Daniel Le Métayer Inria Claude Castelluccia. Impact analysis of facial recognition: Towards a rigorous methodology, 2020.

[2] Jiankang Deng, Jia Guo, Zhou Yuxiang, Jinke Yu, Irene Kotsia, and Stefanos Zafeiriou. Retinaface: Single-stage dense face localisation in the wild. In *arxiv*, 2019.

[3] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[5] Matthew Keegan. Big brother is watching: Chinese city with 2.6m cameras is world's most heavily surveilled, 2019.

[6] Guangcan Mai, Kai Cao, Pong C. Yuen, and Anil K. Jain. On the reconstruction of face images from deep face templates, May 2019.

[7] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015.

[8] Latanya Sweeney. Simple demographics often identify people uniquely. 2000.

Towards Version Controlling in RefactorErl

Jenifer Tabita Ciuciu-Kiss, István Bozó and Melinda Tóth

Abstract: Static source code analysers are operating on an intermediate representation of the source code that is usually a tree or a graph. Those representations need to be updated according to the different versions of the source code. However, the developers might be interested in the changes or might need information about previous versions, therefore, keeping different versions of the source code analysed by the tools are required. RefactorErl is an open-source static analysis and transformation tool for Erlang that uses a graph representation to store and manipulate the source code. The aim of our research was to create an extension of the RefactorErl's Semantic Program Graph that is able to store different versions of the source code in a single graph. The new method resulted in 30% memory footprint decrease compared to the available workaround solutions.

Keywords: Erlang, RefactorErl, graph version control, optimisation

Introduction

Static source code analyser tools are heavily used in software development and maintenance processes. RefactorErl [5, 3] is an open-source static source code analyzer and transformer tool for Erlang [2]. During the initial analysis, the tool builds a Semantic Program Graph (SPG) [4]. In order to make the source code analysis result available for further use, we need to save the graph. The SPG is stored in a database. If the source code is changed, the tool updates the graph, so in the new graph, only the new information is available. Any kind of backups about different versions are really expensive. This is the reason why we need to save the backups in a much more effective way. An adequate solution to this problem is to save the differences in the same graph. In this way, we do not need to make a whole graph for every small change. This solution leads to an average 30% of memory save, but it always depends on the analyzed source code.

The SPG is a three-layered rooted, directed, labelled graph structure that contains the lexical, syntactic and semantic information about the source code. The different kinds of data are stored in the nodes of the graph that are linked through tagged edges. These edges contain structural information about the nodes. The graph has a specified structure. It starts with a root node, which is followed by a node containing the information about the file (the name, absolute path, etc.). They are linked with an edge tagged with the file. The file node is linked to its forms: the function definitions and attributes. The function definition forms are linked to its clauses, and so on. In addition, there are module and function semantic nodes to represent the semantic information as well. Based on the structure of the graph we were able to define an algorithm to find the differences between two versions of the source code by traversing the representing syntax trees in parallel.

In Section 2 we demonstrate this algorithm and present the graph storing two versions of the source code. In Section 3 we present some related work. Finally, Section 4 concludes the paper.

Detecting and storing the source code change

The algorithm gets two SPG subgraphs and it does a breadth-first search in parallel on the two subgraphs to find the differences between them. When a difference is found we save it and separate the deletion and the insertions.Then we go through on the list of found differences.

We might found a similar structure in the graph, where the attribute values or the types of the nodes are different. These cases considered as an update in the source code. When we found a node that does not have a version in the other graph with the different value we consider that is was either a deletion or insertion depending on where was it found: in the old or the new graph. So we insert the updated part of the graph first (as we will describe later in this section), and we delete these updates from the saved differences. After this, we know it for sure that the left part is an insertion if it appears only in the new graph and it is a deletion if it can be found only in the old graph. Since we saved them separately, the only thing we do now is to insert them using the algorithms mentioned below.

After finding and identifying the different types of changes we need to represent them in the SPG. The basic idea is that we introduce two different types of tags for the edges. Updates, deletions and insertions are handled in a similar way. Thanks to RefactorErl, we can easily and efficiently insert new nodes and edges. In the following, we will demonstrate the algorithm for each case.

Figure 1. presents two simple Erlang functions and Figure 3. shows the corresponding graph representation. Although the latter figure shows only the syntactic information, it is already quite big, nonetheless, the analyses are fast.

```
foo(0) -> zero.
op(A, B) -> A * B.
```

Figure 1: Example Erlang function

Update

We talk about an update when we have had something in a version, and we changed something in it, without inserting a new part or deleting a whole part of the source code.

Let us consider Figure 2. as a modified version of our original example. We can see that we have a small update between Figure 1. and Figure 2., which is the change of the symbol from $*$ to $+$ in the body. In this specific case, we are going to find that difference. We insert the whole subgraph under the found difference with a versioned edge (Figure 3.). An edged is versioned when it has the same name as before except it has a v at the beginning of its name. In the resulted graph under the difference node, we can find the content of both versions in the different subgraphs.

```
op(A, B) -> A + B.
bar(1) -> one.
```

Figure 2: Modified source code

In this way, we can easily find the versioned part in the graph and we can keep all the previous features of the tool. An obvious question may arise, whether it is possible to store more than two versions in this way. Yes, it is possible, but we need to make it customisable by the users.

Deletion

A deletion is when we have had something in the source code in a previous version, and we deleted it in a version after that.

The handling of this difference is very similar to the algorithm from the previous subsection. Let us consider the modified example in Figure 2. again. We can see that the first function

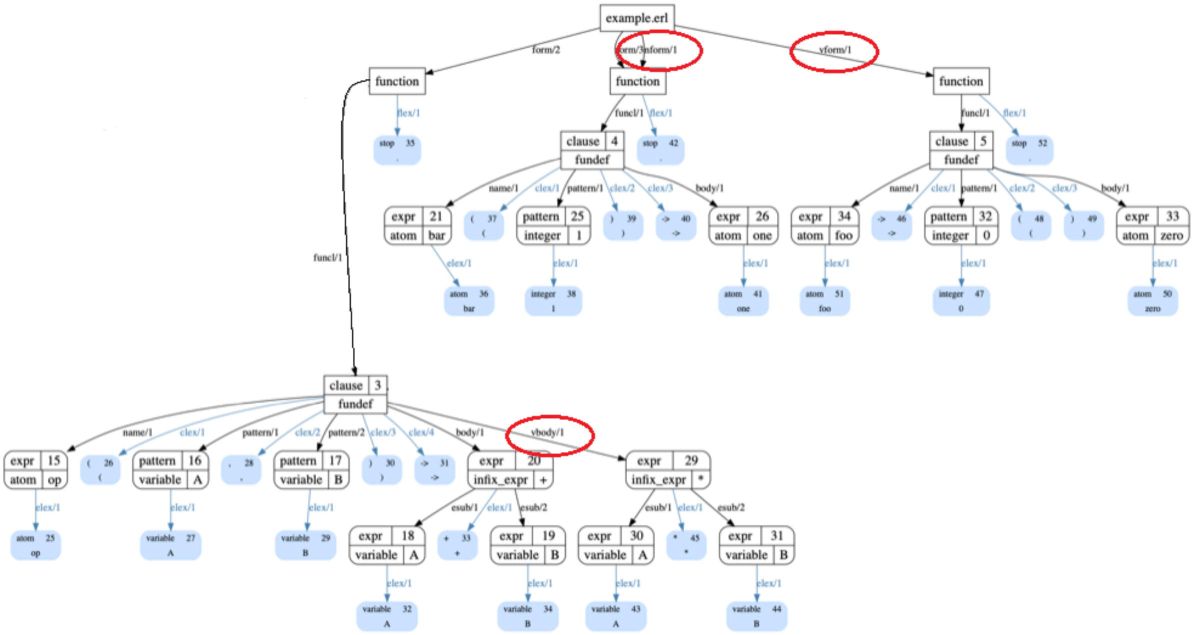


Figure 3: The resulted SPG

(`foo(0) -> zero.`) has been deleted. To handle this we insert the deleted subgraph under its parent node (the original function node) with a versioned edge.

Insertion

An insertion is, when we insert a totally new part in the source code, what did not exist in the previous version.

As an obvious example we can insert `bar(1) -> one.` as a new function into our module (Figure 2.). For that, we insert the difference in the same way but for the tag of the edge, we insert an n at the beginning instead of the v. When we do that, we keep the not versioned edge as well, because it does not bother us and it is needed to have a syntactically correct graph. Now we can easily find the insertions since it had a different concept than the deletions and updates.

Related work

In computer science, a finger tree is a purely functional data structure that can be used to implement other functional data structures. A finger tree gives amortized constant time access to the "fingers" (leaves) of the tree, which is where data is stored, and concatenation and splitting logarithmic time in the size of the smaller piece. It also stores in each internal node the result of applying some associative operation to its descendants. This "summary" data stored in the internal nodes can be used to provide the functionality of data structures other than trees [6]. The SPG (Semantic Program Graph) structure does not have the "summary" data property, thus the finger tree change detection mechanism cannot be adopted easily for our use case.

A component of software configuration management, version control, is the management of changes to documents, computer programs, large web sites, and other collections of information.

Changes are usually identified by a number or letter code, termed the "revision". For example, an initial set of files is "revision 1". Today, the most capable (as well as complex) revision control systems are those used in software development, where a team of people may concurrently make changes to the same files [1]. In RefactorErl the version control is very important because the analyzed graphs are huge. We would like to extend our algorithm to handle multiple revisions.

Conclusion

The proof of concept implementation of the algorithm is integrated within the RefactorErl environment. In general, it has an overhead on the run-time, but it has a smaller memory footprint. The previous graphs for previous versions do not need to be saved, because the annotated new graph is storing all the required information about the different versions. The algorithm can be optimized for different resources and different projects. We are investigating the possible usages of the versioned graph now.

Without this algorithm, the version control could be done with another software like git but it is not as efficient and does not assure any features that RefactorErl does. Analysis of different versions of the code with RefactorErl by storing the whole graph for all states is also possible, but it is slow, and takes a lot of memory.

Acknowledgements

The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002), and by the project no. ED_18-1-2019-0030 (Application-specific highly reliable IT solutions) has been implemented with the support provided by the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme funding scheme.

References

[1] Eric Sink: Version Control by Example (1st. ed.). PYOW Sports Marketing. 2011.

[2] Joe Armstrong: Programming Erlang. The Pragmatic Bookshelf. 546 pages. 2013.

[3] RefactorErl: Static source code analyser and refactoring tool for Erlang. https://plc.inf.elte.hu/erlang

[4] Zoltán Horváth and László Lövei and Tamás Kozsik and Róbert Kitlei and Anikó Nagyné Víg and Tamás Nagy and Melinda Tóth and Roland Király: Modeling Semantic Knowledge in Erlang for Refactoring. Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques, KEPT 2009, Cluj-Napoca, Romania, Studia Universitatis Babes-Bolyai, Series Informatica, vol. 54(2009), Sp. Issue

[5] Bozó, I. and Horpácsi, D. and Horváth, Z. and Kitlei, R. and Köszegi, J. and Tejfel, M. and Tóth, M: RefactorErl - Source Code Analysis and Refactoring in Erlang, In Proceedings of the 12th Symposium on Programming Languages and Software Tools, ISBN 978-9949-23-178-2, pages 138–148, Tallin, Estonia, October 2011

[6] Ralf Hinze and Ross Paterson: Finger Trees: A Simple General-purpose Data Structure, Journal of Functional Programming 16(2):197–217, 2006.

Using the Fisher Vector Approach for Cold Identification

José Vicente Egas-López and Gábor Gosztolya

Abstract: In this paper, we present a computational paralinguistic method for assessing whether a person has an upper respiratory tract infection (i.e. cold) by using their speech. Having a system that can accurately assess a cold can be helpful in the sense of being able to predict its propagation. For this purpose, we utilize Mel-frequency cepstral coefficients (MFCC) as audio-signal representations, extracted from the utterances, which allowed us fitting a generative Gaussian Mixture Model (GMM) that serves to produce an encoding based on the Fisher Vector (FV) approach. The classification is done by a linear kernel SVM (Support Vector Machines), which jointly learns from the training and development set via a Stratified Group K-fold Cross Validation. Owing to the high imbalance of classes on the training dataset, we opt for under-sampling the majority class, that is, reducing the number of samples to those of the minority class. We find that applying Power Normalization (PN) and Principal Component Analysis (PCA) on the Fisher vector features is an effective strategy as a score of 67.81% of Unweighted Average Recall (UAR) on the test set is achieved.

Keywords: computational paralinguistics, speech processing, machine learning, fisher vector

Introduction

Upper respiratory tract infection (URTI) is an infectious process for any of the components of the upper airway; e.g., the common cold, a sinus infection, amongst others. Being able to automatically assess whether a subject has a cold can be relevant when one wants, for example, to prevent the spread of a cold by predicting its patterns of propagation. In this study, we make use of the Upper Respiratory Tract Infection Corpus (URTIC) [4] that contains 630 recordings, which will be described in Section 2. Thus, the task requires healthy and cold speech to be classified from the utterances.

Frame-level features (Mel-frequency cepstral coefficients), extracted from the utterances, are utilized to fit a generative GMM (Gaussian Mixture Model). Next, the computation of low-level patch descriptors together with their deviations from the GMM gives us an encoding (features) called Fisher Vector. FV features are learned using a SVM classifier, where the prediction is 1 (cold) or 0 (healthy). In order to search for the best complexity value (C) of the SVM, Stratified Group k-fold Cross Validation (CV) was applied on the training and development sets. UAR scoring was used to measure the performance of the model. To the best of our knowledge, this is the first study that focuses on making use of a FV representation in order to detect a cold.

Data

The entire dataset consists of 382 male speakers, 248 female speakers, with a mean age of 29.5 years; and a sampling rate of 16kHz. The following tasks (the number varied for each speaker) were performed by the speakers: reading short stories, producing voice commands, and narrating spontaneous speech. The recordings were split into 28652 chunks of 3 to 10 seconds in length. Specifically, the division of the chunks was carried out in a speaker-independent fashion, each partition (i.e. train, development, and test) having 210 speakers. The training and development sets are both comprised by 37 subjects having a cold and 173 subjects not having a cold. See more details in [4].

Methodology

Feature extraction

The features we employed were the well-known MFCCs. With a dimension of 13, plus their first and second order derivatives, with frame-length and frame-shift of 25 ms and 10 ms, respectively.

Fisher Vector (FV)

The FV approach is an image representation that pools local image descriptors [1]. It was originally intended for image classification but here we exploit its ability to generate a complete representation of the samples which are later characterized by their deviation from a generative GMM. The samples can be thought as of local patch descriptors of an image; in our case, they are the frame-level features of an audio signal. FV is an improved version of the general case called the Fisher Kernel (FK) [5], which measures the similarity of two objects from a parametric generative model of the data. FV basically assigns a local descriptor to elements in a visual dictionary, got using generative GMM. This approach stores visual word occurrences and takes into account the difference between dictionary elements and pooled local features, and stores their statistics as well.

Let $X = \{X_t, t = 1 \dots T\}$ be the set of D-dimensional local SIFT descriptors extracted from an image and let us assume that the samples are independent. Then we have the following formula:

$$\mathscr{G}_\lambda^X = \sum_{t=1}^{T} L_\lambda \bigtriangledown_\lambda \log \upsilon_\lambda(X_t), \tag{1}$$

where $\lambda = [\lambda_1, \dots, \lambda_M]' \in R^M$ stands for the parameter vector υ_λ [1]. The assumption of independence permits the FV to become a sum of normalized gradients statistics $L_\lambda \bigtriangledown_\lambda \log \upsilon_\lambda(x_t)$ calculated for each SIFT descriptor:

$$X_t \rightarrow \varphi_{FK}(X_t) = L_\lambda \bigtriangledown_\lambda \log \upsilon_\lambda(X_t), \tag{2}$$

which describes an operation that is a higher dimensional space embedding of the local descriptors X_t.

The FV representation, regardless of the number of local features (i.e. SIFT), or in our case, frame-level features (MFCCs), extracts a *fixed-sized* feature representation from each image (i.e. from each MFCC representation). Here, we use FV features to encode MFCC features extracted from audio-signals of HC and PD subjects. FV allows us to give a complete representation of the sample set by encoding the count of occurrences and higher-order statistics associated with its distribution.

Classification

SVM is the classification algorithm that was used to assess the recordings (this algorithm can achieve a good performance when used with FV [1, 6]), where the prediction is 1 (*cold*) or 0 (*healthy*). In order to search for the best complexity value (C) of the SVM, Stratified Group k-fold Cross Validation (CV) was applied over the training and development sets. This type of CV allowed us avoiding the same speaker from being present in more than one specific fold, while simultaneously preserving the percentage of samples for each class within each fold. UAR is the proper way to measure the performance of our model, principally because it is commonly used when there is the need to handle class imbalance situations. We employed the libSVM implementation [8] with a linear kernel and, as in our previous studies [9, 10, 11], the C complexity parameter was set in the range 10^{-5}, . . ., 10^{1}.

Table 1: UAR scores obtained when SVM classifies the data using Fisher vectors.

Features	GMM size	Performance (%) Cross-Val	Test
ComParE (baseline)	-	64.54%	68.16%
Fisher Vectors	64	63.98%	66.12%
Fisher Vectors + PCA	64	64.72%	67.65%
Fisher Vectors + PN + PCA	64	64.92%	67.81%
ComParE + Fisher Vectors (+PN+PCA)	-	63.01%	70.17%

Experiments and results

The datasets utilized have a high class imbalance, this can diminish the performance of the SVM classifier. Namely, the training dataset consists of 9505 utterances, where 8535 (89.8%) are labeled as *healthy* and the rest, 970 (10.2%), are labeled as *cold*. To overcome this, we used a random undersampling technique, which reduces the number of samples associated to all classes to that of the minority class, i.e. *cold*. We relied on imbalanced-learn [7] which is a Python package offering several resampling methods used in datasets that have a between-class imbalance. Due to the large number of dimensions, before fitting the SVM classifier, the features (Fisher vectors) were normalized ($l2$) and later we reduced their dimensions via Principal Component Analysis [2] with a variance of 0.95. Chatfield et al. demonstrate that applying PCA before classification enhances the discrimination task with FV and reduces the memory consumption as well [13]. The Power Normalization (PN) was found to be helpful for FVs representations [1]. Here, we applied PN before normalizing the features.

The GMM used in our experiments to compute the FVs was set to operate with varying number of components: G_c ranged from 2 to 128. Hence, the construction of the FV representations was made with the help of a Python-wrapped version of the VLFeat library [12]. Table 1 shows the results obtained when using Fisher vectors with their complete number of features as a function of their reduced number of features. As can be seen, when the classifier learned the *raw* Fisher features it achieved a UAR score of 63.98% within the Stratified Group k-fold Cross Validation (CV) using a complexity value of 0.1. On the test set the performance was higher (64.72%). PCA proved to be useful in our study by providing the higher UAR scores of 66.12% and 64.65% in CV and test, correspondingly. On the other hand, we encountered that applying PN was effective as the UAR score increased to 67.81%. Next, we used the ComParE [4] feature set combined with the (power-normalized and reduced) Fisher vectors, that is, we averaged their posteriors. Results indicate an increase to 70.17% of UAR score on test set.

Conclusions

Here, we presented the Fisher Vector approach as a method that was able to achieve reasonable results on cold speech assessment. We found that, in contrast to studies done by other authors using the same dataset [4, 14, 15], our pipeline is much simpler and the performance is good and competitive. With the SVM classifier we managed to achieve a UAR score of 66.12% on the test set. The scores were later improved by applying PN and a dimension reduction on the Fisher vector features, i.e., PCA with a variance of 0.95. PN and dimensionality reduction could give a better UAR score (67.81%) on test set; such results are higher compared to those got using the Bag-of-Audio-Words approach (67.3%) described in [4]. We therefore say that PCA with the SVM (linear kernel) allowed us to perform a better classification of the actual data while taking care of the memory consumption.

References

[1] Sánchez, J., Perronnin, F., Mensink, T., and Verbeek, J. (2013). Image classification with the fisher vector: Theory and practice. International journal of Computer Vision, 105(3),

222-245.

[2] Wold, S., Esbensen, K., and Geladi, P. (1987). Principal component analysis. Chemometrics & intelligent laboratory systems, 2(1-3), 37-52.

[3] Hearst, M. A., Dumais, S. T., Osuna, E., Platt, J., and Scholkopf, B. (1998). Support vector machines. IEEE Intelligent Systems and their applications, 13(4), 18-28.

[4] Schuller, B., Steidl, S., Batliner, A., Bergelson, E., Krajewski, J., Janott, C., ... & Warlaumont, A. S. (2017). The interspeech 2017 computational paralinguistics challenge: Addressee, cold & snoring. In Computational Paralinguistics Challenge (ComParE), Interspeech 2017 (pp. 3442-3446).

[5] T. S. Jaakkola and D. Haussler, Exploiting generative models in discriminative classifiers, in Proceedings of NIPS, Denver, CO,USA, 1998, pp. 487-493.

[6] D. C. Smith and K. A. Kornelson, A comparison of Fisher vectorsand Gaussian supervectors for document versus non-document image classification, in Applications of Digital Image Processing XXXVI, vol. 8856.International Society for Optics and Photonics, 2013, p. 88560N.

[7] Lemaitre, G., Nogueira, F., & Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. The Journal of Machine Learning Research, 18(1), 559-563.

[8] C.C. Chang and C.J. Lin, LIBSVM: A library for supportvector machines, ACM Transactions on Intelligent Systems and Technology, vol. 2, pp. 1-27, 2011

[9] López, E., Vicente, J., Orozco-Arroyave, J. R., & Gosztolya, G. (2019). Assessing Parkinson's Disease From Speech Using Fisher Vectors.

[10] G. Gosztolya, Conflict intensity estimation from speech using greedy forward-backward feature selection, in Proceedings of Interspeech, Dresden, Germany, Sep 2015, pp. 1339-1343.

[11] G. Gosztolya, T. Grósz, R. Busa-Fekete, and L. Tóth, Determining native language and deception using phonetic features and classifier combination, in Proceedings of Interspeech, San Francisco, CA, USA, Sep 2016, pp. 2418-2422.

[12] A. Vedaldi and B. Fulkerson, VLFeat: An open and portable library of computer vision algorithms, in Proceedings of the 18th ACM international conference on Multimedia. ACM, 2010, pp.1469-1472.

[13] Chatfield, K., Lempitsky, V. S., Vedaldi, A., & Zisserman, A. (2011, September). The devil is in the details: an evaluation of recent feature encoding methods. In BMVC (Vol. 2, No. 4, p. 8).

[14] Huckvale, M. A., & Beke, A. (2017). It sounds like you have a cold! Testing voice features for the Interspeech 2017 Computational Paralinguistics Cold Challenge. International Speech Communication Association (ISCA).

[15] Cai, D., Ni, Z., Liu, W., Cai, W., Li, G., Li, M., ... & Cai, W. (2017). End-to-End Deep Learning Framework for Speech Paralinguistics Detection Based on Perception Aware Spectrum. In INTERSPEECH (pp. 3452-3456).

A Novel JWT Revocation Algorithm

László Viktor Jánoky, János Levendevoszky and Péter Ekler

Abstract: JSON Web Tokens (JWT)[1] provide a scalable, distributed way of user access control for modern web-based systems. The main advantage of the scheme, is that the tokens are valid by themselves - through the use of digital signing - also imply its greatest weakness. Once issued, there is no trivial way to revoke a JWT token. In our work, we present a novel approach for this revocation problem, overcoming some of the problems of currently used solutions.

Keywords: JWT, JSON Web Tokens, User access control

Introduction

In our previous paper in the field, titled as An analysis on the revoking mechanisms for JSON Web Tokens[2], we examined the currently used solutions of token revocation and laid out the mathematical foundations and introdued a novel approach. In this paper, we further elaborate on the new solution and provide general guidelines how to use it in a real-world application.

This paper is structured as follows: (I) in this first section, we quickly recap the currently used revocation schemes and their main characteristics, (II) in the second section, in which we provide a detailed description of our new approach and give some recommendations for its real-world use, (III) the third section deals with the evaluation of performance in different cases, comparing our solution with existing one, and finally (IV) the fourth and final section wraps the discussion by providing an overview of the work done.

A JWT token used to determine access for a protected resource is called an access token. The token is usually digitally signed, or otherwise cryptographically secured [3], in both cases we simply refer to the signing key or the public key as secret. In most scenarios, the access tokens are issued along with a second, more traditional; server-side token called a refresh token. This second token makes it possible for the client to acquire a new access token in the future.

When a client logs out from the system, the refresh token is destroyed, and existing JWT tokens are revoked. There are three main methods of this revocation that are currently used[4].

Short-lived tokens: Each generated JWT token has a finite, usually very short lifespan. In this scheme, a token is never directly revoked, but the means of acquiring new tokens are made unavailable, hence when the short lifespan runs out, no further access is possible to the system.

Blacklist: In case of a blacklist, revoked tokens are placed in a shared location (typically a database), where each consuming service can check for invalidated tokens. The big downside of this approach is that it requires data access for each request served - even for ones with valid tokens thus, the validity of the token can no longer be determined in itself.

Secret change: A rarely used solution for invalidation, is the changing of the cryptographic secret used to issue and check the validity of tokens. Changing this secret leads to all tokens being revoked, but still logged in users can apply for new ones using their refresh token.

The revocation algorithm

Our novel revocation strategy is based on the extension of the third option, which is changing the secret. In this section, we give a quick overview of this approach.

Basic principle

When a JWT secret is changed, all the tokens issued with it become invalid. In practice, this means that if a user logs out, every other user in the system must request a new token.

This effect can be controlled by arranging the users in groups (for example, by hashing their usernames) and assigning a different secret for each group. If a token is revoked in a group, only tokens signed with the group secret will be revoked, instead of all the tokens.

As log-outs are typically infrequent events, one can use statistical methods (such as described in our previous paper), to calculate an optional group size, which minimizes the number of unnecessary revocations, while maintaining a manageable number of secrets.

Propagating secret change events

With this method, revocation is instantaneous, and the basic premise of JWT tokens remain intact, that the tokens are valid by themselves.

This solution requires the existence of a channel for propagating the information about the change of the JWT secret. The channel must be available between the token issuer and each service consuming the tokens.

For cases when such a channel is unavailable, we have come up with an alternate solution. In this approach the JWT Secret is generated as a rolling code by a pseudo random number generator[6], each service keeping track of the currently active value. When a token is revoked and the group's secret is changed, the new tokens are issued with the new secret. When a service, still using the old code, receives a token signed with the new secret (a next value from the rolling code), it will also update the secret accordingly.

This method provides eventual consistency for the system in the long run, without the need for a dedicated channel for JWT secret change event prorogation. As a trade-off, the instantaneous revocation is lost, a token is only revoked after the code is rolled (another token, using the new secret is received).

Cost model of running the algorithm

In our previous work on the topic, we prove that any system can be parametrized in a way, for our solution to be better in terms of performance than the previous solutions.

In this paper, we further examine the performance and the costs of our solution and provide a mathematical model to aid in system design and capacity planning.

To accurately model the performance impact of different solutions, a common framework must be set. As the basis of this framework, we determined a set of basic operations, which make up each approach. The costs of these operations can be parametrized, which can be based on estimations or measurements. The following main costs are identified:

- C_i **(Issue cost):** the cost of issuing a new token.
- C_v **(Validation cost):** the cost of checking the validity of a token.
- C_c **(Communication cost):** the cost of system modules communicating with each other.
- C_d **(Data access cost):** the cost of accessing data stored in a persistent storage.

In order to predict the performance of a system, it is not enough to know the cost of these atomic operations, one must also calculate how many times they will occur. The performance cost of a system comes from the clients consuming its service. By characterizing the client sessions, one can come up with predictions for their impact on the system[5]. In order to calculate the former, the following properties must be known (by measurements or estimation).

- N: the number of clients in the system.
- $f_i(t)$: probability distribution of client session lengths.
- r: average number of protected resource access / client / second.

As we have demonstrated in our earlier work on the topic, from these metrics, one can calculate the average time between token revocations in a group of clients. This value is denoted as T_{rvk}.

Knowing both the cost and occurrences of typical operations in the system, one can come up with a cost function, which describes the average cost of operation.

Performance evaluation

To predict the performance characteristics of a system, one must first determine the costs associated with the operations defined in the previous section. The second step is to measure the characteristics of the client population. The third step is to choose a revocation algorithm.

These three steps together determine the overall performance metrics. Each revocation algorithm has a unique cost function, which maps the client pool behavior to system operations, which in turn are used to calculate the overall performance cost of a given solution. Some revocation algorithms have variable parameters, which can be optimized trough the usual means.

Short-lived tokens

The short-lived approach has one parameter, T_{life}, which denotes the lifetime of a token. As for maximizing the performance of this approach, this T_{life} should be chosen as the longest tolerable time for token revocation after logout.

The overall cost function consists of two parts, the cost of validating the tokens of the incoming requests, and the cost of issuing new tokens to replace the expiring ones.

$$C = (N * r * C_v) + (N * \frac{1}{T_{life}} * C_i)$$

Blacklist

In the case of a blacklist, the main cost factor comes from the necessity to check whether a token is on the blacklist for each request. This extra checking makes this approach the worst in terms of scaling in the case of an increasing number of requests.

After accounting for this factor, there are no other costs associated with this method. Therefore the cost function can be defined as the following.

$$C = N * r * C_v * C_d$$

Secret change

In case of a secret change, the baseload of authorizing incoming request is still present, but it is accompanied by the load of new token generation for each client in case of each revocation.

$$C = (N * r * C_v) + (N * \frac{1}{T_{rvk}} * C_i)$$

Notice that the formula is very similar to the short-lived cost function. This is not a coincidence; in both cases, the number of token revocations depends heavily on the average lifespan of a token. In the first case its purely determined by the age of token itself, while in the second case, client logout events trigger it.

The cost evaluation of our method

As our method builds on the secret change event, the cost function is similar too. The main difference being, the introduction of parameter K, which denotes the number of groups the clients are separated to. Because of this separation, the T_{rvk} calculated for the whole client population size must be recalculated to the number of $\frac{N}{K}$ clients. This value is denoted by T_{rvk}. As K increases, T_{rvk} monotonously increasingly approaches the mean value of $f_i(t)$.

$$C = (N * r * C_v) + (K * \frac{1}{T'_{rvk}})(\frac{N}{K} * C_i + C_c)$$

Overview

In this paper, we described the main approaches of JWT revocation and introduced our novel solution. We provided a toolset for characterizing different systems based on the cost of common operations when dealing with JWT tokens. We outlined the necessary parameters to measure in a client population of such a system. We determined the cost functions for each solution, based on the previously described characteristics and client behaviour.

With the mathematical framework at hand, one can find the optimal revocation solution for any system, by choosing the minimal cost function. In cases where the function has additional variable parameters, traditional approaches like linear search can be used to find the optimal solutions.

Ultimately, we hope that our work will aid capacity planning and system design of distributed systems, as the JWT based solutions have the highest potential in this area.

Acknowledgements

This work was supported by the BME-Artificial Intelligence FIKP grant of EMMI (BME FIKP-MI/SC) and by the János Bolyai Research Fellowship of the Hungarian Academy of Sciences.

References

[1] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," RFC Editor, RFC 7519, May 2015.

[2] L. V. Jánoky, J. Levendovszky, and P. Ekler, "An analysis on the revoking mechanisms for JSON Web Tokens," International Journal of Distributed Sensor Networks, vol. 14, no. 9, p. 1550147718801535, Sep. 2018, doi: 10.1177/1550147718801535.

[3] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Signature (JWS)," RFC Editor, RFC 7515, May 2015.

[4] dWTV, "Learn how to revoke JSON Web Tokens," developerWorks TV, 2017. Available: https://developer.ibm.com/tv/learn-how-to-revoke-json-web-tokens/.

[5] M. Arlitt, "Characterizing web user sessions," ACM SIGMETRICS Performance Evaluation Review, vol. 28, no. 2, pp. 50–63, 2000.

[6] M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudorandom bits," SIAM journal on Computing, vol. 13, no. 4, pp. 850–864, 1984.

Towards Secure Remote Firmware Update on Embedded IoT Devices

Márton Juhász, Dorottya Papp and Levente Buttyán

Abstract: An important security problem in IoT systems is the integrity protection of software, including the firmware and the operating system, running on embedded IoT devices. Digitally signed code and verified boot only partially solve this problem, because those mechanisms do not address the issue of run-time attacks that exploit software vulnerabilities. For this issue, the only known solution today is to fix the discovered vulnerabilities and update embedded devices with the fixed software. Such an update should be performed remotely in a secure and reliable way, as otherwise the update mechanism itself can be exploited to install compromised software on devices at large scale. In this work, we propose a system and related procedures for remotely updating the firmware and the operating system of embedded IoT devices securely and reliably.

Keywords: Embedded Systems, Internet of Things, Security

Introduction

IoT systems are built from network connected embedded devices, and their security heavily rely on the security of those embedded devices. One of the most important security aspects in this context is the integrity of the software running on embedded devices. The reason is that unauthorized modification of software can result in arbitrary behavior of those devices, and as a consequence, loss of trust in the entire IoT system built upon them. In particular, protecting low level software, such as the operating system (OS) and the firmware, is important, because typically these components are responsible for implementing many security controls and they provide trusted services (e.g., in the form of system calls) to higher layer software.

Digitally signing software components, including the firmware and the OS kernel, and important data, such as configuration files, combined with some hardware based *root-of-trust* and a secure boot process, which ensures that software components are loaded and executed only if their signature is valid, can help protecting the integrity of software, but does not entirely solve the problem. In particular, signed code and verified boot ensure that the device runs intact code right after a reset, but software can also be compromised at run-time by exploiting design and implementation level vulnerabilities in it. For instance, an attacker may be able to execute arbitrary injected code on a device by exploiting software bugs, such as not checking the amount of data copied into a limited size buffer or using dangling pointers, leading to memory corruption [1].

When software vulnerabilities are discovered, they need to be fixed, and embedded devices need to be updated with the fixed software. This applies to the OS and the firmware too. In addition, due to the potentially large number of embedded devices used in IoT applications and their often special operating environment, it is preferable that the update process can be carried out remotely, without the need to physically approach each and every device. Remote firmware and OS update is sometimes also called *over-the-air* (OTA) update, because the update may be downloaded by the devices over wireless communication links.

Security of the remote update process itself is of paramount importance [2], as we would like to avoid that attackers exploit an insecure update mechanism to install a compromised OS or firmware remotely at large scale. Potentially, such compromised updates may prevent any further legitimate update, leaving the control of all compromised devices in the hand of the attacker. Recovering from such a situation would require manual update of every device, which would be time consuming and expensive.

Besides security, the update process must be reliable and *fail-safe*, by which we mean that an unsuccessful update should not leave the devices in a state where they cannot boot and operate properly, but it should be possible to detect if the update failed and to load the last stable version of the updated software. At the same time, attackers should not be able to force a version rollback when the devices run a stable version of the software, because if that was possible, then they could force the devices to re-install an old, potentially vulnerable version of the software through which they can compromise the devices again.

In this extended abstract, we introduce a remote firmware and OS update system and mechanism for embedded IoT devices that satisfy the above requirements on security, reliability, and rollback protection.

Architecture

In this section, we give a high level overview on our update system architecture, which consists in partitions on the persistent storage (e.g., flash disk) of the embedded device, different images stored on those partitions, and a few log files.

We use 3 partitions with different access restrictions. The device boots from the *boot* partition, which holds the firmware image of the device and the kernel images of 2 OSs, the MainOS (typically some embedded Linux) and the UpdateOS (a trusted OS with minimal functionality, e.g., a stripped down Linux or some formally verified microkernel such as seL4[1]), as well as the root file system of the MainOS and 3 log files, called *updatelog_Firmware*, *updatelog_UpdateOS*, *updatelog_MainOS*, where different update events are logged and a control file, called *nextOStoboot*, indicating which OS to boot. Application data is stored on a separate *data* partition. Firmware and OS kernel images, as well as root file system images for the MainOS, downloaded from an update server are logged in the *downloadlog* on the *images* partition and stored on the *images* partition, from which they are copied on the *boot* partition during the secure update process. Another log file, called *selftestlog*, can also be found on the *images* partition that stores information about the result of self-testing by a freshly updated MainOS.

In order to protect the update log files (i.e., *updatelog_Firmware*, *updatelog_UpdateOS*, and *updatelog_MainOS*) from being updated by a potentially compromised MainOS, the *boot* partition can be written only by the firmware and the UpdateOS. The MainOS can write on the *data* and *images* partitions; the latter is needed in order for the MainOS to be able to download and store updates.

Boot process

Our architecture supports a secure boot process. After reset, code in a boot ROM verifies the digital signature of the firmware image on the *boot* partition, and on success, it loads and executes the firmware. The hash of the signature verification public key used by the boot ROM code is stored in a special, one-time programmable memory, which is written during device customization, after which this signature verification public key can no longer be modified. The firmware performs low level system verification, initializes trusted software components, such as a Trusted Execution Environment, and eventually executes the OS boot loader (e.g., U-Boot[2]).

The OS boot loader has another signature verification public key, which is used to verify the digital signatures of the OS kernel images on the *boot* partition. The OS boot loader always checks the *nextOStoboot* control file, and acts according to what is indicated in that file. When

[1] https://sel4.systems

[2] https://www.denx.de/wiki/U-Boot/

the MainOS is about to be booted, the OS boot loader writes in the *nextOStoboot* that the UpdateOS should be loaded next time, makes the *boot* partition write protected, and gives control to the MainOS kernel. Finally, the MainOS kernel verifies the integrity of the root file system image on the *boot* partition, and on success, it mounts the root file system, after which the device is up and running. When the UpdateOS is about to be booted, the OS boot loader writes in the *nextOStoboot* file that the MainOS should be loaded next time, and gives control to the UpdateOS kernel.

Update process

Updates are downloaded and written on the *images* partition by an update service running on the MainOS. Upon the next boot, the UpdateOS detects the update image from the *downloadlog*, verifies its digital signature, and on success, it places the update image on the *boot* partition. In case of an update of the firmware or the UpdateOS, the update image replaces the old version, as we assume that these are thoroughly tested images that function properly. However, in case of a MainOS update, both the update image and the old image are kept on the *boot* partition. If the digital signature verification fails, the update image is deleted by the UpdateOS. In any case, an appropriate log entry is created in the corresponding *updatelog* file and the device is rebooted.

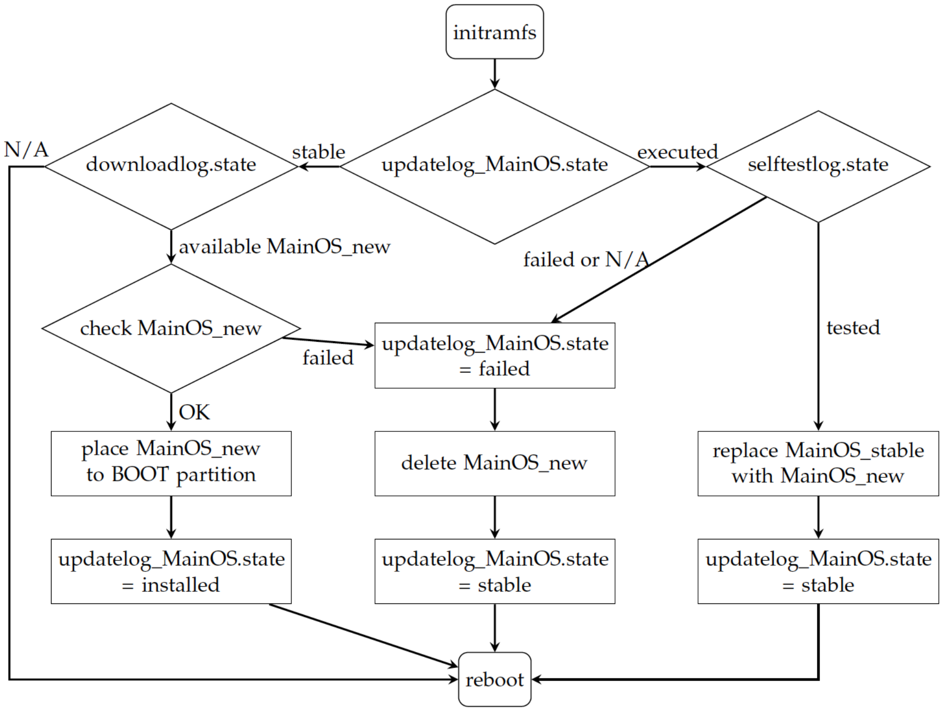

initramfs
updatelog_MainOS.state
stable
executed
N/A
downloadlog.state
selftestlog.state
available MainOS_new
failed or N/A
tested
check MainOS_new
failed
updatelog_MainOS.state = failed
OK
place MainOS_new to BOOT partition
delete MainOS_new
replace MainOS_stable with MainOS_new
updatelog_MainOS.state = installed
updatelog_MainOS.state = stable
updatelog_MainOS.state = stable
reboot

Figure 1: Simplified flowchart of updating the MainOS

In the sequel, we explain how the the MainOS is updated. The process is illustrated in Figure 1. When the device executes the boot process next time, the OS boot loader detects from the *updatelog_MainOS* that it should load and start an updated MainOS for the first time. It writes in the *updatelog_MainOS* the version to be booted, makes the *boot* partition write protected, and transfers control to the MainOS kernel. The MainOS mounts the root file system and performs self-testing. If everything goes well, the result of the self-testing is written in the *selftestlog*. Upon next boot, the UpdateOS detects that the update was successful by observing the *updatelog_MainOS* and the *selftestlog*, so it deletes the old MainOS image from the device and logs in the *updatelog_MainOS* that the update was successful.

However, the self-testing may fail or the new version of the MainOS may hang or crash. Such hangs or crashes are handled with a watchdog mechanism that reboots the device. In this case, the UpdateOS detects the failed self-testing of the updated MainOS by observing in the *updatelog_MainOS* that an update was booted, while missing any indication of a successful self-test in the *selftestlog*. When this happens, the UpdateOS deletes the failing update from the device, logs the failure in *updatelog_MainOS*, and reboots the device. After the reboot, the latest stable MainOS is loaded and executed.

Security is achieved by installing only properly signed updates by the trusted UpdateOS and logging all relevant events to the *updatelog* files that cannot be modified by the potentially compromised MainOS or any applications running on it. Trust in the UpdateOS is based on the following factors: (1) the UpdateOS is signed and its signature is verified before loading and executing it; (2) the UpdateOS executes only for a limited amount of time; (3) the UpdateOS has a reduced functionality (e.g., all unnecessary features and services are disabled, including even network access); and (3) the UpdateOS can potentially be formally verified due to its stripped functionality.

Fail-safety is achieved by using a watchdog mechanism that reboots the device upon failures and by using log files in order to detect a failed self-test after an update. Moreover, the latest stable version of an updated component is kept on the device until the success of the update can be verified, so in case of failure, the device can still boot the latest stable version.

Finally, rollback protection is achieved by keeping information about updates in the *updatelog* files, which cannot be modified by the potentially compromised MainOS or the applications running on it, and by removing old versions from the device after a successful update.

The described architecture and update process are complex enough to warrant for a thorough verification. For this reason, we used the UPPAAL[3] model checker to model the update process and to formally verify its correctness. We checked the following two properties:

- **Update is possible:** When a given version of the MainOS is running and there is a functioning update available, it is possible to reach a state where this update is successfully installed.

- **Rollback is impossible:** It can never occur that a given version of the MainOS is successfully installed when a newer version was running and marked as stable in the past.

Our update process described above satisfies both properties.

Acknowledgment

The presented work was carried out within the SETIT Project (2018-1.2.1-NKP-2018-00004), which has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2018-1.2.1-NKP funding scheme.

References

[1] L. Szekeres, M. Payer, L. T. Wei, and R. Sekar. Eternal war in memory. *IEEE Security and Privacy Magazine*, 12, May-June 2014.

[2] TCG IoT-SG. TCG Guidance for Secure Update of Software and Firmware on Embedded Systems – version 1.0, revision 64. TCG Reference Document – Draft, July 2019.

[3] http://www.uppaal.org/

Quadric tracing: A geometric method for accelerated sphere tracing of implicit surfaces

Mátyás Kiglics and Csaba Bálint

Abstract: Sphere tracing is a common raytracing technique used for rendering implicit surfaces defined by a signed distance function (SDF). However, these distance functions are often expensive to compute, prohibiting several real-time applications despite recent efforts to accelerate it. This paper presents a method to precompute a slightly augmented distance field that hugely accelerates rendering. This method supports two configurations: (i) accelerating raytracing without losing precision, so the original SDF is still needed; (ii) entirely replacing the SDF and tracing an interpolated surface.

Keywords: Computer Graphics, Quadrics, Signed Distance Function

Introduction

A signed distance function (SDF) $f : \mathbb{R}^3 \to \mathbb{R}$ returns the Euclidean distance from the surface $\{f = 0\} = \{\boldsymbol{x} \in \mathbb{R}^3 \mid f(\boldsymbol{x}) = 0\}$ for a given point in space. Any $\boldsymbol{p} \in \{f < 0\}$ point is inside, while any $\boldsymbol{p} \in \{f > 0\}$ is outside the geometry. Various sphere tracing algorithms exist for surface visualization. These raytracing techniques often start a ray through each pixel of the virtual camera and march along the ray, taking distance sized steps [1]. This is because, for any $\boldsymbol{p} \in \mathbb{R}^3$ point, there are no surface points within $f(\boldsymbol{p})$ distance: this is called the unbounding sphere. When the point-to-surface distance becomes negligible, the surface is reached.

However, all sphere tracing algorithms slow down near the surface regardless of the direction taken [2, 3, 7]. The only exception is when the derivative of f is known and the geometry is convex in which case very large steps can be taken [1]. This is because, instead of an unbounding sphere, we can draw a separating plane with normal $\nabla f(\boldsymbol{p})$ and surface point $\boldsymbol{p} - f(\boldsymbol{p}) \cdot \nabla f(\boldsymbol{p})$. Nonetheless, the surface is often concave, and the derivative might be undefined or unknown. For this reason, we generalize the unbounding sphere and separating plane approach to unbounding quadrics [5] for any SDF.

Our method consists of two steps. During precomputation, we store distance values in a regular or an octree grid. For each cell, the eight distance values are stored along with a single $k \in [-1, 1]$ parameter describing the shape of the quadric the cell defines. During the rendering step, *quadric tracing* intersects the ray with the precomputed quadric to accelerate tracing convergence.

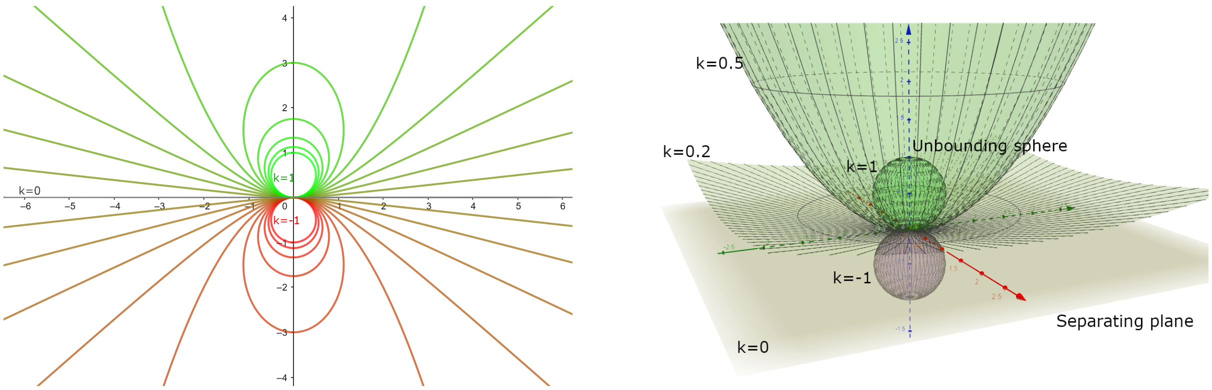
Conic sections of $k \in [-1, 1]$

We define a series of conic sections parameterized by $k \in [-1, 1]$. Since the curve will be symmetric about the y-axis, its implicit equation has the following form:

$$A(k) \cdot x^2 + B(k) \cdot y^2 + C(k) \cdot y = 0 \qquad (A, B, C : [-1, 1] \to \mathbb{R}). \tag{1}$$

We define the functions above to have the desired eccentricity, shape, and curvature at the origin, such as $A := A(k) := |k|$, $B := B(k) := 2(|k| - 0.5)$, and $C := C(k) := -k$, as seen on Figure 1a. For brevity we omit the function notation.

We can obtain a parameterization $\boldsymbol{s}(t)$ of this conic by intersecting it with lines through the origin that has $t \in [0, 2\pi)$ angle with the x-axis. That is, by substituting polar coordinates into the implicit form in (1) we can solve for $r(t)$:



(a) Conic sections with different $k \in [-1, 1]$ values (b) Our unbounding quadrics of revolution

Figure 1: Unbounding quadrics: unbounding sphere $k = 1$, unbounding ellipsoid $k \in (0.5, 1)$, unbounding parabola $k = 0.5$, unbounding hyperboloid $k \in (0, 0.5)$, separating plane $k = 0$, bounding hyperboloid $k \in (-0.5, 0)$, bounding parabola $k = -0.5$, bounding ellipsoid $k \in (0, -0.5)$, and bounding sphere $k = -1$.

$$A \cdot (r(t) \cdot \cos t)^2 + B \cdot (r(t) \cdot \sin t)^2 + C \cdot r(t) \sin t = 0 \ .$$

Assuming $r(t) \neq 0$ yields the polar parametrization of the conic section

$$r(t) = \frac{-C \cdot \sin t}{A \cdot \cos^2 t + B \cdot \sin^2 t} \quad \Longrightarrow \quad \boldsymbol{s}(t) := \begin{bmatrix} \cos t \cdot r(t) \\ \sin t \cdot r(t) \end{bmatrix} \qquad (t \in [0, 2\pi)) \tag{2}$$

For values of $k \in \left(-\frac{1}{2}, \frac{1}{2}\right)$, the $\boldsymbol{s}(t)$ describes a hyperbola with an unwanted branch. Let $L(k) \in [0, 2\pi]$ denote the value where $r(t)$ has a singularity. Thus, we can restrict $\boldsymbol{s}(t)$ to the $[0, L(k))$ interval where

$$L(k) := \begin{cases} \frac{\pi}{2}, & \text{if } A(k) \cdot B(k) \geq 0, \\ \arctan \sqrt{\frac{-A}{B}}, & \text{otherwise} \end{cases} \qquad (k \in [-1, 1]) \ .$$

Note that for all of the equations above, the somewhat heuristic $A(k)$, $B(k)$, and $C(k)$ functions may be redefined if needed.

Unbounding quadrics

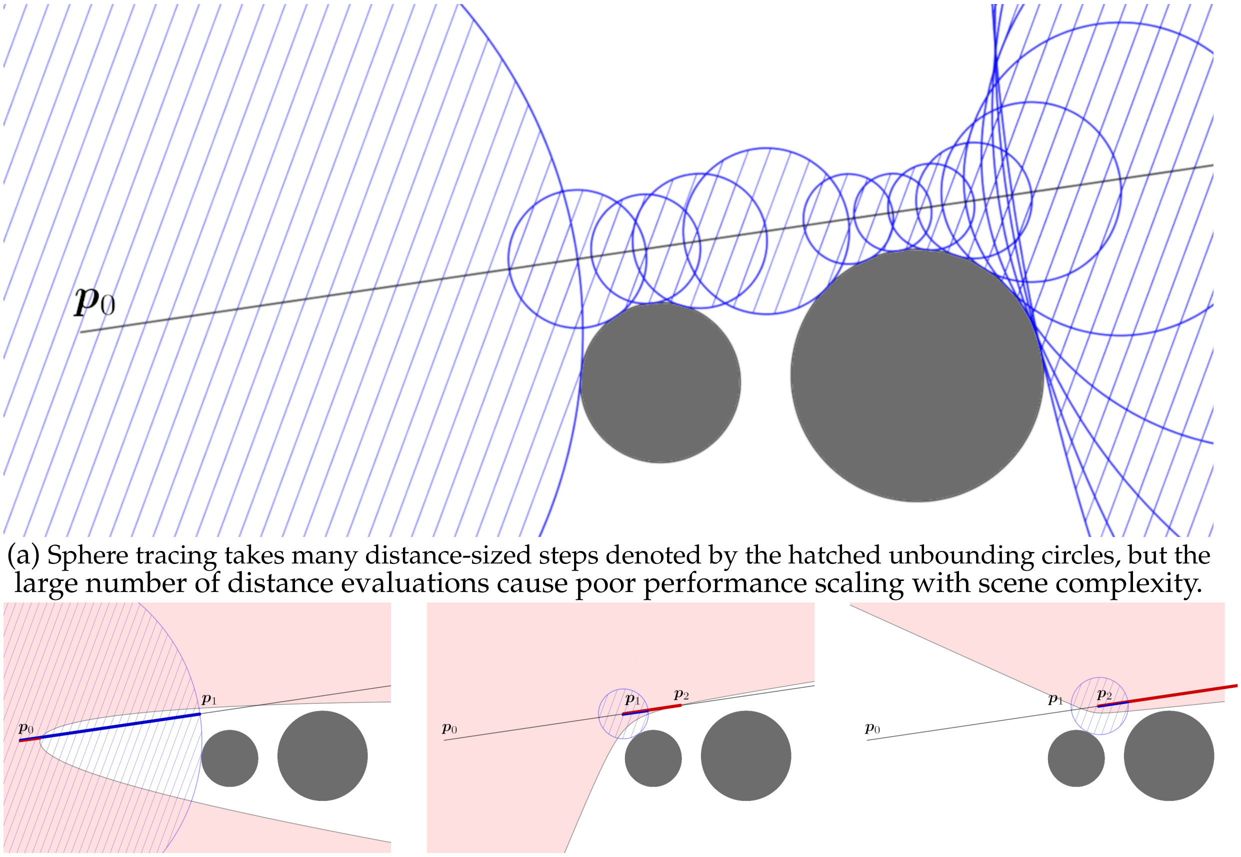
We parameterize quadrics of revolution by rotating $\boldsymbol{s}(t)$ in (2) around the vertical axis:

$$P(u, v) = r(u) \cdot \begin{bmatrix} \cos v \cdot \cos u \\ \sin v \cdot \cos u \\ \sin u \end{bmatrix} \qquad (u \in [0, L(k)), v \in [0, 2\pi)).$$

These quadrics can be seen on Figure 1b. Similarly, the implicit equation becomes

$$A \cdot (x^2 + y^2) + B \cdot z^2 + C \cdot z = 0.$$

Applying the above, we can calculate the intersection of a ray and the quadric surface. The ray is given by a point and a vector ($\boldsymbol{p}_0, \boldsymbol{v}_0 \in \mathbb{R}^3$), and any $\boldsymbol{p}$ point of the ray can be written as $\boldsymbol{p} = \boldsymbol{p}_0 + t\boldsymbol{v}_0, t \in \mathbb{R}$. Substituting $\boldsymbol{p}$ into the implicit equation of the quadric surface, we get a quadratic equation for t.



(a) Sphere tracing takes many distance-sized steps denoted by the hatched unbounding circles, but the large number of distance evaluations cause poor performance scaling with scene complexity.

(b) A sphere tracing step was taken because it was larger

(c) Quadric tracing step is now larger then the distance

(d) Quadric step to infinity terminates the trace

Figure 2: Comparison of sphere tracing (a) to three steps of quadric tracing (b,c,d) on the same scene composed of two circles in 2D. The preprocessing step created the pink unbounding regions to accelerate raytracing whenever the quadric step is larger (c,d).

Quadric tracing

Preprocessing First, a three-dimensional grid is computed to store the signed distance values. For each cell, we compute the approximate normal direction from the eight SDF values stored in the corners of the cell. If the normalization cannot proceed due to division by a small number, then a fixed direction will suffice, such as $[1, 0, 0]^T$. Then, the largest unbounding quadrics that revolve around this normal direction are obtained for each cell using an algorithm relying on sphere tracing. The exact method is to be published in another paper.

Raytracing The benefit of quadric tracing is that it takes much larger steps along the ray as illustrated on Figure 2. For each step, the $\boldsymbol{p}_0+t\cdot\boldsymbol{v}_0$ ray is intersected with the quadric defined by the cell that contains $\boldsymbol{p}_0$. If there are $t_1 < 0 < t_2$ solutions, then we can accelerate the distance-sized step with a $\Delta t = \max(t_2, f(\boldsymbol{p}_0))$ step-size. For this notation, $\boldsymbol{p}_0' = \boldsymbol{p}_0 + \Delta t \boldsymbol{v}_0$ needs to be recomputed at the end of the step. Here, $f(\boldsymbol{p}_0)$ can be either the original distance function or the one interpolated from grid values.

Conclusion

The unbounding quadrics can turn “inside-out” bounding the whole surface within. This means that most of the rays that miss the surface will only take one or two iterations to trace, compared to the hundreds of iterations that sphere tracing usually takes. In summary, this method trades render time for preprocessing time and memory usage. Even though the memory usage can be negligible with a sparse representation such as an octree and using the exact method without interpolation, the surface cannot evolve in time during rendering without constantly updating all the cells. Applications, empirical results, and quadric construction for the preprocessing step will appear in a subsequent paper.

Acknowledgements EFOP-3.6.3-VEKOP-16-2017-00001: Talent Management in Autonomous Vehicle Control Technologies — The Project is supported by the Hungarian Government and co-financed by the European Social Fund. Supported by the ÚNKP-19-3 New National Excellence Program of the Ministry for Innovation and Technology. National Research, Development and Innovation Office HUNGARY Új Nemzeti Kiválóság Program

References

[1] John C. Hart. *Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces.* The Visual Computer, 12:527–545, 1994. https://doi.org/10.1007/s003710050084

[2] Benjamin Keinert, Henry Schäfer, Johann Korndörfer, Urs Ganse, and Marc Stamminger. *Enhanced Sphere Tracing*. In Smart Tools and Apps for Graphics 1–8, Andrea Giachetti (Ed.) The Eurographics Association, 2014. http://dx.doi.org/10.2312/stag.20141233

[3] Csaba Bálint, Gábor Valasek. *Accelerating Sphere Tracing*. EG 2018 - Short Papers 19–32, Olga Diamanti and Amir Vaxman (Eds.) The Eurographics Association, 2018. http://dx.doi.org/10.2312/egs.20181037

[4] Róbert Bán, Csaba Bálint, Gábor Valasek. *Area Lights in Signed Distance Function Scenes*. EG 2019 - Short Papers 85–88, Paolo Cignoni, Eder Miguel (Eds.) The Eurographics Association, 2019. https://doi.org/10.2312/egs.20191021

[5] Silvio Levy. *Geometry Formulas and Facts*. 30th Edition of CRC Standard Mathematical Tables and Formulas, CRC Press, 1995.

Private/Public Resource Discovery for IoT: A Two-Layer Decentralized Model

Mohammed B. M. Kamel, Peter Ligeti and Christoph Reich

Abstract: Billions of resources are connected to each other through the Internet of Things (IoT). Due to the huge number of connected resources in IoT, the challenge is how to discover the appropriate resources efficiently. The task becomes more challenging by taking into consideration the limited storage and computation power of devices and the distributed nature of the network. In addition, some of the resources such as a private camera are private to one or a set of clients. Therefore, making these private resources discoverable through the whole network increases the danger of being compromised by an unauthorized entity. In this paper, we proposed a two layer model of resources discovery for the IoT. The proposed model is based on the Distributed Hash Table (DHT) overlay. It allows a resource to be registered publicly or privately, and to be discovered in a decentralized scheme in the IoT network.

Keywords: Resource Discovery, IoT, Decentralized Scheme

Introduction

In order to make a resource (i.e. an IoT thing, a meta-data or a service provided by an IoT thing) be discoverable, it has to be registered in the IoT network. Later on and depending on the used architecture in the network (i.e. centralized or decentralized), this registered resource is discoverable through a single or multiple points in the network. The resource discovery is a mechanism to return the address of a resource (e.g. its URI, other metadata and further links about the resource) based on the information provided during the lookup operation.

Important requirement of the IoT is the avoidance of single point of failures as it can be a centralized discovery service, even if implemented using redundancy and replication. One of the main goals of a decentralized discovery approach is to make the data distributed among multiple entities. Therefore, avoidance of single point of failure and keep the data close to its origin point are two main features of using a decentralized rather that a centralized scheme. As the connected devices become more powerful in term of connectivity and computing power, this goal becomes a realistic and necessary to achieve.

While some of the resources in the IoT are public resources, such as a temperature sensor attached to a public building, but there are some other resources that are private to one or a set of clients. Therefore, making these private resources discoverable through the whole network increases the danger of being compromised by an unauthorized entity. A private camera does not have to be discoverable by the whole network.

In this paper, we proposed a two-layer discovery model in which a resource can be registered publicly and therefore be discoverable through the whole network. In addition, a resource can be registered privately which makes it discoverable only to a subset of clients in the network. While this work uses the same methodology used in [1] and [2], it shares the main concepts with [3] such as the adoption of the Distributed Hash Table (DHT) with improvements in different aspects of the model.

Proposed Model

There are three main sets in the proposed model: set of clients ($\mathcal{C}$), set of objects ($\mathcal{O}$) and set of gateways ($\mathcal{W}$). The finite set $\mathcal{C}$ consists of the IoT clients in the network. An object $o \in \mathcal{O}$ is any device in the IoT network with proper computational power that handles a resource u. The set

N is a combination of C and O ($\mathcal{N} = \mathcal{C} \cup \mathcal{O}$). Subsets of $\mathcal{N}$ are connected to different IoT gateways in $\mathcal{W}$. A gateway w's responsibility may vary from handling a few nodes (e.g. smart home) to hundreds of nodes (e.g. environmental monitoring). The proposed model uses a p2p structure with DHT that provides a structured method of addressing and discovery of the peers. The members of $\mathcal{W}$ represents the peers in the p2p overlay. Let $H(.)$ be a collision resistant one-way hash function with d bits message digest, $Enc_k(m)$ be an encryption of the message m using symmetric key k and $Sign_w(m)$ be a digital signature for message m generated by $w \in \mathcal{W}$ gateway.

Resource Registration

A resource u in the network has its specific address and a set of attributes that describe its properties (e.g. its type, location, etc.). When an object $o \in \mathcal{O}$ wants to put its resource u in the network, it has to add the required information in the p2p overlay through a member of $\mathcal{W}$. The required information consists of a pair of <*tag, data*> that is described later in details. Each resource in the network has one or more *tags* which are used to discover it in the network. The resource's tags are generated based on the set of attributes that describe the resource. The access *data* of a resource u is stored in the IoT network based on the its tags with a predefined number of replicas (The actual number depends on the *replication factor* rp). Choose an appropriate rp parameter depends on the nature of the network. As a general rule for choosing the appropriate rp value, the probability of existence of a subset of offline peers *Offline* $\subset \mathcal{G}$ with cardinality greater than the number of replicas has to be negligible ϵ. This is shown in equation 1. In addition, the existence of replicas increases the system performance by reducing the access load on any specific peer in the overlay.

$$P(\|\mathit{Offline}\| \geq rp) < \epsilon \tag{1}$$

An object $o \in \mathcal{O}$ puts its resource u in the network as pairs of <*tag, data*>. The *data* parameter consists of the URI and other metadata about the resource u. The set of the attributes that describe resource u are fed into the hash function to generate the *tag* parameter. The direct peer in the overlay (i.e. the IoT gateway w that the object o is connected to) stores these pairs locally for a specific time depending on the caching expiry parameters. Additionally, the *close peers* in $\mathcal{W}$ to tag parameter of the generated pairs are responsible for storing <*tag, data*> pair. The model does not depend on any specific *distance function (dst)* to computer the closeness, it can be any particular distance function.

Let $\mathcal{I}_d$ be a set of all possible sequence of d-bit binary digit (i.e. identifiers) and each peer $w \in \mathcal{W}$ has an identifier $id_w \in \mathcal{I}_d$ and each resource u has a $tag_u \in \mathcal{I}_d$. Let define the following set of peers

$$M(u) = \{w : tag_u \approx id_w, \nexists w' \mid dst(id_{w'}, tag_u) < dst(id_w, tag_u)\} \tag{2}$$

The set $M(u)$ links each resource u depending on its attribute identifiers tag to a peer $w \in \mathcal{W}$ that its identifier $id_w \in \mathcal{I}_d$ is close or equal to tag_u. The cardinality of $M(u)$ depends on the replication factor rp parameter. The rp indicates the number of close peers to w that are responsible for storing a replica of the pair <*tag, data*>. The procedure of registering a resource u in the network consists of three steps:

- `Tag Definition and Generation`: The object o determines the address data and the attributes that describe the resource u and send the generated hash value of the attributes (i.e. *tags*) along with the corresponding data to directly connected w, structured as <*tag, data*>.

- `Tag pair Signing`: In this step the appropriate set of pairs of the resource u is signed by $w \in \mathcal{W}$.
- `Resource Registration`: The gateway w puts the set of pairs in the overlay by storing the pairs at the corresponding peer in the overlay. The possible set of generated pairs of a temperature sensor is shown in table 1.

Table 1: set of pairs for a temperature sensor in the overlay

Attribute	tag	Tag pair
building d	ba7..984	<ba7..984, { $resource_u uri$ } >
hmp7	1ad..8e8	<1ad..8e8, { $resource_u uri$ } >
room v	665..fd3	<665..fd3, { $resource_u uri$ } >
temperature	d96..664	<d96..664, { $resource_u uri$ } >

Public Resource Discovery

The resources that are registered without any restrictions in the IoT network can be discovered by all clients in the network based on their attributes. The proposed model allows search for one or more attributes. A client $c \in \mathcal{C}$ lookup for a resource by sending a lookup request with the required set of attributes to the gateway w that is directly connected to it. The gateway w after receiving a discovery request from a client c generates the appropriate tags for the lookup process in the overlay based on the received attributes. The discovery process contains three main steps as follows.

- `Query Generation`: In the first step, the gateway w generates the set of tags based on the received attributes from a client c. This is done by hashing each of the requested attributes in the client's request.
- `Lookup`: The second step starts by issuing the lookup request by w in the overlay. The result $\mathcal{R}_i$ of each of the lookup operations is a set of data parameters that indicates the resources with the specific attribute i.
- `Result Gathering`: After receiving the results and verifying them based on their attached digital signatures, the intersected members of sets $\mathcal{R}_0 \cap \mathcal{R}_1 \cap \cdot\cdot \cap \mathcal{R}_n$ will be returned as a result to the requested client c.

Private Resource Discovery

Every object o in the IoT network is able to keep a resource private and discoverable only by a predefined set $\mathcal{F}_o$. An object o has a set of its friends $\mathcal{F}_o = \{f_1,...,f_n\} \subset C$ that can be communicated with in a secure and trusted way. The members of a friend set $\mathcal{F}_o$ of an object o are connected through members of $\mathcal{W}$ but they are not part of the p2p overlay itself. Each private resource has a private identifier id_u that is chosen uniformly at random from a given range, e.g. from bit strings of length 512. An identifier id_u is known only by the members of $\mathcal{F}_o$. Additionally, each $c \in \mathcal{C}$ has also a private identifier id_c that is chosen uniformly at random from a given range. The object o stores the private identifiers of each $f \in \mathcal{F}_o \subset \mathcal{C}$ locally. In addition, for every object o and for each $f \in \mathcal{F}_o$ an initial value (IV_{of}) and a common secret key (k_{of}) are generated and shared between them on a secure channel. The key k_{of} is used to encrypt the transmitted data between them. These keys are stored at each node locally at the setup phase and its future distribution scheme is out of the scope of this paper.

If an object o registers its resource u privately, only the members of $\mathcal{F}_o$ can discover and access this specific resource of o. To do so, an object has to generate a specific identifier $privateTag$ for a resource u using equation 3. The communication address of the resource is then encrypted using the shared key k_{of}. Then, the corresponding gateway after receiving the resulted pairs <$privateTag$, $encryptedData$> (a single pair for each $f \in \mathcal{F}_o$) put them in the overlay. Later, the members of $\mathcal{F}_o$ can discover this private resource by computing its private tag $privateTag$. Each object has different private tags. These private tags are not permanent and used only once. The $privateTag_{new}$ parameter can be calculated using $privateid_{old}$, id_u and id_f values. At any given time, the current private tag of a resource is computed as 3:

$$privateid_{new} = H(privateTag_{old} \oplus id_u \oplus id_f) \tag{3}$$

where $privateTag_{old}$ is the previous private tag of the resource u (i.e. the output of the previous hash) and $privateTag_0 = H(IV \oplus id_u \oplus id_f)$. While the discovery process of a private resource in the network resembles the public discovery, but there are two differences. First is that in order to be able to discover a resource u handles by o, a client has to be able to compute its private tag, i.e. being a valid member of $\mathcal{F}$. Secondly, after receiving the result of the lookup, the returned data is confidential and can be read only by knowing the secret key k corresponding to this specific node.

Discussion

In this paper a two layer discovery model is proposed. The model enables a resource to be registered in the network in a decentralized scheme. This improves the availability of a resource and removes the bottleneck of a centralized resource discovery entity. In addition to the public resources, the private resources can be registered and discovered in the network. The private resources are discoverable by a predefined set of clients in a secure approach.

Acknowledgements

This research has been partially supported by project no. ED_18-1-2019-0030 (Application-specific highly reliable IT solutions) has been implemented with the support provided from the National Research, Development and Innovation Fund, financed under the Thematic Excellence Programme funding scheme and by the European Union, co-financed by the European Social Fund. (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Infocommunications) and by SH Programme. The authors thank ELTE Eotvos Lorand University and HFU Hochschule Furtwangen University for their support.

References

[1] Kamel, M., Ligeti, P., Nagy, A., 2019 An Address Propagation Model in P2P and F2F Networks, *Studia Informatica* **64** (1), pp. 91-101.

[2] Kamel, M., Ligeti, P., Nagy, A., 2018 Improved Approach of Address Propagation for F2F Networks, *Proc. of IEEE 2nd European Conference on Electrical Engineering & Computer Science*

[3] Kamel, M., Crispo, B., Ligeti, P., 2019. A Decentralized and Scalable Model for Resource Discovery in IoT Network, *IEEE 15th International Conference on Wireless and Mobile Computing, Networking and Communications*

Improving keyword spotting with limited training data using non-sequential data augmentation

Mohammed Mohammed Amin and István Megyeri

Abstract: Nowadays, deep learning models achieve state-of-the-art results in many fields. A common criticism of these algorithms is their need for a very large training set. In this work, we consider two augmentation strategies to mitigate this issue. We applied them for enhancing classification performance on the Speech Commands database using only 10 samples per class. The sequential combination is widely applied when every transformation performed consecutively. In contrast, we recommend the non-sequential approach that uses transformations uniformly but only one at a time. Our method achieves 78% test accuracy. 9% improvement comparing to the non-augmented case and outperforms the sequential one.

Keywords: data augmentation, speech recognition, convolutional neural network

Introduction

In recent years deep learning brings huge improvements in speech recognition [5, 4]. Perhaps the reasons behind these improvements are the era of big data, improved optimization methods, and the ability to train very deep networks. The point is that these methods give similar performance to classical approaches or may overperform them. However, these algorithms are not without drawbacks. It is known that training a deep learning model requires a sufficient amount of data and computational power. Further, these models are not directly applicable when the task is the same, but the data distribution differs slightly from training data. In speech, it is common difficulty to obtain a speaker agnostic model that able to generalize well. In order to successfully adopt deep learning, these problems need to be solved.

For mitigating the training data problem, there are several approaches like [7, 6]. Those take advantage of other data sources. In few-shot learning at training time, there are only a "few" examples of the targeted class but many others of non-targeted classes. Those can help to improve the targeted class performance. Another direction is unsupervised or semi-supervised model pre-training on a large unlabeled data set [6]. Later this model can be adjusted to the target problem and usually gives better results than using only the available training data.

In a sense, pre-training or existing few-shot methods do not handle the main problem. Those still rely on large amounts of training data in the form of other classes or unlabeled examples. On the other hand, there are a lot of augmentation methods like [10, 3]. Those can naturally increase training data using only given samples. However, due to some reasons, they are applied mainly to improving some already well-performing models.

In this study, we use multiple audio augmentation methods to increase test accuracy. We formed a 10-shot problem from the Speech Commands data set [9] using ten speakers and those recording for all classes. Note, this training set is very challenging because of both sample size and number of speakers. Usually, deep learning is not used in such cases.

Data augmentations can be combined in many ways as long as they preserve the sample label. Here, we define two strategies. The first is sequential, which uses all the transformations consecutively. The second performs non-sequential augmentation, using only one at a time. One may form new transformations by composing some existing ones or changing the transformation order. The number of possibilities is almost infinite. To the best of our knowledge, there is no good practice of how to exploits the potential of several transformations, even for the simple question using them sequentially or non-sequentially.

We carry out multiple experiments to answer this question. We show that sequentially combining transformations may achieve poor results. Further, we demonstrate the non-sequential approach can increase the test accuracy significantly on the 10-shot learning problem.

Materials and method

We experimented with the Speech Commands data set [9]. It contains one second long audios of 30 possible classes like bed, bird, yes or no, etc. In total, there are 64723 samples and six additional for background noises. The test and validation set include 6836 and 6799 samples, respectively. Those samples are from speakers that are not present in the training data.

As feature extraction, we considered Mel-frequency cepstral coefficients (MFCCs) and Mel-spectrogram. We used the librosa[1] library. MFCCs were extracted using a 16000 sampling rate, and the remaining parameters are the defaults. The mel-spectrogram computed on 80 mel filters with 256 FFT windows length and 128 as hop size. The sampling rate is the same.

Here, we considered four transformations: noise injection, time-shifting, pitch changing, and time stretching. Noise injection has done with available noises. Each transformation has a range of modifications. Those can be seen in Table 2. The noises have different ranges to prevent making the sample unrecognizable. We ascertained the ranges empirically preserve class information and still recognizable when all the transformations applied.

All the transformations were applied in training phrase before feature extraction. The amount of modification was uniformly sampled from the given ranges. To combine the transformations, we investigate two strategies. First is *sequential* when all the available transformations are applied consecutively. This option is provided by many deep learning libraries as default. This way, the sample number increases exponentially by the availability of transformations. However, we have only one transformation that is the composite of all the modifications.

The second is called *non-sequential* when, unlike in the previous, only one transformation is used at a time. The applied one is selected uniformly. This approach may obtain fewer new examples, but those are in different regions of the space. Hence it may absorb larger capacity of the model.

We used a convolutional neural network for comparing the strategies. It has five convolution layers; each applies 5×5 convolutions with increasing filter sizes $32, 64, 128, 256$ and 512. Each convolution is followed by batch normalization [2] then pooling. Each pooling is a 2×2 max-pooling with strides 1 except the last one, which is a global max pooling. The last convolution is followed by two dense layer. Those have 512 and 30 neurons with relu and softmax activation, respectively. The relu dense layer also followed by a batch normalization. The network has 4637854 parameters in total.

The model was trained using the categorical cross-entropy loss function and Adam [1] as the optimizer. The batch size was 10 for the 10-speaker set otherwise 30. We used early stopping during optimization based on validation accuracy. The patience was 300 for 10-speaker, 30 for the rest. The reason behind different tolerance and batch sizes for 10-speaker data is two way. Training on smaller data was faster; hence we afford longer runs. Second, we were interested in finding the best model setting that may need larger patience of early stopping. From all training, those model states were restored that gave the best validation accuracy. For further regularization, we used Dropout [8] after the first dense layer with a dropping rate of 0.5.

[1] https://librosa.github.io

Table 1: Training and test set accuracy using different training data sizes. Test accuracy drops significantly for the 10-speaker case.

Train-data	Train-acc	Test-acc
100%	0.999	0.965
25%	0.999	0.956
15%	1	0.948
10%	0.999	0.936
5%	0.999	0.876
10-speaker	0.993	0.69

Table 2: Test accuracy and ranges of separately applied augmentations using the 10-speaker training set. Each transformation improves the test accuracy comparing to the unmodified 10-speaker case.

ID	Transformation(s)	Test-acc	Ranges
T1	pink and white noise	0.723	$[0, 0.03]$
	bike, running		$[0, 0.1]$
	dishing, cat		$[0, 0.03]$
T2	time shifting	0.707	$[0, 0.2]$
T3	stretching	0.701	$[0.95, 1.05]$
T4	pitch changing	0.716	$[-0.5, 0.5]$

Results

We evaluated the two feature set MFCCs and Mel-spectrogram. Those give 0.942 and 0.965 test accuracy with our convolutional network. Therefore we used the Mel-spectrogram. Then we artificially reduced the training set size to find where our model performance drops significantly. The results can be seen in Table 1. Interestingly the test performance is the same till the 10% data. Below this, it starts to fall down. As we expected, the 10-speaker set gives the poorest result. The goal is to improve it using the two augmentation strategies.

The test accuracy using the transformations separately are in Table 2. Each transformation improves accuracy. Especially T1 and T4 seem useful, those increase accuracy by 3%. Figure 1 contains the results of multiple transformations using the two strategies. It is not expected that combining some transformation may worsen the results. Interestingly the non-sequential method outperforms the sequential in all cases. The test accuracy difference is 5% between them when all transformation used. Recall that the sequential approach gives more varied combinations and is widely adopted. Nevertheless, the non-sequential strategy increased the test accuracy by 9% comparing to the non-augmented case.

Conclusions

We compared two methods for combining several augmentation transformations. We found that the non-sequential strategy outperforms the widely adopted sequential method. After using all the four transformations combined with it, the test accuracy improvement is 9% comparing to the not augmented case. We decreased the gap significantly between the full data and 10-speaker performance. Evaluating the non-sequential strategy with more transformations and using other data sets is underway.

References

[1] Jimmy Ba and Diederik Kingma. Adam: A method for stochastic optimization. In *3rd Intl. Conf. on Learning Representations (ICLR)*, 2015.

[2] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *ICML*, volume 37 of *JMLR Workshop and Conf. Proc.*, pages 448–456. JMLR.org, 2015.

[3] Tom Ko, Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. Audio augmentation for speech recognition. In *INTERSPEECH*, pages 3586–3589. ISCA, 2015.

[4] Xin Lei, Andrew W. Senior, Alexander Gruenstein, and Jeffrey Scott Sorensen. Accurate

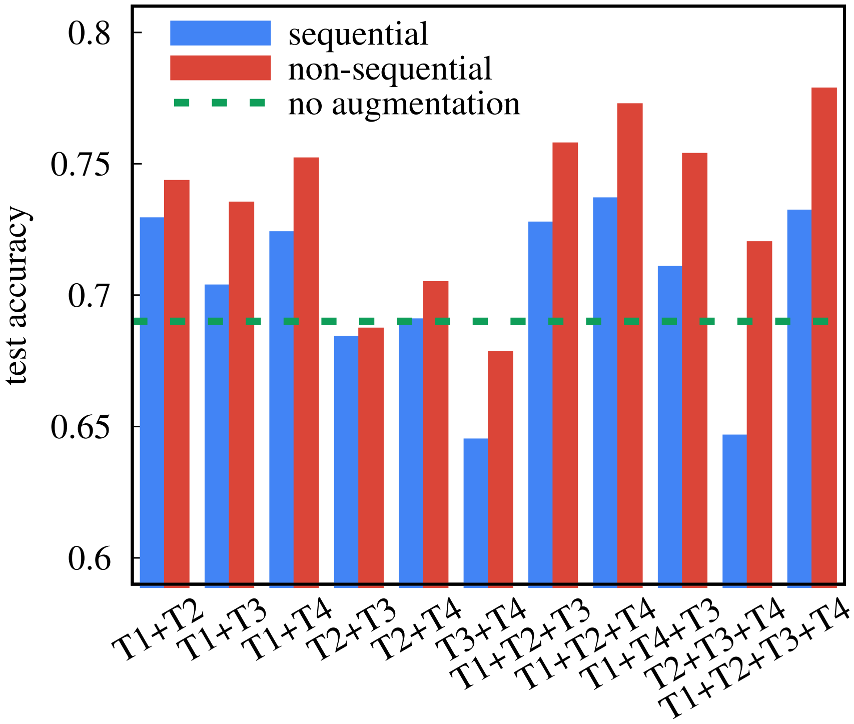


Figure 1: Sequential and non-sequential augmentation test accuracy as a function of applied transformations. Non-sequential outperforms sequential in all cases. It achieves a 9% improvement over the non-augmented case.

and compact large vocabulary speech recognition on mobile devices. In *Proc. Interspeech*, 2013.

[5] Tara Sainath and Carolina Parada. Convolutional neural networks for small-footprint keyword spotting. In *Interspeech*, 2015.

[6] Steffen Schneider, Alexei Baevski, Ronan Collobert, and Michael Auli. wav2vec: Unsupervised Pre-Training for Speech Recognition. In *Proc. Interspeech 2019*, pages 3465–3469, 2019.

[7] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4077–4087. Curran Associates, Inc., 2017.

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[9] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. 04 2018.

[10] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *Int. Conf. on Learning Representations*, 2018.

Social network characteristics from the viewpoint of centrality measures

Orsolya Kardos and Tamás Vinkó

Abstract: Identifying important actors in a social network or predicting the possible connections between them can be of high practical importance regarding many real-life applications in the field of social networks. One of these is the Influence Maximization problem, which aims at finding the most influential users or nodes in a network i.e. the ones whose opinion can spread through the network in the most successful way. Another interesting application is the field of link prediction where algorithms attempt to decide wether a connection will be present in a future version of the network. Centrality measures on the other hand are node characteristics that assign a real number to every node in the network which denominates its importance by different aspects depending on the concrete measure. In this paper besides using different parameters in order to better estimate the influence spreading capability of the nodes, we also introduce a more general model that can be used for graph attribute characterization with the use of centrality measures.

Keywords: centrality measures, social networks, influence maximization, link prediction

Introduction

Given a social network like the online social media platforms or real-life friendships the topology of the underlying graph can easily provide an insight about the most important contributors of the network. In the context of complex networks node centrality measures are the metrics designed in order to identify those nodes which provide a better understanding regarding the functional and structural behavior of the network. Importance in general being a rather vague concept resulted in several different interpretations and thus countless coexisting centrality measures. This is the main reason why we can state that not the values themselves are the important factors, but rather the node rankings provided by these centrality measures. The most commonly known and used centrality measures are degree [15], closeness [2, 17], eigenvector [3], betweenness [7], PageRank [4] and HITS [11].

Different studies put the emphasis on trying to find a connection between the ranking by centrality measures and their influence spreading capability. The Influence Maximization (IM) problem in general tries to find a small number of nodes that are able to influence as many other nodes as possible in a graph or network. Several algorithmic approaches were developed in order to address the mentioned $\mathcal{NP}$-hard combinatorial optimization problem. Evaluating the nodes by their influence spreading capability can result in a ranking among nodes. These evaluations are the results of computationally costly propagation simulations. When considering directed graphs and speaking about the spread of an idea, gossip, opinion or similar we can categorize the nodes as active or inactive i.e. the ones who are affected by an idea and the ones who are not. In the Independent Cascade Model [9] when a node v becomes active in time step t, it is given only one chance to activate its each currently inactive neighbor w. Based on a probabilistic approach, if the activation is successful w will become active in time step $t+1$, whereas if the activation resulted in failure, node v will not get the chance to activate node w again. If in a given time step t more than one attempt from different neighboring nodes will try to influence w these attempts will be processed as an arbitrary ordered sequence. The process only terminates if no more activations can happen in the network.

Preliminaries and Datasets

In our experiments we consider both directed and weighted graphs $G = (V, E, W)$ and also directed graphs without edge weights $G = (V, E)$. Regarding the influence simulations both randomized weight assignment and also the Weighted Cascade Setting (WCS) model is used. The WCS model states that for every edge $e = (u, v) \in E$ the edge weight value is defined as $P(e) = 1/(IDC(v))$ where $IDC(v)$ is the in-degree centrality value of node v.

Jia et al. [8] introduced an improvement method for degree centrality and its extending centralities in directed networks. They proposed the usage of an α parameter in order to take in consideration not only the existence of an edge when calculating the centrality measure vector, but also its direction. The main motivation behind this is the general behavior of the Influence Maximization models, i.e. the fact that a nodes ability to infect and to being infected can be analyzed separately. In order to evaluate their results they used the SIR model. Their extension model for centralities is proposed as follows.

$$C_D^{\alpha}(v) = d_{out}(v)^{\alpha} d_{in}(v)^{1-\alpha},$$

where $C_D^{\alpha}(v)$ is the extended version of the degree centrality, $d_{out}(v)$ and $d_{in}(v)$ denominate the out-degree and in-degree of node v respectively. In our experiments the above extension will provide the foundation to proposing different new combinations of centrality measures.

In order the compare the ranking provided by the combined centrality measures and the one provided by the IC model the Kendall's Tau rank correlation coefficient was used that measures the relationship by considering the number of concordant and discordant pairs. A pair of observations $(r_i(t), r_j(t)), (r_i(t+1), r_j(t+1))$ is concordant if $r_i(t) > r_j(t)$ and $r_i(t+1) > r_j(t+1)$ or if $r_i(t) < r_j(t)$ and $r_i(t+1) < r_j(t+1)$. It is discordant if $r_i(t) > r_j(t)$ and $r_i(t+1) < r_j(t+1)$ or if $r_i(t) < r_j(t)$ and $r_i(t+1) > r_j(t+1)$. A pair is said to be tied if $r_i(t) = r_j(t)$ and $r_i(t+1) = r_j(t+1)$. In order to take in consideration the ties the formula for calculating the correlation coefficient as in [10] is as follows:

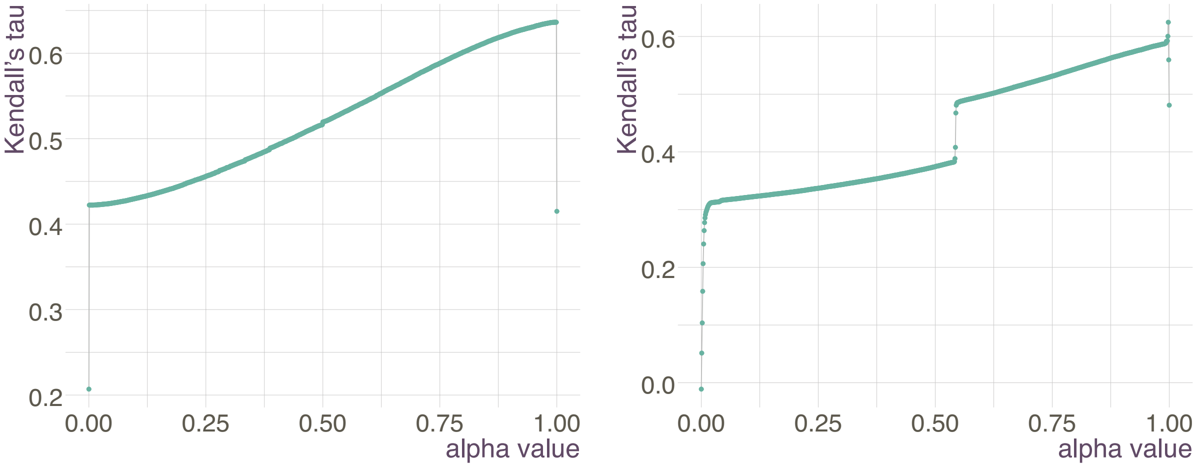
$$\tau = \frac{n_c - n_d}{\sqrt{(n_0 - n_1)(n_0 - n_2)}},$$

where n_c is the number of concordant pairs, n_d is the number of discordant pairs, $n_0 = \frac{n(n-1)}{2}$, $n_1 = \sum t_a(t_a - 1)/2$, $n_2 = \sum u_a(u_a - 1)/2$, t_a is the number of values in the ath group of ties for variable i, u_a is the number of values in the ath group of ties for variable j.

Three real-world networks from "the Koblenz Network Collection" project (KONECT) [13] were used in our experiments being the Blogs [1], Ucsocial [16] and Adolescent Health [14] networks. Besides the KONECT project graphs two artificial networks were used the first being a network generated by the Cooper-Frieze graph process [6]. Lastly the SF-1 [5] network was analyzed which is an empirical scale free network whose out-degree and in-degree both follow power law distribution.

Experiments

Regarding our numerical experiments the first attempt was to extend the above described α-parameter setup to a broader spectrum of centrality measures. Firstly we selected shortest path based metrics like closeness and betweenness centrality measures in order to investigate the Kendall's Tau rank correlation coefficient between the ranking of the measures and the information spreading capability of the nodes. On Figures 1a and 1b the Kendall correlation coefficient values are present for the combination of out-degree and in-degree and out-closeness



(a) out-degree and in-degree

(b) out-closeness and in-closeness

Figure 1: The evolution of the Kendall's tau correlation coefficient based on the change of the alpha-parameter on the Blogs network

and in-closeness respectively. Afterwards the estimate versions of these measures were calculated, and interestingly higher correlation coefficients appeared when calculating with the estimated values rather than the concrete values. By determining the most important nodes based on the different centrality measures we randomly chose pairs and triplets among them to analyze the correlation between the order of the pairs when it comes to group centrality and the diffusion capability of the given set of nodes used as initial seed nodes in the influence maximization model. In order to do so a tensor like structure was implemented that contained the values for the given pairs and triplets of nodes. Based on our experiments a generalized model was built aiming at finding the best α-parameter for the input vectors in order to approximate a given feature, like for example the ranking vector based on the propagation model. In this setup the first two input vectors are centrality measure vectors and the third vectors is the node ranking based on the propagation results from the IC model. The optimization problem is designed to find the best fitting α-parameter in order to maximize the Kendall correlation coefficient between the rank vector provided by the IC model and the ranking provided by the extended centrality measure calculated with the given α-parameter. In this setup the two parametrized vectors are multiplied in an element-wise manner. Based on this idea the well-known fact regarding the several different approaches of calculating the product of two vectors from linear algebra got involved. For example when talking about edge weight prediction, if we want to calculate the edge weights for every possible pairing of the nodes that results in a matrix, thus in this situation we are calculating the outer product of the two input vectors. We applied the α-parameter to a link prediction problem introduced in [3] in order to find better estimations for signed edge weights. The two input vectors in this situations were the fairness and goodness measures proposed by the mentioned article. The third option is taking the inner product of the two vectors, which results in a scalar. If we think of this scalar as a graph attribute, the optimization model will search for the best fitting parameter in order to approximate a global graph attribute based on two centrality measure vectors.

Conclusion

Motivated by the use of α-parametrized centrality measures we have been building a more general optimization model aiming to find the best possible parameter for its input vectors in order to approximate a given graph property. In our future work we aim to apply the model on different real-world social networks driven by the above mentioned applications like selecting a group of seed nodes for the Independent Cascade influence maximization model, or improving a missing link prediction method, but also calculating the inner dot product of the input vectors in order to make experiments with global graph features like graph diameter of clustering coefficient.

Acknowledgements

The project has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002) and by grant TUDFO/47138-1/2019-ITM of the Ministry for Innovation and Technology, Hungary.

References

[1] Lada A Adamic and Natalie Glance. The political blogosphere and the 2004 us election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, pages 36–43, 2005.

[2] Murray A Beauchamp. An improved index of centrality. *Behavioral Science*, 10(2):161–163, 1965.

[3] Phillip Bonacich. Factoring and weighting approaches to status scores and clique identification. *Journal of Mathematical Sociology*, 2(1):113–120, 1972.

[4] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[5] Young Sul Cho, Jin Seop Kim, Juyong Park, Byungnam Kahng, and Doochul Kim. Percolation transitions in scale-free networks under the achlioptas process. *Physical review letters*, 103(13):135702, 2009.

[6] Colin Cooper and Alan Frieze. A general model of web graphs. *Random Structures & Algorithms*, 22(3):311–335, 2003.

[7] Linton C Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977.

[8] Peng Jia, Jiayong Liu, Cheng Huang, Lin Liu, and Chunyang Xu. An improvement method for degree and its extending centralities in directed networks. *Physica A: Statistical Mechanics and its Applications*, 532:121891, 2019.

[9] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146, 2003.

[10] Maurice G Kendall. The treatment of ties in ranking problems. *Biometrika*, 33(3):239–251, 1945.

[11] Jon M Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[12] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. Edge weight prediction in weighted signed networks. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pages 221–230. IEEE, 2016.

[13] Jérôme Kunegis. Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1343–1350, 2013.

[14] James Moody. Peer influence groups: identifying dense clusters in large networks. *Social Networks*, 23(4):261–283, 2001.

[15] UJ Nieminen. On the centrality in a directed graph. *Social Science Research*, 2(4):371–378, 1973.

[16] Tore Opsahl and Pietro Panzarasa. Clustering in weighted networks. *Social networks*, 31(2):155–163, 2009.

[17] Gert Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966.

Toolset for supporting the number system research

Péter Hudoba and Attila Kovács

Abstract: The world of generalized number systems contains many challenging areas. In some cases the complexity of the arising problems are unknown, but computer experiments are able to support the theoretical research. In this talk we introduce a new toolset that helps to analyze number systems in lattices. The toolset is able to analyze the expansions; decide the number system property; classify and visualize the periodic points; calculate correlations between system data, etc.

The generalized number system functionality is implemented in Python, published alongside with a database that stores plenty of candidates of number systems and is able to store the custom properties like signature, matrix eigenvalues, etc. The researchers can connect to the server and request candidates by custom filters to perform experiments on them or upload new properties/candidates.

We present an introductory usage of the toolset and detail the experimental obervations that can be achieved with the toolset and the database.

Keywords: radix system, generalized number system, simulation

Introduction

Let Λ be a lattice in $\mathbb{R}^n$ and let $M : \Lambda \to \Lambda$ be a linear operator such that $\det(M) \neq 0$. Let furthermore $0 \in D \subseteq \Lambda$ be a finite subset. Lattices can be seen as finitely generated free Abelian groups. In this talk we consider number expansions in lattices.

Definition 1. *The triple (Λ, M, D) is called a* number system *(GNS) if every element x of Λ has a unique, finite representation of the form*

$$x = \sum_{i=0}^{L} M^i d_i \ ,$$

where $d_i \in D$ and $L \in N$.

Here M is called the *base* and D is the *digit set*. It is easy to see that similarity preserves the number system property, i.e., if M_1 and M_2 are similar via the matrix Q then (Λ, M_1, D) is a number system if and only if $(Q\Lambda, M_2, QD)$ is a number system at the same time. If we change the basis in Λ a similar integer matrix can be obtained, hence, no loss of generality in assuming that M is integral acting on the lattice Z^n.

If two elements of Λ are in the same coset of the factor group $\Lambda/M\Lambda$ then they are said to be *congruent* modulo M. If (Λ, M, D) is a number system then

1. D must be a full residue system modulo M;
2. M must be expansive;
3. $\det(I_n - M) \neq \pm 1$ (*unit condition*) .

If a system fulfills the first two conditions then it is called a *radix system*.

Let $\varphi : \Lambda \to \Lambda$, $x \overset{\varphi}{\mapsto} M^{-1}(x - d)$ for the unique $d \in D$ satisfying $x \equiv d \pmod{M}$. Since M^{-1} is contractive and D is finite, there exists a norm $\|.\|$ on Λ and a constant C such that the orbit of every $x \in \Lambda$ eventually enters the finite set $S = \{x \in \Lambda \mid \|x\| < C\}$ for the repeated application of φ. This means that the sequence $x, \varphi(x), \varphi^2(x), \ldots$ is eventually periodic for all

$x \in \Lambda$. Clearly, (Λ, M, D) is a number system iff for every $x \in \Lambda$ the orbit of x eventually reaches 0. A point p is called *periodic* if $\varphi^k(p) = p$ for some $k > 0$. The orbit of a periodic point p is a *cycle*. The set of all periodic points is denoted by $\mathcal{P}$. The *signature* $(l_1, l_2, \ldots, l_\omega)$ of a radix system is a finite sequence of non-negative integers in which the periodic structure $\mathcal{P}$ consists of #l_i cycles with period length i $(1 \leq i \leq \omega)$.

The following problem classes are in the mainstream of the research.

Definition 2. *For a given (Λ, M, D) the* decision problem *asks if the triple form a number system or not.*

Definition 3. *For a given (Λ, M, D) the* classification problem *means finding all cycles (*witnesses*).*

Definition 4. *For a given (Λ, M, D) the* parametrization problem *means finding parametrized families of number systems.*

Definition 5. *For a given (Λ, M, D) the* construction problem *aims at constructing a digit set D to M for which (Λ, M, D) is a number system. In general, construct a digit set D to M such that (Λ, M, D) satisfies a given signature.*

The algorithmic complexity of the decision and classification problems are still unknown.

The toolset

To support the theoretical research of the area, we built a Python based toolset that helps the investigations and experiments. The toolset implements the base features of a number systems, like addition, multiplication with any matrix based system. It gives multiple possibilities to solve a decision or a classification problem, starting with a simple brute force, to probabilistic solutions.

To speed up the solving of these problems we can initiate optimization algorithms, that rotates the lattice into a more optimal state, where we can solve the problems faster. We can visualize the expansion of the numbers in system to understand the deeper structure, like we can see on the Figure 1.

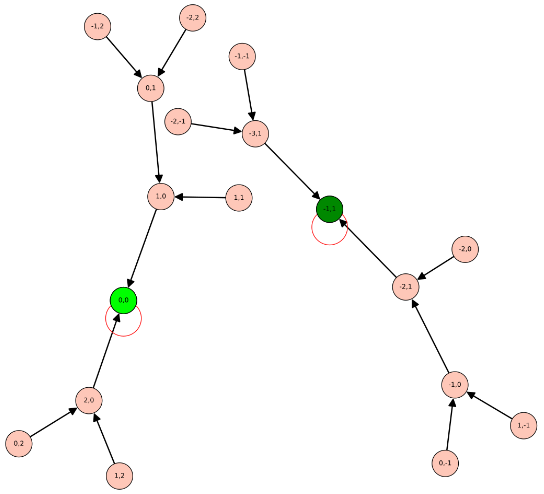


Figure 1: An expansion graph of a non-number system candidate, drawn by the toolset

The area has a plenty of unsolved problems in general, but most of the problems have solutions for a specific forms of number systems. To test conjectures of specific forms it is necessary to be able to collect candidates, filter them and organize the results. Therefore we implemented a server-side application which is able to store various data on number system candidates. The database already contains more than 10 000 candidates. We uploaded cases by generating companions of expansive polynomials with constant terms ±2,±3,±5, ±7). We used canonical, shifted canonical, symmetric digit sets as well and we calculated many combinations of product systems.

The data server allows to read data from the server publicly via a JSON API and the registered users with own API token can send new properties to the database. The candidates can be filtered by any custom property.

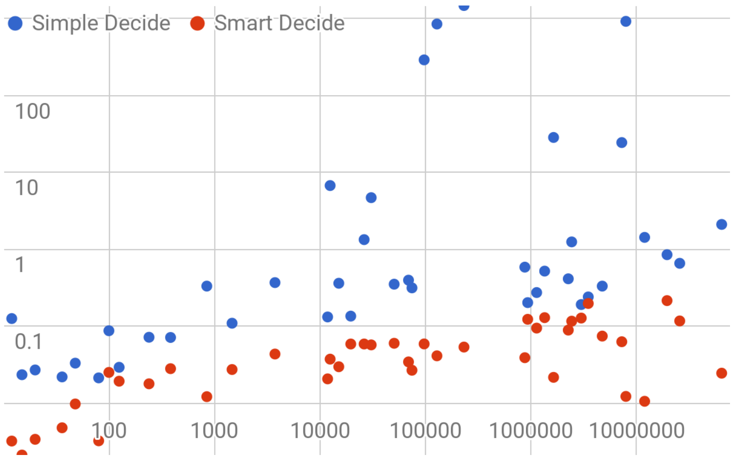


Figure 2: Runtime comparison of the simple decide method and the smart one by the size of the space where the witnesses can be found

The server already stores plenty of properties, like eigenvalues, eigenvectors, periodic points and orbits, classification details, etc. In this talk we present an introductory usage of the toolset.

Experimental observations

We investigated many candidates and filled up the publicly available database. With the aid of the database any researcher can easily check his/her conjecture.

Using the toolset we found out multiple conjectures. We generalized a previous conjectures that says if the absolute value of the contant term of the polynom is 2, then we can find a witness that shows if it is a number system or not. Our conjecture is that we can find a witness to counterproove the number system property of a candidate within a constant term absolute value sized box.

In 2-dimensional product system cases we found a connection between the eigenvector directions and the number system property. As we can see in Figure 3, if the eigenvalues can be enclosed in a π degree max, then the product will be a number system as well.

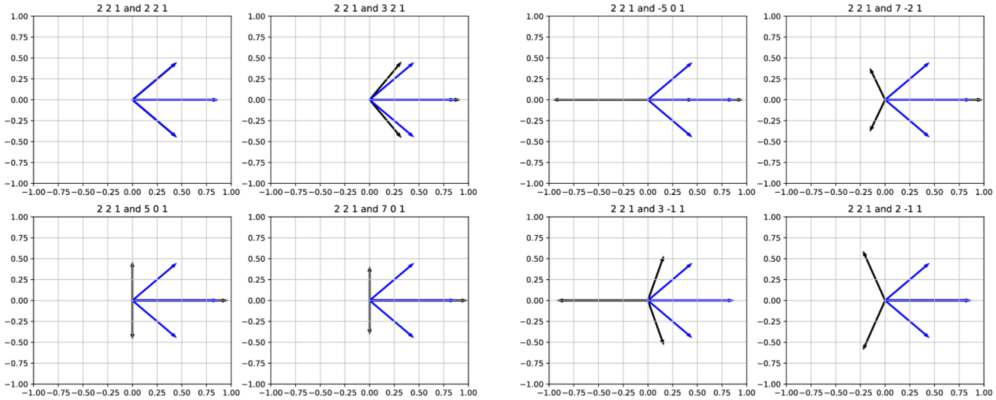


Figure 3: 2-dimensional examples of base matrix eigenvalues in GNS or non-GNS cases. Eigenvalues visualized on the complex plane. The candidates are denoted by the polynomials that generated them.

References

[1] Kovács, A., *On computation of attractors for invertible expanding linear operators in Z^k* , Publ. Math. Debrecen, **56**/1–2, (2000), 97–120.

[2] Kovács, A., *On number expansions in lattices*, Math. and Comp. Modelling, **38**, (2003), 909–915.

[3] Burcsi, P., Kovács, A., Papp-Varga, Zs., *Decision and Classification Algorithms for Generalized Number Systems*, Annales Univ. Sci. Budapest, Sect. Comp., **28**, (2008), 141–156.

[4] Hudoba, P., Kovács, A., *Some improvements on number expansion computations*, Annales Univ. Sci. Budapest, Sect. Comp., **46**, (2017), 81–96.

Geometric Distance Fields of Plane Curves

Róbert Bán and Gábor Valasek

Abstract: This paper introduces a geometric generalization of signed distance fields for plane curves. We propose to store simplified geometric proxies to the curve at every sample. These proxies are constructed based on the differential geometric quantities of the represented curve and are used for queries such as closest point and distance calculations. We investigate the theoretical approximation order of these constructs and provide empirical comparisons between geometric and algebraic distance fields of higher order. We apply our results to font representation and rendering.

Keywords: Computer Graphics, Signed Distance Fields, Plane Curves

Introduction

Signed distance functions are a versatile implicit representation of shapes that possess important practical advantages over standard implicit expressions [3, 4, 5]. Formally, they map a signed distance to every point in space, i.e. $f : \mathbb{E}^n \to \mathbb{R}$ is a signed distance function of a $F \subset \mathbb{E}^n$ volume in the n dimensional Euclidean space $\mathbb{E}^n$ iff

$$f(\boldsymbol{x}) = \begin{cases} -d(\boldsymbol{x}, \partial F) & \text{if } \boldsymbol{x} \in F \\ d(\boldsymbol{x}, \partial F) & \text{if } \boldsymbol{x} \notin F \end{cases}$$

where $d(\boldsymbol{x}, F) = \inf_{\boldsymbol{y} \in F} \|\boldsymbol{y} - \boldsymbol{x}\|_2$. Despite their attractive properties, factoring free-form geometries in terms of signed distance functions is not tractable in closed form in general; the signed distance function of a complex scene is only available procedurally. This poses difficulties in their real-time applicability.

Signed distance fields overcome this difficulty by sampling the signed distance function and using a reconstruction filter to compute a local approximation to the actual signed distance for all points in space. Performance concerns usually limit this filtering to bi- and trilinear filtering for planar and spatial distance fields, respectively. Signed distance fields saw a wide adaptation in high quality font rendering [1] and they are used in high performance game engines for effects such as soft shadows and ambient occlusion [6, 7].

Our paper introduces a higher order geometric generalization of signed distance fields. Instead of storing an algebraic approximation to the signed distance function at every sample, we propose the use of a geometric approximation to the local geometry. We show that our geometric approach is equivalent to the higher order approximation of the signed distance function itself but at a reduced storage cost.

Theoretical background

Algebraic distance fields

Algebraic distance fields are generalizations of distance fields. Instead of storing the distance value at every sample, we store a polynomial approximation of the distance function around the sample point.

First, let us show that the distance samples of an arbitrary distance field can be considered as zero order, i.e. a constant Taylor approximations to the distance field, as presented in [2].

Let $\alpha = (\alpha_1, \ldots, \alpha_n)$ denote a multi-index where $|\alpha| = \alpha_1 + \cdots + \alpha_n$, $\boldsymbol{x}^\alpha = x_1^{\alpha_1} \cdot \ldots \cdot x_n^{\alpha_n}$, $\partial^{\boldsymbol{\alpha}} f = \partial_1^{\alpha_1} \ldots \partial_n^{\alpha_n} f$, $\boldsymbol{\alpha}! = \alpha_1! \cdot \ldots \cdot \alpha_n!$. The degree k multivariate Taylor polynomial about $\boldsymbol{x}_0$ is

$$T_{k,\boldsymbol{x}_0}(\boldsymbol{x}) = \sum_{|\alpha| \leq k} \frac{\partial^{\boldsymbol{\alpha}} f(\boldsymbol{x}_0)}{\boldsymbol{\alpha}!} (\boldsymbol{x} - \boldsymbol{x}_0)^{\boldsymbol{\alpha}} \ .$$

The degree 0 Taylor-polynomial is the value of the function itself: $T_{0,\boldsymbol{x}_0}(\boldsymbol{x}) = f(\boldsymbol{x}_0)$, that is indeed, the classical distance fields are order 0 algebraic distance fields. By the *order of a distance field* we refer to the highest order derivative of the signed distance function incorporated into the distance field.

In higher orders, we store the coefficients of the polynomial Taylor approximation. At degree 1, the polynomial $ax + by + c$ is represented by the (a, b, c) tuple of coefficients in power basis. This linear polynomial implicitly describes a line. To express its power basis coefficients with the derivatives of the signed distance function $f : \mathbb{E}^2 \to \mathbb{R}$, let $\boldsymbol{x} = (x, y)^T \in \mathbb{E}^2$, $\boldsymbol{x}_0 = (x_0, y_0)^T \in \mathbb{E}^2$, then

$$\begin{aligned} T_{1,\boldsymbol{x}_0}(\boldsymbol{x}) &= f(\boldsymbol{x}_0) + \partial_x f(\boldsymbol{x}_0)(x - x_0) + \partial_y f(\boldsymbol{x}_0)(y - y_0) = \\ &\underbrace{\partial_x f(\boldsymbol{x}_0)}_{a} \cdot x + \underbrace{\partial_y f(\boldsymbol{x}_0)}_{b} \cdot y + \underbrace{f(\boldsymbol{x}_0) - \partial_x f(\boldsymbol{x}_0) \cdot x_0 - \partial_y f(\boldsymbol{x}_0) \cdot y_0}_{c} \end{aligned}$$

Note that the constant term c is not simply the function value $f(\boldsymbol{x}_0)$, but a translated value.

Similarly, the coefficients of the second degree polynomial $ax^2 + by^2 + cxy + dx + ey + g$ are expressed by the first $(\partial_x f, \partial_y f)$ and second order $(\partial_{xx} f, \partial_{xy} f, \partial_{yy} f)$ derivatives of the signed distance function. This means that we have to store six coefficients for every second order algebraic distance field sample.

By increasing the per sample Taylor approximation order, we can increase the accuracy of the distance field. However, as the order increases, we need to store more coefficients per sample: an order n Taylor approximation uses the derivatives of the signed distance function up to order n. As such, it requires $\binom{n+2}{n}$ coefficients in two variables.

To counter this coefficient explosion, we propose the use of geometric distance fields.

Geometric distance fields

For our proposed geometric distance fields, we store different geometric proxies that approximate the local differential geometry of the represented shape at the closest point to the sample position.

The main theoretical insight of this paper is that the order n geometric contact of surfaces guarantees that the derivatives of the signed distance functions of said geometries coincide at the position of the join.

In order 0, the geometric proxy is the foot point, which is the closest point on the curve. Compared to the 0th order algebraic distance field, a geometric sample contains two scalars instead of one. The foot point can be encoded either in a global coordinate system or relative to the sample point – a vector from the sample point to the foot point. This storage method doesn't contain the sign of the distance field, and thus it is less useful than the other constructions.

The first order geometric proxies to a plane curve are its tangent lines. For every first order sample we store the tangent line of the foot point. We investigate several possible representations of this tangent line, as well as the half-spaces that these tangent lines encode in case of signed distance functions. We evaluate these in terms of expressive power and construction.

The second order geometric proxy is the osculating circle of the plane curve. This circle is tangent to the curve at the foot point and it has the same curvature (κ). A circle – as seen with the tangent line – can be represented in many ways. We carry out the same evaluation of these various representations as with the tangent lines.

Conclusion

This paper generalizes classic planar signed distance fields to higher order algebraic distance fields. We show that the increased accuracy comes at a cost of coefficient increase. We propose geometric distance fields to retain accuracy at a decreased per-sample coefficient requirement.

We compare these constructs in terms of theoretical approximation properties. We propose practical construction algorithms for said fields and also carry out empirical comparisons between algebraic and geometric distance fields.

Acknowledgement

EFOP-3.6.3-VEKOP-16-2017-00001: Talent Management in Autonomous Vehicle Control Technologies – The Project is supported by the Hungarian Government and co-financed by the European Social Fund.

References

[1] Chris Green. *Improved alpha-tested magnification for vector textures and special effects*. In ACM SIGGRAPH 2007 courses, 9–18, 2007.

[2] Róbert Bán and Gábor Valasek. *First Order Signed Distance Fields*. Eurographics 2020 - Short Papers 33–36, Wilkie, Alexander and Banterle, Francesco (Ed.) The Eurographics Association, 2020.

[3] John C. Hart. *Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces.* The Visual Computer, 12:527–545, 1994.

[4] Bernhardt, A., Barthe, L., Cani, M.-P. and Wyvill, B. *Implicit Blending Revisited*. Computer Graphics Forum, 29:367–375, 2010.

[5] Benjamin Keinert, Henry Schäfer, Johann Korndörfer, Urs Ganse, and Marc Stamminger. *Enhanced Sphere Tracing*. In Smart Tools and Apps for Graphics, Giachetti A., (Ed.) The Eurographics Association, 2014.

[6] Daniel Wright. *Dynamic Occlusion with Signed Distance Fields*. Advances In Real-Time Rendering, SIGGRAPH 2015. http://advances.realtimerendering.com/s2015/DynamicOcclusionWithSignedDistanceFields.pdf . Retrieved 26. 03. 2020.

[7] Róbert Bán, Csaba Bálint, Gábor Valasek. *Area Lights in Signed Distance Function Scenes.* EG 2019 - Short Papers, Cignoni P., Miguel E. (Ed.) The Eurographics Association, 2019.

Towards Rootkit Detection on Embedded IoT Devices

Roland Nagy and Levente Buttyán

Abstract: Rootkits are malicious programs that try to maintain their presence on infected computers while remaining invisible. They have been used to attack traditional computers (PCs and servers), but they may also target embedded IoT devices. In this work, we propose a rootkit detection approach for such embedded IoT devices, where the detection mechanism is executed in an isolated execution environment that protects it from manipulation by the rootkit. Our rootkit detection approach is focused on detecting Direct Kernel Object Manipulation (DKOM) and it is based on detecting inconsistencies caused by the presence of a rootkit in various Linux kernel data structures such as the process list, the process tree, and different scheduling queues. We also report on the current status of our implementation using OP-TEE, an open source Trusted Execution Environment.

Keywords: Embedded Systems, Internet of Things, Security, Malware

Introduction

Connecting embedded devices to the Internet (a.k.a. the Internet of Things or shortly IoT) enables new applications such as smart homes, intelligent transportation systems, and personalized healthcare. However, many of these new applications have stringent security and privacy requirements. Unfortunately, IoT systems are notoriously insecure. One of the reasons is that IoT devices are rather easy to compromise by exploiting weaknesses in the way they are operated and vulnerabilities of the software components running on them. A consequence of this is that malware for IoT has appeared [1, 4] and gaining momentum [5].

Sophisticated malware tries to maintain its presence on infected devices while remaining invisible for the operators of those devices. This sort of malware is called *rootkit*. Typically, rootkits run with elevated (root) privileges and they modify system commands and/or low level data structures in the operating system (OS) kernel such that their files and running processes do not appear in the output of various system tools used to monitor the operation of the devices. Detecting such a rootkit is challenging, mainly because any detection program running at the same or lower privilege levels than the rootkit may also be compromised or may be misled by the tricks used by the rootkit to hide itself.

In this work, we aim at rootkit detection on embedded IoT devices, and we address the above challenge by running our rootkit detection tool in a Trusted Execution Environment (TEE), which is isolated from the main OS of the device, and hence the rootkit – even running with root privileges on the main OS – cannot interfere with its operation. In this extended abstract, we introduce the concept of TEE and describe how our rootkit detection tool running in the TEE detects active rootkits.

Trusted Execution Environments

A TEE is an execution environment which is isolated from the main OS and applications running on the device (the so called Rich Execution Environment or shortly REE) by software and hardware mechanisms (e.g., based on the ARM TrustZone[1] technology). Within the TEE, trusted applications (TAs) run on top of a trusted OS. The isolation mechanisms ensure, that system resources (e.g., memory) of the REE can be accessed from the TEE, but not vice versa. Thus, secrets (e.g., keys) can be kept and critical computations can be executed within the TEE

[1] https://developer.arm.com/ip-products/security-ip/trustzone

without the risk of being observed or manipulated by potentially malicious software running in the REE.

Our thesis is that an active rootkit must introduce inconsistencies in the data structures of the main OS kernel, since it must remove its own processes from some data structures used by system monitoring tools in order to maintain stealthiness, while it must keep its own processes in other data structures for being eventually scheduled and executed. Hence, our rootkit detection approach is based on detecting inconsistencies in OS kernel data structures by a TA running in the TEE. For this, our TA needs to access the memory of the main OS running in the REE.

As a TEE implementation, we use OP-TEE[2], which is an open source TEE, compliant with a widely accepted standard[3] for TEEs. In OP-TEE, by default, simple TAs are not capable for accessing the memory regions of the REE; such an access requires a so called Pseudo-TA (PTA). Our PTA is running with the privileges of the trusted OS kernel, and we can instruct this kernel to map the memory regions used by the main OS (typically Linux on embedded devices) in the REE, such that our PTA can access them.

Rootkit detection

Rootkits use different cloaking mechanisms to hide their presence on infected systems. A simple idea, for instance, is to hide something by corrupting the tool used for gathering information about it. On Linux, an example would be modifying the `ps` program such that it does not list some specific processes. This type of attack can be easily detected by verifying the integrity of important system programs, which we do not discuss here.

In this work, we focus on rootkits that use Direct Kernel Object Manipulation (DKOM) [3]. Such rootkits modify the underlying data structures that the kernel uses to maintain information about its specific components. For instance, if one can determine what data structure is used to populate the `/proc` virtual filesystem on Linux, then he may be able to remove a specific process from that data structure, which will then remain hidden from the `ps` command.

As, in the IoT domain, the main OS running on embedded devices is often Linux, we describe some relevant Linux kernel data structures that may be manipulated by DKOM:

task_struct: Inside the Linux kernel, this structure holds most of the information associated with processes. Internally, tasks are equivalent to threads, and any process may have several threads. Tasks of the same process share one virtual address space and many more resources.

Process list: The task structures inside the kernel memory are chained into a doubly linked circular list. In previous kernel versions, the kernel iterated through this list to populate the `/proc` directory.

Process tree: Processes are related to each other via a parent-child relationship. Every process has a parent that created it, and processes might start other processes that become their children. The `task\_struct` holds a pointer to the parent of the given task, a list of pointers for its children, and another list of pointers for its siblings.

Pid namespace, IDR and the struct pid: Each namespace maintains a radix tree[4], containing pointers to pid structures[5]. These structures have lists of pointers for the tasks using them. This data structure is responsible for accounting for taken pids and for fast access

[2] https://www.op-tee.org

[3] https://globalplatform.org

[4] https://lwn.net/Articles/175432/

[5] https://lwn.net/Articles/195627/

to tasks via their pids[6]. In recent kernel versions, this mechanism populates the `/proc` directory.

Run queues: Each CPU has a runqueue structure, holding inline structures for the available schedulers. These have their own methods to keep track of runnable processes. The CFS and DL schedulers are using red-black trees[7] for this, while the real-time scheduler has a so-called `rt\_prio\_array`; a bitmap and an array of lists for each priority level[8].

Wait queues: Every time a process must wait for something, it is placed in a waitqueue[9], containing wait entries. Each such entry has a pointer to the task waiting, and to a function to execute when it is time to wake up the task.

The basic idea of our rootkit detection mechanism is simple: We iterate through the previously mentioned data structures, collect pids into different lists, and look for inconsistencies in the obtained lists. For instance, it is abnormal if a pid can be found in the process tree, but it is missing from the process list.

Implementation

Our rootkit detection solution is implemented in an OP-TEE TA, which uses a PTA to access the Linux kernel's memory in the REE as we mentioned before. In order to be able to use types and structures of the Linux kernel, we generated header files from the DWARF section of a dummy kernel module with the `dwarfparse` script[10]. In addition, we were able to retrieve useful addresses from the `System.map` file of the compiled kernel, with which we were able to locate the data structures described in the previous section.

So far, we managed to implement the following: We can request a copy of the `init` task, from which we can iterate through the list of all processes using the process list. We save the pid of each task found in the process list into an arraylist. Then, from the `init` task again, we run a depth-first search on the process tree, saving the pids found there into a separate arraylist. For each element of these lists, we look for that pid in the other list, and if not found, we save it to yet another list. If this differential list is not empty, then we found suspicious processes that appear in one of the kernel data structures but missing from the other one. After this check, a unified list is created from the first two collections of pids, and we check if all the pids found in the schedulers of all the CPUs are also a part of this list.

We tested our implementation with a simple rootkit, which creates a bind shell and removes it from the process list. We detected the rootkit by identifying a pid in the process tree that was missing from the process list.

Conclusion and future work

Rootkits are malicious programs that try to maintain their presence on infected devices while remaining invisible for the operators of those devices. They have been used to attack traditional computers (PCs and servers), but they may also target embedded IoT devices. In this work, we proposed a rootkit detection approach for such embedded IoT devices, where the detection mechanism is executed in a TEE, which protects it from manipulation by the rootkit running in the REE. Current technologies (e.g., ARM TrustZone) supports the implementation

[6] https://lore.kernel.org/patchwork/patch/834401/
[7] https://lwn.net/Articles/184495/
[8] https://www.linuxjournal.com/article/10165
[9] https://lwn.net/Articles/577370/
[10] https://github.com/realmoriss/dwarfparse

of TEEs on embedded devices, hence, our approach does not rely on far fetched assumptions, but can be readily used even today on commodity embedded boards.

The work presented here is work-in-progress. We are currently experimenting with iterating through the IDR of the initial pid namespace and collecting pids from waitqueues. We also plan to extend the functionality of our detection tool with features beyond DKOM, and we would like to cover kernel resources other than processes. Finally, we would like to test our implementation against real rootkits captured in the wild.

Acknowledgment

The presented work was carried out within the SETIT Project (2018-1.2.1-NKP-2018-00004), which has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2018-1.2.1-NKP funding scheme.

References

[1] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou. Understanding the Mirai botnet. In *USENIX Security Symposium*, August 2017.

[2] W. Blunden. *The Rootkit Arsenal: Escape and Evasion: Escape and Evasion in the Dark Corners of the System*. Jones & Bartlett Learning, 2009.

[3] J. Butler. DKOM – Direct Kernel Object Manipulation. In *BlakHat USA*, 2004.

[4] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin. Measurement and analysis of Hajime, a peer-to-peer IoT botnet. In *Network and Distributed Systems Security (NDSS) Symposium*, 2019.

[5] P.-A. Vervier and Y. Shen. Before toasters rise up: A view into the emerging IoT threat landscape. In *IoT Security Foundation Conference*, 2018.

Use data mining methods in quality measurement in the education systems

Sándor Balázs Domonkos and Németh Tamás

Abstract: Our basic problem is rooted in the educational systems, where measurement and evaluation of the pedagogical work and pedagogical developments is a must from year to year. These measurements can be used for the individuals to get information on which field they need to improve and can be used for rewarding systems. We get real data from 58 different schools from 2007-2016, from nearly 8000 educators. All these survey's ratings and other data got collected and processed to get in a usable form. The schools make statistics with the collected data every year, but the results depend on individuals, and because of that these have a lot of distortions for example personal dependencies or connections. To get a workaround of the "personal" dependencies of these statistics, we use the PageRank algorithm with the Comparability graph[12]. With the comparability graph, we could compare two attributes with each other not heavily depend on the persons who fill the survey.

Keywords: data mining, pagerank, comparability graph

Introduction

The main problem is raising from the education system quality and evaluation system about the educators. This system is really important for the schools, because only in this way they can monitor the educators' progression and other workers' qualities and working capabilities. These data that they collected via surveys are not just used to personal evaluation and ratings, but to get a view of the workload for each individual, for example, it also can help to make a better timetable for everyone with this data. So we examined the data and the statistical style old system and we got the conclusion that the statistics-based system is can be very noisy and heavily depends on the person to person connections and depends on a considerable luck factor for the drawing system.

Examination and explanation

So to take an overview of the whole process and the problems, in the beginning, the schools use a default survey system with given attributes, and the evaluator teachers can give 1-5 scores to every attribute. There is a drawing for every teacher, everyone gets a few evaluator teachers but not the same ones - and this is the main problem. Because if Teacher A is lucky and Teacher B is unlucky at the drawing system, that not so life like thing/distosion can happen that Teacher A only gets higher point giver evaluators and get only good points for every attribute, and Teacher B get unlucky and get only strict evaluators and only get lower points for everything. One mor factor that is complicate and invalidate the statistical results, that things can happens if Teacher A get X amount of evaluators and Teacher B get a lower amount, that can distort the year to year and teacher to teacher comparsions.

Graph Based Solution

After we researched the other researcher articles that mentioned before and these other articles [1] [2] [3] [5] [6] [7]. We decide that Our solution for this problem will based on graphs and a combination of pagerank algorithm. We analize the raw datas try out different methods and learning from the other articles [4] to break up the datas for different viewpoints and build different types of graphs. After a lots of experiencing we decide to use Comparability graphs for every attributes. First of all we need to break up the datas for 3 parts: evaluating lecturer, attributes for the evaluated lecturer, evaluated lecturer. So in this way we can construct comparability graphs for every attributes. In this way 2 evaluation can be compared if they are from the same evaluator lecturer and for the same attributes, in this way the data noise is dampening down,because its compare data from the same evaluator. From these datas we construct Comparability graphs for every attributes. As the first figures show it is building the Comparability graph for a single attributes after we build a simple graph where the edges contains the given evaluator values and we paired them up to each evaluator lecturer with each evaluated lecturer as you can see in the figure. After that we build up the Comparability graphs in this way, for example Lecturer1 get 3 points and Lecturer2 got 4 points from the same nice evaluator so they can be compared so on the Comparability graph they got an edge labeled 1 from Lecturer1 to Lecturer2 so they can be compared because they get evaluated by the same evaluator for the same attributes. The next step Lecturer2 is compared with Lecturer3 when they get numbers from a Strict Evaluator, Lecturer2 gets 3 and Lecturer3 gets 1. So these two values are can be compared again on the Comparability attributes graph and the edge is pointing to Lecturer2 from Lecturer3 and get a label 2 that means Lecturer2 is "bette" here with 2 points. In these values we put them on the edges, and the edges are directed.

On these new graphs we can use the PageRank algorithm as mentioned before [8][9][10], to get quantified results for every evaluated lecturer for every attribute without the distortions of the personal conflicts or any other type of distortions like the luck factor of the drawing system mentioned above. We modify the PageRank algorithm a little bit that when building up the values the modified PageRank algorithm can be set to take into account the edge values of the graphs, not only the incoming and outgoing edge numbers and the importance of them. In this way, we can use the plus information that we generated with the comparability, graph to compare the teachers' attributes for each other only if they have a common evaluator teacher. The modified PageRank algorithm gives more data about the evaluating lecturers too for each attribute after the modified PageRank is done these values are just more well adjusted for every attributes for every evaluated lecturer. For example on the old-style method if someone is getting a few bad results because of the drawing system luck factor or any personal issues, these bad ratings can really pull down the person numbers, but with our method the comparability graph for attributes processed with the modified PageRank methods these bad ratings can ease down the input data, cause the algorithm will ease out if someone gives only bad points for an individual and good points for everyone else or give bad points for everyone because it is a strict evaluator or give only good points for everyone these evaluators can only be compared on the teacher attributes level if they have a common teacher evaluated, so in this way we double defend against the people and the luck factor. Because the comparability graph generation only compares the teachers with common evaluators for common attributes, and the modified PageRank was taken into account these edge values too that we generated in the comparability graph between the nodes.

My Page Rank algorithm

Input Analog paper type survey coming from evaluator teachers year to year Output Digital and comparable digital data for every evaluated teacher for every year.

1. Collect the analog data from the schools

2. Group the datas by teacher and secondary by year

3. Make the first simple graph for evaluator to evaluated teacher attributes

4. Use the simple graphs to build up the Comparability Graph for every attributes with the edges contains the compare values numerically

5. On the Comparability Graph for every attributes for every teacher we can run the modified page ranking algorithm that consider the edge values too to calculate the correct values. The algorithm has a sub part that count the numericall values on the edges, like if there are multiple edges there.So if the edge value from A to B is 2, than the algorithm count it as 2 piece of A to B edge, like on value 3 it counts as 3 piece of A to B edge. This sub part everytime run down and give the simulated data back to the original page rank algorithm.

6. We get the output for every teacher for every attributes that can be compared and these values doesn't distorted by the luck factor of the drawing system.

Results

The quantified values are less likely to depend on the evaluator person and much more like depends on the evaluated person cause the comparability graph and modified PageRank method helps us to get the right value for every teacher and defend against the drawing system luck and personal factor distortions. This method the year to year improvements are taking shape in a better way and more readable way and more lifelike. This way we not just only get the results for the evaluated people attributes it can work back way too, we can get data from the evaluators too. So we can make some conclusions in the lecturers have a personal or work-related problem with each other. If the results give back significant value distortions to A teacher to B teacher versus A teacher to all other teachers, this can give a warning sign. For results we got a much lower dispersion level and with a lower dispersion and very low fluctuation level we can much easier fit line to yearly improvement attributes too. These numbers are much more like living in a mathematical way too because it is not life-like if teacher performance is going down for half of them to a year to next year or go up by double. On figure 10, for example, we can see that from 2009 to 2010 there is a big jump in statistical data and that wasn't very acceptable and that was clearly a statistical method failure by data distortions. The average dispersion level for the results can be 30-50 percent better than the statistics ones summarized for the whole school level, and these data not just look better but checked by persons at the schools and they confirm that these data are much more like real life and give a better understanding.

Future plans

In the future we want to spread these methods with these data we gathered and processed by successfully, on the student level as well.

For example on the student level, they need to be a better compare system as another article mentioned before to use PageRank methods on the values too. But with our method, we can use the processed comparability graph values to give another comparability graphs for student-level where we can only compare students where they get the same teacher and with similar attributes. On the evaluator's side, we can get extra pieces of information about the strictness of the teachers too. This info also can get into account when we want to compare student grade results. So widely we try to make a better system to follow the students' progress on each subject and not only based on the grades. Above all we try to make a manual override for the algorithm that we can give importance for certain evaluators for example principals or HR managers that they values can be multiplied and counted in with more weight on it.

Acknowledgements

This work was supported by Enaplo and snw systems.

References

[1] C. Romero, S. Ventura, M. Pechenizkiy and R. Baker, Ryan, Handbook of educational data mining, CRC Press, 2011.

[2] C. Romero and S. Ventura, Educational data mining: A survey from1995 to 2005 , Expert systems with applications volume 33 no. 1 pp. 135-146 2007

[3] C. Heiner, N. Heffernan, and T. Barnes, Educational data mining , In Supplementary of Proceedings of the 12th International Conference of Artificial Intelligence

[4] Fiala, D., Rousselot, F., and Jezek, K., Pagerank for bibliographic networks. Scientometrics 76, 1 (2008)

[5] Isaac, Stephen, and William B. Michael.,Handbook in research and evaluation: A collection of principles, methods, and strategies useful in the planning, design, and evaluation of studies in education and the behavioral sciences. Edits publishers, 1995.

[6] Kuzmanovic, Marija, Gordana Savic, Milena Popovic, and Milan Martic.,"A new approach to evaluation of university teaching considering heterogeneity of students preferences." Higher Education 66, no. 2 (2013): 153-171.

[7] Braga, Michela, Marco Paccagnella, and Michele Pellizzari., "Evaluating students evaluations of professors." Economics of Education Review 41 (2014): 71-88.

[8] Bar-Yossef, Z., and Mashiach, "Evaluating students evaluations of professors." Economics of Education Review 41 (2014): 71-88.

[9] Brin, S., and Page, L. The anatomy of a large-scale hypertextual web search engine. Computer networks and ISDN systems 30, 1 (1998)

[10] Chen, Y.-Y., Gan, Q., and Suel, T., Local methods for estimating pagerank values. In Proceedings of the 13th ACM International conference on Information and knowledge management (2004)

[11] A. London. A local PageRank algorithm for evaluating the importance of scientific articles ,*Annales Mathematicae et Informaticae* 44:131-140 2015.

[12] A. London. A local PageRank algorithm for evaluating the importance of scientific articles ,*Annales Mathematicae et Informaticae* 44:131-140 2015.

[13] S. Brin and L. Page, engine The anatomy of a large-scale hypertextual Web search , Computer networks and ISDN systems, vol. 30, no. 1, pp. 107-117, 1998

Feature Extraction from JavaScript

Tamás Aladics, Judit Jász and Rudolf Ferenc

Abstract: Source code analyzation is generally a challenging task and it is especially true for loosely typed languages like JavaScript. Traditionally analyzation is done by hand with the help of static analyzation tools which has many disadvantages - one of which is the lack of robustness. The recent advances in machine learning are promising to increase the robustness of source code analysis, however for ML models to work a meaningful and compatible representation is needed. We propose a specific way of extracting features of JavaScript source code based on it's underlying structure (AST) then we embed these features to a fixed length vector using Doc2Vec. Applying this method on a dataset of 150 000 Java Script source files we found this representation to be meaningful as the semantically similar AST nodes are grouped together after the embedding.

Keywords: JavaScript, Feature Assembler, Deep Learning, Doc2Vec

Introduction

Machine learning and recently deep learning has set foot in most subfields of computer science and source code analysis is no exception. Machine learning can be used by itself or as a supplementary tool for various tasks in this domain, for example code summarization, code completion and error detection. For this purpose it is important to find meaningful representation and attributes of the source code that can be fed to machine learning models as features.

This is a challenging task for various reasons. For example the lately highly successful natural language processing methods to process inputs are optimized for natural languages leaving the domain of artifical languages with potentially better working approaches. Another challenge is the fact that different programming languages may be better represented in different ways as they are highly diverse in their sytax and semantics. Loosely typed languages are especially challenging in this regard.

In this work we propose a method to extract meaningful features from source code focusing on JavaScript methods and functions. We took advantage of the strictly structured nature of programming languages and used source code's underlying tree repesentation (Abstract Syntax Tree or AST) to derive features. This involved embedding each function's AST to variable length vectors in ways that try to preserve the graph's structure.

Using this method to produce the input sentences we trained a Doc2Vec embedding [1] on a database containing 150 000 JavaScript source files [2]. In the end we find that meaningful representation is indeed apparent as in the resulting embedding space the semantically related nodes are close to each other.

Related

In the field of source code analysis different methods have been proposed to find ways to represent source code digestible by machine learning models. One way to categorize these approaches is their granuality, as it is done by Zimin Chen et al. [3] which we use to present the related work in this topic.

Various attempts have been made to extract features based on the tokens that build up the source code. Harer et al. [4] uses these tokens as inputs for Word2Vec to generate word embedding for C/C++ tokens for software vulnerability prediction. Chen & Monperrus [5] use Word2Vec to generate java token embedding for finding the correct ingredient in automated program repair . They use cosine similarity on the embeddings to compute a distance between pieces of code.

Other approaches are based on embedding functions or methods, which is not as fine grained as representations based on tokens. Devlin et al. [6] use function embedding as input to repair variable misuse in Python. They encode the function AST by doing a depth first traversal and create an embedding by concatenating the absolute position of the node, the type of node, the relationship between the node and its parent and the string label of the node. DeFreez et al. [7] generate function embeddings for C code using control-flow graphs. They perform a random walk on interprocedural paths in the program, and used the paths to generate function embeddings. The embedding is used for detecting function clones.

In the realm of JavaScript source analyzation a related work is done by Theeten et al. [8] who presented Import2Vec, an embedding of software libraries for (among other languages) JavaScript. They create vectors based on import statements to indicate similarity between software libraries. A lot of other methods to tackle source code analysis are based on metrics. Gregor Richards et al. [9] uses various (both general and dynamic language related) metrics to analyse the dynamic behaviour of JavaScript.

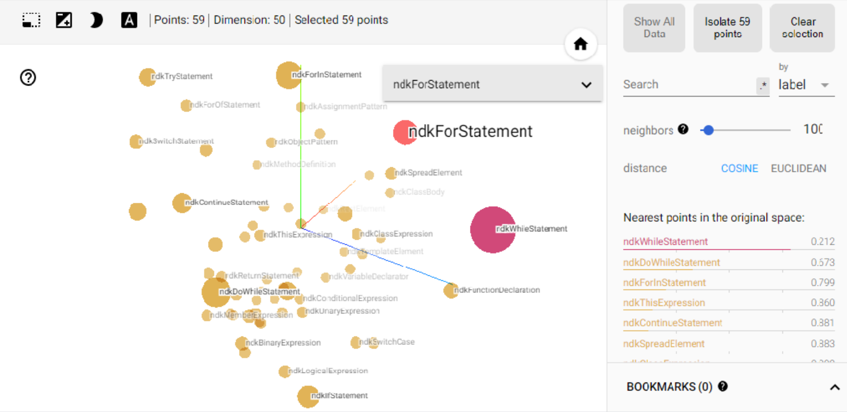
Taking the work of Zimin Chen et al. [3] as basis, numerous papers have been published in the field of feature extraction but only a small subset of these use AST as we propose. Also, most of the publications are focused around Java, C/C++ or Python; the publications involving JavaScript are very limited in quantity. This small number of publication for JavaScript are solely based on metrics and static analysis tools [9, 10], or they are problem specific [8]. Our method tries to specialize for JavaScript (as we took extra steps to take care of the heavy use of anonymous functions, which is not that elemental in the more object oriented languages) but still remain general.

Methodology and results

The first step of deriving important features of source code was to get the underlying structure of it. For this we used SourceMeter, an open source static analysis framework [11]. Analyzing with SourceMeter generates various results and one of them is an AST representation which we used. However, AST in itself is not a compatible format for machine learning algorithms as it is a graph.

To map each AST to a form that is usable by ML algorithms we flattened them to AST node sequences with the use of SourceMeter tools. Constructing the sequence consists of traversing the AST, getting each node's kind (ie. identifier, assignment, for loop etc.) then outputting these types. Furthermore, we needed to find a way to take the overall structure of the code into account so that *if(cond) exp1; expr2;* and *if(cond) expr1; expr2;* will not be mapped to the same node sequence. For this end we introduced a separate identifier that is inserted into the node sequence everytime the scope is changed. With this extra step we tried to ensure that the resulting vector will be specific enough to reflect the structure but still remain general and simple. Another note be added that JavaScript has heavy use of nested functions. We took the path of only flattening the most outer function, the inner functions are flattened in place and are not added as separate node sequences.

After flattening the AST we acquired a node sequence, each element in the sequence is an identifier of a node kind. However, these sequences hold no semantic information yet. We used Doc2Vec [1] to map these node sequences into meaningful representations in form of vectors. Doc2Vec in general is an extension of Word2Vec [12], and it's purpose is to generate vectors for documents and for the words that build the documents up, using a specific mapping. This mapping's objective is to map semantically similar words and documents to vectors whose distance is minimal for words/documents that are semantically similar. There are two main approaches for this mapping: PV-DM and PV-DBOW. Both algorithms use a sliding windows (context) around each word (center), and a paragraph vector for each document. The difference is in the way of vector generation: PV-DM adjusts vectors so that context words and the



Points: 59 | Dimension: 50 | Selected 59 points

ndkForStatement

Search .* by label

neighbors 100

distance COSINE EUCLIDEAN

Nearest points in the original space:

ndkWhileStatement	0.212
ndkDoWhileStatement	0.573
ndkForInStatement	0.799
ndkThisExpression	0.360
ndkContinueStatement	0.381
ndkSpreadElement	0.383

Figure 1: For Statement in the dimension reduced embedding space

paragraph vector predict the center word. PV-DBOW tries to do the opposite in a generative manner: it calibrates vectors so that the central word generates the context words and the paragraph vector. In the end both PV-DM and PV-DBOW finds an embedding where word vectors will hold semantic information on word level, and document (or paragraph) vectors will hold information on document level.

The embedding must be trained on a corpus that is large enough. We used a dataset made up of 150 000 JavaScript source codes from more than 9300 projects [2]. This codebase is promising to be diverse enough: it contains projects that consist of only a few configuration files to complex systems. Using SourceMeter we generated the ASTs for the whole code base, then on each AST we ran our AST flattener which resulted in 597 074 functions flattened into node kind sequenes.

In our work, using the Doc2Vec lingo the "documents" are the node kind sequences (that corresponds to a function), and the "words" are the node kinds in them. We used the genism [13] library's PV-DM algorithm implementation to generate the embedding, using the vector size of 50, windows size of 10 with 8 epochs over the 597074 functions acquired from the 150k JavaScript dataset.

We found that using the described AST Flattening on a function level then utilizing Doc2Vec to get the embedding for the vectors resulted in a meaningful representation of the source code. To illustrate these results we used Tensorboard, Tensorflow's profiler and visualization toolkit, which has built-in ways to present higher dimension vectors in 3D, in our case we used PCA to do the dimensionality reduction. For exmaple, on Figure 1 we can see that the node kind corresponding to *for loops* is most similar to loops and other control flow statements.

Conclusions and future works

To concolude our work it is apparent that this way of representing JavaScipt source code can be promising as it preserves semantic information to an extent. However for it to be truly usable as part of an ML model it must be modified for the specific task: a database with fine tuned features and labels and different parameters can be tried for the Doc2Vec embedding to find the most optimal settings.

Some modification in the AST Flattener may also be benefical based on the task, for example the handling of nested function methods. In our work we flattened the inner methods in place, however using references and adding them as separate instances of methods could prove useful as this could potentially increase the number of available function methods at the cost of losing some structural information.

Another possible modification in future works could be to use different ways to learn the embeddings than Doc2Vec. Graph2Vec for example is a good candidate as it may be better at capturing the behaviour of the AST as it is a special graph.

Acknowledgements

The presented work was carried out within the SETIT Project (2018-1.2.1-NKP-2018-00004)[1].

References

[1] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[2] ETH Zürich. 150k Javascript Dataset. https://www.sri.inf.ethz.ch/js150.

[3] Zimin Chen and Martin Monperrus. A literature study of embeddings on source code. *CoRR*, abs/1904.03061, 2019.

[4] Jacob Harer, Louis Kim, Rebecca Russell, Onur Ozdemir, Leonard Kosta, Akshay Rangamani, Lei Hamilton, Gabriel Centeno, Jonathan Key, Paul Ellingwood, Marc McConley, Jeffrey Opper, Sang Chin, and Tomo Lazovich. Automated software vulnerability detection with machine learning. 02 2018.

[5] Zimin Chen and Martin Monperrus. The remarkable role of similarity in redundancy-based program repair. *CoRR*, abs/1811.05703, 2018.

[6] Jacob Devlin, Jonathan Uesato, Rishabh Singh, and Pushmeet Kohli. Semantic code repair using neuro-symbolic transformation networks. *CoRR*, abs/1710.11054, 2017.

[7] Daniel DeFreez, Aditya V. Thakur, and Cindy Rubio-González. Path-based function embedding and its application to specification mining. *CoRR*, abs/1802.07779, 2018.

[8] Bart Theeten, Frederik Vandeputte, and Tom Van Cutsem. Import2vec - learning embeddings for software libraries. *CoRR*, abs/1904.03990, 2019.

[9] Gregor Richards, Sylvain Lebresne, Brian Burg, and Jan Vitek. An analysis of the dynamic behavior of javascript programs. volume 45, pages 1–12, 05 2010.

[10] Wei-Hong WANG, Yin-Jun LV, Hui-Bing CHEN, and Zhao-Lin FANG. A static malicious javascript detection using svm. In *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering*. Atlantis Press, 2013/03.

[11] FrontEndArt Software Ltd. Sourcemeter. https://www.sourcemeter.com/.

[12] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[13] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

[1] Project no. 2018-1.2.1-NKP-2018-00004 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2018-1.2.1-NKP funding scheme.

Minimal solution for ellipse estimation from sphere projection using three contour points

Tekla Tóth and Levente Hajder

Abstract:

This paper introduces a minimal solution for ellipse estimation in images interpolating only three inlier points in the fitting algorithm. In classical methods, five contour points are required for this task. However, applying two additional constraints based on the properties of the special case because of the sphere projection, we can reduce the minimal size of the inlier pointset. Hence, the computation cost can be reduced, and the minimal method can be more robust against noise which gives more accurate result.

Keywords: Ellipse parameter estimation, Conic section detection in images, Geometric computer vision

Introduction

In computer vision, the camera matrix containing the intrinsic camera parameters defines the link between a 3D environment and the projected 2D camera image. Our main assumption is that if this intrinsic variables are known, a fitting algorithm requires fewer contour points in minimal case to estimate the parameters of the two-dimensional projected shapes. One benefit of this modified algorithm is that using them in e.g. RANSAC [3]-like robust algorithms need fewer iterations. Hence, the computational time is shorter. This concept can be tested against the existing fitting methods.

Our goal is to detect sphere projections in images. These conic sections require five points for the interpolation, and the spheres projections are usually ellipses in real-word images. In this paper, specific constraints are defined which come from the special properties of the perspective projection. These constraints are applied to the searched coefficients of the implicit ellipse representation. Our aim is to modify a minimal solver, of the general case, by exploiting these additional conditions to reduce the required point number.

Related work

There are many alternative methods available for solving ellipse fitting problems. These algorithms can be divided into two groups : (i) Hough transformation or (ii) edge following based approaches. The classical method using Hough transformation has a five dimensional parameter space, where the detection has very high time and memory demands [1]. A huge amount of scientific papers have shown that this problem can be overcome by using fitting methods processing some contour points of the ellipse. One of the fastest method is a direct fitting from Fitzgibbon et.al. [2]. Moreover, few other contour point based studies have focused on using four or three inliers; however, these points cannot be selected randomly and applied in a general case because of some geometric restrictions. On the other hand, our algorithm needs only three independently chosen points without any restriction.

The fitting procedures can be applied as the part of a robust sampling algorithm like RANSAC [3], where the minimally required inlier number is a critical point considering the iteration number as it is shown in Table 1. Hence, our main goal is to reach a minimal solution to this problem.

N	50%	65%	80%	90%	95%
1	7	11	20	44	90
3	35	106	573	4602	$\sim 10^4$
4	72	305	2875	$\sim 10^4$	$\sim 10^5$
5	146	875	$\sim 10^4$	$\sim 10^5$	$\sim 10^7$

Table 1: Iteration number required for RANSAC [3] algorithm if confidence is set to 99%. N denotes the number of sample points. A general ellipse needs 5 points. Using one or two constraints this number can be reduced to 4 or 3. Columns are varying w.r.t. outlier ratio from 50% to 95%. It can be observed that these cases are drastically faster when fewer sampling points are required.

Parameter constraints

This section introduces the theoretical background of our proposed method based on calibrated camera and sphere projection in the image.

Our model uses a calibrated pin-hole camera. Thus the intrinsic parameters are known. Let $\mathbf{K}$ denote the camera matrix. Its elements are as follows:

$$\mathbf{K} = \begin{bmatrix} s_u f & 0 & u_0 \\ 0 & s_v f & v_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{1}$$

where s_u, s_v, f, and location $[u_0 \quad v_0]^T$ are the horizontal and vertical sensor scales, the focal length and the principal point [4], respectively.

The general implicit equation of a conic section contains six parameters:

$$Au^2 + Buv + Cv^2 + Du + Ev + F = 0, \tag{2}$$

where u and v are the pixel coordinates in the image along x and y axes. The degree of freedom of ellipse is five, but in case of spherical projection, two constraints have to be fulfilled.The first one describes that the main axis of ellipse always intersects the centre of image. The second one defines a relation between the camera parameters and the ellipse coefficients.

Axial constraint

The first constraint C_1 can be defined as follows [7, 8, 5] :

$$C_1 : D(BD - AE) - 2E(BE - CD) = 0. \tag{3}$$

This is the formal definition of the fact that the main axis of the ellipse should intersect the principal point if the ellipse is the contour of a projected sphere. A general scenario is visualised in Figure 1. It can be observed that the ratio of the axes only depends on the relative distance of the ellipse from the principal point, and it is independent of the direction of the ellipse center position to the principal point.

Projection constraint

The second constraint C_2 can be defined as follows [5] :

$$C_2 : B(AE - BD)f^2 - E(DE - BF)(l^2 - 1) = 0, \tag{4}$$

where $l = 0$ if a pinhole camera model is considered and f is the focal length. This statement is the formal description of the connection between the ellipse coefficients and the intrinsic parameters of the camera.

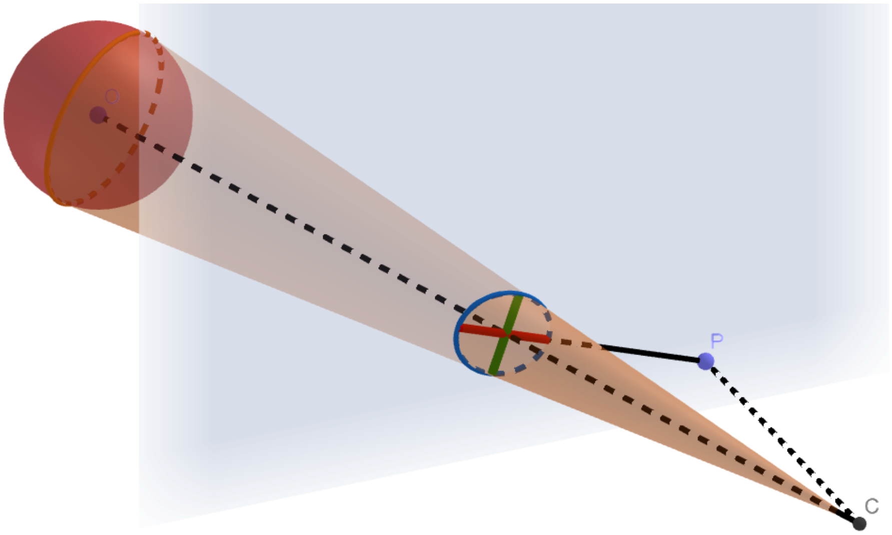


Figure 1: Schematic image of the problem containing a sphere in a 3D scene, an image plane and a camera. The projected sphere image is an ellipse. The main axis (int red) should go through the principal point of the image. The camera center, the principal point, and the sphere center are denoted by C, P, and O, respectively.

Minimal solution

The key question becomes how to define the algorithm using three independent points and the two constraints. Normalized image coordinates are used because of the numerical robustness. Thus the first step is this normalization. Then the equation system can be written in matrix form, based on Equation 2, as follows:

$$\begin{bmatrix} u_1^2 & u_1v_1 & v_1^2 & u_1 & v_1 & 1 \\ u_2^2 & u_2v_2 & v_2^2 & u_2 & v_2 & 1 \\ u_3^2 & u_3v_3 & v_3^2 & u_3 & v_3 & 1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \tag{5}$$

where $[u_i \quad v_i]^T$ are the normalized coordinates of the ith sample point, $i \in \{1, 2, 3\}$. In general, it can be written as

$$\mathbf{P}\mathit{\Phi} = \mathbf{0}, \tag{6}$$

where $\mathbf{P}$ is the matrix containing the polynomial coordinate functions and $\mathit{\Phi}$ is the vector containing the ellipse coefficients.

Matrix $\mathbf{P}$ has three right null vectors, the linear combination of those yields the solution for the ellipse parameters. The weights for the linear combination should be computed considering the two constraints, discussed above. When the parameters are calculated, a RANSAC-like algorithm can separate the inlier and outlier points related to the fitted model. We will compare the results of the proposed method with the traditional methods using synthetic and real-world tests. The ground truth data of the synthetic tests will clearly exhibit the performance of our algorithm both in time and precision.

Conclusion

In summary, this paper argued that 3 randomly chosen ellipse contour points are sufficient to construct a fitting method to estimate the ellipse parameters in case of a projected sphere. The novelty of the method is this minimal solution in opposition to the widely use five-point based model. This algorithm will be applied in our sphere-based camera calibration task [6]. Hence, in further research, the calibration pipeline can be more fast and accurate.

Acknowledgements

Tekla Tóth was supported by the project `EFOP-3.6.3-VEKOP-16- 2017-00001`: Talent Management in Autonomous Vehicle Control Technologies, by the Hungarian Government and co-financed by the European Social Fund. Levente Hajder was supported by the project no. `ED_18-1-2019-0030` (Application domain specific highly reliable IT solutions subprogramme). It has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme funding scheme.

References

[1] R. O. Duda and P. E. Hart. Use of the Hough Transformation to Detect Lines and Curves in Pictures, *Commun. ACM*, 1972

[2] A.W. Fitzgibbon and M. Pilu and R.B. Fisher, Direct Least Square Fitting of Ellipses, *IEEE Trans. on PAMI*, 1999

[3] M. Fischler and R. Bolles, RANdom SAmpling Consensus: a paradigm for model fitting with application to image analysis and automated cartography, *Commun. Assoc. Comp. Mach.*, 1981

[4] R. I. Hartley and A. Zisserman,Multiple View Geometry in Computer Vision, *Cambridge University Press*, 2003

[5] X. Ying and Z. Hu, Catadioptric Camera Calibration Using Geometric Invariants *IEEE PAMI* , 2004

[6] L. Hajder, T. Toth and Z. Pusztai, Automatic estimationof sphere centers from images of calibrated cameras, *InProceedingsof the 15th International Joint Conference on Computer Vision,Imaging and Computer Graphics Theory and Applications, Imagingand Computer Graphics Theory and Applications*, 2020

[7] G. Shivaram, G. Seetharaman, A New Technique for Finding the Optical Center of Cameras, *ICIP1998*, 1998

[8] N.Daucher, M.Dhome, J.T.Laprets, Camera Calibration From Spheres Images, *ECCV1994*, 1994

Side road to axioms for two-dimensional centralities: Network reconstruction from hub and authority values

Viktor Homolya and Tamás Vinkó

Abstract: In a network, determine which node is the most important in an aspect is a frequent task. We call centrality the function which gives the values to ranking. A lot of centralities have been created, they have their method to compute. In a recent paper of Sciearra et al. [4] it is shown how to calculate for directed graphs the degree and the HITS (Hyperlink-Induced Topic Search)centralities by using matrix factorization. Extending the centrality measures for weighted signed networks (WSN) is far from trivial (consider, e.g., the degree centrality: different nodes can easily get 0 value). On the other hand, signed-HITS has been introduced for WSNs [5]. We demonstrate that by using matrix factorization we do not obtain the signed-HITS values. In this case the factorization does not give a centrality, or is the signed-HITS not a decent extension? A set of axioms need to be proposed in order to see what properties might be requested for a function $C : V \to R^n$ to be centrality. On the way to find these axioms we have formulated an optimization model for the following problem: Given the hub and authority vectors of a directed network G. Is it possible to reconstruct the adjacency matrix A of graph G?

Keywords: graph reconstruction, centrality, HITS algorithm

Introduction

Node centrality is a function of form $C : V \to R^n$, where V is the set of nodes of graph G. Centrality values can be used to determine the order of importance of nodes. The well-known centrality measures are defined on directed weighted graphs. These assign values from $[0, 1]$ to every node. There are centralities that assign more than one value to a node. We can call them multi-dimensional centralities. Some examples for 2-dimensional functions are: in- and out-degree, HITS (Hyperlink-Induced Topic Search), Fairness-Goodness [3], PageRank-CheiRank. Some of these can be used on weighted signed networks (WSN), where weights can be negative. Sometimes the codomain of these functions is not the interval $[0, 1]$, negative values are allowed. For example, fairness value can be in $[-1, 1]$.

There are only a few proposed centralities for WSNs. The HITS algorithm's generalizations for WSN are the signed-HITS and the Fairness-Goodness. The Fairness-Goodness values are efficient in edge prediction, which formalism is resembling to a low-rank factorization of the adjacency matrix. Other centralities can be calculated via low-rank factorization [4]. We made tests to see that signed-HITS, Fairness-Goodness and other values from factorization correlate. The result is that the signed-HITS has no similarity with other tested values.

The question is, if they are different, which one is better for the generalization of HITS? To decide the question we create axioms such as in [1] to one-dimensional centralities on unsigned networks. Most of the well-known centralities do not fulfill all the mentioned axioms. It is also shown that these centralities made for various problems.

Trustworthiness between identities usually be represented as WSN. In these networks the general task is ordering the nodes by reliability or utility. Our goal is to make axioms to judge 2D centralities for reliability problems on WSNs. First of all, we decided to examine what are the expected attributions that a 2D centrality has to fulfill on weighted (unsigned) networks.

Axioms for centralities

We got some idea for our work from [1]. The mentioned axioms' shorter, informal versions:

- Size axiom: can change the size of a (special) subgraph to decrease the rank of nodes from this subgraph (be first).
- Density axiom: value depends on the number of neighbors connected by incoming edges
- Score / Rank-monotonicity axiom: with added, new in-edges to a node, its score won't be lower / it will get higher rank.

These axioms do not consider weights and deal only with incoming edges. To HITS also, because they tested only the hubs values.

Other ideas for WSNs came from [2], e.g., what is the minimum/worst node in a network where negative and zero weights exist? The isolated node or node with lowest sum of incoming weights? In the current work we pay attention axioms for 2D centralities on directed weighted networks. Some of these axioms, where f and g mean the first and second dimension centralities, are the followings:

- Homogeneity axiom: when two nodes have equal quantities of edges and the weights are same in pairs the centrality values have to be equal.
- Quantity (Density) axiom: if the f values of node x and y are the same and different from 0, but the number / expected value of weights / variation of weights of incoming edges are different, then g valuees of x and y have to be different or equal to zero.
- Quality axiom: if f (or g) centrality values of node x and y are equal and the number of incoming edges are the same but the weights of these edges are not, then the f (or g) centrality values of the neighbors have to be different, except when the values are 0.

Multi-dimensional centralities made for sort nodes by various criteria. With Quantity and Quality axioms if two nodes are equal in an aspect can sort by another one.

While testing if HITS fulfills the Quantity axiom, a problem has arisen. To prove that the axiom fails (a guess), we tried to create an unweighted graph where two nodes have the same (non-zero) hub and authority values, but have different number of incoming edges. It was an idea to reconstruct the graph from the known conditions. In the two vectors, hub and authority values' vectors, the ith and the jth elements are the same (can be different between the vectors).

Graph reconstruction from degree sequence is an active research field. The degree is a centrality value. Is it possible to reconstruct graphs from other centrality values? The new problem is creating graph when we know only the hub and authority values.

Graph reconstruction

The HITS centrality measure for directed unweighted graph G is calculated in the following way. Let $A \in [0,1]^{n\times n}$ be the adjacency matrix of G, $\overline{h} \in [0,1]^{1\times n}$ is the hub values' vector, and $\overline{a} \in [0,1]^{1\times n}$ is the vector of authority values. Let $C_1 = A \cdot A^T$ and $C_2 = A^T \cdot A$. Then $\overline{h}$ is the dominant eigenvector (eigenvector of the largest eigenvalue in absolute terms) of C_1 and $\overline{a}$ is the dominant eigenvector of C_2.

A real, symmetric M matrix always has an eigendecomposition: $M = VDV^T$, where D is real, diagonal matrix formed from eigenvalues of M, and V is real, orthogonal matrix, its columns are the corresponding eigenvectors. If A is not symmetric because directed edges, even so C_1 and C_2 will be symmetric. All elements of C_1 and C_2 are non-negative integers. The eigenvalues of these matrices are the same and non-negatives.

We created an optimization model that contains the conditions to find a C_1 matrix whose dominant eigenvector is $\overline{h}$. The diagonal elements of C_1 are the out-degree, non-diagonal elements show how many joint out-neighbors have the two nodes. For C_2 it is similar, but with in-degree and in-neighbors. From C_1 (or C_2) we can create an A' matrix (similarly as in binary tomography).

This approach has some problems. The search space of this model to find C_1 is non-convex. Solvers usually cannot guarantee a solution in acceptable time. With knowledge of C_1 and C_2 matrix A' can be different from A. As an illustrative example, let's consider the following adjacency matrices:

$$\begin{array}{l} A: \\ \begin{array}{ccccc} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{array} \end{array} \qquad\qquad \begin{array}{l} A': \\ \begin{array}{ccccc} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{array} \end{array}$$

Note that both have the same C_1 and C_2 matrices. The hub and authority scores are the same. If we work only from one of these C_i matrices the problem is even worse. We can get an A' matrix, whose in-degree sequence is the same as that of A, but the out-degree sequences are different. The reason for work from one vector (hub or authority values) that it is an easier problem, with less variables and less conditions. Another motivation is we want to reconstruct a graph, when some values are missing, like the majority of authority values.

Brief conclusion and future plans

In this work we draw up some axioms to 2-dimensional centralities for weighted networks. To test one of the axioms another problem has appeared: reconstruct a graph from its hub and authority values. Some problems with this task have been presented.

This work offers two paths for future plans: research reconstruction from other centralities and create axioms for 2-dimensional centralities on WSNs.

Acknowledgements. This research has been partially supported by the project "Integrated program for training new generation of scientists in the fields of computer science", no EFOP-3.6.3-VEKOP-16-2017-0002. The project has been supported by the European Union and co-funded by the European Social Fund.

References

[1] Paolo Boldi & Sebastian Vigna, Axioms for Centrality, *Internet Mathematics*, 10:3-4, 22-262, 2014.

[2] Csató László "Rangsorolás páros összehasonlításokkal." Kiegészítések a felvételizői preferencia-sorrendek módszertanához. *Közgazdasági Szemle* 60 (2013): 1333-1353

[3] S. Kumar, F. Spezzano, V. Subrahmanian, and C. Faloutsos. Edge weight prediction in weighted signed networks. In *16th International Conference on Data Mining (ICDM)*, pages 221–230. IEEE, 2016.

[4] C. Sciarra, G. Chiarotti, F. Laio, and L. Ridolfi. A change of perspective in network centrality. *Scientific reports*, 8(1):15269, 2018.

[5] M. Shahriari and M. Jalili, Ranking nodes in signed social networks, *Social Network Analysis and Mining*, 4(1), 2014.

A Combination of Attribute-Based Credentials with Attribute-Based Encryption for a Privacy-friendly Authentication

Yuping Yan and Péter Ligeti

Abstract: Attribute-based credentials (ABC) is an authentication scheme to identify an entity without revealing its identity. Privacy-friendly ABC mechanism does not ask for the revelation of users' whole attributes, instead it only requests a predicate over the attributes which is different from traditional PKI and ABC. No full identification is required in privacy-ABC by following the zero-knowledge property and combining identity attributes. Attribute-based encryption (ABE) is a new cryptographic primitive with additional functionalities in fine-grained access control mechanisms and one-to-many flexible encryption mode. However, most of the ABE schemes are not privacy-friendly. In this paper, we propose a privacy-friendly ABC system with ABE mode, which has several in attribute revocation problem, privacy-friendly ABE scheme, flexible access control in ABC, data protection, and user authentications.

Keywords: Attribute-based encryption, Attribute-based credentials, Authentication, Access Control

Introduction

In contrast with traditional authentication methods, the privacy-friendly attribute-based credentials (ABC) uses attributes rather than identities to authenticate an entity. In this way, authentication authority does not need to verify all the properties of users. For example, usually, the name, age, sex, birthday, and identity card numbers are standard information to guarantee a person's identity. However, if we want to know whether a person can buy alcohol, age is the only attribute that matters. Thus, instead of reveling the whole identity, it provides some level of privacy by just revealing few attributes. The sets of attributes signed by an issuer are called credentials. However, all the ABC systems require authentication from a third party, which can cause privacy problems.

Attribute-based encryption (ABE) is an extension of public key cryptography and identity-based encryption. In ABE scheme, both the ciphertext and the key are related to a set of attributes. According to the characteristics of information and the attributes of receivers, the encryptor can customize an encryption strategy. The generated ciphertext can be decrypted only by the users whose attribute satisfies the encryption policy. If one could use attributes - like the ones used in ABC - and define access polices over them, fine-grained access control could be obtained [4].

Privacy-enhancing Attribute-Based Credentials (Privacy-ABCs) are innovative technologies that both provide authentication and access control for the digital services, and enhance user's privacy [5]. By combining attribute-based credentials with attribute-based encryption it is possible to achieve privacy and data protection together. In such a system, the data is authorized by the user attributes. On the other hand, there no third party is required, which solves the privacy problem of ABC.

In this paper, we focus on a technique of the combination of ABCs with ABE for a privacy-friendly authentication. In section we introduce the necessary background on access control, attribute-based encryption modes, the revocation problem, and the privacy-enhancing attribute-based credentials techniques. In section we propose a combination of ABC and ABE achieving privacy-friendly authentication by multi-authority Cipher Policy Attribute-based encryption.

Background

This chapter provide a brief introduction to the used cryptographic primitives.

Attribute-based Credentials

In the general architecture of Privacy-ABC technology, there are three main parties, a User, an Issuer, and a Verifier. The issuer will issue the credentials to the users for further authentication by the Verifier. A credential is a set of attributes that is (digitally) signed by an issuer.[1]

The most critical features of the Privacy-friendly ABCs scheme are the untraceability and unlinkability between the credentials of different users. Untraceability means that users' credentials can not be traced between different interactive protocols and credentials claims. Unlinkability means that different sets of credentials of the same user can not be linked.

Access Control

Access control mechanisms are methods that imply to enforce the authentication policies which define what an individual can get access to. It refers to a set of policies for restricting access to information. The main traditional access control techniques are discretionary access control, mandatory access control, role-based access control, and other models. However, there are two problems with these classical access control systems. The first one is that it asks for an online authentication server, which is vulnerable to Denial of Service Attacks. Additionally, once the server gets attacked, attackers can get access to all the stored plain-text.

In order to solve these problems, we switch into encryption-based and attribute-based access control. We encrypt data in the web-server instead of storing plain data directly, and then users can decrypt the content with decryption key associated with several attributes. Thus, access control is defined by key issuers. The main difference of encryption access control is that the authorization takes place in users' decryption phase.

Attribute-based Encryption

As a public key encryption algorithm, ABE's decryption object is a group, not a single user. It uses the combinations of groups' attributes as the public key to encrypt all the data. In contrast, the private key is calculated and assigned to the individual by the attribute authority based on the user attribute. Standing on the bilinear pairing techniques, the ABE builds various access structures to achieve fine-grained access control of data. The basic ABE consists of the following four phases: $Setup()$ to initialize, $KeyGen()$ to generate the keys, $Encrypt()$ to encrypt and $Decrypt()$ to decrypt.

In traditional ABE mode, there is only one trusted organization to manage all attributes. However, it is impractical to use a single authority to distribute all attributes in distributed and large-scale environment. Chase et [2] propose Multi-authority attribute-based encryption(MA-ABE), in which multiple organizations manage different attribute sets and distribute the keys within their authority.

In the CP-ABE mechanism, we embed the tree access structure into ciphertext, and combining attributes sets to generate the users' secrete keys. The CP-ABE is different from the basic ABE algorithm. The length of public keys and public parameters are independent of the number of system attributes.

Revocation problem

In ABE systems, it is hard to revoke the attributes in the existing applications, which makes ABE modes complicated to implement. In stock, the rights or the attributes of the users change, and we have to modify the access structure accordingly. However, the revocation of attributes is quite tricky.

There are two main current approaches of keys and attributes revocation of ABE modes: indirect revocation and direct revocation. In direct revocation mode, the sender specifies the revocation list when encrypting the messages, achieving the revocation directly. In indirect revocation mode, the authorized institution releases the key periodically, and only the non-revoked users can update the key.

Our contribution: combination of ABC and ABE

ABCs have a comprehensive implementation of smart cards, combining with IBM's Identity Mixer (Idemix) technology. In [3], a privacy-friendly key generation for smart card-based attribute-based encryption is proposed. It uses CP-ABE to encrypt data while adopts ABCs to authenticate users' identities.

In the following paragraphs, we will propose a solution for combination of attribute-based credentials and attribute-based encryption as illustrated in Figure 1.

In our proposed system, the principal participant is the revocation referee. By working together with the Authority in CP-ABE, the users' attributes can be changed accordingly, which improve the efficiency and solve the problem of attribute revocation in encryption and decryption. Secondly, the ABCs itself does not support access control. When we combine these two techniques, CP-ABE provides a more flexible way for users by checking whether the attributes satisfy the access policies. Additionally, in this mode, users do not need to reveal all of its attributes in the decryption step. Once an attribute requirement is met, the user is able to decrypt since CP-ABE can support "or" gate operation. Hence, in this case, the privacy of CP-ABE is also improved.

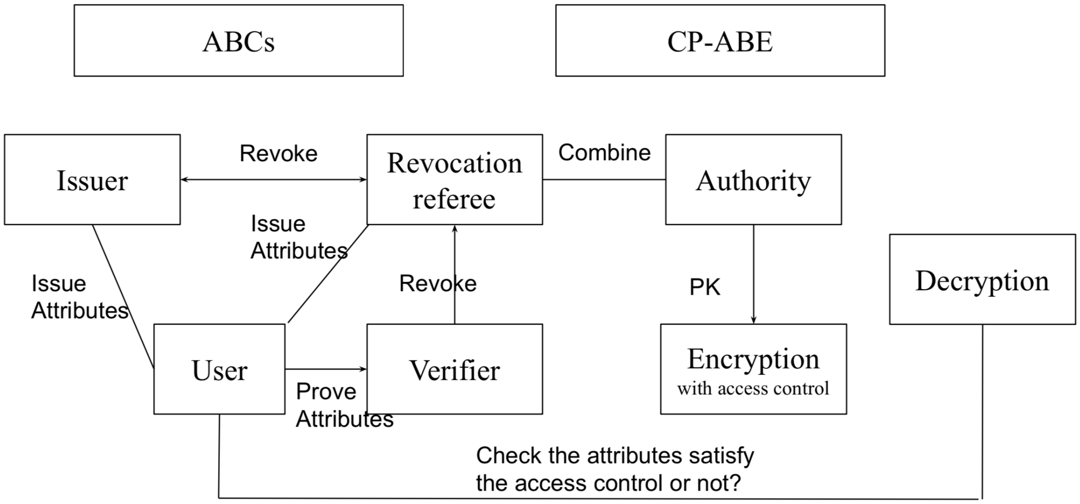


Figure 1: The combination of ABCs and ABE

Attribute revocation is a technical problem in the implementation of ABE schemes. However, ABC can help to solve this problem with the presentation token. Some credentials are revocable because, in the ABC system, the revocation authorities are stated and specified. A user can form a presentation token that contains a subset of the certified attributes, provided that the corresponding credentials have not been revoked. See [1] for more details.

Let us note that the simple ABCs and ABE can not reach the privacy-friendly authentication and encryption requirements. It is easy to see that ABCs can not prevent collusion attacks; different users can join their attributes and reach the attribute access control requirements. However, attribute-based encryption does prevent against collusion attack. No group of users can share and combine their keys to decrypt the ciphertext from the encryption and decryption mode of ABE. Thus, the combination of ABCs and ABE can efficiently prevent collusion attacks for users authentication.

Cipher-policy ABE is not privacy-friendly in a sense, that we have to fully reveal the attributes when we want to encrypt and decrypt with users' attributes. However, if we combine ABCs inside, only a certain of attributes are satisfied, we can use ABE to encrypt and decyrpt. By this method, the privacy and security can be efficiently improved. Meanwhile, ABE mode can support different access controls and structures, while ABC can not realize any access control. Thus, if we combine these two techniques, we can provide a flexible access structure in authentication and encryption.

Another problem in most ABE schemes is the frequent usage of attribute authorities and key generation centers. In this case, we need to ensure the honesty and efficiency of Attribute Authorities(AAs) and Key Generation Center(KGC). However, by implying a blind ABE scheme, this problem can be efficiently solved.

Note that, this paper contains a proof-of-concept of the combination of ABC and ABE only. Further research is needed to analyze and validate the proposed method from theoretical and practical point of view, like security analysis and implementation on IoT devices or smart cards. Additionally, this protocol model has some possible applications in cloud environment to achieving privacy and user authentication.

Acknowledgements

This research has been partially supported by project no. ED_18-1-2019-0030 (Application-specific highly reliable IT solutions) has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary. This work was supported by Péter Ligeti and Mohammed B. M. Kamel. With all of their help, I have a better understanding of how to organize my PhD life and how to do researches.

References

[1] Camenisch, J., Concepts Around Privacy-Preserving Attribute-Based Credentials.*IFIP PrimeLife International Summer School on Privacy and Identity Management for Life. Springer, Berlin, Heidelberg, 2013.*

[2] Chase, M., Multi-authority attribute based encryption. *Proc of Theory of Cryptography Conf. Berlin: Springer, 2007: 515-534*

[3] Hanzel, M., and Clément, C., Blockchain for Clinical Decision Support Systems.

[4] Kamp, T. R., Combining ABCs with ABE Privacy-Friendly Key Gengeration for Smart Card Based Attribute-based Encryption, *Master's thesis Computer Science, University of Twente*, 2014.

[5] Veseli, F., and Serna, J., Evaluation of privacy-ABC technologies-a study on the computational efficiency,*IFIP International Conference on Trust Management. Springer, Cham, 2016.*

Crosslayer Cache for Telemedicine

Zoltán Richárd Jánki and Vilmos Bilicki

Abstract: In modern Web applications, it is essential to be robust, responsive and present consistent data with as low latency as possible. Regarding the category of telemedicine Web applications, indeed, consistency and low latency need the highest attendance. Since the amount of electronic healthcare records is rapidly increasing, it is also required to store data in a distributed database system. By taking into account Eric Brewer's CAP theorem [1], we have to find the proper balance among consistency (C), availability (A) and partition-tolerance (P). We have collected real world telemedicine use cases and elaborated an easily adaptable system model in which the level of consistency can be subtly tuned. Based on a given scenario and the accepted staleness of data, we can provide recommendations for consistency configuration, caching strategy and cache points on data path.

Keywords: consistency, latency, cache, telemedicine, CAP theorem

Introduction

As Web became worldwide, it has come to be a platform for modern applications. Today, a Web application has to be not only robust and responsive, but it also has to respond with low latency. In order to support users and applications accessing and sharing remote resources easily and in no time, server-side is more and more frequently consists of well-scalable distributed systems [2]. Besides considering the advantages, a distributed system has weaknesses too.

As telemedicine entered in the cloud era, most of telemedicine Web applications are based on distributed systems. It is a requirement, since the amount of electronic healthcare records (EHR) is rapidly growing. According to Eric Brewer, a distributed system can guarantee at most two of three desirable properties: consistency (C), availability (A) and partition-tolerance (P) [1]. Finding the appropriate balance among the above mentioned properties is not easy at all, but in order to set up an efficient configuration, the capabilities has to be measurable. We did research on metrics of distributed system properties. Peter Bailis et al. introduced the so-called Probabilistically Bounded Staleness (PBS) technique that offers metrics for measuring consistency and availability [3]. Based on their metrics, we elaborated a system model that is applicable for many telemedicine use cases.

State of the art

We have discussed in our paper published in 2018 [4] that modern Web applications have to be robust, responsive and have to give responses as soon as possible. As technologies are developing, users are becoming less and less tolerant for latency. Regarding distributed systems and CAP theorem, we have to make a trade-off between consistency and availability because there are no distributed systems that can achieve consistency and availability at the same time [5].

Meanwhile, PBS - presented by Peter Bailis - introduced a metric set for measuring consistency and availability of quorum-replicated data stores, like Apache Cassandra [6]. *t*-visibility and *k*-staleness are the metrics of this technique that can help to describe other type of distributed systems as well after a small adaptation. For evaluation, they introduced a model of message latency called WARS that stands for Write, Ack, Read and Response. WARS was implemented in an event-drive simulator for use in Monte Carlo methods. They noticed that changing the quorum sizes of read and write can radically affect *k* and *t* properties. Differences in latencies can be even hundreds of milliseconds high.

Microsoft designed Azure Cosmos DB in such a way that consistency can be parameterized. They defined 5 levels of consistency: strong, bounded staleness, session, consistent prefix and eventual. They specified and verified their work with a formal specification language called TLA+ [7][8].

In the next section, we will review our motivations and list telemedicine use cases in which consistency is critical. Regarding our motivations and use cases we will present a system model that provides a solution for consistency and availability trade-off problem. After describing our model, we will prove its correctness.

Motivation

Since most of Web applications can be opened on mobile platforms too, we cannot leave out of consideration the trends that are followed by the main participants of mobile marketplace. As an example, application users are gotten used to low latency since Android is continuously monitoring rendering time of user interface (UI) frames. Improving app quality, if an application cannot render a frame within 700 milliseconds (ms), then the given UI frame is considered frozen. Google recommends apps to render a frame within 16 ms in order to have smooth UI [9].

On the other hand, latency can arise from data transmission too. High transport delay can occur because of network issues, huge distances or simple server and database problems. Also, the delay of transatlantic cables is about 100 ms in one direction. We can approach the 16 ms limitation, if we use Content Distribution Networks (CDNs). Through the use of CDNs, Web caches are becoming more and more important. CDNs are responsible for installing many geographically distributed caches throughout the Internet, so that much of traffic could be localized [10]. Additionally, we can cache at different levels of data path in order to reduce latency and increase availability.

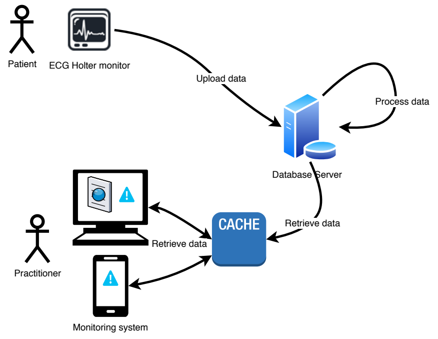


Figure 1: ECG use case

However, during our former telemedicine projects, we met scenarios that required consistency against availability. We collected an incomplete list of non-invasive measurements that can be carried out at home and need strong consistency in given circumstances.

1. Electrocardiography
2. Blood pressure monitoring
3. Blood glucose monitoring

From an *electrocardiography* (ECG) measurement, several evidences of a sickness can be taken, like atrial fibrillation, ventricular fibrillation, extrasystole, and so on. For instance, atrial fibrillations can lead to stroke when every second counts. The following scenario is shown in Figure 1.

ECG measurements can be done at home by using a holter monitor that can upload realtime data to the cloud. In practice, uploaded samples have a few seconds length, and the data process and evaluation is executed in longer windows. The processed data is sent back to the practitioner who is notified too if his or her patient has problems. In this case, missing a window of data due to weak consistency can lead to the patient's serious health damages. Naturally, besides consistency, there are other factors that can block or influence data transmission.

Summary of our work

Our goal is to elaborate a system that makes a trade-off between consistency and availability, although we take into account that it has to be easily tuned. In order to make sure that our system works as it should, we need to prove its correctness. One approach could be a simple simulation in which we can test a finite set of possible inputs. Simulating a networked system that works with small latencies is not only very difficult but also several cases remain untested. For this purpose, we choose logical modelling. Writing down a system specification with a modelling language, we can create a state space that can be explored via state graph. We created a logical model using TLA+, and made executions in different environments. Our specification was written in PlusCal which is a formal specification language that transpiles to TLA+. PlusCal resembles imperative programming languages in which writing a specification is easier for us.

```
read ≜  ∧ pc[“crud”] = “read”
        ∧ IF lat1 ≥ cacheLat
              THEN ∧ lat1′ = 0
                   ∧ history′ = Append(history, “client_read”)
                   ∧ IF (cacheVer < Len(ecgData) − MaxStaleness)
                         THEN ∧ cacheContent′ = ecgAF
                              ∧ cacheVer′ = Len(ecgData)
                         ELSE ∧ TRUE
                              ∧ UNCHANGED ⟨cacheVer, cacheContent⟩
                   ∧ readData′ = Append(readData, cacheContent′)
              ELSE ∧ lat1′ = lat1 + 1
                   ∧ UNCHANGED ⟨readData, cacheVer, cacheContent, histor
        ∧ pc′ = [pc EXCEPT ![“crud”] = “client_actions”]
        ∧ UNCHANGED ⟨ecgData, ecgDataSum, ecgAF, ecgAFData, dbLat, cach
                      calcLat, lat2, lat3, numOp, af, dataTail⟩
```

Figure 2: Read operation

```
write ≜  ∧ pc[“crud”] = “write”
         ∧ IF lat2 ≥ dbLat
               THEN ∧ lat3′ = lat3 + 1
                    ∧ lat2′ = 0
                    ∧ ∃ d ∈ 1 .. DataRange :
                         ecgData′ = Append(ecgData, d)
                    ∧ history′ = Append(history, “client_write”)
               ELSE ∧ lat2′ = lat2 + 1
                    ∧ UNCHANGED ⟨ecgData, lat3, history⟩
         ∧ pc′ = [pc EXCEPT ![“crud”] = “read”]
         ∧ UNCHANGED ⟨ecgDataSum, ecgAF, ecgAFData, readData, cacheVer
                       cacheContent, dbLat, cacheLat, calcLat, lat1, numOp
                       af, dataTail⟩
```

Figure 3: Write operation

In our spec, we defined 3 entities that can communicate with each other. These are client, database and cache. We implemented the possible actions that describe the entities' operations. A client can read and write data, a database can make data aggregation and cache can update its content. The implementation of client read and write operations are shown in Figures 2 and 3. In our model, we have two parameters, one is the staleness that can influence consistency and the other one is the latency that can affect availability. Latency is assigned to all entities, staleness is taken into account on client reads. We examined three K values, $K = 0$, $K = 1$ and $K = 2$. $K = 0$ means that client reads the most up-to-date version of data, while $K = 1$ and $K = 2$ let client read 1 and 2 version older data. K values can be used to configure cache. Since we combined the staleness with delays, we also checked the probability of retrieving inconsistent data depending on latency.

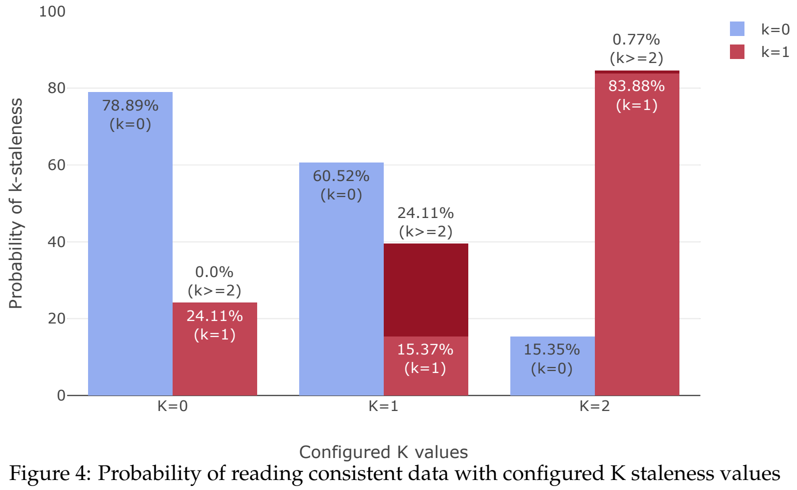


Figure 4: Probability of reading consistent data with configured K staleness values

Considering our telemedicine use cases, we focused on consistency because it is the most critical parameter in scenario. We executed TLC model checker on our model with given parameters and that resulted in a state graph. We processed the raw graph markup file that was written in DOT graph description language and we constructed a two-dimensional tabular data structure called dataframe. The dataframe processing resulted in 1,270,930 distinct states that contained 985,020 client read operations. After studying the whole state space, we drew the following inferences. Our consequences are also visualized in Figure 4.

As we are increasing the allowed staleness parameter, the probability of retrieving consistent data is getting lower. With $K = 0$ configuration parameter we can guarantee having consistent data in 78.89%. If K is set to 2, this probability is under 20%. So with higher staleness parameters, we can miss data windows with huge percentage. Recalling ECG monitoring, with higher K values, it is possible that a queue of data samples that contained atrial fibrillation is missed, which can lead to patient's health damage. Another interesting fact is that the staleness of retrieved data (k) is varying due to latencies.

In table 1, we collected the situations when delay matters. As we can see, setting up $K = 0$ for consistency, the probability of fetching not up-to-date data is 24.11% which is quite high regarding our configuration. However, 100% of these cases occured due to latency. We can also see that, we can further reduce latency if we place caches on data path that decrease delay of data transmission.

Table 1: Retrieved data with given staleness affected by latency

Configuration (K)	Staleness of data (k)	latency: 0	latency > 0
$K = 0$	$k = 0$	65.89%	34.11%
	$k = 1$	0%	100%
	$k = 2$	0%	0%
$K = 1$	$k = 0$	69.92%	30.08%
	$k = 1$	50%	50%
	$k = 2$	0%	100%
$K = 2$	$k = 0$	50%	50%
	$k = 1$	50%	50%
	$k = 2$	50%	50%

Conclusions

In our paper, we introduced a system model that offers a solution for consistency and availability trade-off problem in distributed systems. We listed such telemedicine use cases where consistency is crucial. The consistency level of our system is easily tuneable, but we noticed that consistency is highly influenced by latency even if the consistency level is set to strong. As a result, we can guarantee strong consistency with configuration, and also can increase availability with caches. In our future work it is planned to create a program that can construct system specifications from real telemedicine implementations. With such specs, we can prove the correctness of our implementations amd we can detect bugs, like deadlocks.

Acknowledgements

This work was supported by the construction EFOP-3.6.3-VEKOP-16-2017-00002. The project was supported by the European Union, co-financed by the European Social Fund.

References

[1] E. Brewer, *CAP Twelve years later: How the "Rules" have Changed*, Computer, vol. 45, February 2012, pp. 23–29

[2] M. van Steen and A.S. Tanenbaum, *A brief introduction to distributed systems*, Computing 98, pp. 967–1009 (2016). https:doi.org10.1007s00607-016-0508-7

[3] Bailis, Peter, et al., *Probabilistically Bounded Staleness for Practical Partial Quorums*, Ion., Proceedings of the VLDB Endowment. 5., 2012., 10.14778/2212351.2212359.

[4] Z. R. Jánki and V. Bilicki, *Full-stack FHIR-based MBaaS with Server- and Client-side Caching Capable WebDAO*, CSCS, The 11th Conference of PhD Students in Computer Science, 2018, pp. 179–183

[5] M. Kleppmann, (2015)., *A Critique of the CAP Theorem*

[6] A. Lakshman and P. Malik., 2008., *Cassandra - A Decentralized Structured Storage System*, InLADIS, pp. 35–40

[7] L. Lamport, J. Matthews, M. R. Tuttle, Y. Yu. (2002.) *Specifying and verifying systems with TLA+*, Proceedings of the 10th Workshop on ACM SIGOPS European Workshop, EW 10. 45–48., 10.1145/1133373.1133382.

[8] L. Lamport. 2002. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*, Addison-Wesley Longman Publishing Co., Inc., USA.

[9] *Frozen frames*, Available: https://developer.android.com/topic/performance/vitals/frozen, Accessed: 20 March 2020

[10] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 7th Edition, 2017, Pearson Education Limited, Edinburgh Gate, Harlow, Essex CM20 2JE, England

EHR Data Protection with Filtering of Sensitive Information in Native Cloud Systems

Zoltán Szabó and Vilmos Bilicki

Abstract: In recent years, software development with native cloud support became a more and more popular practice, despite providing several serious challenges for developers, especially in the field of healthcare software development, where the applications used to originally run in isolated, well-protected environments where the handling of crucial, sensitive information in readable, standardized format without the necessary authorization was nearly impossible. The most popular current standard concerning the format of digital healthcare data is the Fast Healthcare Interoperability Resources, or FHIR, developed by the Health Level 7 organization, which unfortunately provides only drafts, recommendations and tools [2] for the crucial authorization and security management. In this paper, we summarize the various requirements of healthcare application security, establish metrics from the collected requirements, then use the metrics to compare and validate the services of several popular PaaS providers while also introducing the concept of a custom, cloud-independent solution for these problems.

Keywords: FHIR, full-stack, security, healthcare, PEP, policy engine, Google Cloud, Azure, AWS

Introduction

The most prominent standard in healthcare software development is without any doubt the Fast Healthcare Interoperability Resources, of FHIR [1] for short from the HL7 organization, which establishes a set of customizable documents to describe the entire medical infrastructure from general practitioners to administration level in a standardized XML or JSON format with a pre-defined REST API to retrieve and modify said resources. Despite the popularity, FHIR in its current state still possesses some shortcomings, especially in the area of data protection and security, where developers are only provided with a set of security labels and some recommendations, but nothing more. This makes the adaptation of FHIR more and more challenging, given the growing concerns with the usage and protection of personal data, and the establishment of international acts and regulations such as GDPR [11] and HIPAA [10]. The sensitive nature of healthcare datasets continually proves to be a difficult aspect of application development to overcome, resulting in solutions, which even though satisfy the legal requirements, not necessarily provide correct answers to realistic scenarios and use cases [4].

State of the Art

The popularity of the FHIR standard is well-indicated by the fact that more and more of the great cloud providers start to adapt it and offer the format along with the REST API as the de facto solution for storing and communication healthcare data. The majority of these solve the security problem with the extension of their default role-based access control methodology to the healthcare resources, but, as we are going the showcase it, mere access control is not a complete answer to every question and requirement established by the users and developers of the healthcare applications. The most complex solution of the cloud-based FHIR services is offered by Microsoft Azure [5] in the form of the FHIR Services for Azure, which not only provides a custom FHIR REST API with the integration of the popular SMART on FHIR [3] proxy and the Azure Active Directory to handle the identification of various users, groups and applications, but also makes it possible to integrate a custom virtual FHIR server in the same environment. The Google Cloud [6] also supports the FHIR format as the default healthcare communication format of the Cloud Healthcare API, while the Amazon AWS [7], although offers the virtual deployment of an FHIR server, it provides the lowest level of support to develop healthcare applications in their cloud with the standard. In the comparison below we defined the requirement metrics as the following categories based on several sources examining the problem [9] [12] [13]: first, the cloud has to provide the tools to separate the various roles and attach them to users, groups or applications, second the cloud has to provide tools to include inner attributes and context-based information while evaluating the authorization, third, there has to be a way to mark several documents or fields to be removed from the results or replaced by placeholder values and finally, the cloud has to provide a way to implement solutions for break-the-glass scenarios, events, when an unauthorized user has to gain access to critical information to provide the necessary care. We also included in the comparison the Open Policy Agent [8] policy engine, our chosen engine for the implementation of a cloud neutral PEP.

Table 1: Cloud Provider Comparisons

Requirement	Google Cloud	Microsoft Azure	Amazon AWS	OPA
Separation of Roles	Yes	Yes	Yes	Yes
Context-based Authorization	Partially with Native Rules or Full With Cloud Functions	With Azure Functions	With Lamdba Functions	Yes
Modification of the Result Set	Only With Cloud Functions	Only With Azure Functions	Only With Lamdba Functions	Yes
Break-the-Glass	Yes, but without specific access restriction	Yes, but without specific access restriction	Yes, but without specific access restriction	Yes

Evaluation

As shown on Table 1, although the major cloud providers offer high level security solutions, when it comes the to complex requirements of the healthcare applications, they can only meet the requirements by extending their logic with custom query hooks implemented through scripting solutions. This is the main reason why for our own approach we chose to implement a cloud neutral solution in the form of an external PEP. The storage is only accessible through a proxy which checks the user identity and has to ability to modify the query and even the results. Open Policy Agent is a rather new, open source policy engine with an emphasis on high performance during policy decisions. OPA separates the decision making and the policy enforcement modules, requiring only the input dataset it needs to apply the rules (written in OPA's custom Rego language) on. To evaluate the effectiveness of a solution as shown on Figure 1 we have defined a set of Rego rules, modeling the four main categories of security requirements and run them on separate input datasets containing json objects of the FHIR Observation resource, describing various vital signs of the patient, such as bloodpressure, body weight and bloodalcohol level. During the evaluation we measured the maximum CPU usage and memory usage of the OPA process while applying the rules to the input data. We also chose to increment the size of the dataset with each input set - while the initial only contained a mere 10 records, the next one had a 100 and the final exactly 1 million Observations. The tests were run in an environment with a 4-core Intel i5-3750K processor, 8 GB DDR4 RAM and 120 GB SSD.

In the first category, the rule only filtered the Observations based on the owner aka. the identity of the patient, in the second we also filtered using several inner attributes including the exact value, and to put the system under a heavier load, we first calculated the average of the entire input set, then returned only the Observations which were created after a specific date with a value higher than the average. The third not only applied several filters but also removed one of the fields from every document passing the query, while the fourth, the break-the-glass scenario removed and rewrote specific fields in every instance.

As shown in Table 2 and Table 3, the size of the dataset had a much more significant impact on the resource consumption of the OPA process, than the complexity of the rules. In all cases the smaller datasets required nearly insignificant resources, while with the 100.000 and 1

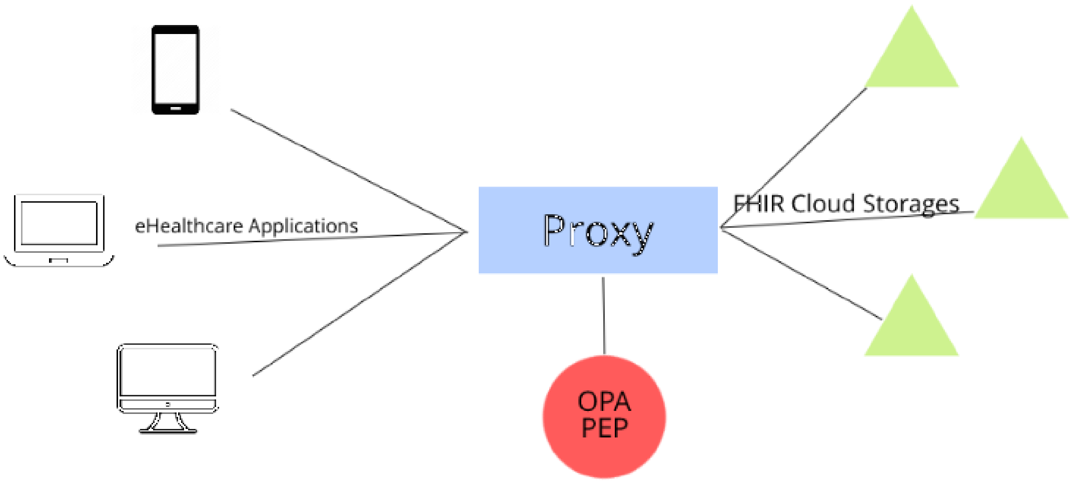


Figure 1: The layout of our planned architecture including the OPA PEP

Table 2: Maximum Percentage of CPU Usage

Input Size	Role Evaluation	Context with Aggregation	Context with Modification	Break Glass
10	0,00	0,00	0,00	0,00
100	0,00	0,00	0,00	0,00
1000	1,56	6,34	1,56	1,56
10.000	10,84	28,07	17,43	23,37
100.000	130,86	127,74	135,53	129,43
1.000.000	326,94	357,63	334,93	328,10

Table 3: Memory Requirement in MBytes

Input Size	Role Evaluation	Context with Aggregation	Context with Modification	Break Glass
10	15,039	14,867	13,66	14,8
100	15,094	14,949	16,234	14,87
1000	16,187	16,488	17,757	15,46
10.000	44,863	44,929	44,835	47,367
100.000	398,968	395,3	329,043	400,227
1.000.000	3806,277	3933,394	3937,359	3839,019

million sized sets the consumption levels quickly increased. If we take it into account that in practice, most queries use paging solutions, and the queried datasets are rarely downloaded from the cloud to local devices in batches larger than 250 or at most 500 at a time, these attributes seem very promising. It is also worth mentioning that while the latency seemed to have a random variance during each evaluation, which is why we chose not to include it among the test results, but with the exception of the 1 million size, it never took longer than 1-2 seconds, clearly showcasing the promised low-latency feature of OPA. Based on these results, we are ready to take the next step forward and implement a full implementation of the planned architecture in a healthcare development project with the integration of our general healthcare ACL syntax [14], and further evaluate our solution by implementing a complete, real-life requirement set and measuring it against a normal workload.

Acknowledgement

This research was supported by the EU-funded Hungarian grant EFOP-3.6.1-16-2016-00008.

References

[1] *FHIR Overview*, Available: https://www.hl7.org/fhir/overview.html, Accessed: 21 March 2020

[2] *FHIR Data Segmentation for Privacy*, Available: http://build.fhir.org/ig/HL7/fhir-security-label-ds4p/branches/master/index.html, Accessed: 21 March 2020

[3] Mandel, Joshua C., et al., *SMART on FHIR: a standards-based, interoperable apps platform for electronic health records.*, Journal of the American Medical Informatics Association 23.5, 2016, pp. 899-908.

[4] Altamimi, Ahmad Mousa. *Security and privacy issues in eHealthcare systems: towards trusted services.* International Journal of Advanced Computer Science and Applications 7.9 (2016): 229-236.

[5] *Azure API for FHIR Documentation*, Available: https://docs.microsoft.com/en-us/azure/healthcare-apis/, Accessed: 21 March 2020

[6] *Cloud Healthcare API*, Available: https://cloud.google.com/healthcare/docs, Accessed: 21 March 2020

[7] *Building a Serverless FHIR Interface on AWS*, Available: https://aws.amazon.com/blogs/architecture/building-a-serverless-fhir-interface-on-aws, Accessed: 21 March 2020

[8] *Open Policy Agent Official Documentation*, Available: https://www.openpolicyagent.org/docs/latest/, Accessed: 18 March 2020

[9] Marcus, Andrew *Security in FHIR* at DevDays Redmond 2019, Available: https://tinyurl.com/ryk9zlu, Accessed: 17 March 2020

[10] Ness, R. B., and Joint Policy Committee. (2007). *Influence of the HIPAA privacy rule on health research.* Jama, 298(18), 2164-2170.

[11] Orel, A., and Bernik, I. (2018, October). *GDPR and Health Personal Data; Tricks and Traps of Compliance.* In EFMI-STC (pp. 155-159).

[12] Gajanayake, R., Iannella, R., and Sahama, T. R. (2012, March). *Privacy oriented access control for electronic health records.* In Data Usage Management on the Web Workshop at the Worldwide Web Conference. ACM.

[13] Finance, Beatrice, Saida Medjdoub, and Philippe Pucheral. *Privacy of medical records: From law principles to practice.* 18th IEEE Symposium on Computer-Based Medical Systems (CBMS'05). IEEE, 2005.

[14] Z. Szabó, V. Bilicki *Felhőben tárolt egészségügyi adatok védelme ABAC modellel* in 31th Neumann Colloquium, November 2018

List of Authors

Ádám Fodor: Eötvös Loránd University, Hungary

Ahmad T. Al-Anaqreh: University of Szeged, Hungary

Ali Al-Haboobi: University of Miskolc, Hungary

András Márkus: University of Szeged, Hungary

Artúr Poór: Eötvös Loránd University, Hungary

Attila Szatmári: University of Szeged, Hungary

Balázs Szűcs: AUDI HUNGARIA Zrt., Hungary

Bence Bogdándy: Eszterházy Károly University, Hungary

Biswajeeban Mishra: University of Szeged, Hungary

Csaba Bálint: Eötvös Loránd University, Hungary

Dániel Balázs Rátai: Eötvös Loránd University, Hungary

Dániel Pásztor: Budapest University of Technology and Economics, Hungary

Dávid Papp: Budapest University of Technology and Economics, Hungary

Dilshad Hassan Sallo: University of Miskolc, Hungary

Dorottya Papp: Budapest University of Technology and Economics, Hungary

Ebenezer Komla Gavua: University of Miskolc, Hungary

Gábor Karai: University of Szeged, Hungary

Gábor Székely: Budapest University of Technology and Economics, Hungary

Gabriella Tóth: Eötvös Loránd University, Hungary

György Papp: University of Debrecen, Hungary

Hamza Baniata: University of Szeged, Hungary

Hayder K. Fatlawi: Eötvös Loránd University, Hungary

István Fábián: Budapest University of Technology and Economics, Hungary

Jenifer Tabita Ciuciu-Kiss: Eötvös Loránd University, Hungary

José Vicente Egas-López: University of Szeged, Hungary

László Kopácsi: Eötvös Loránd University, Hungary

László Viktor Jánoky: Budapest University of Technology and Economics, Hungary

Levente Buttyán: Budapest University of Technology and Economics, Hungary

Márton Juhász: Budapest University of Technology and Economics, Hungary

Mátyás Kiglics: Eötvös Loránd University, Hungary

Mohammed B. M. Kamel: Eötvös Loránd University, Hungary

Mohammed Mohammed Amin: University of Szeged, Hungary

Orsolya Kardos: University of Szeged, Hungary

Péter Hudoba: Eötvös Loránd University, Hungary

Péter Ligeti: Eötvös Loránd University, Hungary

Róbert Bán: Eötvös Loránd University, Hungary

Roland Nagy: Budapest University of Technology and Economics, Hungary

Sándor Balázs Domonkos: University of Szeged, Hungary

Tamás Aladics: University of Szeged, Hungary

Tekla Tóth: Eötvös Loránd University, Hungary

Viktor Homolya: University of Szeged, Hungary

Yuping Yan: Eötvös Loránd University, Hungary

Zoltán Richárd Jánki: University of Szeged, Hungary

Zoltán Szabó: University of Szeged, Hungary

Zsolt Tóth: Eszterházy Károly University, Hungary

Notes

CSCS^2

Supported by the project *"Integrated program for training new generation of scientists in the fields of computer science"*, № **EFOP-3.6.3-VEKOP-16-2017-00002**. The project has been supported by the European Union and co-funded by the European Social Fund.