A P2P Architecture for a Large-Scale Dynamic Grid?

Márk Jelasity

University of Bologna
Italy
P2P and Grid

- **P2P**
  - ad hoc group
    - independent
  - fixed purpose
  - very large-scale
  - dynamic
  - unreliable

- **Grid**
  - more control over resources
  - general purpose (middleware)
  - not as large-scale
  - not as dynamic
  - not as unreliable
P2P
- ad hoc group
  - independent
  - uncontrolled
- fixed purpose
- very large-scale
- dynamic
- unreliable

Grid
- more control over resources
- general purpose (middleware)
- not as large-scale
- not as dynamic
- not as unreliable
Scenario from a user's point of view

- User prepares application against an API
  - API contains calls to services such as search, multicast, aggregation, monitoring, etc

- User is assigned a P2P network which supports the services needed by the application
  - assigned network is a real P2P network: dynamic, unreliable, maybe very large-scale, etc
  - but with the required services and contracts

- User executes the application

- The P2P network that was assigned is recycled
How to support this?

- A global pool needs to be maintained
- Partitions of this pool need to be separated and assigned
- The infrastructure that implements the services over the given partition needs to be built
- The application of the user needs to be executed
- Cleanup needs to be performed
For completeness...

The protocols that implement these functions need to be:
- lightweight
  - simple
  - efficient
- effective
- reliable
- self-managing (only high level human involvement)
The global pool

- requirements
  - provide a group abstraction
  - minimal functionality but
  - extremely reliable and robust

- solution
  - peer sampling service abstraction
    - getRandomPeer()

- implementation
  - unstructured overlay network maintained by gossip
  - newscast, lpbcast, cyclon, etc
The creation of partitions

- work in progress...
- partitions also implement peer sampling service
  - recursive structure
- partitions are abstract and maintained (filled in) dynamically by actual set of nodes
The creation of partitions

- Partitions are dynamic
  - they are constantly created and dissolved on demand
  - even when the set of partitions is constant, they change due to
    - churn
    - changing properties of nodes
- The main challenge is this dynamism
Building the architecture

- the applications will need services, such as a DHT or proximity overlay
- we need to build those on top of the respective partitions (ie, the peer sampling service)
Building the architecture

- achieved through the bootstrapping service
  - P2P services are based on overlay networks
  - the bootstrapping service must build arbitrary overlay networks (semantic, proximity, several DHT-s, etc) quickly, reliably and cheaply
  - should rely only on the peer sampling service
- we implement it based on a gossiping scheme as well, very similarly to the peer sampling service (T-Man)
Executing the application

- just vague ideas
  - content distribution to spread the clients
  - start them all in a reasonable interval
  - monitoring, administration, etc...
Cleaning up

- The easiest: do nothing
  - all constructs are disposable, the nodes involved in the partition will automatically join other partitions when their partition is removed
Some thoughts

- the components are useful outside of this vision too
  - maintaining a specified partitioning
  - bootstrapping different overlay networks (P2P services) on demand
A global pool needs to be maintained
  - peer sampling service: gossip-based implementation

Partitions of this pool need to be separated and assigned
  - creating the partitions is work in progress
  - separate peer sampling service for partitions (similar gossip-based implementation)

The infrastructure that implements the services over the given partition needs to be built
  - bootstrapping service: T-Man: gossip-based protocol too
Most problems boil down to building and maintaining overlay networks: the protocols then are simple

- peer sampling: random overlay
- bootstrapping: arbitrary overlay

Nodes are computing devices connected to a computer network

- Neighbours are defined by the “knows-about” relation (NOT physical neighbors in the network).
  - Can be easily adapted (unlike physical networks) by simply exchanging information
Overlay network

A → B → C → D → E → F

View of B:
- Descriptor of A
- Descriptor of C
- Descriptor of E
Solution: Self-organization through gossip

- The solution needs to be robust, scalable, efficient and cheap: needs to be fully distributed
  - No single point of failure, no bottlenecks, no critical components, negligible maintenance cost
- Fully distributed: need to involve self-organization
  - Gossip paradigm: a generalization of cellular automata
    - Arbitrary interconnection of nodes (cells)
    - Asynchronous
    - Nodes have arbitrary computing capabilities and send arbitrary messages
Gossip protocols for topology management
Gossip protocols for topology management

SelectPeer
Gossip protocols for topology management

Exchange of views
Gossip protocols for topology management

Both sides apply update thereby redefining topology
Gossip protocols for topology management

- Fully symmetric and decentralized model
- Components of the framework
  - **node descriptors**: in the view we store not only the address but additional information as well about the nodes
  - **selectPeer**: uses the actual view to select a peer to contact
  - **update**(view$_1$, view$_2$): based on information available on the peer nodes in the views (node descriptors) constructs the next view
peer sampling: a gossip protocol for unstructured topologies

- **Goal:** quickly generate and maintain a random network in the face of dynamism (churn, failures, etc)

- **node descriptors:** contain the address and creation time of descriptor

- **selectPeer:** Selects a young descriptor or a random one

- **update:** fills the view with the youngest descriptors or with a random subset
T-Man: a gossip protocol for structured topologies

- **Goal:** quickly generate and maintain a
  - A very wide range of pre-specified or even dynamically specified topologies
  - In the face of dynamism (churn, failures, etc)
- **node descriptors:** contain the **profile** of the node (real number, vector, etc)
- **selectPeer:** Ranks view using a **ranking function** that defines the target topology and selects a **low rank** neighbor
- **update:** fills the view with the **lowest rank descriptors**
Example 1 (ring and line): Let the nodes be real numbers. Let the ranking function be defined by the distance \( d(a,b) = |a-b| \). For the ring, apply periodic boundary conditions, assuming nodes are from \([0,N]\).

Example 2 (mesh and torus): Let the nodes be two dimensional real vectors. Similarly to the ring, let the Manhattan distance define the topology.
Initial state

Cycle 3

Cycle 5

Cycle 8

Cycle 12

Cycle 15
Evolution of a grid from random initial topology (15 cycles)
Time to reach perfect topology
What can we bootstrapped?

Sorting

Clustering
What can we bootstrapped?

- (geographical, semantic, etc) proximity overlays
  - simply with appropriate distance function
- Distributed Hashtable routing substrates
  - typically a distance function (eg ring) and additional data collection during the process
    - ring+fingers (Chord)
    - [ring+] prefix tables (Pastry, Kademlia, etc)
Example 4 (sorting): Let $\leq$ be a total ordering over the nodes. Let the ranking function apply a distance function consistent with $\leq$ separately to those $<$ and $>$ than the base node, and merge the ranked two subsets.

For example $R(10,\{1,2,4,100, 300\})$ could return $(4,100,2,300,1)$. No distance function over the set of nodes generates this ranking function!

Example 5 (2d proximity): Similar to sorting, classifying nodes into four subsets, ordering them according to distance and merging them.
Illustration of clustering and sorting
Summary

- presented an architecture for managing large P2P pools of resources
- presented some protocols that implement some of the functionality
- the architecture so far is very simple and lightweight: mostly gossip-based protocols