A Peer-to-peer Extension of Network-Enabled Server Systems

Eddy Caron[1], Frédéric Desprez[1], Cédric Tedeschi[1]
Franck Petit[2]

1 - GRAAL Project / LIP laboratory

2 - LaRIA laboratory

E-Science 2005 - December 8, 2005

Outline

Outline

Context

- Number of resources grows every day
- Strong need of scalability of the grid middleware
- Network-Enabled Server Systems (GridRPC)

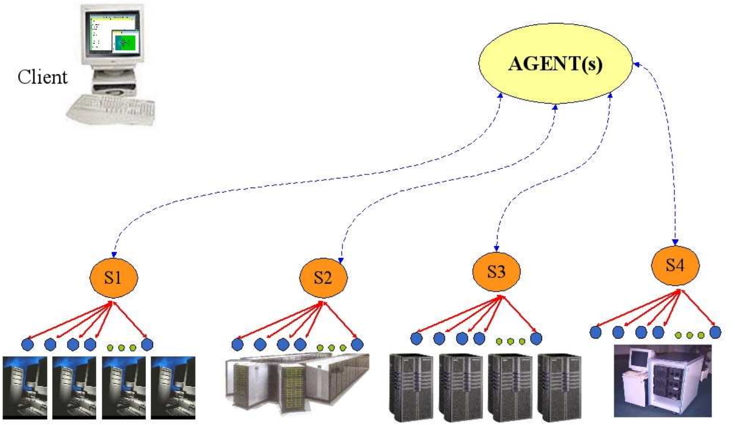
Context

- Number of resources grows every day
- Strong need of scalability of the grid middleware
- Network-Enabled Server Systems (GridRPC)

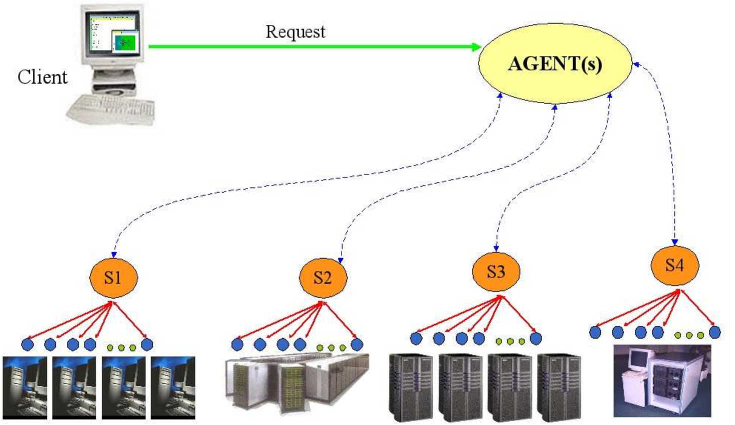
Context

- Number of resources grows every day
- Strong need of scalability of the grid middleware
- Network-Enabled Server Systems (GridRPC)

RPC and grid computing : The GridRPC paradigm



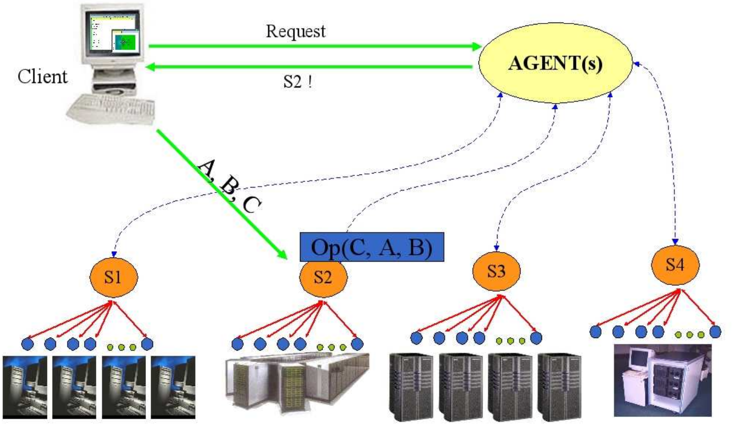
RPC and grid computing : The GridRPC paradigm



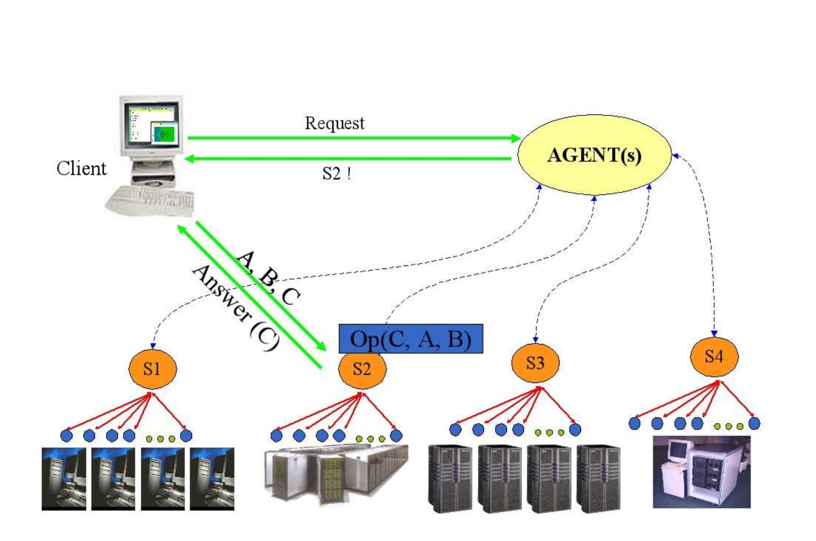
RPC and grid computing : The GridRPC paradigm

RPC and grid computing : The GridRPC paradigm

RPC and grid computing : The GridRPC paradigm



RPC and grid computing : The GridRPC paradigm



Outline

DIET components

- The Master Agent (MA)
  - Root of the tree
  - Entry Point to clients
  - Discovery and scheduling
- Agents (A)
  - Internal nodes of the tree
  - Transmission of the request
- Local Agents (LA)
  - Connected to servers
  - Gather information about servers
- Server Daemons (SeD)
  - Encapsulation of a computational server
  - Register to a LA (its parent)
  - List of data and problems available on it
  - Performance prediction

DIET components

- The Master Agent (MA)
  - Root of the tree
  - Entry Point to clients
  - Discovery and scheduling
- Agents (A)
  - Internal nodes of the tree
  - Transmission of the request
- Local Agents (LA)
  - Connected to servers
  - Gather information about servers
- Server Daemons (SeD)
  - Encapsulation of a computational server
  - Register to a LA (its parent)
  - List of data and problems available on it
  - Performance prediction

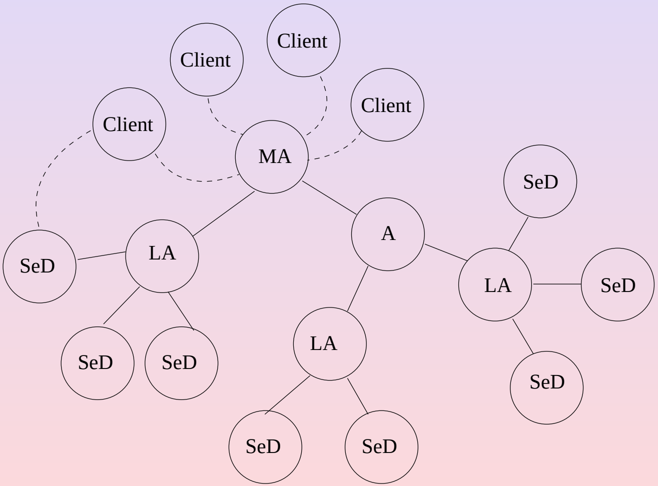
DIET components

- The Master Agent (MA)
  - Root of the tree
  - Entry Point to clients
  - Discovery and scheduling
- Agents (A)
  - Internal nodes of the tree
  - Transmission of the request
- Local Agents (LA)
  - Connected to servers
  - Gather information about servers
- Server Daemons (SeD)
  - Encapsulation of a computational server
  - Register to a LA (its parent)
  - List of data and problems available on it
  - Performance prediction

DIET components

- The Master Agent (MA)
  - Root of the tree
  - Entry Point to clients
  - Discovery and scheduling
- Agents (A)
  - Internal nodes of the tree
  - Transmission of the request
- Local Agents (LA)
  - Connected to servers
  - Gather information about servers
- Server Daemons (SeD)
  - Encapsulation of a computational server
  - Register to a LA (its parent)
  - List of data and problems available on it
  - Performance prediction

DIET hierarchical architecture



DIET Limits

- Master Agent
  - Single point of failure
  - Bottleneck
- Static configuration
  - Don't cope with the dynamic nature of large scale platforms
  - Deployed in one administrative domain
  - Clients are given unique static entry point
- Service discovery
  - DIET deployed in one administrative domain
  - Small scale service discovery
- Enhancing grids with the P2P technology
  - Widely suggested
  - Very few grid middleware have integrated it

DIET Limits

- Master Agent
  - Single point of failure
  - Bottleneck
- Static configuration
  - Don't cope with the dynamic nature of large scale platforms
  - Deployed in one administrative domain
  - Clients are given unique static entry point
- Service discovery
  - DIET deployed in one administrative domain
  - Small scale service discovery
- Enhancing grids with the P2P technology
  - Widely suggested
  - Very few grid middleware have integrated it

DIET Limits

- Master Agent
  - Single point of failure
  - Bottleneck
- Static configuration
  - Don't cope with the dynamic nature of large scale platforms
  - Deployed in one administrative domain
  - Clients are given unique static entry point
- Service discovery
  - DIET deployed in one administrative domain
  - Small scale service discovery
- Enhancing grids with the P2P technology
  - Widely suggested
  - Very few grid middleware have integrated it

DIET Limits

- Master Agent
  - Single point of failure
  - Bottleneck
- Static configuration
  - Don't cope with the dynamic nature of large scale platforms
  - Deployed in one administrative domain
  - Clients are given unique static entry point
- Service discovery
  - DIET deployed in one administrative domain
  - Small scale service discovery
- Enhancing grids with the P2P technology
  - Widely suggested
  - Very few grid middleware have integrated it

Outline

DIET$_J$: A P2P extension of DIET

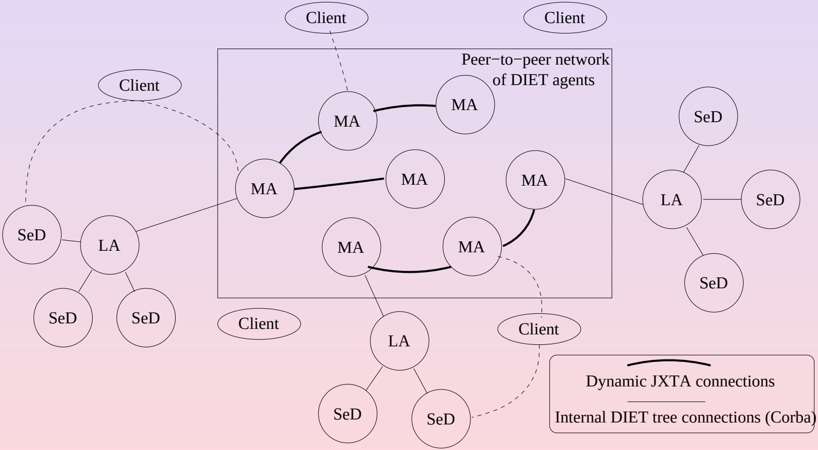
- Dynamically connecting hierarchies
- Balancing the load among the Master Agents
- Gathering services at larger scale

The JXTA Project

- Open source project initiated by SUN Microsystems
- Rich set of protocols for building P2P applications
- Basic logical entity : the peer
  - Edge peer
  - Rendezvous peer
  - Relay peer
- Communication services
  - Endpoint service
  - Pipe Service
  - JXTA Sockets
- Discovery done by advertisements

www.jxta.org

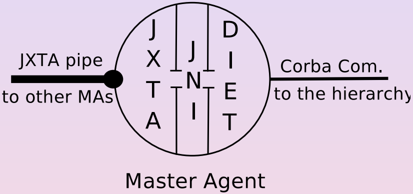
DIET$_J$ architecture



The Multi Master Agent System (1)

The Master Agent is divided into three parts :

- The DIET part
- The JXTA part
- The JNI part, interface between JXTA (Java) and DIET (C++)



The Multi Master Agent :

- Composed of all running MA reachable from a first MA
- All MA have a common advertisement's name

The Multi Master Agent System (2)

The Master Agents apply the following algorithm :

- Initialization of the JXTA part
- Initialization of the DIET part (via JNI)
- Publication of its advertisement
  - Short lifetime
  - Periodic republication
- On receipt of a client request
  - if no SeD matches the request (in its own subtree)
  - Discovers others MA thanks to their advertisement
  - Connects the other MAs
  - Propagates the request through the multi-hierarchy

Outline

Approach

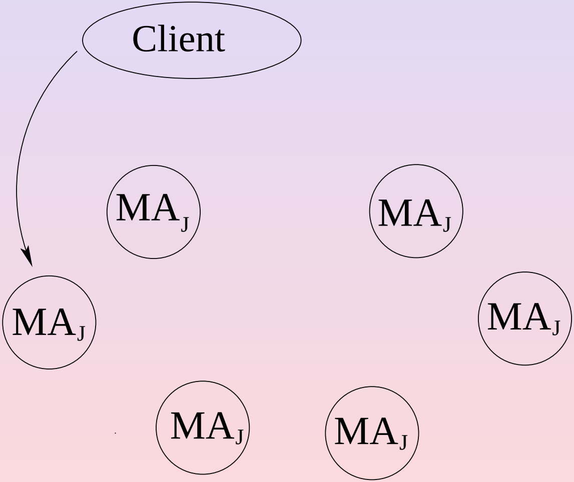
1. Peers (MA) discovery
   - Discovering running MA reachable
   - Thanks to the JXTA discovery process
     - DHT (structured)
     - flooding (unstructured)
2. Service discovery
   - Looking for the requested service in the whole multi-hierarchy
   - Exploration implemented with two algorithms
     - STAR_{async}
     - PIF_{async}

Implementation - STAR$_{async}$ (1)

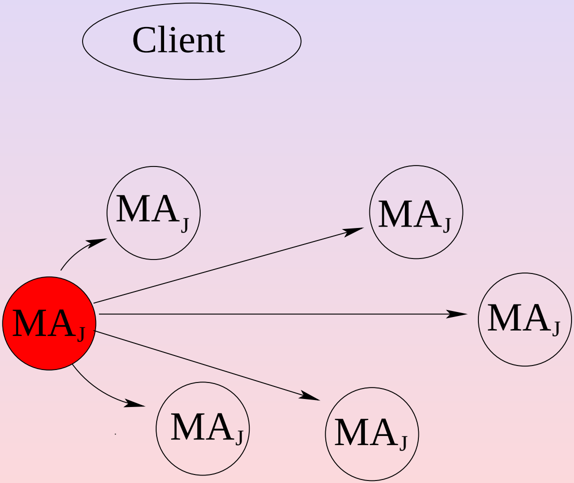
Propagation as an asynchronous star graph traversal

- On receipt of a client's request, fails retrieving the service in its own hierarchy (*root*)
  - Discovers other MAs (JXTA discovery)
  - Propagates the request to other MAs (multicast pipe)
  - Merges the answers sent back to the client
- On receipt of a propagated request
  - Submits the request to the local hierarchy
  - Sends the servers found back to *root*

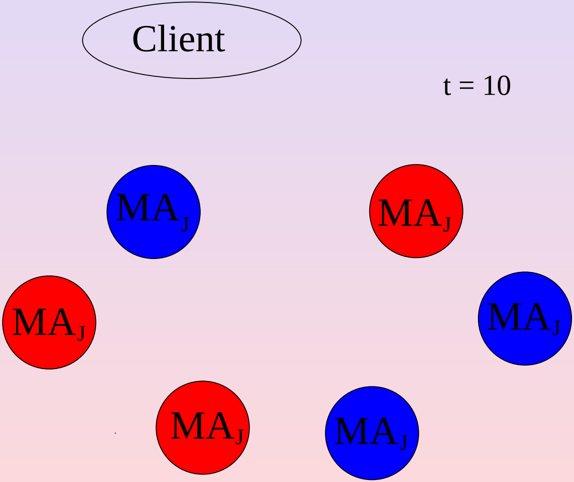
Implementation - STAR_{async} (2)



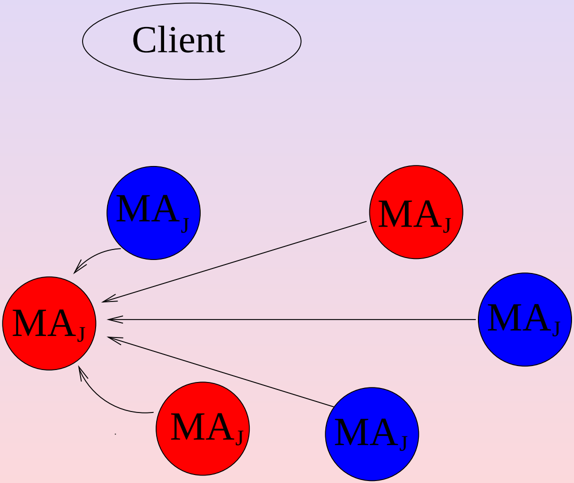
Implementation - STAR$_{async}$ (2)



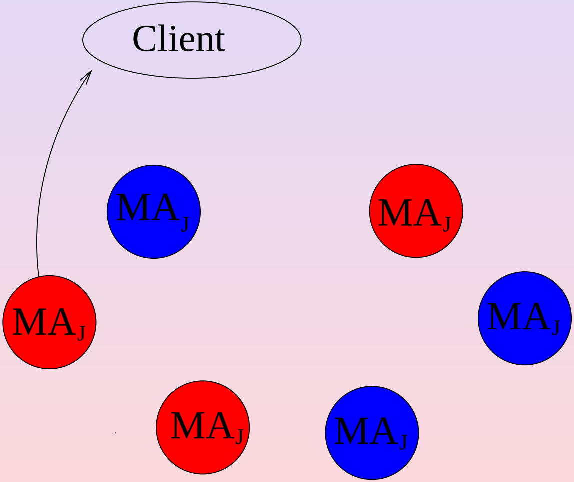
Implementation - STAR$_{async}$ (2)



Implementation - STAR$_{async}$ (2)



Implementation - STAR$_{async}$ (2)



Implementation (PIF$_{async}$) (1)

Propagation as asynchronous PIF scheme.

- Wave algorithm
- Build a time optimal spanning tree
- Made of two phases

Implementation (PIF_{async}) (2)

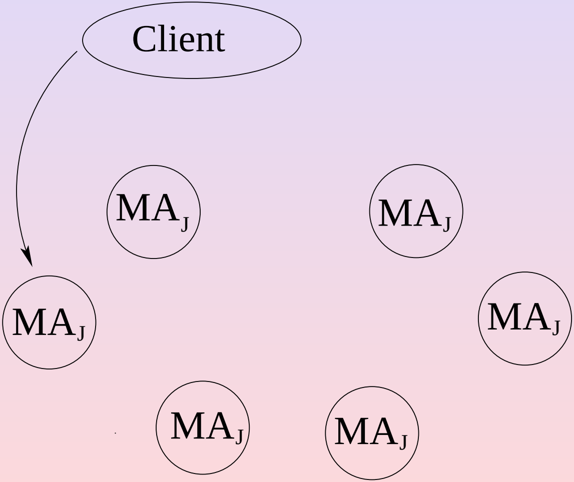
The Broadcast phase

- On receipt of a client's request, fails retrieving the service in its own hierarchy (*root*)
  - Discovers other MAs (*n*)
  - Initiates of the wave
  - Propagates the request to other MAs (JXTA multicast pipe)
  - Waits for *n* replies and sends them to the client
- On receipt of a propagated request
  - If already processed, ignores it.
  - else
    - The sender becomes its parent
    - Re-propagates the request to other MAs except its ancestors

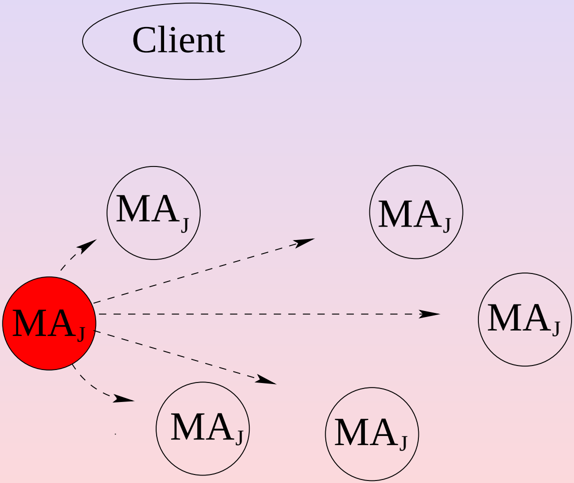
The Feedback phase

- Submits the request to the local hierarchy
- Sends the local servers found back to its parent
- Forwards the answers coming from children

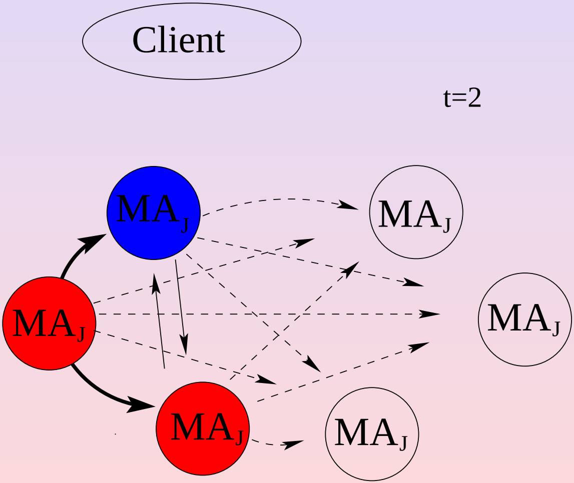
Implementation - PIF$_{async}$ (3)



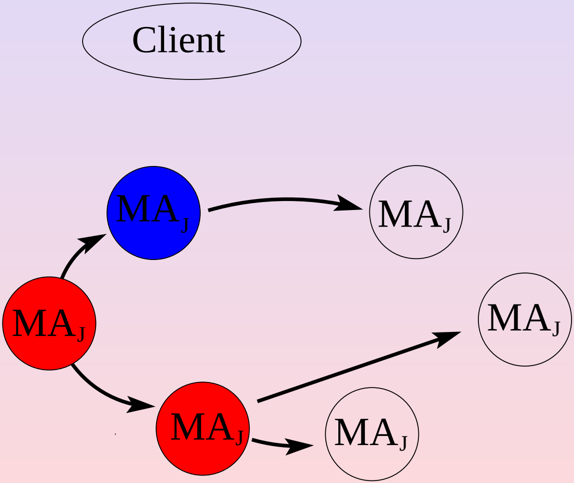
Implementation - PIF$_{async}$ (3)



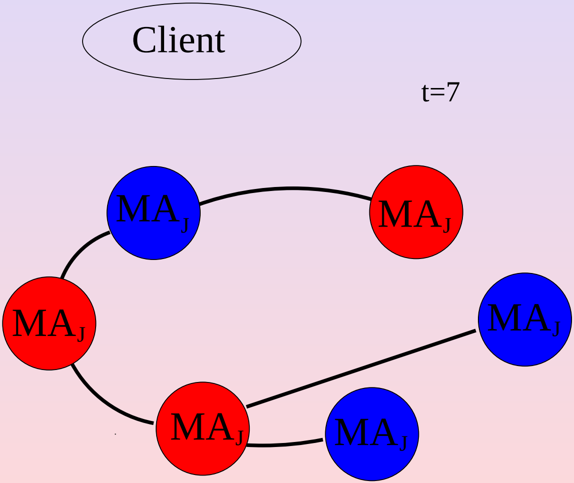
Implementation - PIF$_{async}$ (3)



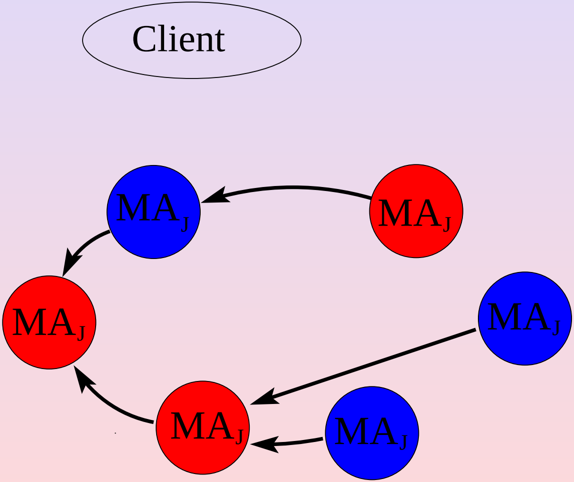
Implementation - PIF$_{async}$ (3)



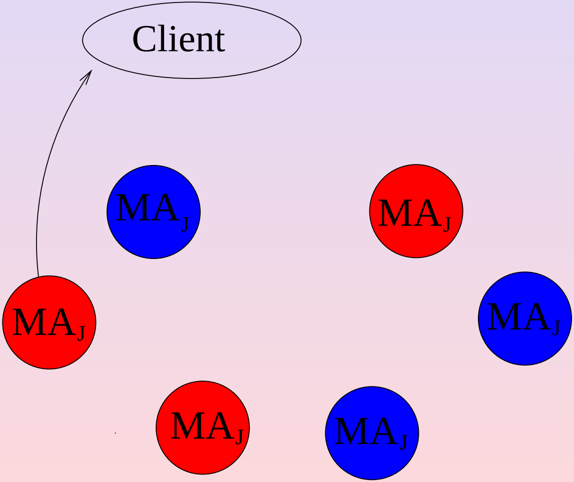
Implementation - PIF$_{async}$ (3)



Implementation - PIF$_{async}$ (3)



Implementation - PIF$_{async}$ (3)



Implementation - PIF$_{async}$ (4)

Quick analysis of the PIF scheme

- Builds dynamically a time-optimal tree for a given root
- Fastest possible to reach every nodes
- Messages follow the spanning tree during the feedback phase
- Consequences :
  - Balances the load among the links
  - Avoids overloaded links
  - Provides more fault-tolerance

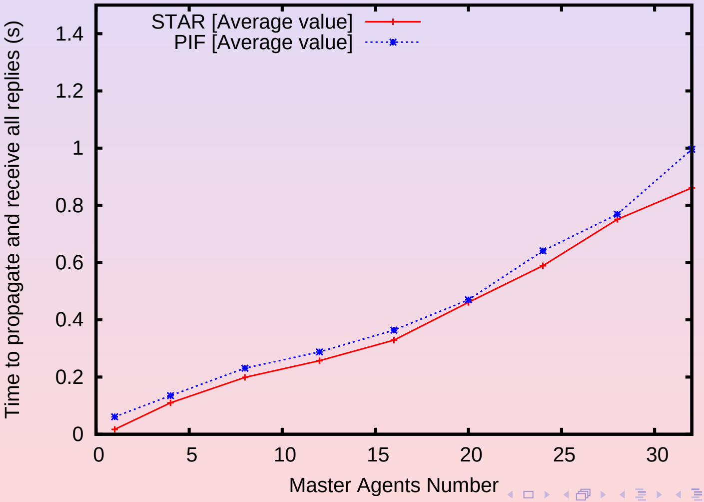
Outline

Experimental Platform

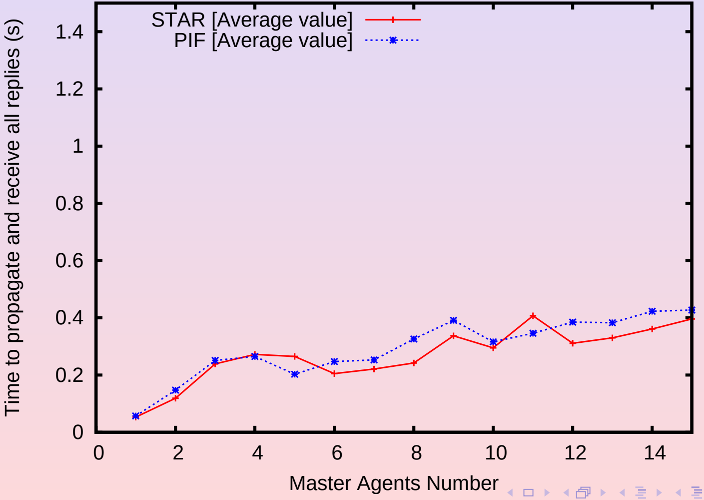
- VTHD Network
  - Wide Area Network
    - Connecting clusters
    - 2,5 Gb/s links between clusters
  - Clusters used
    - Intel quadri-processors 2.4 GHz
    - Intel bi-processors 2,8 GHz

- One MA runs per node
- Without the underlying hierarchy (hundreds of servers under one MA - Caron et al., IPDPS 2003)

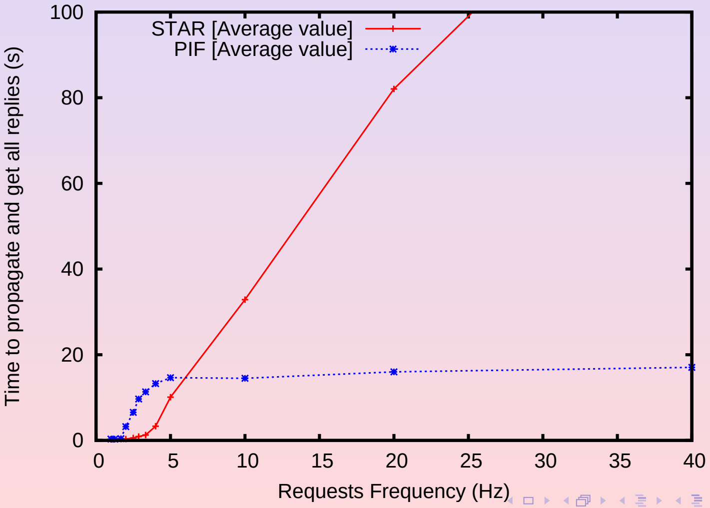
Homogeneous network



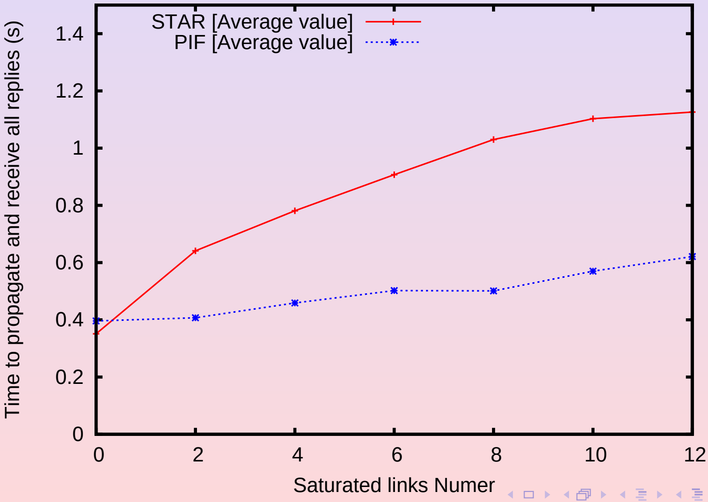
Homogeneous network



Requests flooding



Overloaded links



Outline

Conclusion and future work

- Conclusion
  - DIET$_J$: the first P2P extension of a NES system
  - JXTA and PIF$_{async}$
    - On-demand discovery of available servers at large scale
    - Adapt to the dynamic and heterogeneous nature of future grids platforms
- On-going and future work
  - Validate DIET$_J$ at larger scale
  - Implement other peer-to-peer algorithms
  - Extend this approach to other NES systems (NetSolve, Ninf)
  - Adapt peer-to-peer algorithms for service discovery