Decentralized Ranking in Large-Scale Overlay Networks *

Alberto Montresor
University of Trento
Italy
alberto.montresor@unitn.it

Márk Jelasity
University of Szeged and
Hungarian Academy of
Sciences, Hungary
jelasity@inf.u-szeged.hu

Ozalp Babaoglu
University of Bologna
Italy
babaoglu@cs.unibo.it

Abstract

Modern distributed systems are often characterized by very large scale, poor reliability, and extreme dynamism of the participating nodes, with a continuous flow of nodes joining and leaving the system. In order to develop robust applications in such environments, middleware services aimed at dealing with the inherent unpredictability of the underlying networks are required. One such service is aggregation*. In the aggregation problem, each node is assumed to have attributes. The task is to extract global information about these attributes and make it available to the nodes. Examples include the total free storage, the average load, or the size of the network. Efficient protocols for computing several aggregates such as average, count, and variance have already been proposed. In this paper, we consider calculating the* rank *of nodes, where the set of nodes has to be sorted according to a numeric attribute and each node must be informed about its own rank in the global sorting. This information has a number of applications, such as slicing. It can also be applied to calculate the median or any other percentile. We propose* T-RANK*, a robust and completely decentralized algorithm for solving the ranking problem with minimal assumptions. Due to the characteristics of the targeted environment, we aim for a probabilistic approach and accept minor errors in the output. We present extensive empirical results that suggest near logarithmic time complexity, scalability and robustness in different failure scenarios.*

1. Introduction

The large scale and extreme dynamism of current distributed systems pose special challenges to developers: monitoring and control requires the orchestration of a huge number of nodes, with a continuous flow of nodes joining and leaving the system. Special middleware services are required that shield the application from the resulting unpredictability of the environment.

One such important service is *aggregation* [1]. Aggregation is a common name for a set of functions that provide a summary of some global property in a distributed system. Possible examples include the network size, the total free storage, the maximum load, the average uptime, location and description of hotspots, etc. The computation of simple aggregate values can be used to support more complex protocols. For example, the knowledge of average load in a system can be exploited to implement near-optimal load-balancing schemes [2].

Many existing aggregation solutions are *reactive* [3, 4]: aggregation is triggered by a specific query issued by a node, and the answer is returned to the issuer. Instead, we are interested in *proactive* protocols, where results are continuously made available to all nodes. Proactive protocols are useful when aggregation is used as a building block for other decentralized algorithms, as in the load-balancing example cited above. Furthermore, proactive protocols are completely decentralized and "democratic", with every node participating equally, without any bottlenecks or points of failure.

Previous work exist [5,6] on gossip-based algorithms for computing a large collection of aggregates [7], including maximum, minimum, means, counting, sum, product, variance and other moments. Thanks to the gossip approach, the algorithms are characterized by extreme robustness and scalability, together with a very small communication cost. In this paper we tackle the ranking problem, that is closely related to the *sorting* problem, where the task is to sort the nodes according to their attributes; the additional goal is to inform all nodes about their own index (rank) in the global sorting.

In this paper we propose T-RANK, that, under minimal assumptions, creates and overlay representing a sorted list *and* informs all nodes about their rank in (empirically) logarithmic time using a logarithmic number of messages per node.

There are countless protocols and applications that maintain or rely on a sorted list/ring overlay. We build on T-MAN [8] to create the list, and then we add long range links

*In Proc. IEEE SASOW 2008, DOI 10.1109/SASOW.2008.17. This work was partially supported by the Future & Emerging Technologies unit of the European Commission through Project CASCADAS (IST-027807). M. Jelasity was supported by the Bolyai Scholarship of the Hungarian Academy of Sciences.

to this topology in an informed manner so that ranking information can be propagated in a logarithmic time. Our contribution lies in the *scalability, speed and small cost* of obtaining ranking information *from scratch*, without assuming the existence of a structured overlay.

The outline of the paper is as follows. In Section 2 we define the system model. Section 3 presents the problem, describes the core idea of the protocol and discusses the algorithmic details of the protocol. Section 4 presents simulation results. Finally, related work and conclusions are given in Section 5 and Section 6.

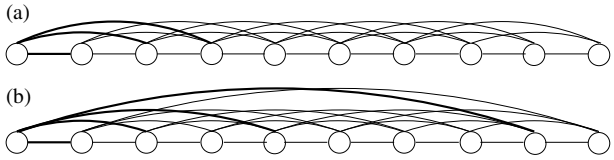

Figure 1. (a) A linear lattice topology, with $K = 3$. (b) A finger-based topology, showing links to nodes whose distance is equal to 2^i, for $i = 0 \ldots 3$. In both cases, the links of the first node are highlighted to ease their identification.

2. System Model

We consider a network consisting of a large collection of *nodes* that are assigned unique identifiers and that communicate through message exchanges. The network is highly dynamic; new nodes may join at any time, and existing nodes may leave, either voluntarily or by *crashing*. For the sake of simplicity, in the following we limit our discussion to node crashes, that is, we treat nodes that leave voluntarily as crashed nodes. This clearly represents a worst case scenario, since we could add special procedures to handle node leaves. Byzantine failures, with nodes behaving arbitrarily, are excluded from the present discussion.

We assume that nodes are connected through an existing routed network, such as the Internet, where every node can potentially communicate with every other node. To actually communicate, a node has to know the identifiers of a set of other nodes (its *neighbors*). A neighborhood relation over the nodes defines the topology of an *overlay network*. Given the large scale and the dynamism of our envisioned system, neighborhoods are typically limited to small subsets of the entire network. The neighbors of a node (and so the overlay topology) can change dynamically.

Communication incurs unpredictable delays and is subject to failures. Single messages may be lost, links between pairs of nodes may break. Occasional performance failures in communication (e.g., delay in receiving or sending a message in time) can be seen as general communication failures, and are treated as such. Nodes have access to local clocks that can measure the passage of real time with reasonable accuracy, that is, with small short-term drift.

3. The Algorithm

This section gives a formal description of the ranking problem and the basic concepts, along with the solution we propose: the T-RANK algorithm.

3.1. Definition of the Problem

As mentioned before, all nodes in the system hold a value that is used in the sorting problem. For the sake of simplifying language, we will often refer to the value as if it was the node itself.

The input of the ranking problem is a set $\mathcal{N}$ of N nodes, together with a total ordering relation $\preceq$, defined over $\mathcal{N}$. We assume that, given two nodes r and q, each node can establish whether $r \preceq q$ or $q \preceq r$, that is, nodes know and can apply the ordering relation. We define a *ranking distance* function $d : \mathcal{N} \times \mathcal{N} \rightarrow \mathbb{Z}$, where $d(r, q)$ is equal to number of "hops" that must be traversed to go from one node to the other:

$$d(r, q) = |\{\ r' \mid \min(r, q) \prec r' \preceq \max(r, q)\ \}|$$

The goal of the protocol is to compute the *ranking position* of each node in the ordered sequence defined by $\preceq$, corresponding to its distance from the first node of the sequence (i.e., the one with the minimum value), and to also inform each node about its rank.

Motivated by the arguments given in the Introduction, we are interested in a completely decentralized solution, where each node participates in a "democratic" way (i.e., with the same amount of resources) in the computation of the ranking, using only local information.

3.2. The Idea

The idea behind the proposed solution is the following: if we can efficiently build a structured overlay topology over the set of nodes that reflects the $\preceq$ order relation, that is, that embeds the ordering as a linked list, we can use it to (i) discover the first node in the sequence (and thus its rank: 1), and (ii) propagate rank information following the overlay links and we can also add shortcuts to the overlay defining the ordering so as to facilitate the propagation of the rank information.

The sorted list/ring overlay, enhanced with shortcuts, is by now a standard component of a wide class of distributed algorithms, mainly distributed hash tables (DHTs). It is therefore important not to confuse our proposal with DHTs. Our goal is to build the structure quickly and cheaply from scratch, dynamically, perhaps for several attributes simultaneously or sequentially. The structure itself will often be only temporary, needed only until ranks have been calculated. The design goal of DHTs, where maintaining the

structure is the key goal, is therefore not appropriate. Accordingly, known DHT algorithms are not applicable, as they solve a different problem.

Let us introduce some notations. The topology that embeds the ordering will be a *one-dimensional linear lattice* topology, illustrated in Figure 1(a). Each node r is connected to the nodes whose ranking distance is less than a configuration parameter $K \geq 1$. We call these nodes *leafs*; each node r will maintain two distinct leaf vectors, called $leaf_P$ and $leaf_S$, respectively containing nodes that preceed (*predecessors*) or succeed r (*successors*):

$$leaf_P{}^r[i] = \begin{cases} r' \text{ if } d(r,r') = i \text{ and } r' \preceq r \\ \perp \text{ if no such } r' \text{ exists} \end{cases}$$

$$leaf_S{}^r[i] = \begin{cases} r' \text{ if } d(r,r') = i \text{ and } r \preceq r' \\ \perp \text{ if no such } r' \text{ exists} \end{cases}$$

The length of these vectors is the number of non-$\perp$ elements. The length is at most K but sometimes smaller: obviously, those nodes that are closer to the beginning or the end of the ordering than K will not have K nodes preceeding them or succeeding them, respectively. Also note that the larger K is, the higher the probability is that the overlay network will not get partitioned due to node or link failures.

Once this network is available, a trivial solution to the ranking problem is the following: the nodes whose $leaf_P$ set is smaller than K can easily compute their rank, which is equal to the number of $leaf_P$ entries. Whenever a node r discovers its rank v, it sends a message to each node $q = leaf_S[i]$, informing q that its rank is equal to $v + i$. It is easy to see that this algorithm will eventually lead to each node knowing its rank in the total order $\preceq$.

The problem with this solution is the number of steps required to complete the algorithm, which is $O(N)$. To improve the speed of convergence, we build a *finger-based* topology, as shown in Figure 1(b), where nodes are connected to "distant" nodes in the ordered sequence. These nodes are called *fingers*. In our solution, we want to build a target topology, where the finger set of a node r contains all nodes whose distance from r is equal to 2^i, for $i \geq 0$. As with leafs, each node r organizes the information about fingers in two vectors $finger_P$ and $finger_S$, with predecessor and successors fingers, respectively:

$$finger_P{}^r[i] = \begin{cases} r' \text{ if } d(r,r') = 2^i \text{ and } r' \preceq r \\ \perp \text{ if no such } r' \text{ exists} \end{cases}$$

$$finger_S{}^r[i] = \begin{cases} r' \text{ if } d(r,r') = 2^i \text{ and } r \preceq r' \\ \perp \text{ if no such } r' \text{ exists} \end{cases}$$

Note that definition of fingers given here is different from the one of Chord [9]. Our fingers are defined based on their distance between their index over the sorted listed of nodes, while Chord fingers are defined based on the distance in the identifier space. This is clearly motivated by the specific goal of our protocol, and can be very significant if the distribution of attribute values (that we cannot control, unlike node IDs in Chord) is far from uniform.

The propagation algorithm can now be modified to exploit also the fingers: a node r with rank v can send a message to its finger $q = finger_S[i]$ informing it that its rank is $v + 2^i$. It is easy to see that, in the absence of failures, the number of steps needed to complete the algorithm is $O(\log N)$, thanks to the exponential distance of fingers, assuming that all fingers are informed in a single timestep.

In the rest of this section, we provide the algorithmic details of the protocol. For building and maintaining the ordering topology, we rely on T-MAN [8]. T-MAN is a gossip-based protocol scheme for the construction of several kinds of topologies. We provide a brief description of T-MAN below; interested readers may refer to the original paper for details [8]. Subsequently we focus on the description of T-RANK, the algorithm used to discover fingers and propagate rank information.

3.3. The T-MAN Algorithm

T-MAN is a gossip-based protocol scheme for the construction of several kinds of topologies. Each node maintains a list of neighbors. This list is of a fixed size, and updated periodically through gossip. In a gossip step, a node contacts one of its neighbors, and the two peers exchange their lists of neighbors, so that both peers have two lists: their old list and the list of the selected neighbor. Subsequently both participating nodes update their lists of neighbors by selecting the new list from the union of the two old lists. The key is how to select peers for a gossip step, and how to update the list of neighbors based on the two lists.

In T-MAN, the peer selection and the list update functions are implemented based on a *ranking function* (not to be confused with the ranking in this paper). The ranking function can be used to sort the list of neighbors to create an order of preference. This order of preference can be used to select peers and to update the list. The ranking function of T-MAN is a generic function and it can capture a wide range of topologies from rings to binary trees, from n-dimensional lattice to sorting. In particular, in the case of sorting, the order of preference is defined by the function $d(r, r')$ as defined previously.

As described in [8], T-MAN is able to construct overlay topologies in logarithmic time, with high accuracy.

3.4. The T-RANK Algorithm

The T-RANK algorithm is illustrated in Figure 2. Even though the system is not synchronous, we find it convenient to describe the protocol execution in terms of consecutive real time intervals of length δ called *cycles*. We describe the algorithm following its organization, namely introducing variables and discussing their initialization first; then, we present the periodic section, whose task is to discover new fingers and propagate ranking information.

As anticipated above, each node maintains four vectors $leaf_P$, $leaf_S$, $finger_P$ and $finger_S$. The first two contain the leafs, as obtained by T-MAN. The last two should contain

// Variables
Node[ ] $leaf_P$, $leaf_S$, $finger_P$, $finger_S$
int[ ] $dist_P$, $dist_S$
Set $next_P$, $next_S$
Set $newleafs$, $newfingers = \emptyset$
int $rank = -1$

// Initialization:
$leaf_P$ and $leaf_S$ are initialized by T-MAN, with K leafs
Init $finger_P$, $finger_S$ based on $leaf_P$, $leaf_S$
foreach i **do** $dist_P[i] = dist_S[i] = 2^i$
$next_P = \{\ i \mid finger_P[i] \neq \perp\ \}$
$next_S = \{\ i \mid finger_S[i] \neq \perp\ \}$
if $(|leaf_P| < threshold)$
  $newleafs = \{\ i \mid leaf_S[i] \neq \perp\ \}$
  $newfingers = \{\ i \mid finger_S[i] \neq \perp\ \}$
  $rank = |leaf_P|$

repeat periodically every δ time units

  // Send rank
  foreach $i \in newleafs$:
    send $\langle$RANK, $rank + i\rangle$ **to** $leaf_S[i]$
  foreach $i \in newfingers$:
    send $\langle$RANK, $rank + dist_S[i]\rangle$ **to** $finger_S[i]$
  $newfingers = newleafs = \emptyset$

  // Send fingers
  $mask = next_P \cup next_S$
  foreach i : $finger_P[i] \neq \perp$ **and** tosend(i)
    send $\langle$VIEW$_S$, $dist_P[i], finger_S \cap mask, dist_S \cap mask\rangle$
      to $finger_P[i]$
  foreach i : $finger_S[i] \neq \perp$ **and** tosend(i)
    send $\langle$VIEW$_P$, $dist_S[i], finger_P \cap mask, dist_P \cap mask\rangle$
      to $finger_S[i]$
  $next_P = next_S = \emptyset$

on receive $\langle$VIEW$_t, d, f_q, d_q\rangle$
  foreach $f_q[i]$:
    $e = d_q[i] + d, l =$ bits(e)
    if $(finger_t(l) == \perp$ **or** $f_q[l] \prec finger_t(l))$
      if $(t == S$ **and** $rank \geq 0)$
        $newfingers = newfingers \cup \{\ l\ \}\ \}$
      $finger_t[l] = f_q[i]$
      $dist_t[l] = e$
      $next_t[l] = next_t \cup \{\ l\ \}$

on receive $\langle$RANK, $r\rangle$
  if $(r > rank)$
    $rank = r$
    $newleafs = \{\ i \mid leaf_S[i] \neq \perp\}$
    $newfingers = \{\ i \mid finger_S[i] \neq \perp\}$

Figure 2. T-RANK Algorithm.

fingers whose distance is equal to 2^i; due to failures, however, discovering nodes at the required distance may be impossible. For this reason, the *finger* vectors are allowed to store nodes whose distance is smaller than required, and two *dist* vectors are created to contain the actual distance of nodes. If a finger is discovered with distance d included in $[2^i, 2^{i+1} - 1]$, it is stored in $finger_t[i]$ (where t corresponds to the appropriate direction); furthermore, value d is stored in $dist_t[i]$.

In addition to these vectors, four variable sets are maintained. Their goal is to reduce the amount of messages sent by the algorithm, by storing information about the nodes that need to be updated. In particular, $newleafs$ and $newfingers$ contain the indexes of the nodes to which the rank information need to be propagated, while $next_P$ and $next_S$ contain the indexes of the new discovered predecessor and successor fingers. All these sets trigger the sending of corresponding messages in the periodic section of the algorithm, after which they are emptied.

Finally, variable $rank$ contains the current estimate of the rank position. $rank$ is initialized to -1 to denote that the node does not know its position yet.

The algorithm initialization is as follows. First, the *leaf* and *finger* vectors are initialized as described in Section 3.2, and the *dist* vectors are set accordingly. Second, nodes that are beginning of the ordered sequence (recognized by a $leaf_P$ set smaller than a given $threshold$) initialize their rank based on the cardinality of $leaf_P$ and update their $newleafs$ and $newfingers$ sets to start sending ranking messages to their neighbors.

The core of the algorithm is given by the periodic sending of messages and their handling. Two kinds of messages are sent: RANK are used to notify nodes with their rank position, while VIEW messages are used to build the finger table. Communication is one-way; as we will see in Section 4, the algorithm is capable of dealing with message losses.

RANK messages are sent to all nodes in $newleafs$ and $newfingers$; the rank value contained in them is computed by adding the distance of the destination node (obtained by the position in $leaf_S$ or the distance in $dist_S$) to the rank of the local node. After the sending of the message, $newleafs$ and $newfingers$ are emptied, to avoid further sending of the same value. When a RANK message containing a new rank value is received, the node updates its local value and stores all leafs and fingers in $newleafs$ and $newfingers$, to propagate the new rank value to its successor neighbors. Note that the rank value is considered new only if it is greater than the previous value; this is because in case of a non-perfect leaf ordering (as the one produced by T-MAN), the estimate of this value can be initially smaller than the real value.

Finger tables are built in the following way. Each node sends a VIEW message containing its predecessor fingers to its successor ones, and a message containing its successor fingers to its predecessor ones. In this way, at each cycle a node discovers nodes that are progressively further away from itself; for example, when a node p receives from its

successor q with distance 2^i a successor finger r of q whose distance is 2^i, it discovers that r is distant 2^{i+1} and can fill the corresponding entry in $finger_s$. In case of failures, if a node p receives a message from a non-perfect successor finger q with distance in $[2^{i-1}, 2^i - 1]$, containing a non-perfect successor finger r with distance in $[2^{i-1}, 2^i - 1]$ from q, the distance from p to r is in the range $[2^i, 2^{i+1} - 1]$ and r can fill the corresponding entry.

To avoid sending an excessive amount of information, just new fingers (the one stored in $mask = next_p \cup next_s$) are sent to the opposite nodes. In the algorithm, we abuse of notation by writing $finger_t \cap mask$ and $dist_t \cap mask$ $(t = P, S)$, to indicate this restriction. Clearly, if the $next_p$ or $next_s$ are empty, the corresponding message is not sent.

Function tosend() is used in the figure to determine the set of fingers to which the VIEW message has to be sent. For the moment, we consider a function that returns always true, meaning that fingers are propagated to all nodes. This is the safest assumption in the case of failures, but also the more costly one. We will see alternative possibilities in Section 4.

When a VIEW message is received, the node verifies whether some of the nodes received may be used to insert a new entry or replace an existing one in the finger table. The predefined function bits(x) returns i if x is contained in $]2^{i-1}, 2^i]$. If a new finger is found, it is added to $next_p$ or $next_s$; if it is a successor finger, and the node has already received a rank estimate (certified by $rank \geq 0$), $newfingers$ is updated as well.

4. Evaluation

All experiments in the paper were performed with PEERSIM, a simulator optimized for executing cycle-based protocols such as T-RANK [2]. In all figures, 20 individual experiments were performed. Averages computed over all experiments are shown as curves. In most of the experiments, the empirical variance of the results is very low. When this is not true, the variance has been shown through error bars.

We present two sets of results. The first set is performed over a perfect regular lattice, where each node is connected to the K nodes that preceed it and to the K nodes that succeed it in the linear ordering. The perfect regular lattice can be produced by T-MAN in the absence of failures; these simulations serve thus as a baseline for comparison with the experiments on non-perfect lattices and to illustrate the robustness of the T-RANK algorithm with respect to node failures. The second set presents more realistic results based on the topologies constructed by the T-MAN distributed protocol. The extreme robustness is confirmed, even with the suboptimal topologies constructed by T-MAN. For all the simulations, the value of K is equal to 20. This means that the leaf degree of the nodes equals to $2K = 40$: this value is also suggested by empirical results with T-MAN [8], and has proven to be sufficient to obtain good approximations even in very large networks.

To evaluate our protocol, we are also interested in the following metrics: *convergence speed* and *communication cost*. Regarding convergence speed, we are interested in how many steps are needed to inform all nodes about their rank. Regarding communication cost, two kinds of messages are sent, *rank* and VIEW. The latter ones represent the higher cost, because they are sent in each cycle to all the current fingers.

Orthogonal to these figures of merit, we are interested also in the scalability and robustness characteristics.

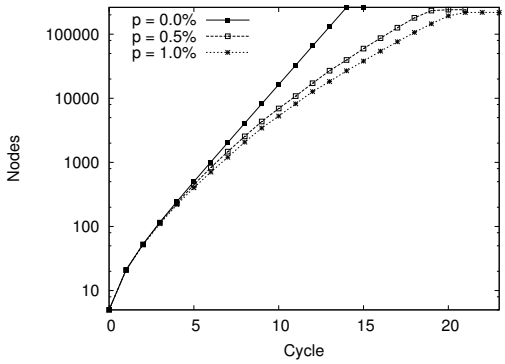

Figure 3. Number of correct nodes that have learned the exact ranking after each cycle. Network size is 2^{18}.

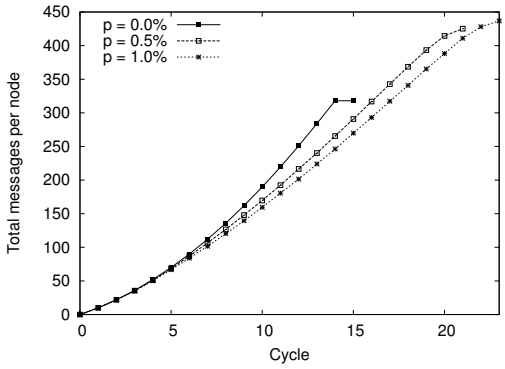

Figure 4. Number of VIEW messages exchanged after each cycle. Network size is 2^{18}.

4.1. Simulation Experiments

Figures 3 and 4 show the behavior of the protocol when executed starting from a perfect lattice. The T-RANK protocol was executed on a simulated network of 2^{18} nodes. Three curves are shown, corresponding to a failure probability of 0%, 0.5% and 1% per cycle If we consider $1s$ cycles, the latest probability is extremely high, approximately two order of magnitudes larger than what you observe in normal

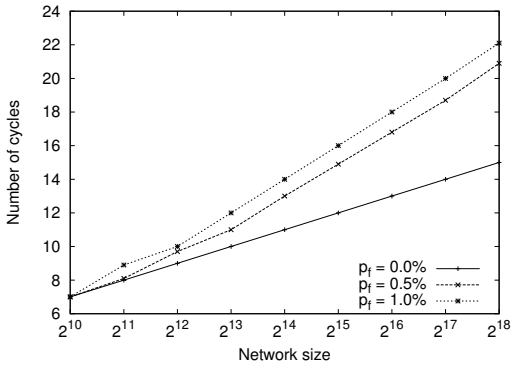

Figure 5. Number of cycles needed to complete the protocol, on networks with size in the range $[2^{10}, 2^{18}]$.

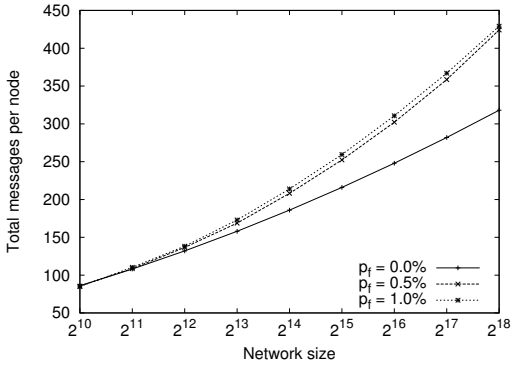

Figure 6. Total number of VIEW messages sent per node to complete the protocol, on networks with variable size in the range $[2^{10}, 2^{18}]$.

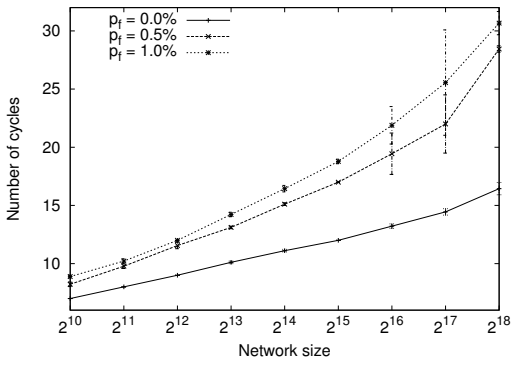

Figure 7. Number of cycles needed to complete the protocol, on networks with variable size in the range $[2^{10}, 2^{18}]$.

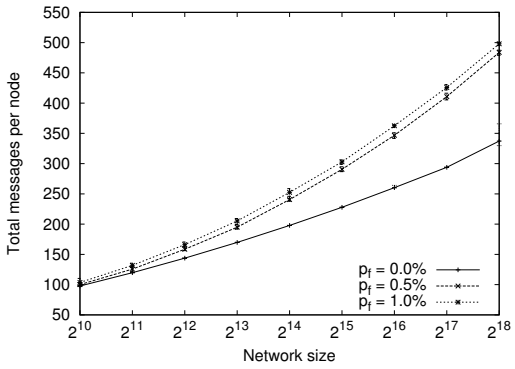

Figure 8. Total number of VIEW messages sent per node to complete the protocol, on networks with variable size in the range $[2^{10}, 2^{18}]$.

P2P systems.

Figure 3 shows the number of nodes that have obtained the correct estimation of the rank after each cycle. In the absence of failures, the number of nodes knowing their correct rank grows exponentially. In case of failures, the growth is slightly slower, due to the impossibility to discover some of the farthest fingers. In both cases, the number of cycles to complete the rank estimation is reasonably low.

Figure 4 shows the total number of VIEW messages exchanged per node after each cycle. In the absence of failures, if a node p knows a finger whose distance is 2^i, at the next cycle it will discover a link whose distance is 2^{i+1} (if such finger exist). This quickly leads to completion of the finger tables of all nodes, after which no VIEW messages are sent. Failures, on the other hand, may slow down the discovery process, as long-range fingers may not be present due to unavailability of nodes.

To illustrate the scalability of our protocol, we have tested it on networks with different sizes ranging between 2^{10} and 2^{18} nodes. Results are shown in Figures 5 and 6. As before, three curves are shown, corresponding to a failure probability of 0%, 0.5% and 1% per cycle. Figure 5 shows the number of cycles needed to complete the protocol, i.e. for all nodes to know their exact rank. Such desirable output has always been reached in all our simulations, independently of size and failure probability. As mentioned earlier, in a static network the number of cycles grows logarithmically with respect to the size of the network. The presence of failures slow down the algorithm, but only by a small constant factor.

Figure 6 shows the total number of VIEW messages per node. In this case, the growth is superlogarithmic with respect to the size of the network. Yet, the number of messages involved (around 300 in a static network with to 2^{18} nodes) is very small when compared to the size of the network itself.

Experiment 1		Experiment 2		Experiment 3	
error	# nodes	error	# nodes	error	# nodes
0	4252	0	144	0	2620
1	12407	1	807	1	187354
2	167688	2	3137		
3	3855	3	135296		
4	1149	4	50918		
5	921	207	1		
13	1	428	1		
1382	1	841	1		
		2652	1		

Table 1. Three independent runs as illustrative examples for the distribution of the error over the nodes.

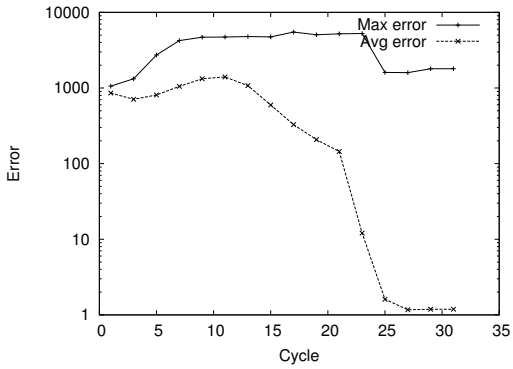

Figure 9. The error of ranking during a single run of T-RANK. Network size is 2^{18} and failure probability is $p = 1\%$. The initial network is obtained by the execution of T-MAN.

Figures 7 and 8 show the same scalability results, but starting from a topology built by T-MAN, instead of a statically generated network. In a static network, convergence is as quick as in the optimal case. The presence of failures, as before, may slow down the convergence. In the larger network, it is also possible that a perfect ranking is not reached. However, we can observe that T-MAN provides a sufficiently good sorting topology as the results are very close to that of the perfect sorting. In fact, the sorting generated by T-MAN is almost perfect, only very few nodes are misplaced. To illustrate this, consider Table 1, where the detailed distribution of the error (difference from correct rank) is shown along with the number of nodes with the difference in question. Three typical experiments are shown, with a network size of 2^{18} and failure probability of $p = 1\%$ per cycle. The number of nodes do not sum up to 2^{18} because of the large number of nodes that have crashed in the meantime. We can observe that there are very few outliers, and most of the nodes are very accurately ranked, especially considering the size of the network.

Figure 9 illustrates the same error distribution as a function of time. It depicts statistics of the error of ranking during a single run of T-RANK. The network size was 2^{18} with a failure probability of $p = 1\%$. The initial network was obtained by the execution of T-MAN. We can see that the average error is very low while the maximal error is reltively hight. Fortunately, as illustrated by Table 1, the maximal value is represented by outliers that form an ignorable minority.

4.2. Optimization

It is possible to reduce the number of messages that need to be sent during the running of T-RANK. In this section we present two ideas and illustrate them through simulation experiments, based on the perfect sorting as input.

As a first possibility for optimization, we can reduce the number of messages sent by changing function tosend(). If tosend(i) returns true only if $i \in mask$, with $mask$ computed as $next_p \cup next_s$, the algorithm converges as quickly as the original algorithm in the absence of failures, as shown in Figure 10. The number of messages exchanged is much smaller however as shown in the same figure.

Unfortunately, in the presence of failures, the convergence is much slower, particularly with large networks. The reason for this is that in the case of failures, if a node does not receive a finger with distance 2^i from a finger at distance 2^i (because the latter is crashed), it cannot find a finger of distance 2^{i+1}. However, it can still receive long-range fingers from other nodes whose distance is not exactly 2^i, and thus jump over the gap.

As a second idea of optimization, consider that we do not need to send messages to all fingers. If we modify function tosend(i) to return $i \in mask \vee toss(r)$, where $toss(r)$ returns true with probality r, we obtain a lighter version of the algorithm that sends messages to all nodes in $mask$, in addition to some other nodes choosen at random. Figure 11 shows the behavior of the algorithm for various values of r (where $r = 1$ corresponds to the tosend function that always returns true). As can be seen, for values of r as small as 20%, the convergence does not suffer, while the number of messages sent is greatly reduced.

5. Related Work

The several manifestations of the problem of sorting and ranking in distributed systems have long been an important area of research. Many, rather different definitions of the problem exist that can be classified according to the nature of the distribution of the data and the features of the networking environment, but in all cases it is the data that moves in the network, not the (overlay) network adapts to the data, as in our case. Nevertheless, to focus on those solu-

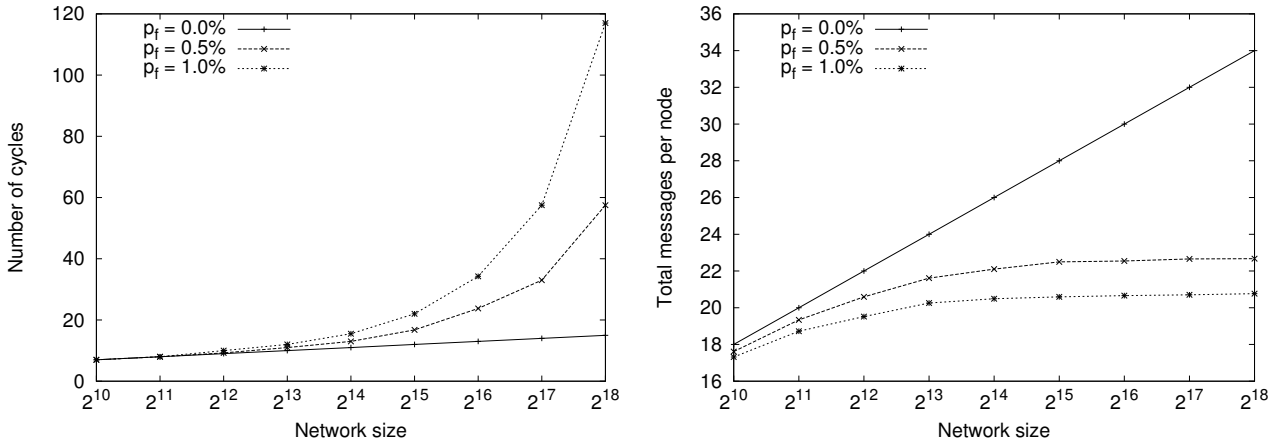

Figure 10. T-RANK with tosend(i) returning true only if $i \in next_P \cup next_S$. Figures show the number of cycles to reach perfect ranking, and the number of messages sent per node, respectively.

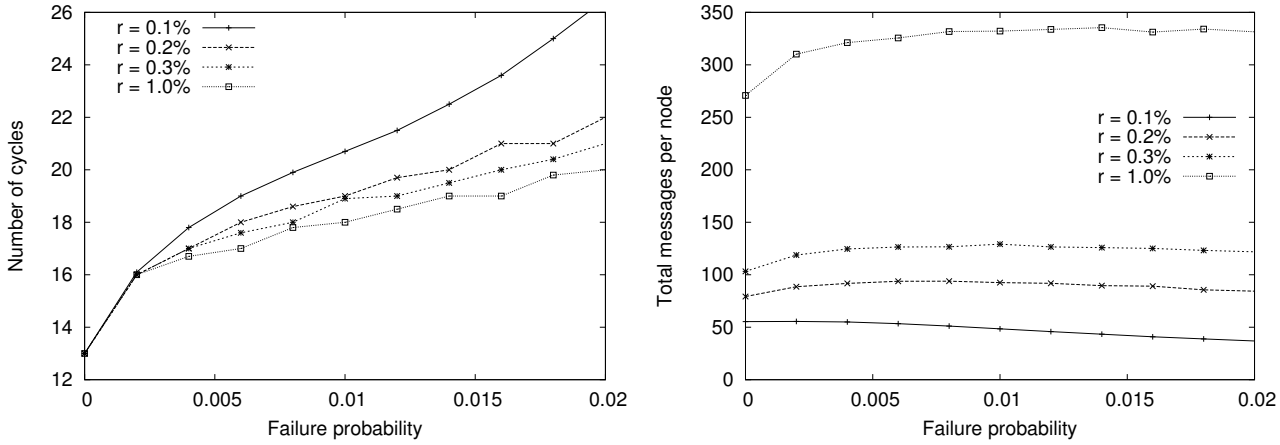

Figure 11. T-RANK with tosend(i) returning true only if $i \in next_P \cup next_S$ or with probability r. Figures show the number of cycles to reach perfect ranking, and the number of messages sent per node, respectively, in a network of 2^{18} nodes.

tions that were designed to operate in an unreliable environment, for example, Byzantine failure has been considered in a fixed hypercube topology [10] and dynamically changing values were tackled using the self-stabilization framework of Dijkstra [11], over a spanning tree topology. In comparison, our approach is probabilistic partly to deal with the extreme failure scenarios we are targeting, and partly because the goal is not ranking *per se*, but to apply ranking information for data aggregation, so absolute precision is not crucial.

Ordered slicing protocols [12, 13] are used to select the "best" $k\%$ nodes from a network. There is a clear relation between the problems of slicing and ranking: ranking is a possible implementation of slicing (although not the only possible implementation). Slicing protocols are often less rigorous, and cannot provide a precise ordering of nodes, even in the absence of failures.

As mentioned previously, the idea of long term links (fingers) added to a large diameter linear structure to facilitate information propagation is extremely common. Two well known examples are Chord and Pastry [9, 14]. The closest structures to our proposal are SkipNet and GosSkip [15,16], which is based on the idea of skip lists [17]. Our contribution however was not the invention of the structure itself but to propose a way to (i) build it very quickly and efficiently from scratch in order to use it in a dynamic setting and (ii) to

propose a protocol to utilize the structure to calculate ranks.

6. Conclusions

In this paper we have proposed T-RANK, a protocol for solving the ranking problem in large-scale, dynamic networks. The protocol bootstraps a one dimensional lattice overlay network representing the sorting of the nodes and assigns the ranks based on propagating rank information in this overlay network while simultaneously enhancing the overlay with long range links to facilitate the propagation process.

It has been pointed out that the speed of rank calculation is logarithmic if a sorted list overlay is given. It is also guaranteed to converge in the absence of failures. Most importantly, apart from these simple theoretical observations, we have presented extensive empirical evidence showing that the protocol can be practically implemented based on T-MAN, that provides the sorted list in approximately logarithmic time, and that it is scalable and robust to node failures (churn).

As of applicability, reasonably cheap information on ranking is potentially important in large scale dynamic distributed systems, where the shape of the distribution of many attributes could be unknown and can be very far from uniform. Ranking provides the basis to derive percentiles of the distribution, that can be used for slicing. We can also use ranking to help identify the distribution of a certain attribute.

References

[1] Robbert van Renesse, "The importance of aggregation," in *Future Directions in Distributed Computing*, André Schiper, Alex A. Shvartsman, Hakim Weatherspoon, and Ben Y. Zhao, Eds. 2003, number 2584 in Lecture Notes in Computer Science, pp. 87–92, Springer.

[2] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu, "A modular paradigm for building self-organizing peer-to-peer applications," in *Engineering Self-Organising Systems*. 2004, vol. 2977 of *Lecture Notes in Artificial Intelligence*, pp. 265–282, Springer.

[3] Indranil Gupta, Robbert van Renesse, and Kenneth P. Birman, "Scalable fault-tolerant aggregation in large process groups," in *Proceedings of the International Conference on Dependable Systems and Networks (DSN'01)*, Göteborg, Sweden, 2001.

[4] Robbert van Renesse, Kenneth P. Birman, and Werner Vogels, "Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining," *ACM Trans. Comput. Syst.*, vol. 21, no. 2, pp. 164–206, 2003.

[5] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM Trans. Comput. Syst.*, vol. 23, no. 1, pp. 219–252, Aug. 2005.

[6] Fetahi Wuhib, Mads Dam, Rolf Stadler, and Alexander Clemm, "Robust monitoring of network-wide aggregates through gossiping," in *Integrated Network Management*. 2007, pp. 226–235, IEEE.

[7] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, "Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals," *Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 29–53, 1997.

[8] Márk Jelasity and Ozalp Babaoglu, "T-Man: Gossip-based overlay topology management," in *Engineering Self-Organising Systems: Third International Workshop (ESOA 2005), Revised Selected Papers*, Sven A. Brueckner, Giovanna Di Marzo Serugendo, David Hales, and Franco Zambonelli, Eds. 2006, vol. 3910 of *Lecture Notes in Computer Science*, pp. 1–15, Springer-Verlag.

[9] Frank Dabek et al., "Building Peer-to-Peer Systems with Chord, a Distributed Lookup Service," in *Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, Schloss Elmau, Germany, May 2001, IEEE Computer Society.

[10] Bruce M. McMillin and Lionel M. Ni, "Reliable distributed sorting through the application-oriented fault tolerance paradigm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 4, pp. 411–420, July 1992.

[11] Gianluigi Alari, Joffroy Beauquier, Joseph Chacko, Ajoy K. Datta, and Sebastien Tixeuil, "Fault-tolerant distributed sorting algorithm in tree networks," in *IEEE International Performance, Computing and Communications Conference (IPCCC 1998)*, 1998, pp. 37–43.

[12] Márk Jelasity and Anne-Marie Kermarrec, "Ordered slicing of very large-scale overlay networks," In Montresor et al. [18], pp. 117–124.

[13] Antonio Fernández, Vincent Gramoli, Ernesto Jiménez, Anne-Marie Kermarrec, and Michel Raynal, "Distributed slicing in dynamic systems," in *ICDCS*. 2007, p. 66, IEEE Computer Society.

[14] Antony Rowstron and Peter Druschel, "Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems," in *Proc. of the 18th Int. Conf. on Distributed Systems Platforms*, Heidelberg, Germany, November 2001.

[15] Nicholas J. A. Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman, "Skipnet: A scalable overlay network with practical locality properties," in *USENIX Symposium on Internet Technologies and Systems*, 2003.

[16] Rachid Guerraoui, Sidath B. Handurukande, Kevin Huguenin, Anne-Marie Kermarrec, Fabrice Le Fessant, and Etienne Riviere, "Gosskip, an efficient, fault-tolerant and self organizing overlay using gossip-based construction and skip-lists principles," In Montresor et al. [18], pp. 12–22.

[17] W. Pugh, "Skip Lists: A Probabilistic Alternative to Balanced Trees," *Communications of the ACM*, vol. 33, no. 6, pp. 668 – 676, 1990.

[18] Alberto Montresor, Adam Wierzbicki, and Nahid Shahmehri, Eds., *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P 2006), 2-4 October 2006, Cambridge, United Kingdom*. IEEE Computer Society, 2006.