Automatic Deployment for Hierarchical Network Enabled Servers

Eddy Caron, Pushpinder Kaur Chouhan, Arnaud Legrand

26 April 2004
Heterogeneous Computing Workshop

Outline

Outline

Grid and GridRPC

Grid : Platform resulted from aggregating distributed computers and storage units

- Renting computation power and memory capacity
- Need of Problem Solving Environments

GridRPC : Implement Remote Procedure Call model over the Grid

- Good and simple paradigm to implement the Grid
- Run computation remotely

Grid and GridRPC

Grid : Platform resulted from aggregating distributed computers and storage units

- Renting computation power and memory capacity
- Need of Problem Solving Environments

GridRPC : Implement Remote Procedure Call model over the Grid

- Good and simple paradigm to implement the Grid
- Run computation remotely

Grid and GridRPC

Grid : Platform resulted from aggregating distributed computers and storage units

- Renting computation power and memory capacity
- Need of Problem Solving Environments

GridRPC : Implement Remote Procedure Call model over the Grid

- Good and simple paradigm to implement the Grid
- Run computation remotely

Grid and GridRPC

Grid : Platform resulted from aggregating distributed computers and storage units

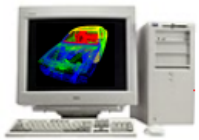
- Renting computation power and memory capacity
- Need of Problem Solving Environments

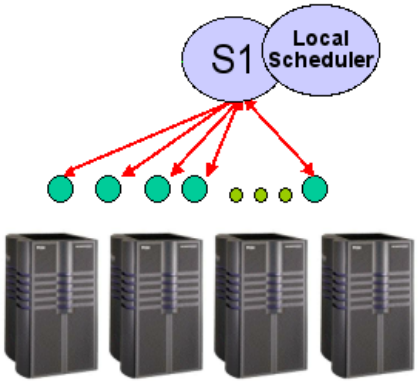
GridRPC : Implement Remote Procedure Call model over the Grid

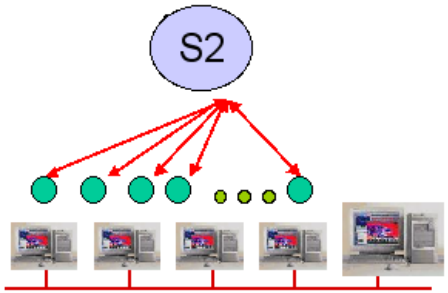
- Good and simple paradigm to implement the Grid
- Run computation remotely

GridRPC Paradigm

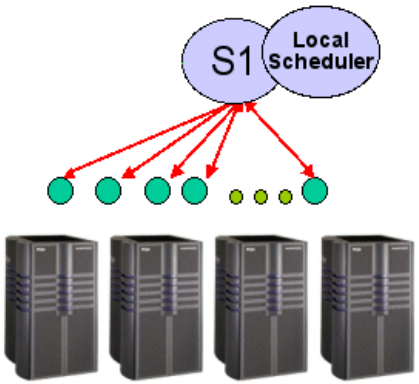
GridRPC Paradigm

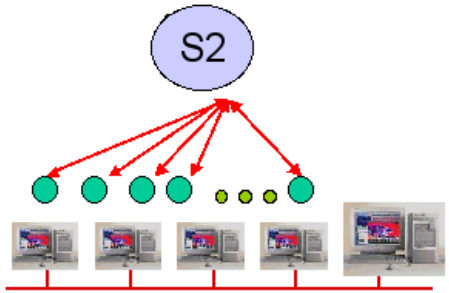

Visualization server





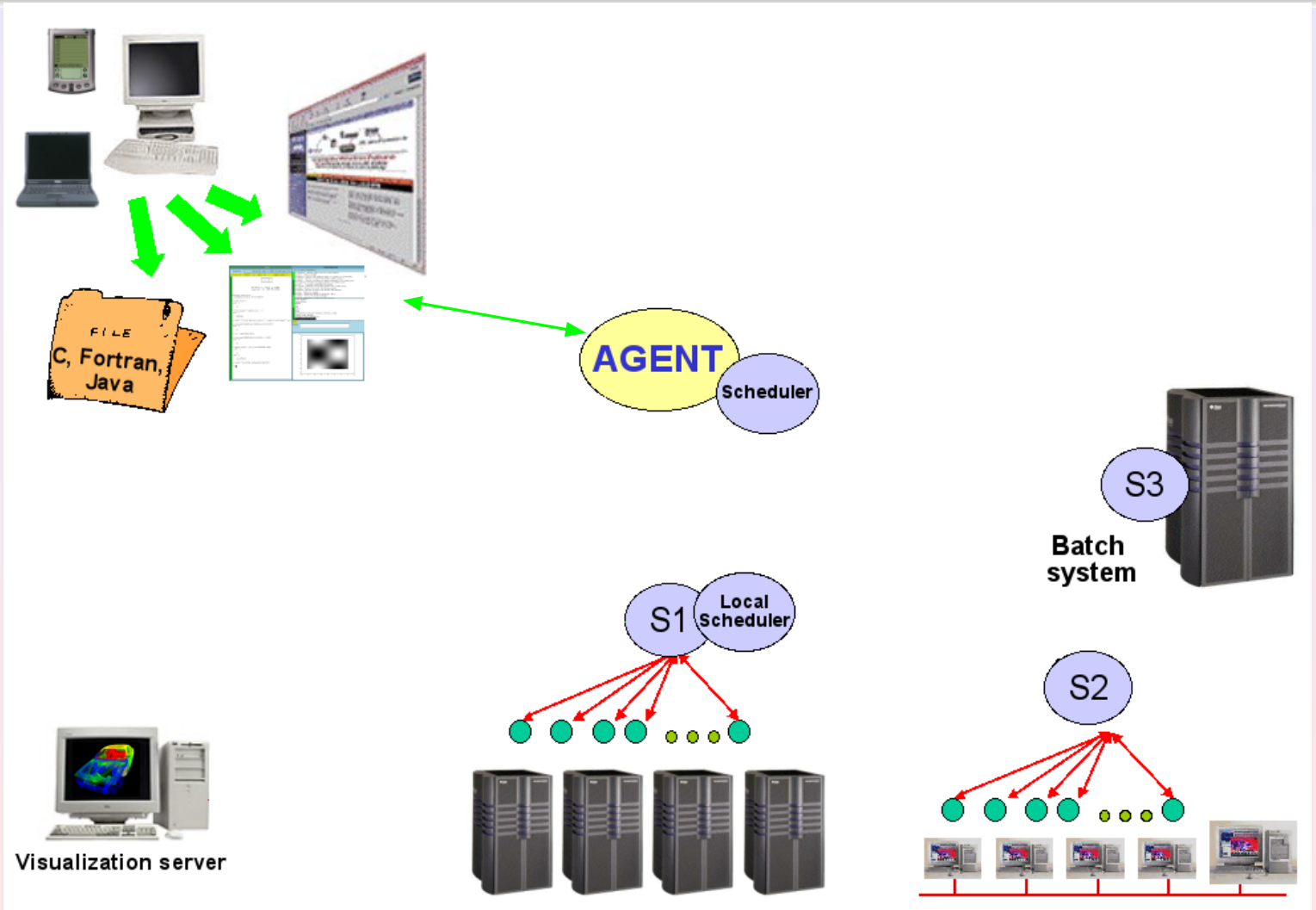
GridRPC Paradigm



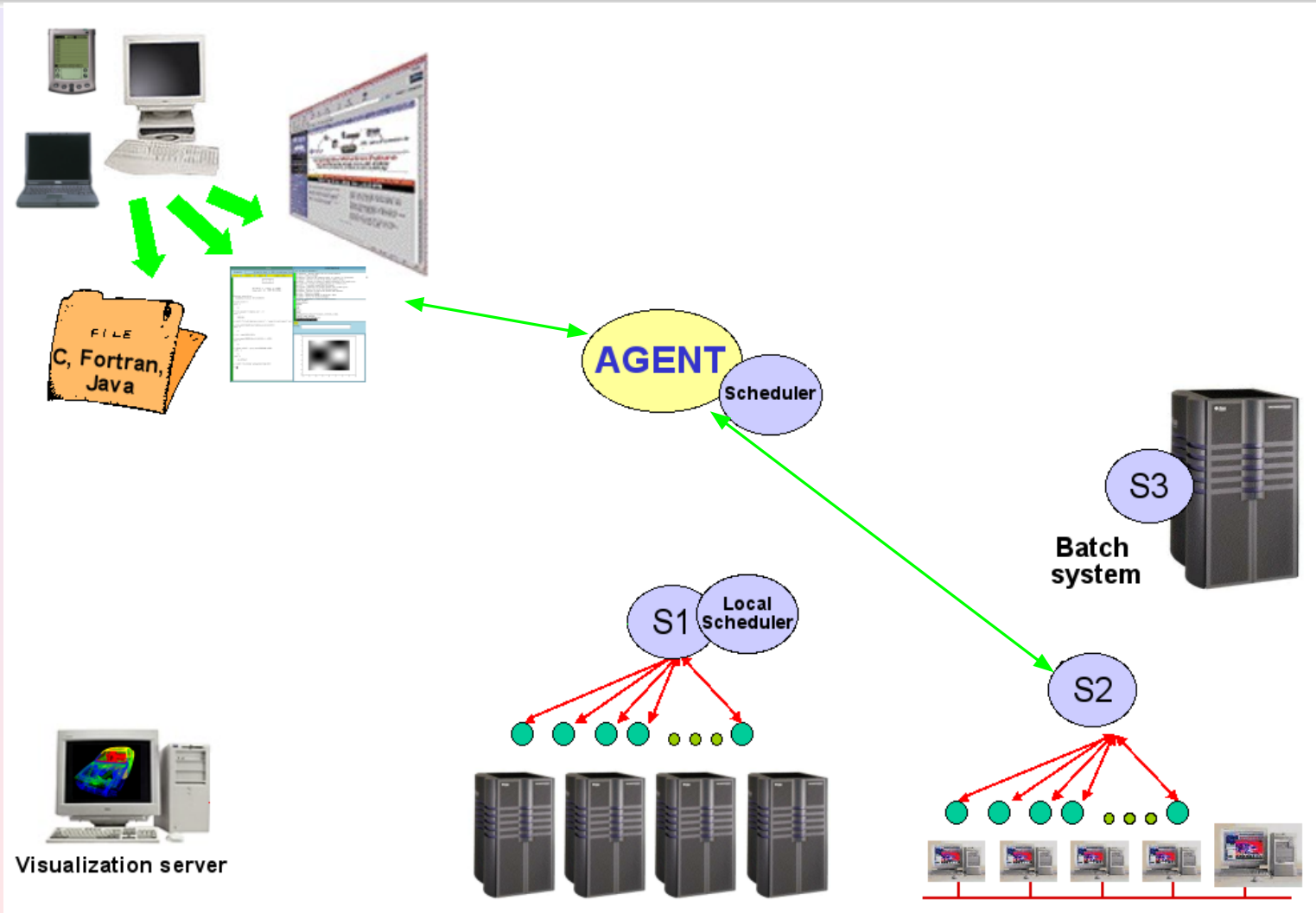



Visualization server

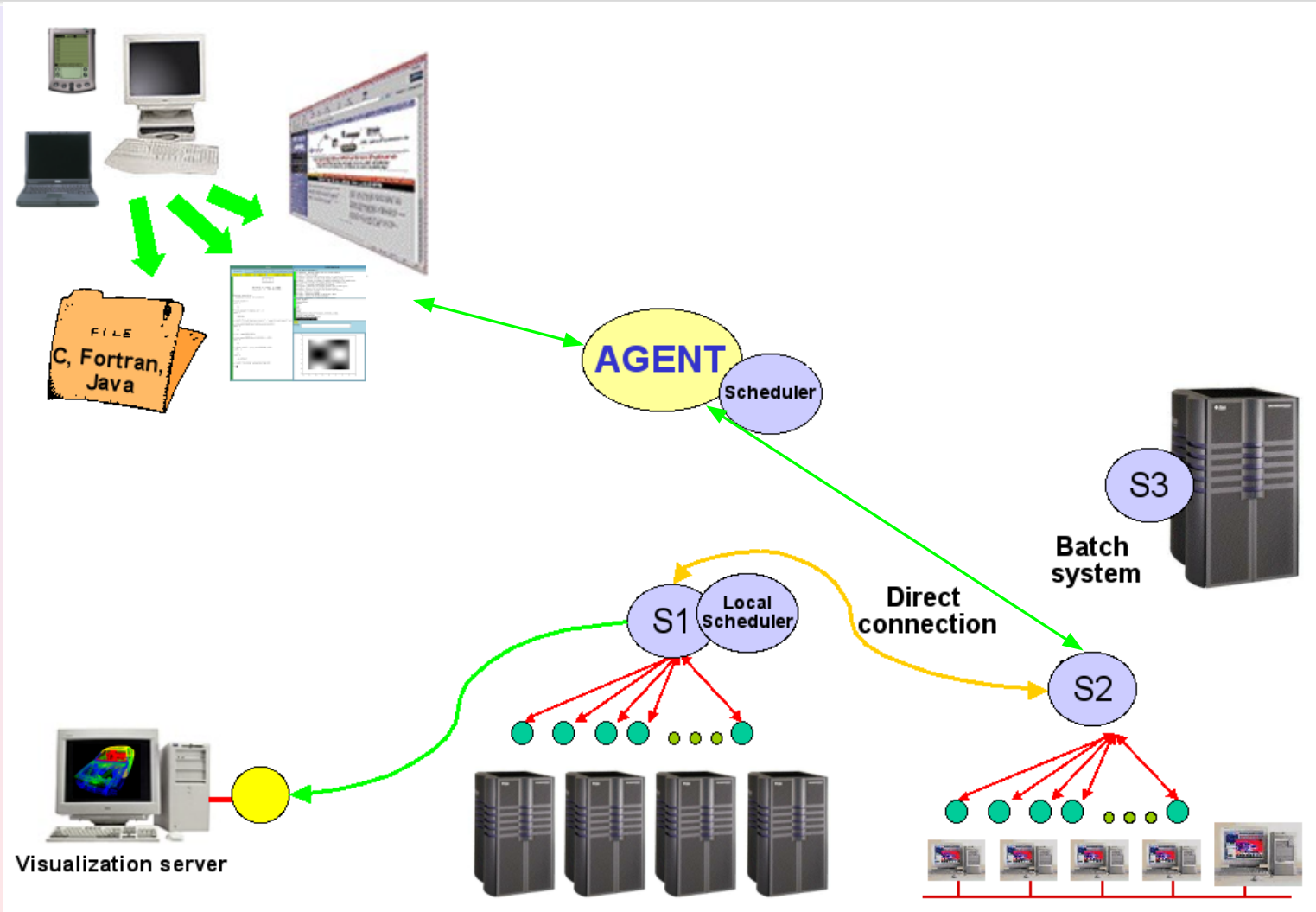
GridRPC Paradigm



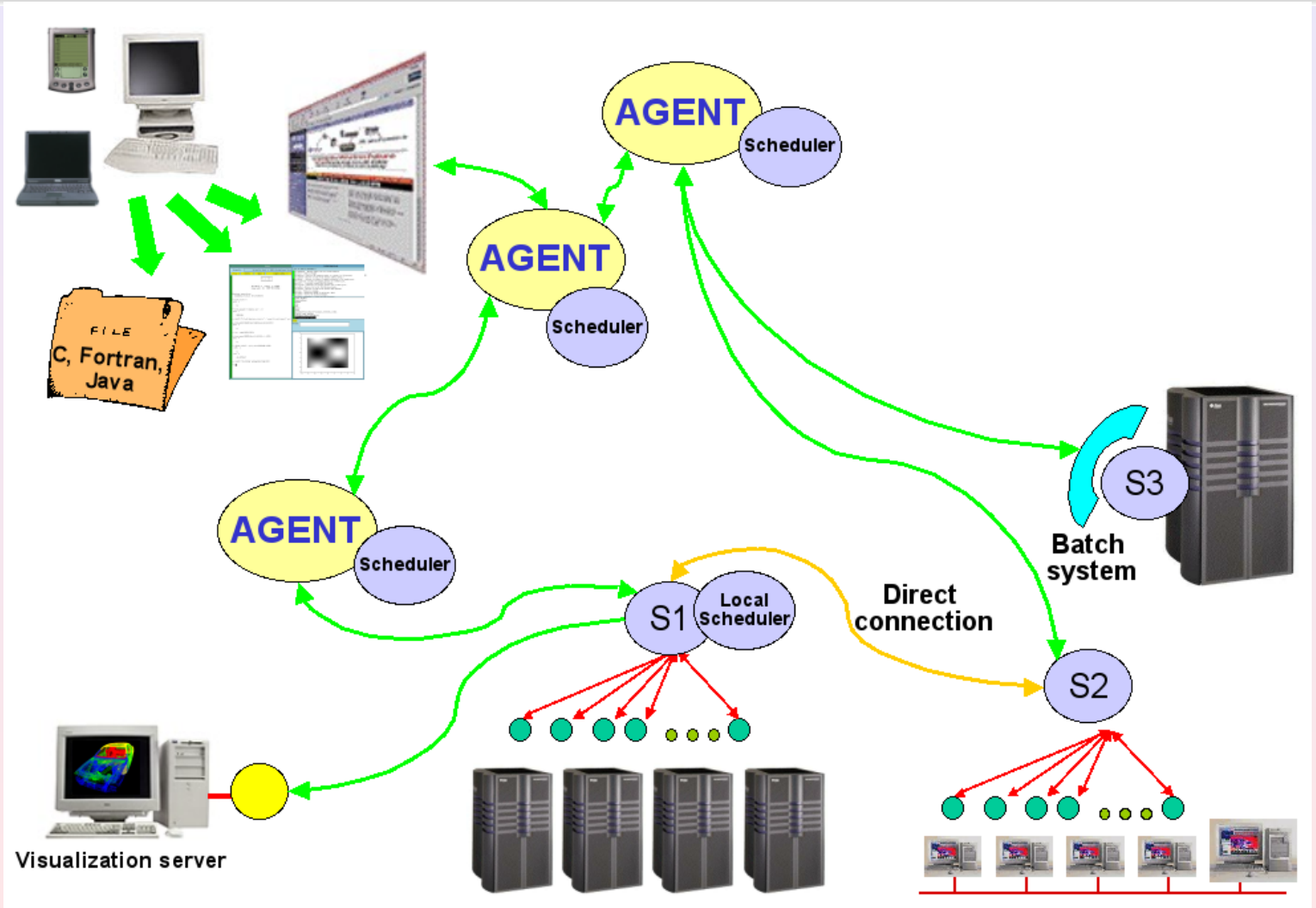
GridRPC Paradigm



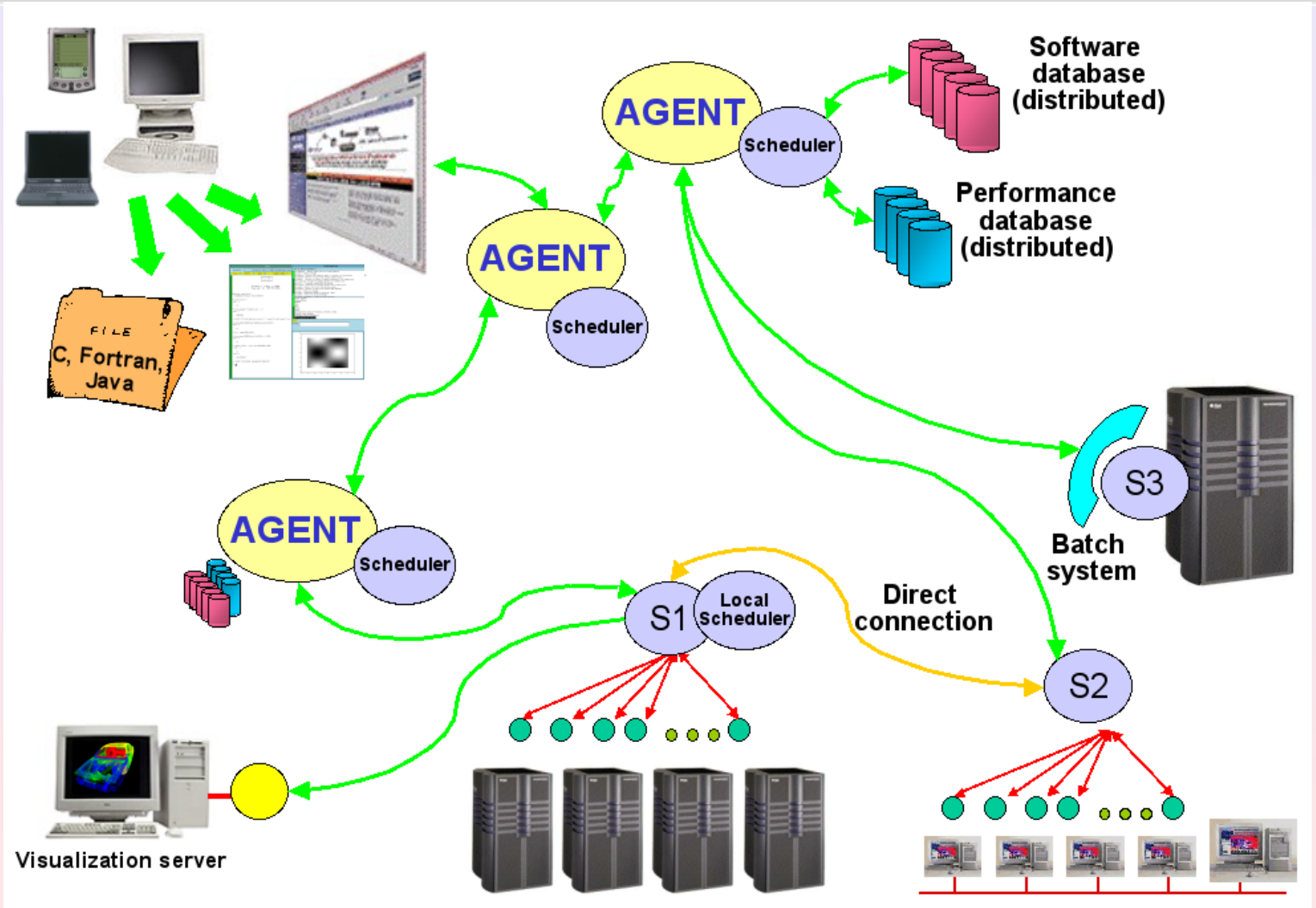
GridRPC Paradigm



GridRPC Paradigm

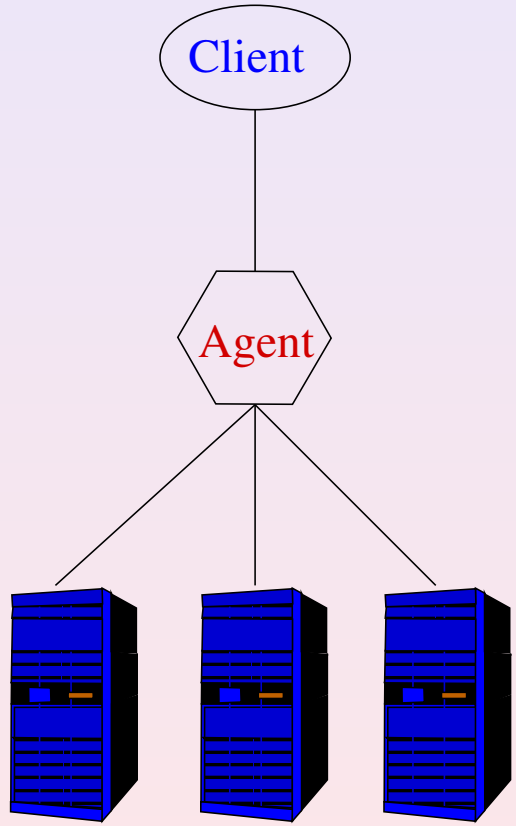


GridRPC Paradigm



Distributed Interactive Engineering Toolbox

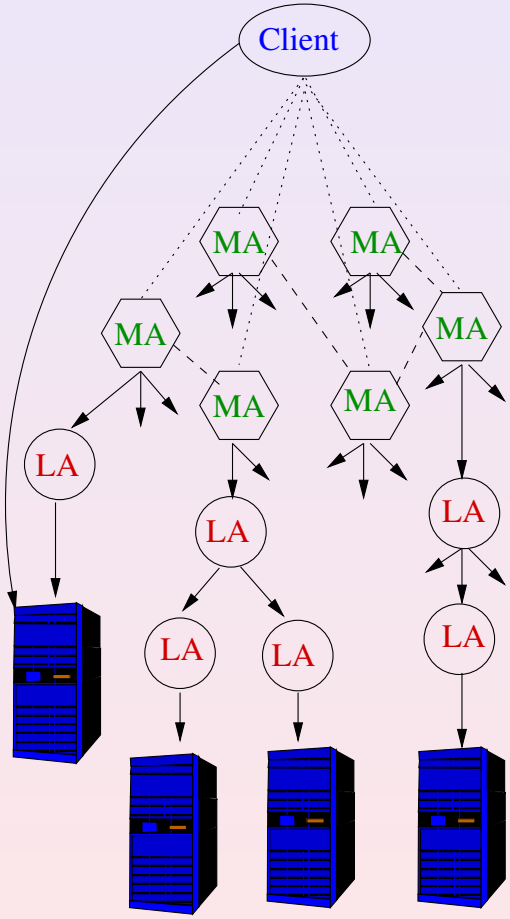
- Hierarchical architecture for an improved scalability
- Distributed information in the tree
- Plug in schedulers



http://graal.ens-lyon.fr/DIET/

Distributed Interactive Engineering Toolbox

- Hierarchical architecture for an improved scalability
- Distributed information in the tree
- Plug in schedulers



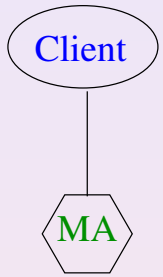
http://graal.ens-lyon.fr/DIET/

DIET Components

- Client : An application which uses DIET to solve problems
- Master Agent (MA) :
  - Receives computation request from clients
  - Collects computational abilities from servers and chooses the best one
  - The reference of the chosen server is returned to the client
- Local Agents (LA) : Act as transmitter between MAs and SeDs
- Servers (SeD) : Perform computations on data sent by a client

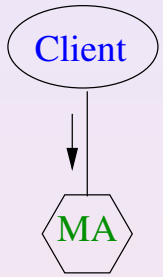
DIET Components

- Client : An application which uses DIET to solve problems
- Master Agent (MA) :
  - Receives computation request from clients
  - Collects computational abilities from servers and chooses the best one
  - The reference of the chosen server is returned to the client
- Local Agents (LA) : Act as transmitter between MAs and SeDs
- Servers (SeD) : Perform computations on data sent by a client



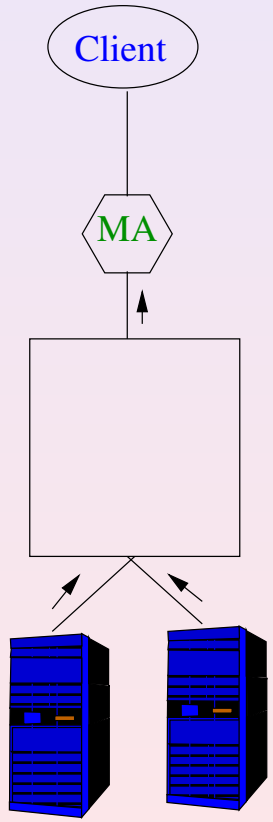
DIET Components

- Client : An application which uses DIET to solve problems
- Master Agent (MA) :
  - Receives computation request from clients
  - Collects computational abilities from servers and chooses the best one
  - The reference of the chosen server is returned to the client
- Local Agents (LA) : Act as transmitter between MAs and SeDs
- Servers (SeD) : Perform computations on data sent by a client



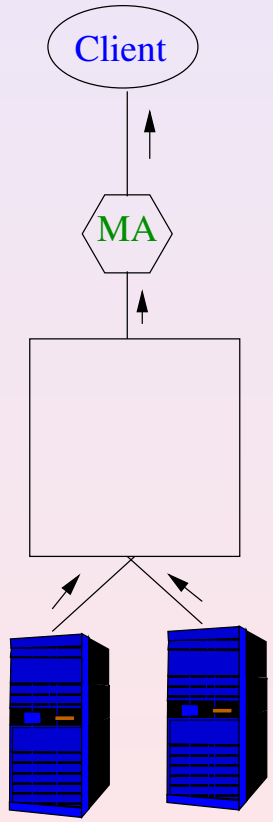
DIET Components

- Client : An application which uses DIET to solve problems
- Master Agent (MA) :
  - Receives computation request from clients
  - Collects computational abilities from servers and chooses the best one
  - The reference of the chosen server is returned to the client
- Local Agents (LA) : Act as transmitter between MAs and SeDs
- Servers (SeD) : Perform computations on data sent by a client



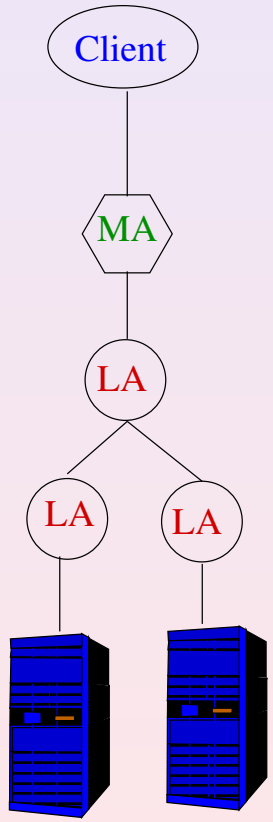
DIET Components

- Client : An application which uses DIET to solve problems
- Master Agent (MA) :
  - Receives computation request from clients
  - Collects computational abilities from servers and chooses the best one
  - The reference of the chosen server is returned to the client
- Local Agents (LA) : Act as transmitter between MAs and SeDs
- Servers (SeD) : Perform computations on data sent by a client



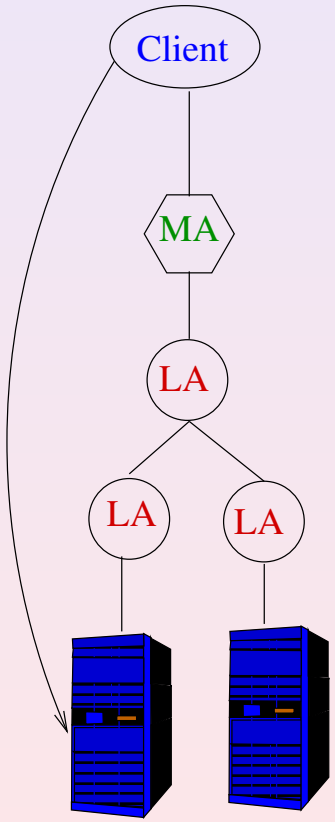
DIET Components

- Client : An application which uses DIET to solve problems
- Master Agent (MA) :
  - Receives computation request from clients
  - Collects computational abilities from servers and chooses the best one
  - The reference of the chosen server is returned to the client
- Local Agents (LA) : Act as transmitter between MAs and SeDs
- Servers (SeD) : Perform computations on data sent by a client



DIET Components

- Client : An application which uses DIET to solve problems
- Master Agent (MA) :
  - Receives computation request from clients
  - Collects computational abilities from servers and chooses the best one
  - The reference of the chosen server is returned to the client
- Local Agents (LA) : Act as transmitter between MAs and SeDs
- Servers (SeD) : Perform computations on data sent by a client



Introduction

- Goal : To increase the performance (number of requests executed per second) of the platform
- Problem :
  To deploy the distributed middleware
  To find the bottleneck
  To add new component at correct place
  To redeploy the distributed middleware
- Motivation : How to deploy distributed middleware on Grid

Introduction

- Goal : To increase the performance (number of requests executed per second) of the platform
- Problem :
  To deploy the distributed middleware
  To find the bottleneck
  To add new component at correct place
  To redeploy the distributed middleware
- Motivation : How to deploy distributed middleware on Grid

Introduction

- Goal : To increase the performance (number of requests executed per second) of the platform
- Problem :
  To deploy the distributed middleware
  To find the bottleneck
  To add new component at correct place
  To redeploy the distributed middleware
- Motivation : How to deploy distributed middleware on Grid

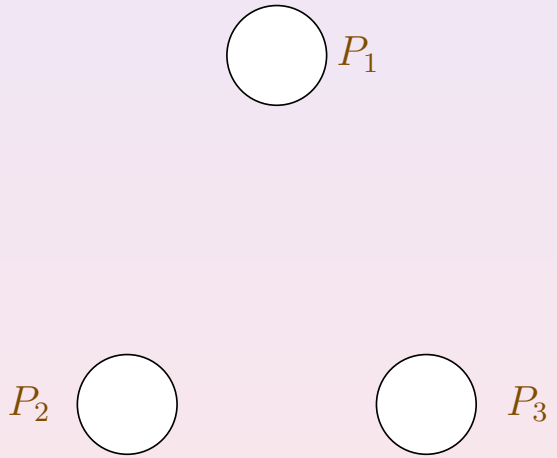
Outline

Notation

- $G = (V, E, w, c)$
- $P_i \in V$: computing resource
- w_i: computing power of resource P_i
- $(i, j) \in E$: communication link between P_i and P_j
- $c(i, j)$: size of data sent per second from P_i to P_j
- Each link is bidirectional

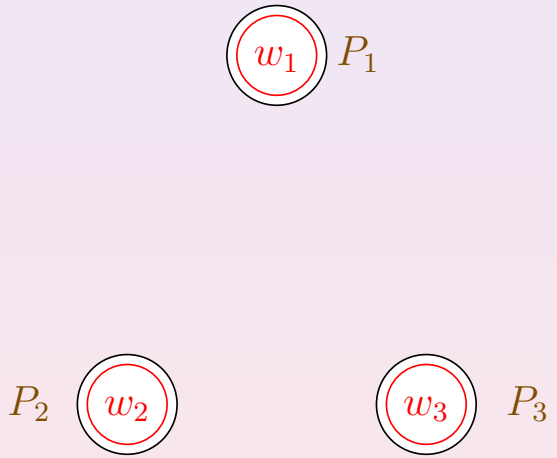
Notation

- $G = (V, E, w, c)$
- $P_i \in V$: computing resource
- w_i: computing power of resource P_i
- $(i, j) \in E$: communication link between P_i and P_j
- $c(i, j)$: size of data sent per second from P_i to P_j
- Each link is bidirectional



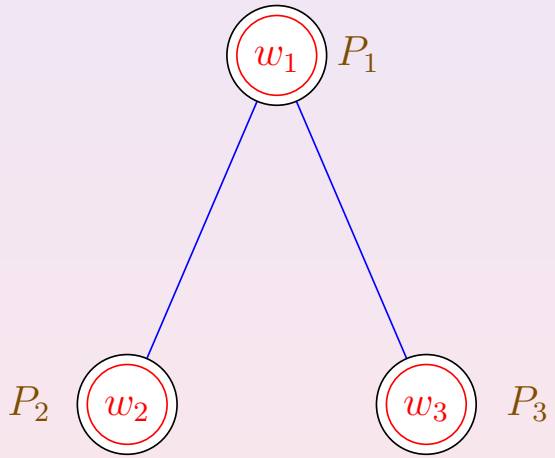
Notation

- $G = (V, E, w, c)$
- $P_i \in V$: computing resource
- w_i: computing power of resource P_i
- $(i, j) \in E$: communication link between P_i and P_j
- $c(i, j)$: size of data sent per second from P_i to P_j
- Each link is bidirectional



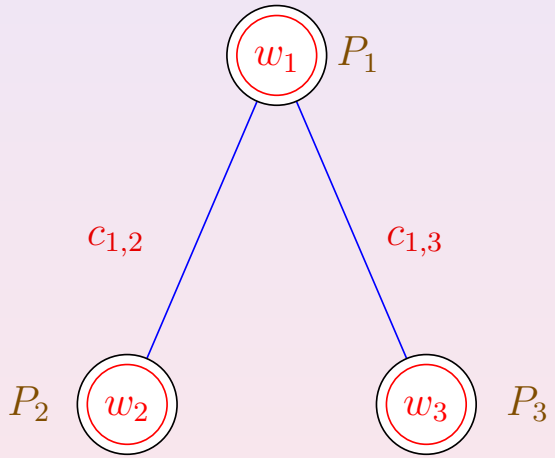
Notation

- $G = (V, E, w, c)$
- $P_i \in V$: computing resource
- w_i: computing power of resource P_i
- $(i, j) \in E$: communication link between P_i and P_j
- $c(i, j)$: size of data sent per second from P_i to P_j
- Each link is bidirectional



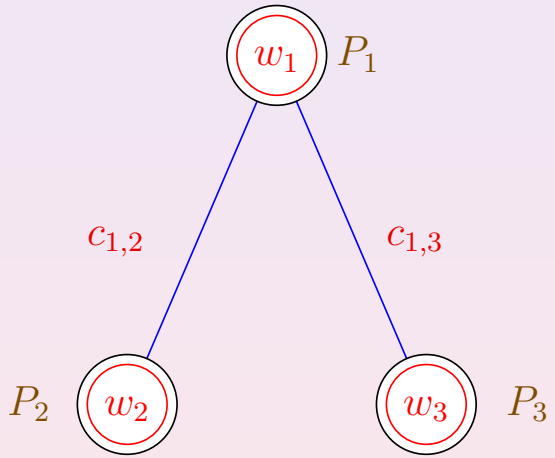
Notation

- $G = (V, E, w, c)$
- $P_i \in V$: computing resource
- w_i: computing power of resource P_i
- $(i, j) \in E$: communication link between P_i and P_j
- $c(i, j)$: size of data sent per second from P_i to P_j
- Each link is bidirectional



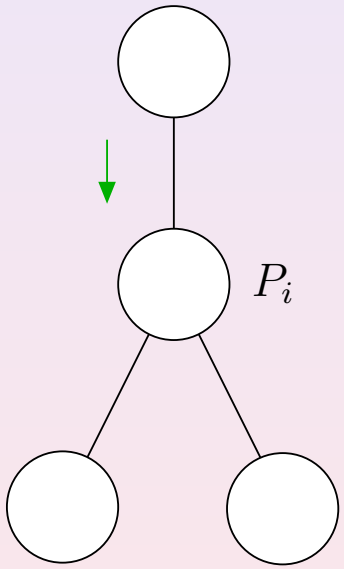
Notation

- $G = (V, E, w, c)$
- $P_i \in V$: computing resource
- w_i: computing power of resource P_i
- $(i, j) \in E$: communication link between P_i and P_j
- $c(i, j)$: size of data sent per second from P_i to P_j
- Each link is bidirectional



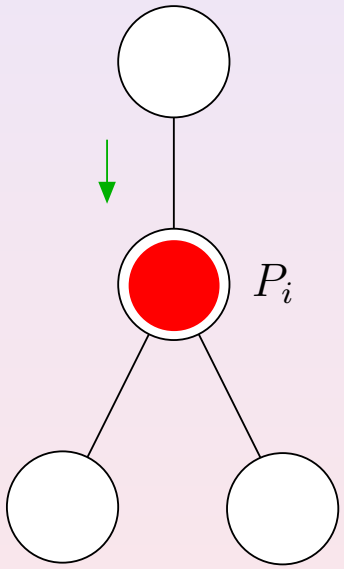
Notation

- $S_i^{(in)}$: size of request generated by client
- $W_i^{(in)}$: computation time to process one incoming request by P_i
- $S_i^{(out)}$: size of replied request generated by P_i
- $W_i^{(out)}$: computation time to merge the reply requests of its children
- $W_i^{(prob)}$: computation amount needed by server P_i to process a generic problem



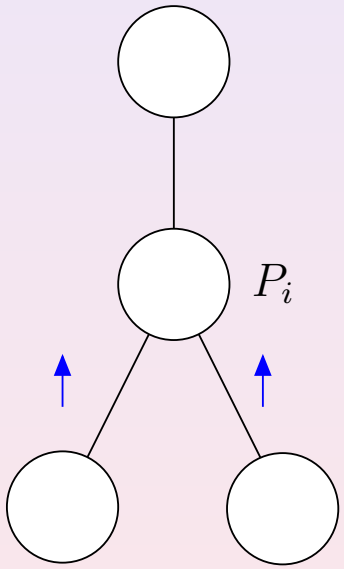
Notation

- $S_i^{(in)}$: size of request generated by client
- $W_i^{(in)}$: computation time to process one incoming request by P_i
- $S_i^{(out)}$: size of replied request generated by P_i
- $W_i^{(out)}$: computation time to merge the reply requests of its children
- $W_i^{(prob)}$: computation amount needed by server P_i to process a generic problem



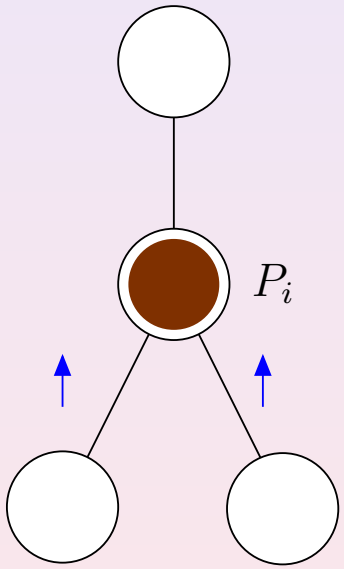
Notation

- $S_i^{(in)}$: size of request generated by client
- $W_i^{(in)}$: computation time to process one incoming request by P_i
- $S_i^{(out)}$: size of replied request generated by P_i
- $W_i^{(out)}$: computation time to merge the reply requests of its children
- $W_i^{(prob)}$: computation amount needed by server P_i to process a generic problem



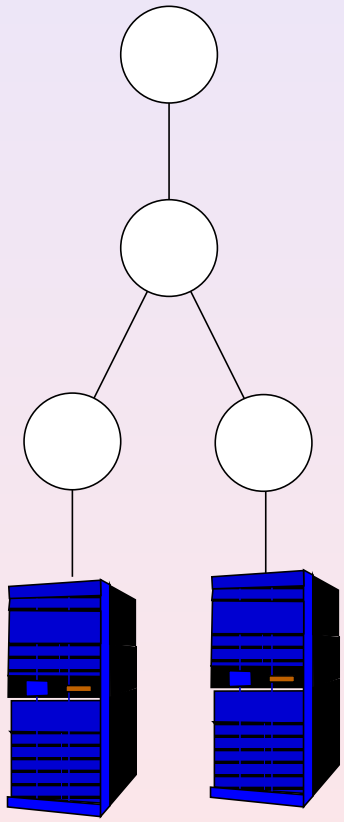
Notation

- $S_i^{(in)}$: size of request generated by client
- $W_i^{(in)}$: computation time to process one incoming request by P_i
- $S_i^{(out)}$: size of replied request generated by P_i
- $W_i^{(out)}$: computation time to merge the reply requests of its children
- $W_i^{(prob)}$: computation amount needed by server P_i to process a generic problem



Notation

- $S_i^{(in)}$: size of request generated by client
- $W_i^{(in)}$: computation time to process one incoming request by P_i
- $S_i^{(out)}$: size of replied request generated by P_i
- $W_i^{(out)}$: computation time to merge the reply requests of its children
- $W_i^{(prob)}$: computation amount needed by server P_i to process a generic problem



Steady state approach

Following the steady-state scheduling work by Beaumont et al. [IPDPS'2002]
we focus on the average quantities, over one time-unit:

- $\alpha_i^{(prob)}$: number of generic problems solved by server P_i
- ρ : throughput of the platform $\left(\rho = \sum_{P_i \text{ is a server}} \alpha_i^{prob}\right)$
- $\alpha_i^{(in)}$: number of incoming requests processed by P_i $\left(\rho = \alpha_i^{(in)}\right)$
- $\alpha_i^{(out)}$: number of outgoing requests computed by P_i $\left(\rho = \alpha_i^{(out)}\right)$

Steady state approach

Following the steady-state scheduling work by Beaumont et al. [IPDPS'2002]
we focus on the average quantities, over one time-unit:

- $\alpha_i^{(prob)}$: number of generic problems solved by server P_i
- ρ : throughput of the platform $\left(\rho = \sum_{P_i \text{ is a server}} \alpha_i^{prob} \right)$
- $\alpha_i^{(in)}$: number of incoming requests processed by P_i $(\rho = \alpha_i^{(in)})$
- $\alpha_i^{(out)}$: number of outgoing requests computed by P_i $(\rho = \alpha_i^{(out)})$

Steady state approach

Following the steady-state scheduling work by Beaumont et al. [IPDPS'2002]
we focus on the average quantities, over one time-unit:

- $\alpha_i^{(prob)}$: number of generic problems solved by server P_i
- ρ : throughput of the platform $\left(\rho = \sum_{P_i \text{ is a server}} \alpha_i^{prob} \right)$
- $\alpha_i^{(in)}$: number of incoming requests processed by P_i $(\rho = \alpha_i^{(in)})$
- $\alpha_i^{(out)}$: number of outgoing requests computed by P_i $(\rho = \alpha_i^{(out)})$

Steady state approach

Following the steady-state scheduling work by Beaumont et al. [IPDPS'2002]
we focus on the average quantities, over one time-unit:

- $\alpha_i^{(prob)}$: number of generic problems solved by server P_i
- ρ : throughput of the platform $\left(\rho = \sum_{P_i \text{ is a server}} \alpha_i^{prob} \right)$
- $\alpha_i^{(in)}$: number of incoming requests processed by P_i $(\rho = \alpha_i^{(in)})$
- $\alpha_i^{(out)}$: number of outgoing requests computed by P_i $(\rho = \alpha_i^{(out)})$

Steady state approach

Following the steady-state scheduling work by Beaumont et al. [IPDPS'2002]
we focus on the average quantities, over one time-unit:

- $\alpha_i^{(prob)}$: number of generic problems solved by server P_i
- ρ : throughput of the platform $\left(\rho = \sum_{P_i \text{ is a server}} \alpha_i^{prob} \right)$
- $\alpha_i^{(in)}$: number of incoming requests processed by P_i $(\rho = \alpha_i^{(in)})$
- $\alpha_i^{(out)}$: number of outgoing requests computed by P_i $(\rho = \alpha_i^{(out)})$

Steady state approach

Following the steady-state scheduling work by Beaumont et al. [IPDPS'2002]
we focus on the average quantities, over one time-unit:

- $\alpha_i^{(prob)}$: number of generic problems solved by server P_i
- ρ : throughput of the platform $\left(\rho = \sum_{P_i \text{ is a server}} \alpha_i^{prob} \right)$
- $\alpha_i^{(in)}$: number of incoming requests processed by P_i $(\rho = \alpha_i^{(in)})$
- $\alpha_i^{(out)}$: number of outgoing requests computed by P_i $(\rho = \alpha_i^{(out)})$

Steady state approach

Following the steady-state scheduling work by Beaumont et al. [IPDPS'2002]
we focus on the average quantities, over one time-unit:

- $\alpha_i^{(prob)}$: number of generic problems solved by server P_i
- ρ : throughput of the platform $\left(\rho = \sum_{P_i \text{ is a server}} \alpha_i^{prob} \right)$
- $\alpha_i^{(in)}$: number of incoming requests processed by P_i $(\rho = \alpha_i^{(in)})$
- $\alpha_i^{(out)}$: number of outgoing requests computed by P_i $(\rho = \alpha_i^{(out)})$

Deployment constraints

- Server computation:

$$\forall P_i : \frac{\alpha_i^{(prob)} \times W_i^{(prob)}}{w_i} \leq 1 \quad \rightsquigarrow \quad \alpha_i^{(prob)} \leq \frac{w_i}{W_i^{(prob)}}$$

$$\rightsquigarrow \quad \rho = \sum_{P_i \text{ is a server}} \alpha_i^{prob} \leq \sum_{P_i \text{ is a server}} \frac{w_i}{W_i^{prob}}$$

Deployment constraints

- Server computation:

$$\forall P_i : \frac{\alpha_i^{(prob)} \times W_i^{(prob)}}{w_i} \leq 1 \quad \rightsquigarrow \quad \alpha_i^{(prob)} \leq \frac{w_i}{W_i^{(prob)}}$$

$$\rightsquigarrow \quad \rho = \sum_{P_i \text{ is a server}} \alpha_i^{prob} \leq \sum_{P_i \text{ is a server}} \frac{w_i}{W_i^{prob}}$$

Deployment constraints

- Server computation:

$$\forall P_i : \frac{\alpha_i^{(prob)} \times W_i^{(prob)}}{w_i} \leq 1 \quad \rightsquigarrow \quad \alpha_i^{(prob)} \leq \frac{w_i}{W_i^{(prob)}}$$

$$\rightsquigarrow \quad \rho = \sum_{P_i \text{ is a server}} \alpha_i^{prob} \leq \sum_{P_i \text{ is a server}} \frac{w_i}{W_i^{prob}}$$

Deployment constraints

- Agents computation:

$$\forall P_i : \frac{\alpha_i^{in} \times W_i^{(in)} + \alpha_i^{out} \times W_i^{(out)}}{w_i} \leq 1$$

- Agents Communication:

$$\forall P_i : \frac{\alpha_i^{in} \times S_i^{(in)} + \alpha_i^{out} \times S_i^{(out)}}{w_i} \leq 1$$

Deployment constraints

- Agents computation:

$$\forall P_i : \rho \times \frac{W_i^{(in)} + W_i^{(out)}}{w_i} \leq 1$$

- Agents Communication:

$$\forall P_i : \frac{\alpha_i^{in} \times S_i^{(in)} + \alpha_i^{out} \times S_i^{(out)}}{w_i} \leq 1$$

Deployment constraints

- Agents computation:

$$\forall P_i : \rho \times \frac{W_i^{(in)} + W_i^{(out)}}{w_i} \leq 1$$

- Agents Communication:

$$\forall P_i : \frac{\alpha_i^{in} \times S_i^{(in)} + \alpha_i^{out} \times S_i^{(out)}}{w_i} \leq 1$$

Deployment constraints

- Agents computation:

$$\forall P_i : \rho \times \frac{W_i^{(in)} + W_i^{(out)}}{w_i} \leq 1$$

- Agents Communication:

$$\forall P_i \rightarrow P_j : \rho \times \frac{S_i^{(in)} + S_j^{(out)}}{c_{i,j}} \leq 1$$

Operating models

- No internal parallelism:

$$\underbrace{\rho\left(\frac{S^{(in)}_{parent(i)}}{c_{parent(i),i}}+\frac{S^{(out)}_{parent(i)}}{c_{parent(i),i}}\right)}_{\text{Communications with the parent}}+\underbrace{\rho\left(\sum_{P_i\to P_j}\frac{S^{(in)}_i+S^{(out)}_j}{C_{i,j}}\right)}_{\text{Communications with the children}}+\underbrace{\rho\left(\frac{W^{(in)}_i+W^{(out)}_i}{w_i}\right)}_{\text{Local computations}}\leq 1$$

- Communication and computation in parallel:

$$\underbrace{\rho\left(\frac{S^{(in)}_{parent(i)}}{c_{parent(i),i}}+\frac{S^{(out)}_i}{c_{parent(i),i}}\right)}_{\text{Communications with the parent}}+\underbrace{\rho\left(\sum_{P_i\to P_j}\frac{S^{(in)}_i+S^{(out)}_j}{c_{i,j}}\right)}_{\text{Communications with the children}}\leq 1$$

Operating models

- No internal parallelism:

$$\underbrace{\rho\left(\frac{S^{(in)}_{parent(i)}}{c_{parent(i),i}}+\frac{S^{(out)}_{parent(i)}}{c_{parent(i),i}}\right)}_{\text{Communications with the parent}}+\underbrace{\rho\left(\sum_{P_i\to P_j}\frac{S^{(in)}_i+S^{(out)}_j}{C_{i,j}}\right)}_{\text{Communications with the children}}+\underbrace{\rho\left(\frac{W^{(in)}_i+W^{(out)}_i}{w_i}\right)}_{\text{Local computations}}\leq 1$$

- Communication and computation in parallel:

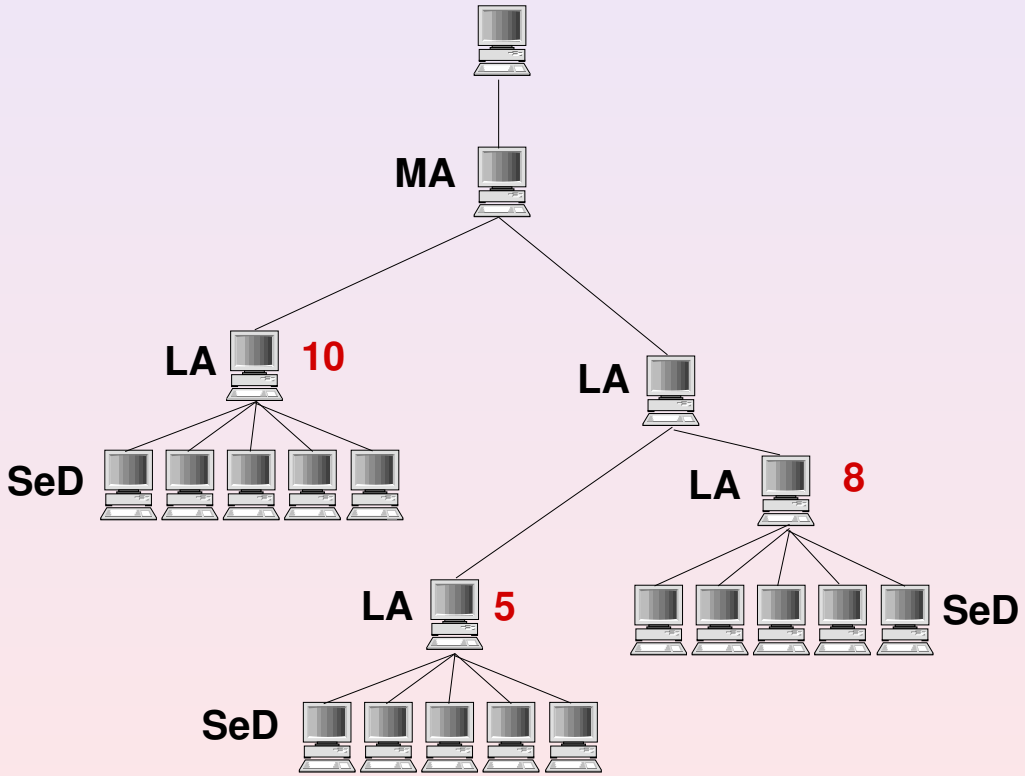
$$\underbrace{\rho\left(\frac{S^{(in)}_{parent(i)}}{c_{parent(i),i}}+\frac{S^{(out)}_i}{c_{parent(i),i}}\right)}_{\text{Communications with the parent}}+\underbrace{\rho\left(\sum_{P_i\to P_j}\frac{S^{(in)}_i+S^{(out)}_j}{c_{i,j}}\right)}_{\text{Communications with the children}}\leq 1$$

Maximum throughput of the platform

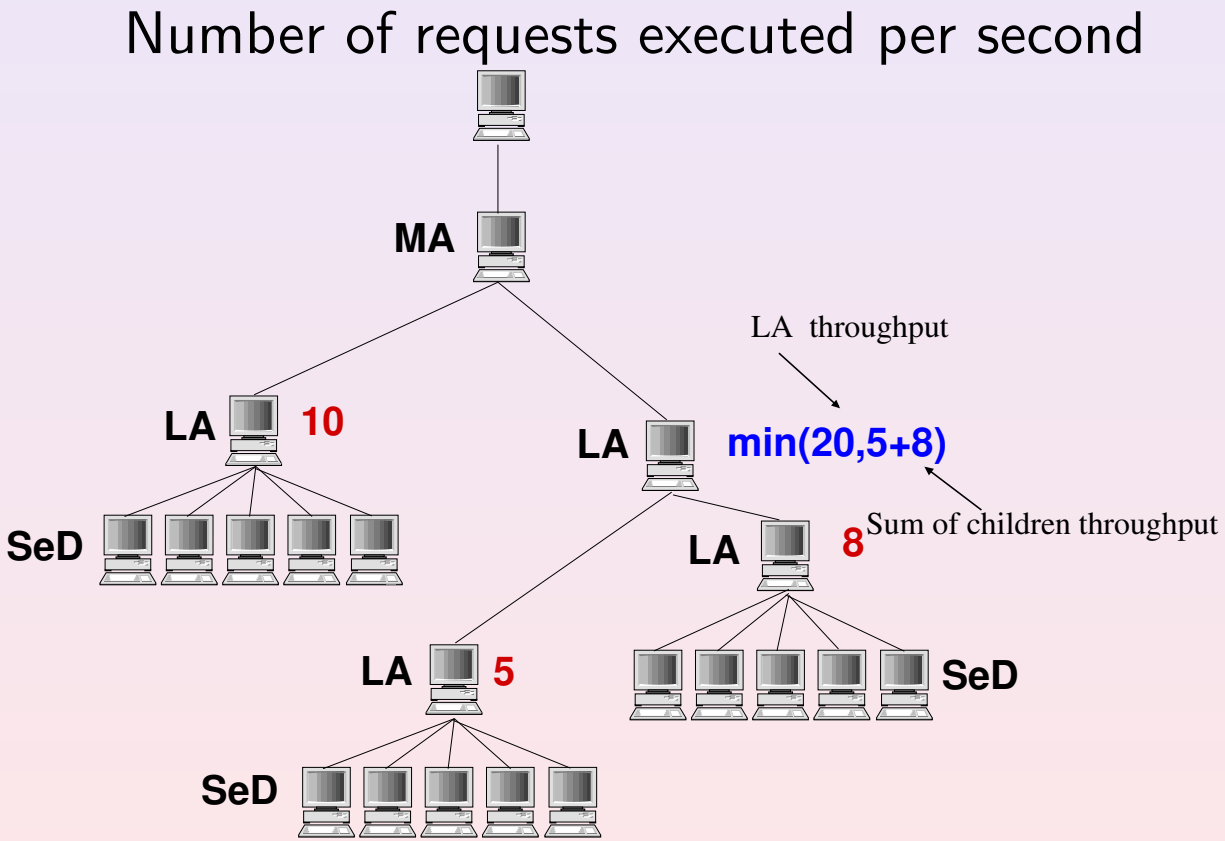
$$\rho = \min \left(\frac{w_i}{W_i^{(in)} + W_i^{(out)}}, \frac{c_{i,j}}{S_i^{(in)} + S_i^{(out)}}, \sum_{P_i | P_i \text{ is a server}} \frac{w_i}{W_i^{(prob)}}, \frac{1}{\frac{S_{parent(i)}^{(in)}}{c_{parent(i),i}} + \frac{S_i^{(out)}}{c_{parent(i),i}} + \sum_{P_i \to P_j} \frac{S_i^{(in)} + S_j^{(out)}}{c_{i,j}} + \frac{W_i^{(in)} + W_i^{(out)}}{w_i}} \right)$$

Automatic deployment

Number of requests executed per second

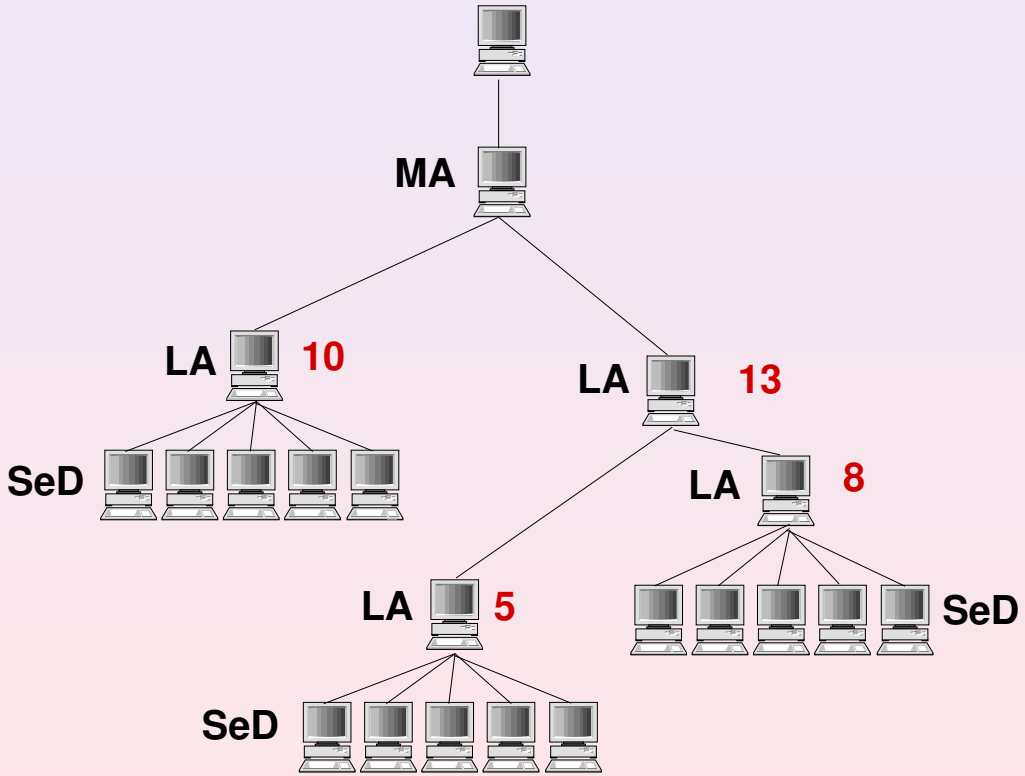


Automatic deployment



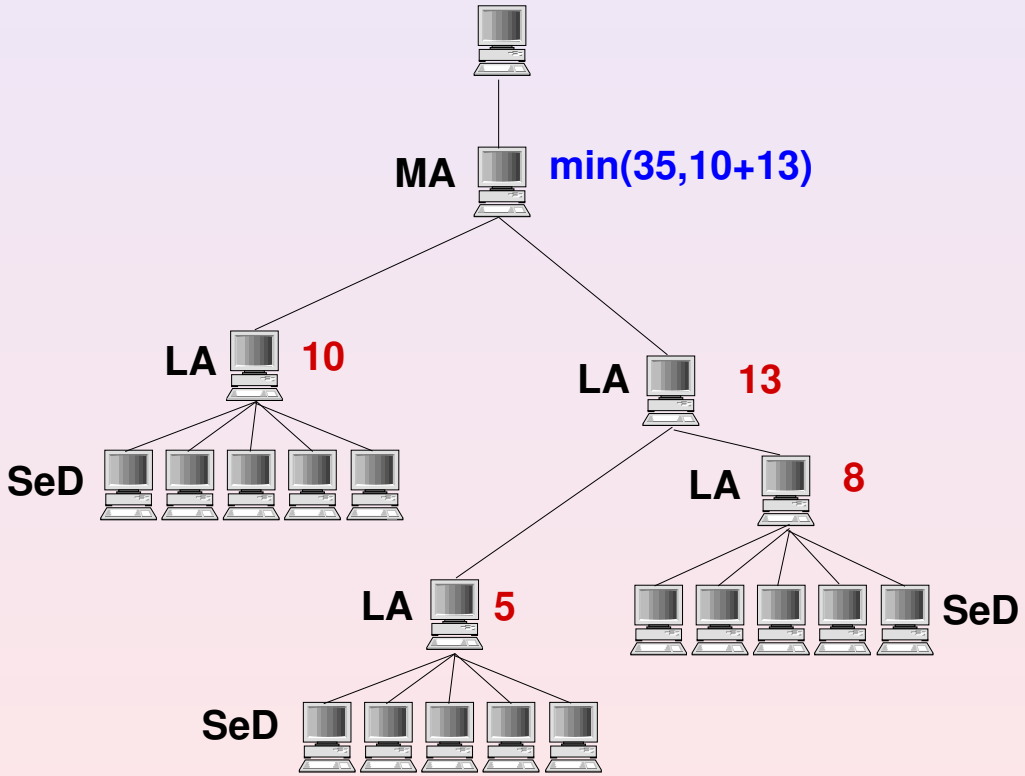
Automatic deployment

Number of requests executed per second



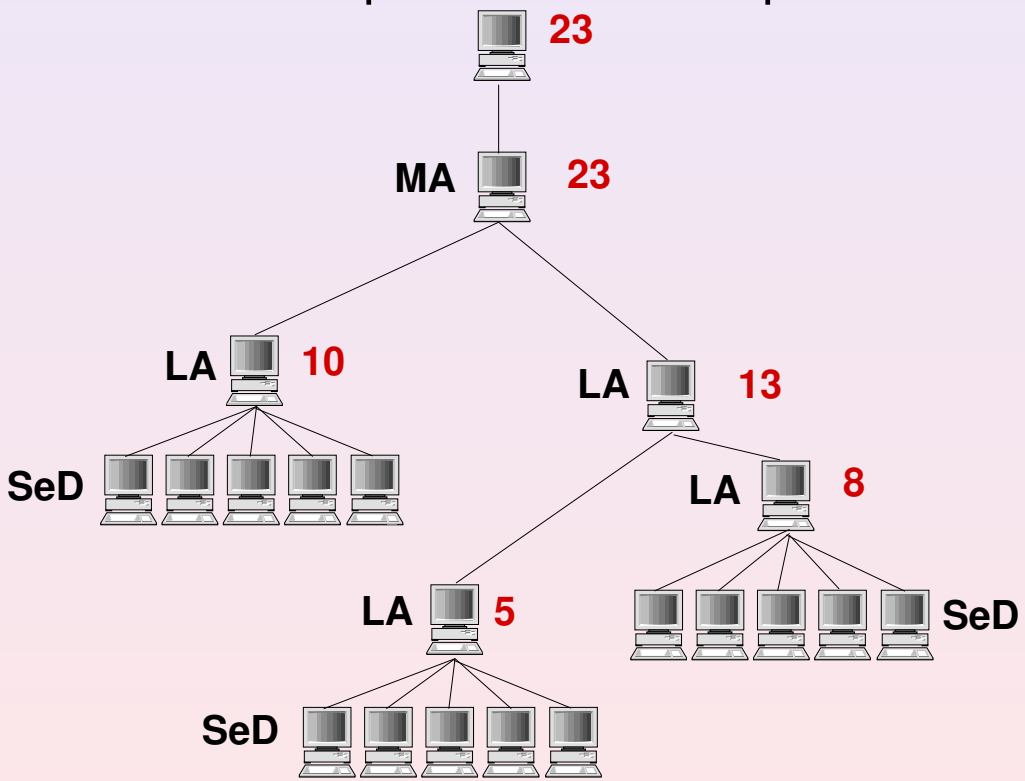
Automatic deployment

Number of requests executed per second



Automatic deployment

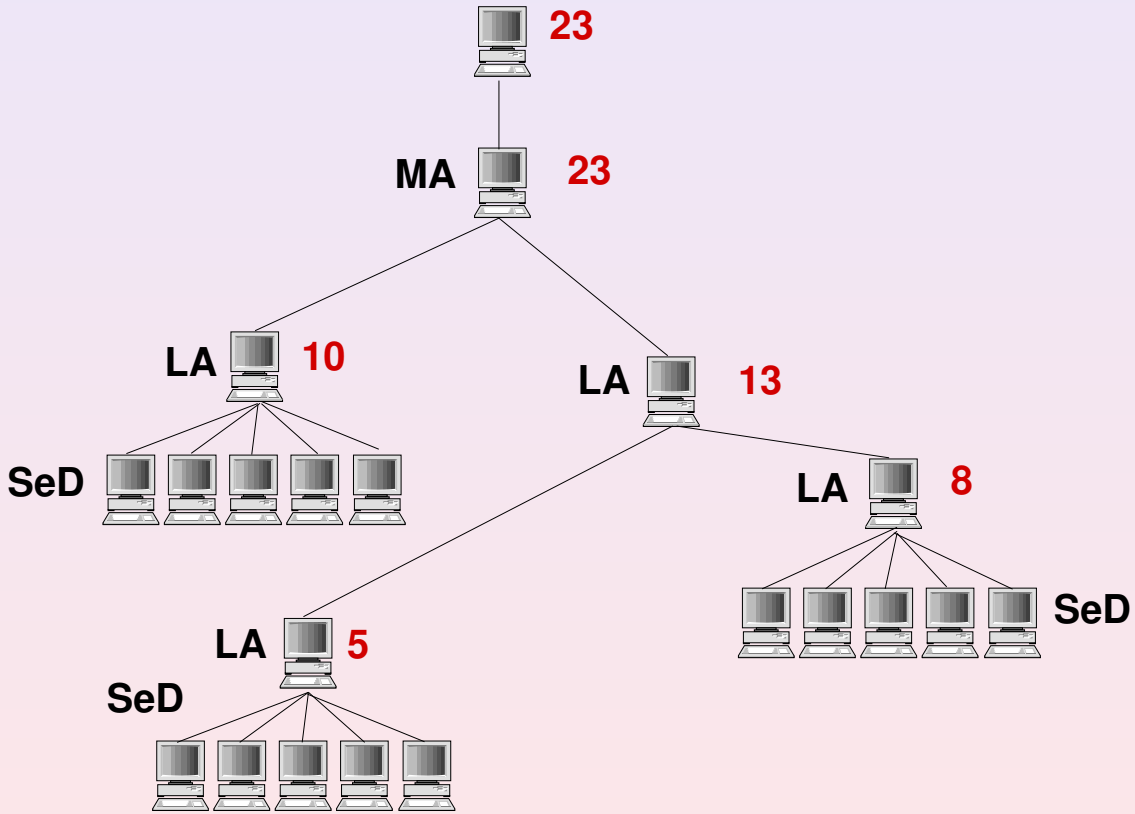
Number of requests executed per second



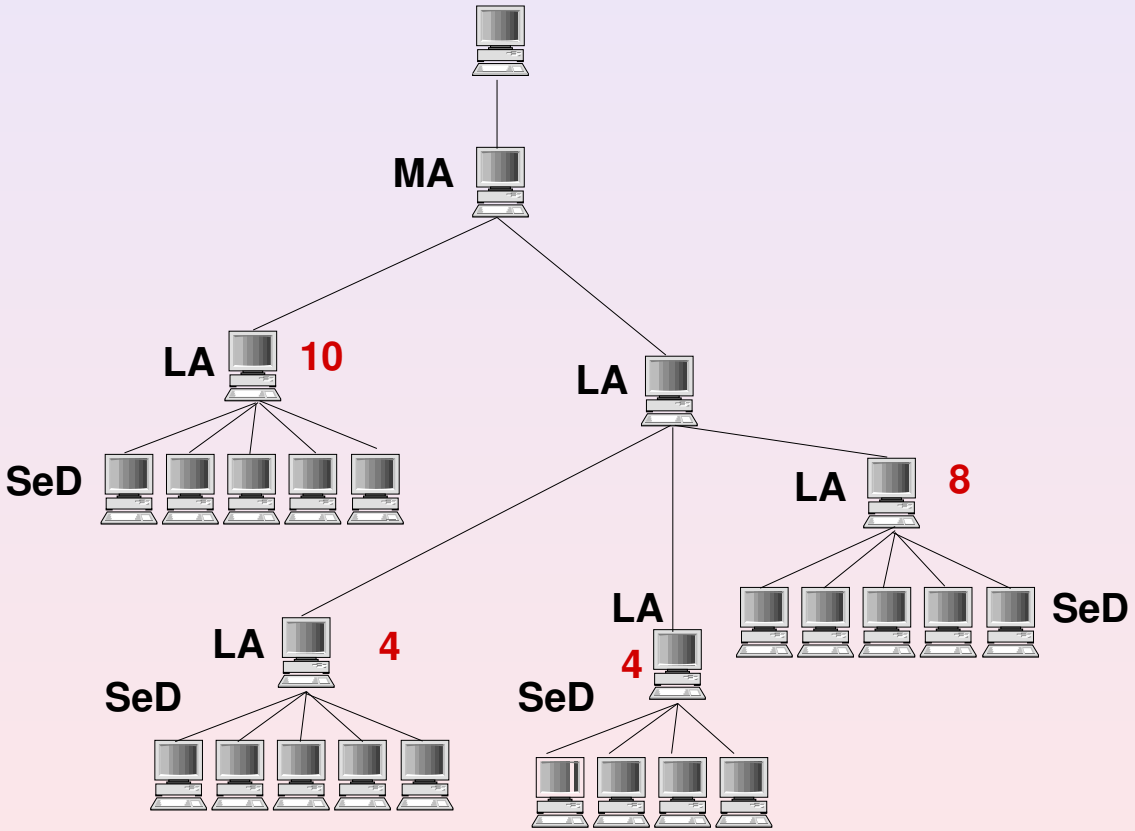
Automatic redeployment

1: **while** (number of available nodes > 0) **do**
2: Calculate the throughput ρ of structure.
3: Find a node whose constraint is tight and that can be split
4: **if** no such node exists **then**
5: The deployment cannot be improved.
6: Exit
7: **end if**
8: Split the load by adding new node to its parent
9: Decrease the number of available nodes
10: **end while**

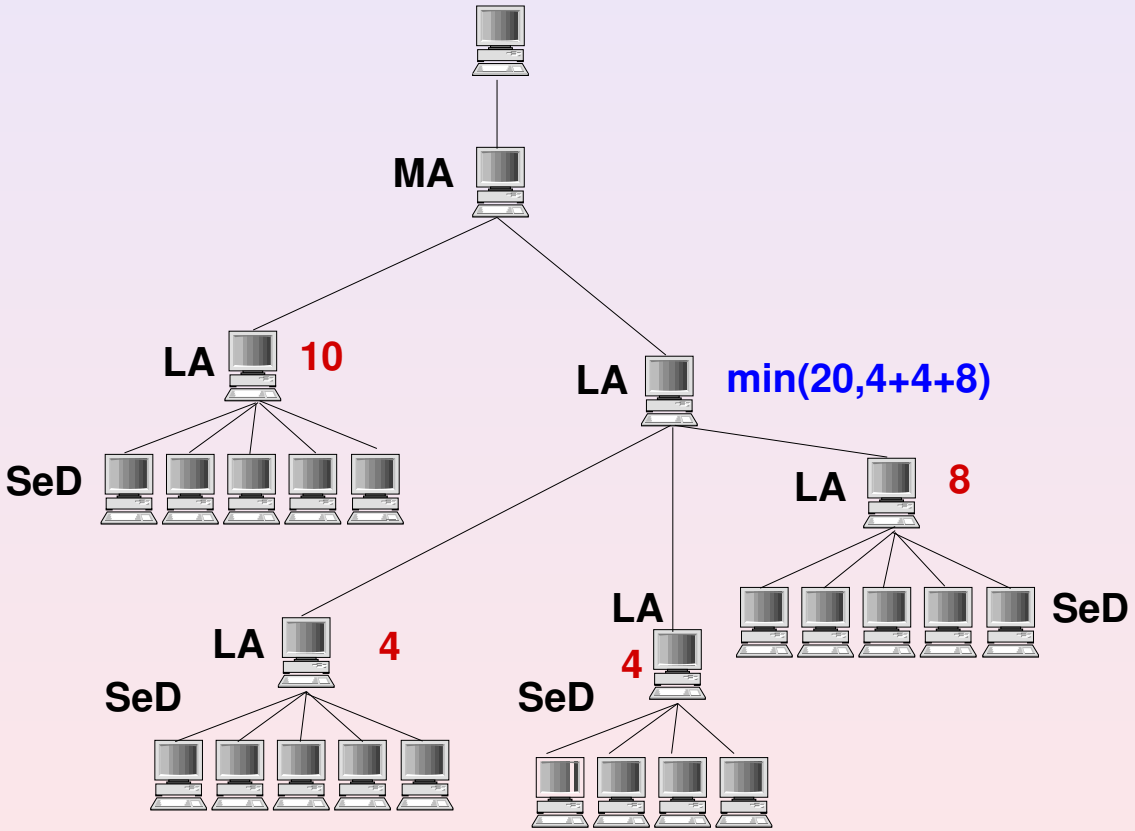
Automatic redeployment



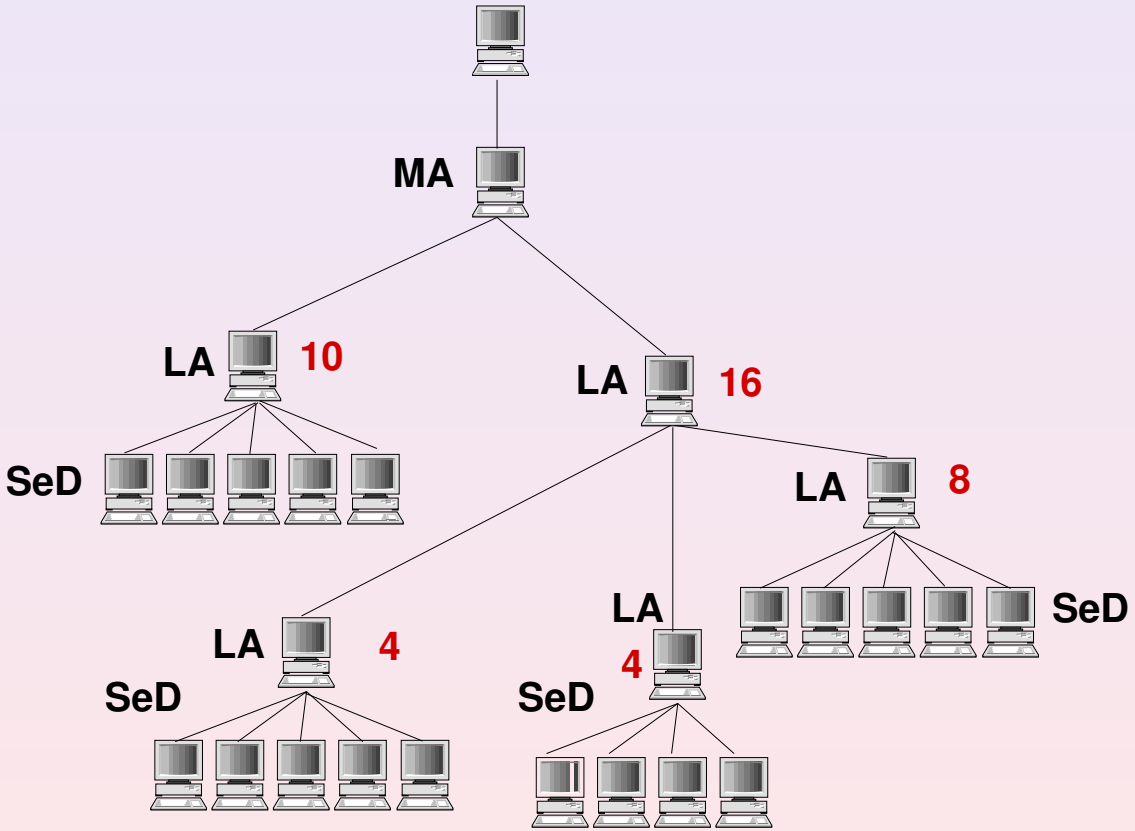
Automatic redeployment



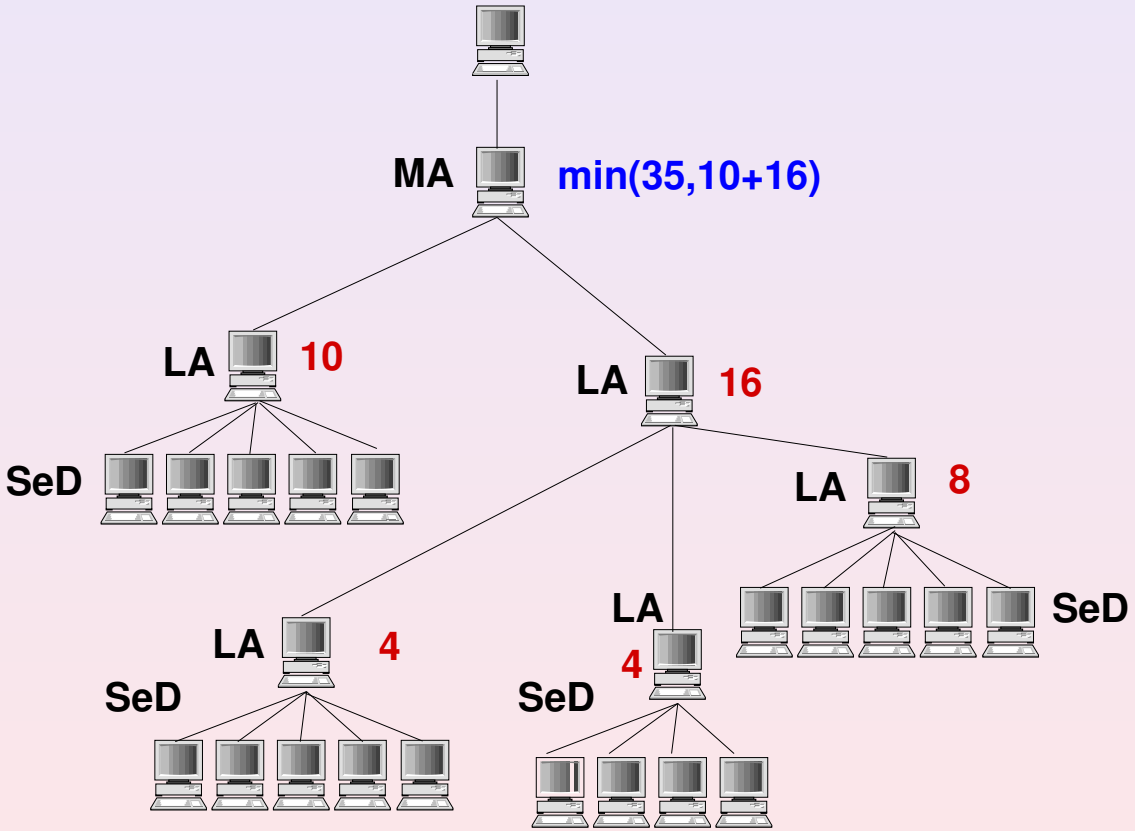
Automatic redeployment



Automatic redeployment

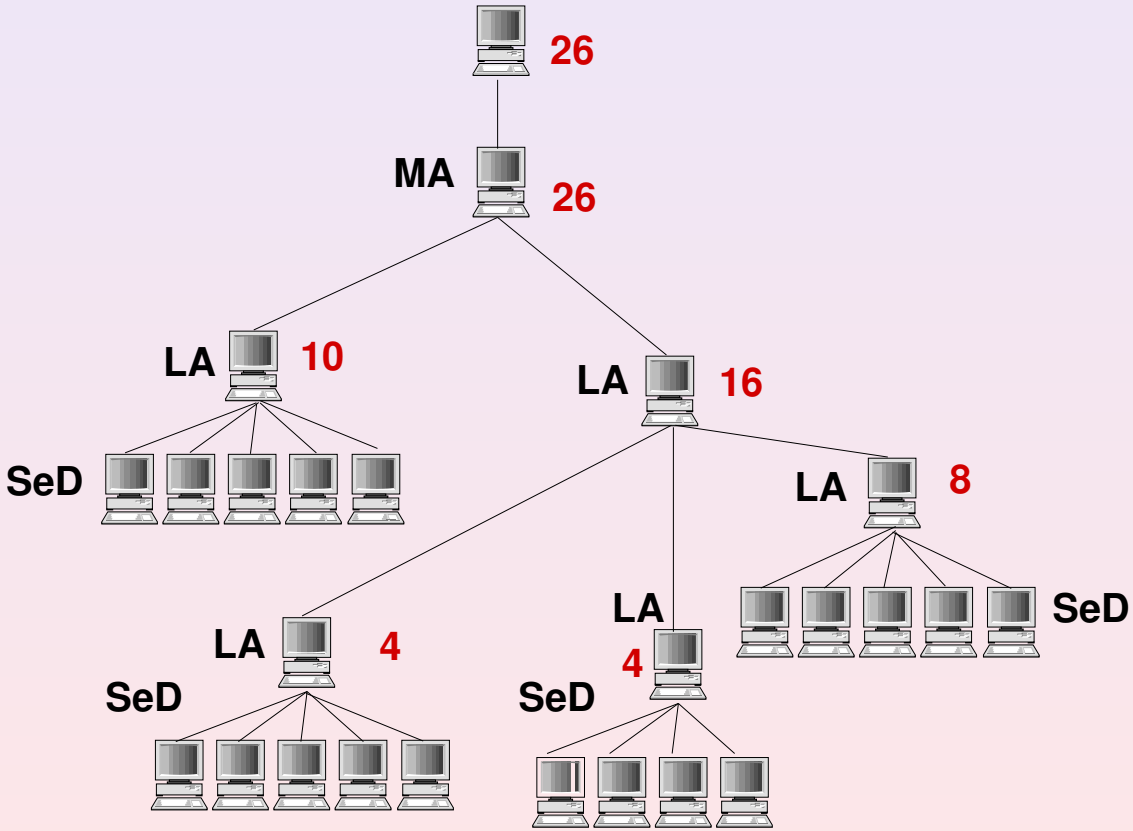


Automatic redeployment



Automatic redeployment

Platform throughput is increased to 26 requests per second



Outline

Measured Parameters

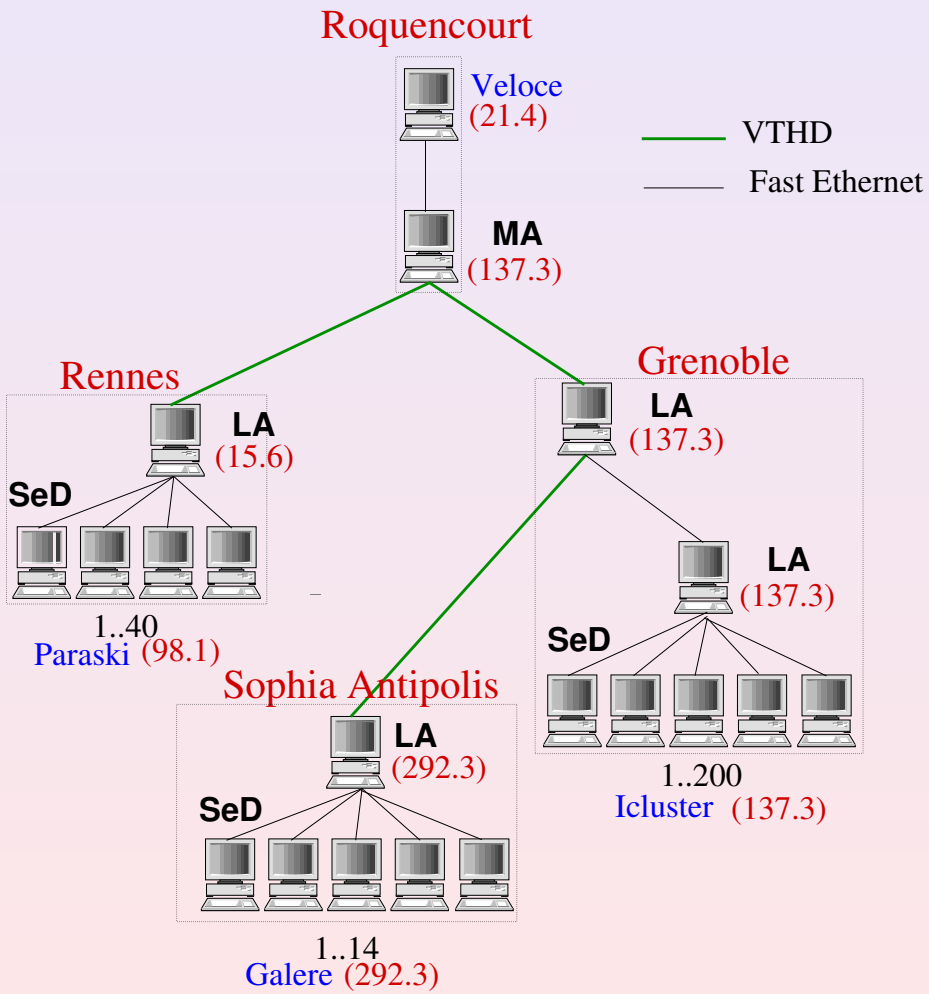
- Cluster of 16 dual-PIII 1.4Ghz at ENS-Lyon

Components	$S_i^{(in)}$	$W_i^{(in)}$	$W_i^{(out)}$
Client	0.339	0.014	0
MA	0.010	0.159	0.78 e-3
LA	0.012	0.079	0.19 e-3

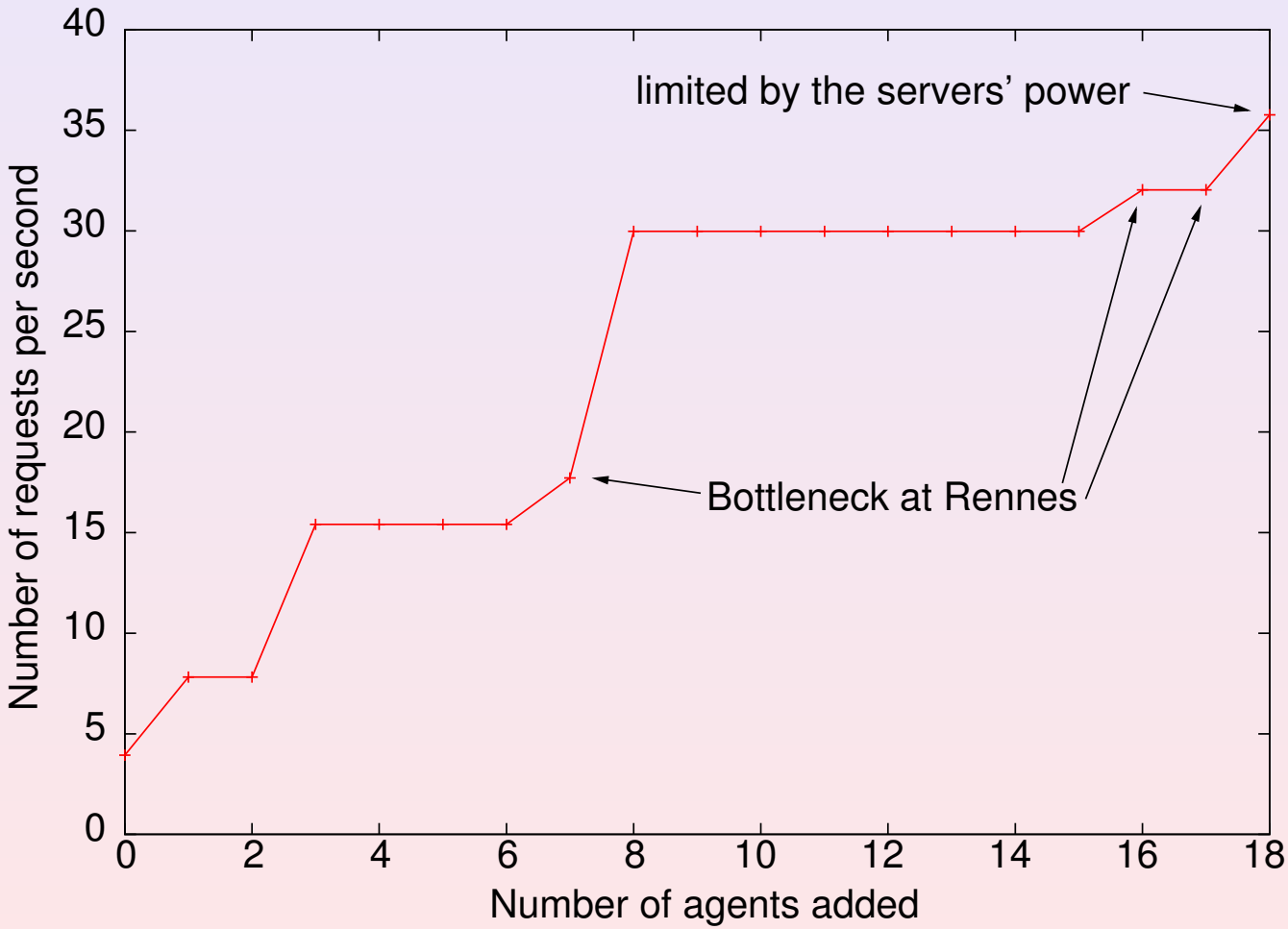
VTHD Network

- High speed network (2.5Gb/s) between INRIA research centers and several other research institutes.

Testbed


Roquencourt
Veloce
(21.4)
VTHD
Fast Ethernet
MA
(137.3)
Rennes
LA
(15.6)
SeD
1..40
Paraski (98.1)
Grenoble
LA
(137.3)
LA
(137.3)
SeD
1..200
Icluster (137.3)
Sophia Antipolis
LA
(292.3)
SeD
1..14
Galere (292.3)

Throughput of the platform



Outline

Conclusion

- Improve the throughput of the network
  - Remove bottlenecks
- Predict the performance of the platform
  - Automatic deployment
- Calculate the performance under different criteria
  - Automatic redeployment
- Select best architecture

Future work

- Use SeD as the supporting LA in place of adding a new component
- Better algorithms for improving the platform throughput
- Calculate the throughput of structures with multi-client and multi-master agents
- Combine scheduling and deployment to increase the performance
- Automatic deployment with component reconfiguration