Decentralized Service Deployment for Collaborative Environments

Daniel Lázaro Iglesias Joan Manuel Marquès i Puig
Josep Jorba Esteve

Open University of Catalonia

First International Workshop on P2P, Parallel, Grid and Internet Computing, 2007

Outline

Context

- A group of people wants to work together through Internet.
- Groups formed spontaneously based on interests:
  - Groups of friends
  - Groups of people who share an interest
  - Campaigns from social and political activists
- No entity that automatically and transparently guarantees resources.

Context

Share resources. Ideally they would work:

- Only using resources provided by the members of the group.
- Guarranteeing that the system can self-organize itself.
- Each participant freely deciding which resources or services to contribute.

Context

They need a system to manage the collective use of the resources.

LaCOLLA middleware

A middleware that offers small groups basic general purpose functionality for collaborative activities:

- Presence information
- Location transparency
- Object storage
- Messaging
- Event dissemination
- Management of members and groups

Motivation

A user wants to offer a service, but lacks resources.

Example

- A Web server.
- Shared calendar.
- Audio/video streaming.

If the user belongs to a group which shares resources, he could use them to offer the service.

Two options

- Implement the service so that it uses the shared resources.
- ***Deploy the service in the group using a deployment mechanism.***

Proposal

We present a system that allows the deployment of stateless services in a group of computers scattered across the Internet,

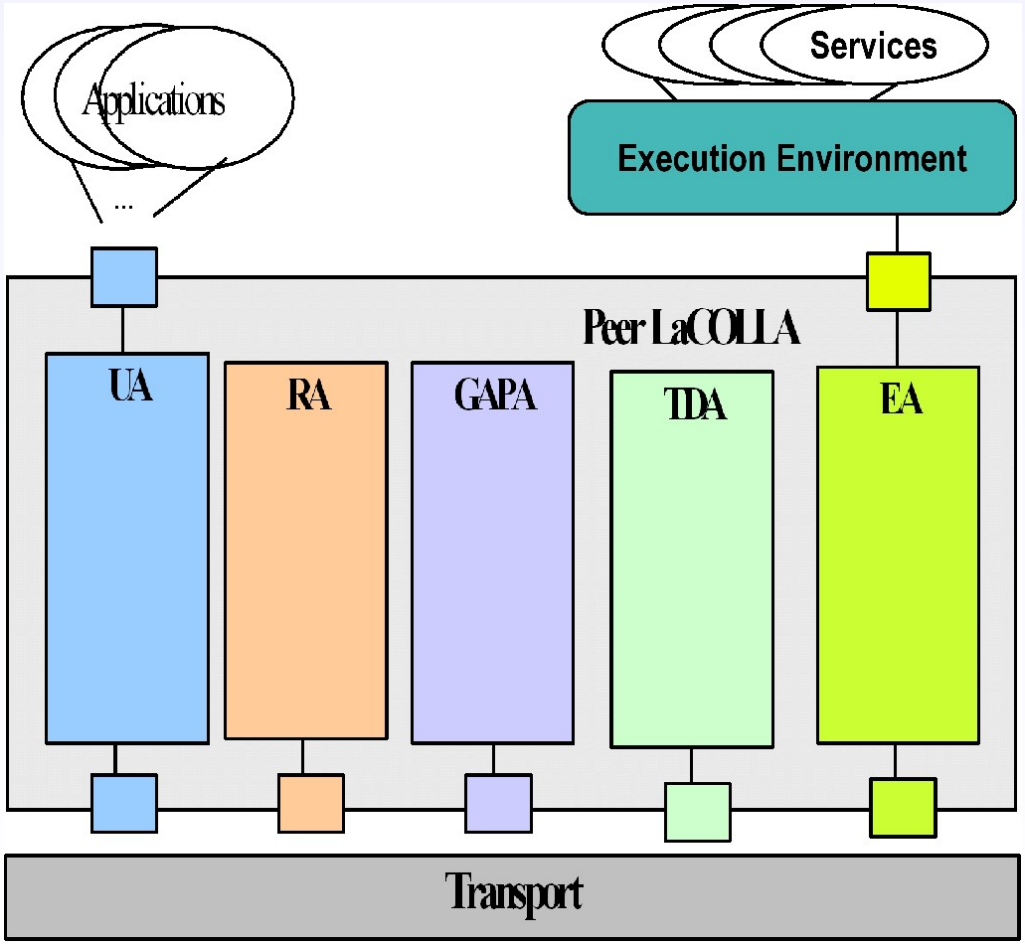
- in a decentralized manner,
- using only the resources provided to the group,
- keeping these services always available as long as there are enough resources provided to the group.

Requirements

- Group self-sufficiency.
- Decentralization.
- Self-organization.
- Individual autonomy.
- Service availability:
  - Replication.
  - Location transparency.

Components

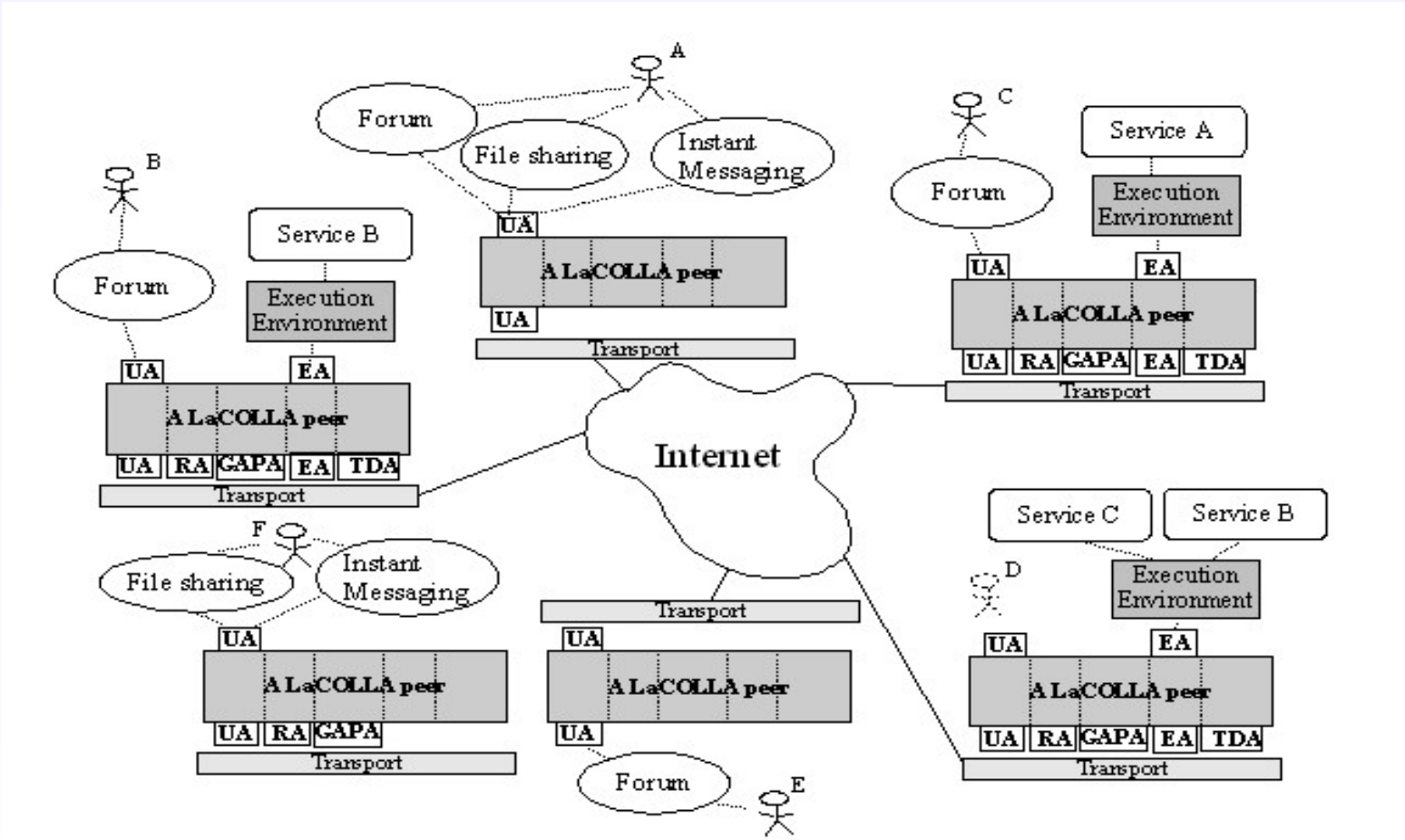
- User Agent (UA)
- Repository Agent (RA)
- Group Administration and Presence Agent (GAPA)
- Task Dispatcher Agent (TDA)
- Executor Agent (EA)



Execution Environment

- A component independent of the rest of the system.
- Connected to the Executor Agent:
  - The EA offers an API with operations the Environment can call to connect to the group.
  - The Environment must implement an interface to allow the EA to communicate with it.
- Several Execution Environment modules can be created and connected to the group to support services written in different languages and for different operating systems.

Snapshot of a group



Mechanisms must manage...

Existing services

Keeping information about existing services.

- Information must be stored persistently.
- It (probably) won't change frequently.

Active services

Managing running services.

- Must keep services available.
- Information potentially changes quickly.

Existing services management

Mechanisms used:

- Application-level multicast:
  - When a new information is produced, a message is sent to all the interested components.
- Epidemic dissemination:
  - TDAs periodically carry out bidirectional anti-entropy sessions with a given number of randomly chosen TDAs.
  - UAs periodically contact a random TDA in order to update their information.

We take an optimistic approach.

Active services management

- Each service is assigned to a TDA at activation.
- This TDA will be the master for the service:
  - Chooses the nodes where the service will be executed.
  - Periodically checks that the current number of replicas is the apporpiate.
  - Detects stopped service replicas.
  - Stops or activates new replicas when needed.
- There's a set of secondary masters which can take the primary's place in case of failure.

Master election

Two methods:

Pseudo-DHT

- Calculate proximity between TDA's id and service id using a hash function.
- The nearest TDA is the master.

Pseudo-CSMA/CD

- Master selected randomly.
- If master fails, secondaries start a negotiation:
  - Send a message and wait a random interval.
  - If no messages are received, become master.

Access to services

When a user wants to use a service:

- The user asks the UA the location of the service.
- The UA gives the user the location.
- The user accesses the service directly at its current location.

Validation

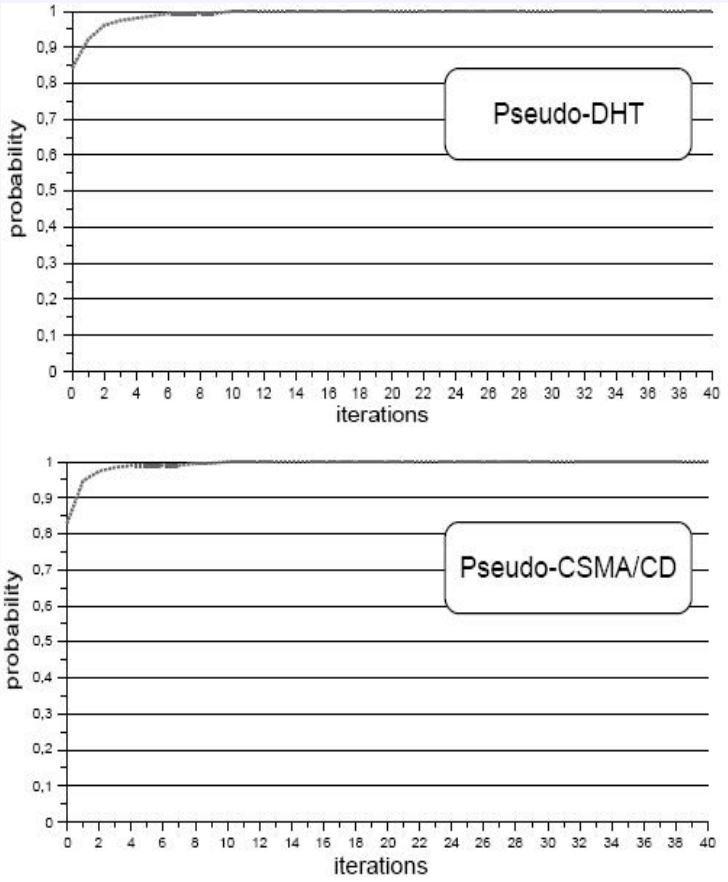
A prototype that implements:

- Service creation.
- Service activation.
- Service deactivation.

Simulations conducted in two phases:

- *Phase 1*: Activity simulation.
  - Service creation/activation/deactivation.
  - Component disconnection/failure.
- *Phase 2*: Only internal mechanisms work.
  - Measure how many iterations it takes to reach a consistent state.

Validation



Very similar results with both election mechanisms.
Consistency is reached in few iterations:

- 85% of cases, already consistent in the first iteration.
- 95% of cases, already consistent in two iterations.

Future work

- Implement service modification and elimination.
- Explore possible modifications of the proposed mechanisms to improve system's scalability.
- Allow the deployment of stateful services.

Questions?