Automatic Middleware Deployment Planning on Clusters

Pushpinder Kaur Chouhan, Holly Dail,
Eddy Caron, and Frédéric Vivien

6 Oct 2005
GRAAL Group Meeting

Outline

Outline

What is Deployment

A deployment is the mapping of a common platform and middleware across many resources.

- Software deployment maps and distributes a collection of software components on a set of resources. Software deployment includes activities such as releasing, configuring, installing, updating, adapting, de-installing, and even de-releasing a software system.
- System deployment involves two steps, physical and logical. In physical deployment all hardware is assembled (network, CPU, power supply etc), whereas logical deployment is organizing and naming whole cluster nodes as master, slave, etc.

What is Deployment

A deployment is the mapping of a common platform and middleware across many resources.

- Software deployment maps and distributes a collection of software components on a set of resources. Software deployment includes activities such as releasing, configuring, installing, updating, adapting, de-installing, and even de-releasing a software system.
- System deployment involves two steps, physical and logical. In physical deployment all hardware is assembled (network, CPU, power supply etc), whereas logical deployment is organizing and naming whole cluster nodes as master, slave, etc.

Problem Statement

- How to carry out an adapted deployment of middleware services on a cluster with hundreds of nodes?
- Which resources should be used?
- How many resources should be used?
- Should the fastest and best-connected resource be used for middleware or as a computational resource?

Problem Statement

- How to carry out an adapted deployment of middleware services on a cluster with hundreds of nodes?
- Which resources should be used?
- How many resources should be used?
- Should the fastest and best-connected resource be used for middleware or as a computational resource?

Problem Statement

- How to carry out an adapted deployment of middleware services on a cluster with hundreds of nodes?
- Which resources should be used?
- How many resources should be used?
- Should the fastest and best-connected resource be used for middleware or as a computational resource?

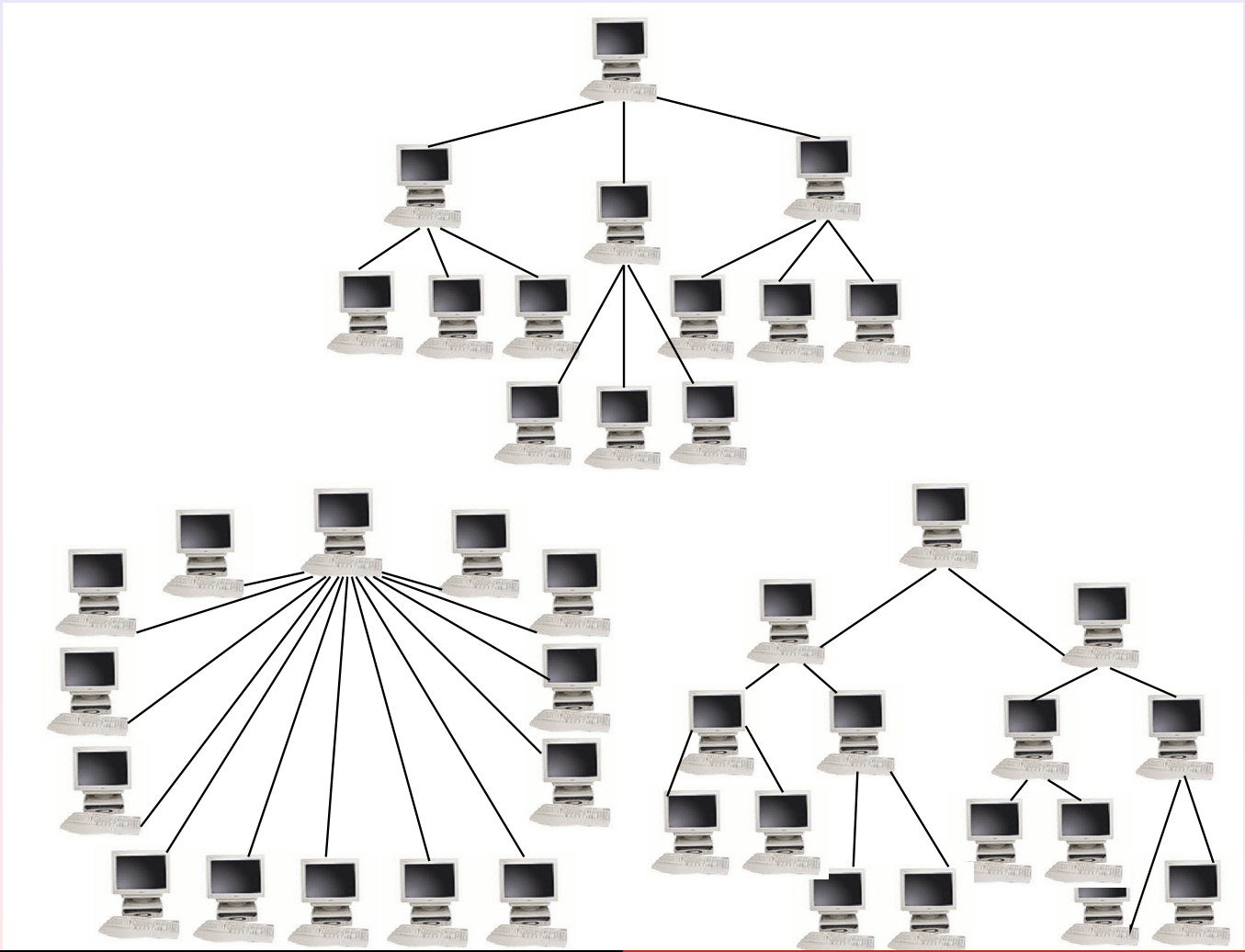
Problem Statement

- How to carry out an adapted deployment of middleware services on a cluster with hundreds of nodes?
- Which resources should be used?
- How many resources should be used?
- Should the fastest and best-connected resource be used for middleware or as a computational resource?

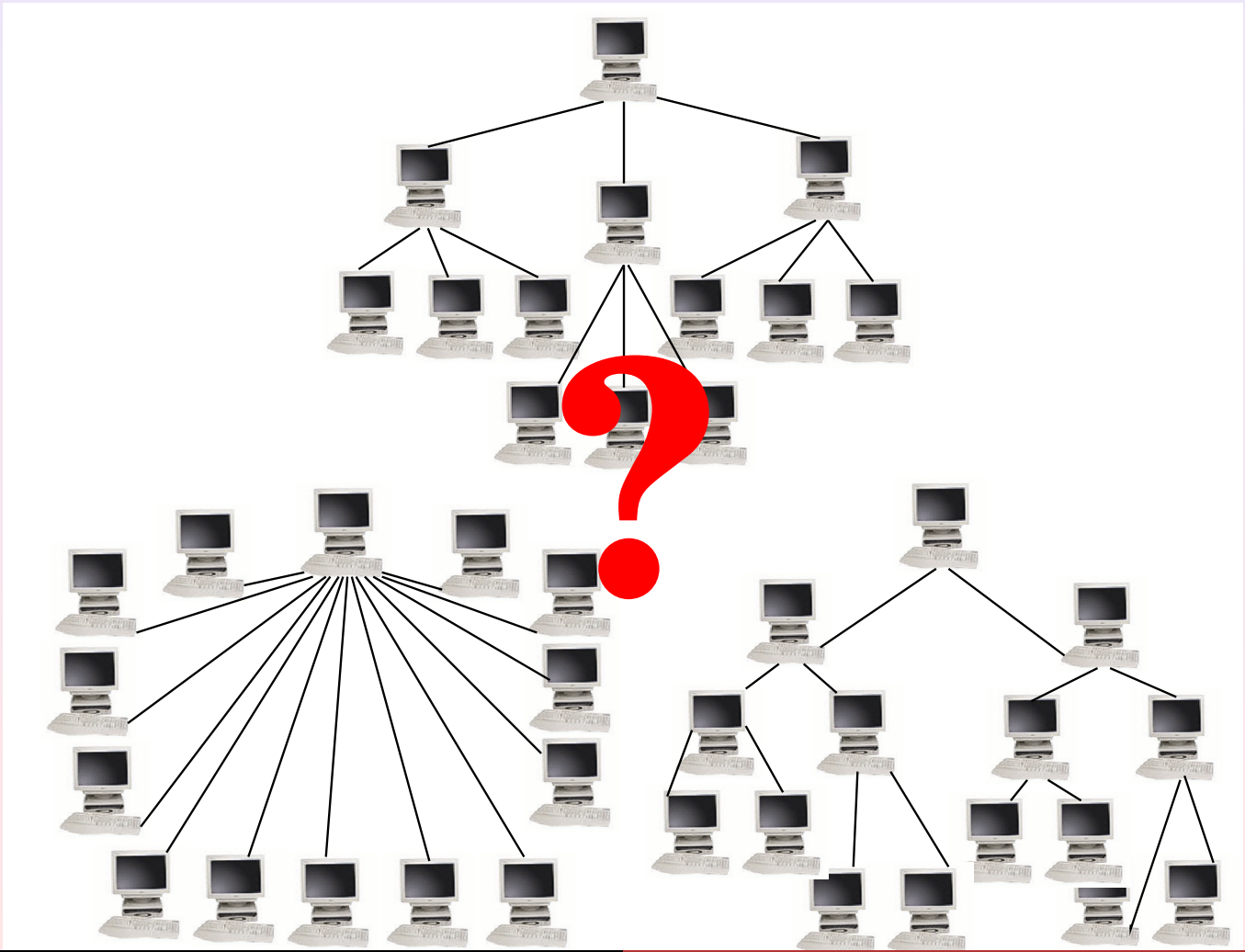
Problem Statement - Diagrammatic Representation

Problem Statement - Diagrammatic Representation

Problem Statement - Diagrammatic Representation



Problem Statement - Diagrammatic Representation



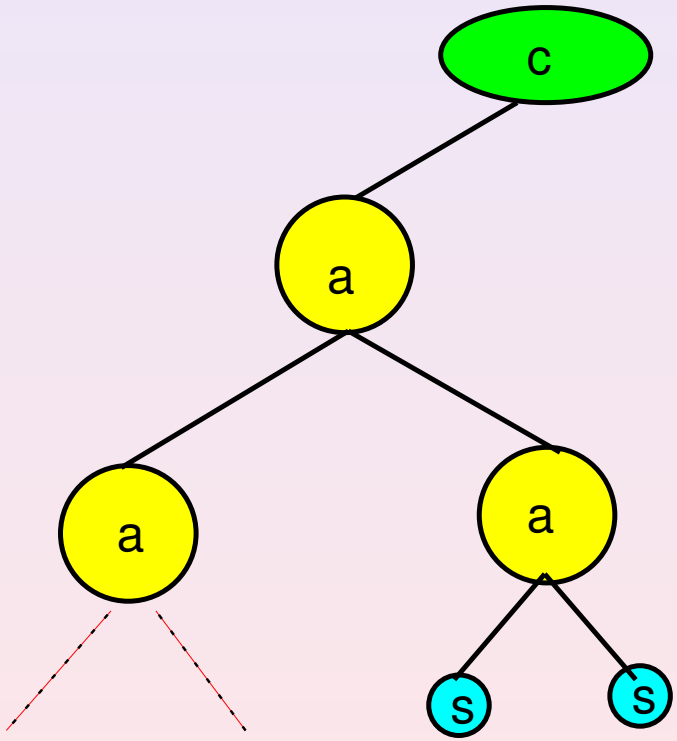
Outline

Objective

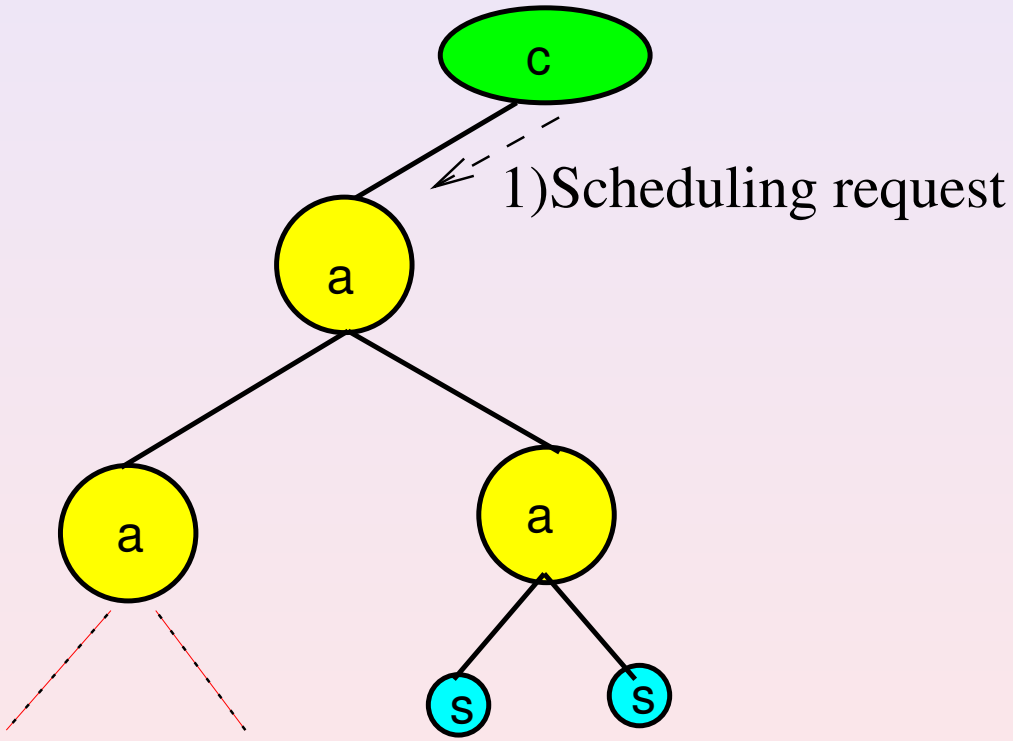
Find an optimal deployment of agents and servers onto a set of resources.

- optimal deployment is the deployment that provides the maximum the throughput.
- ρ is the throughput of the platform calculated as the completed requests per second.

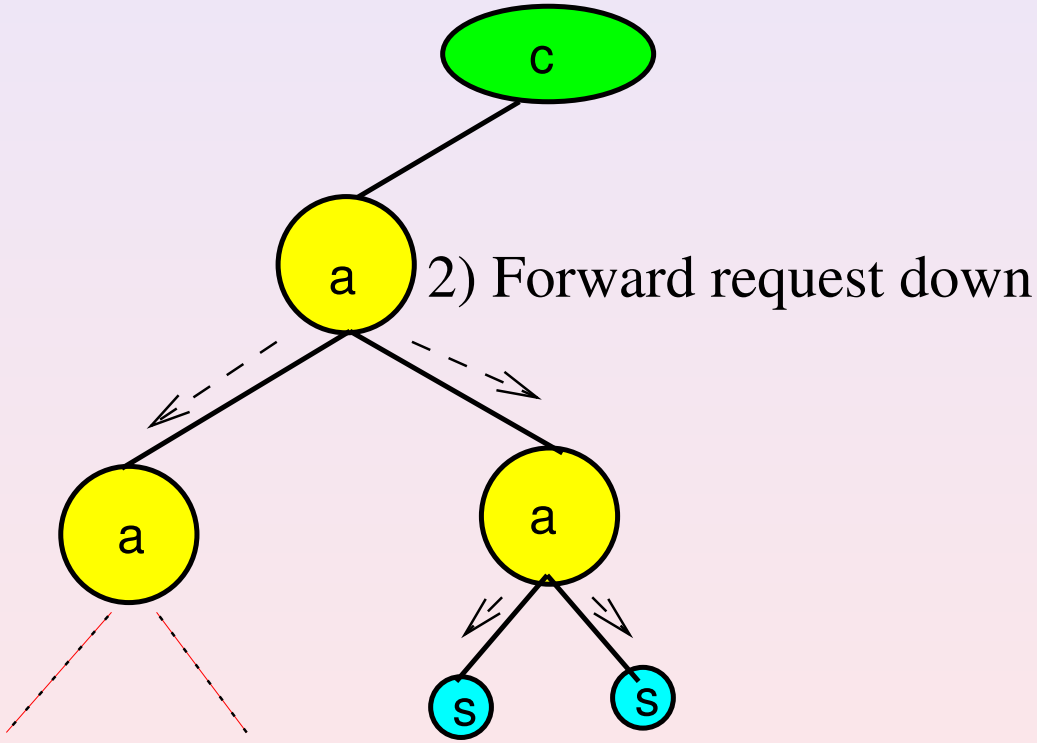
Platform deployment architecture



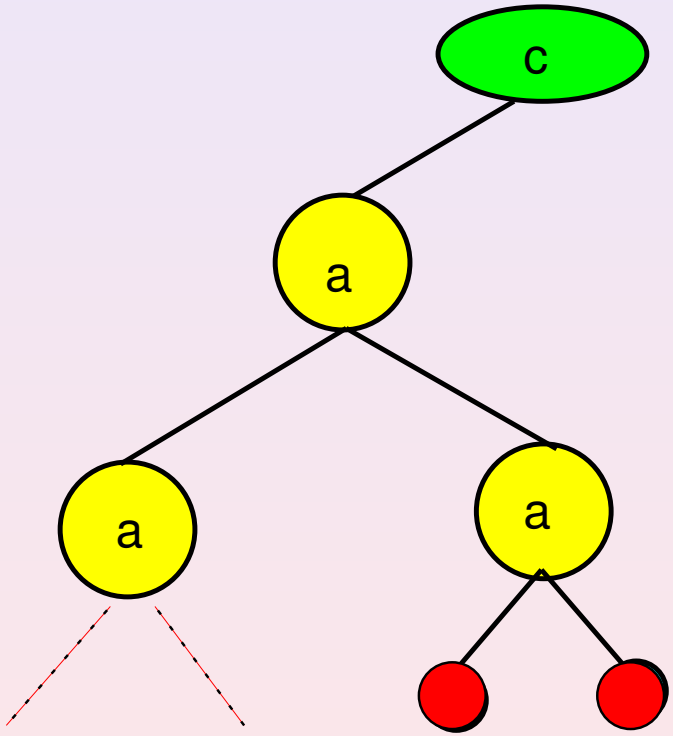
Platform deployment architecture



Platform deployment architecture

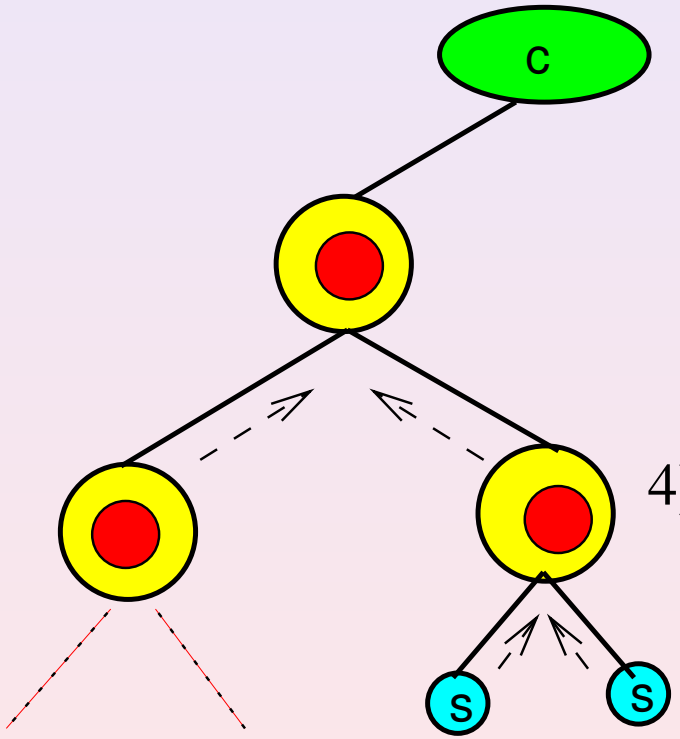


Platform deployment architecture



3) Request prediction
&response generation

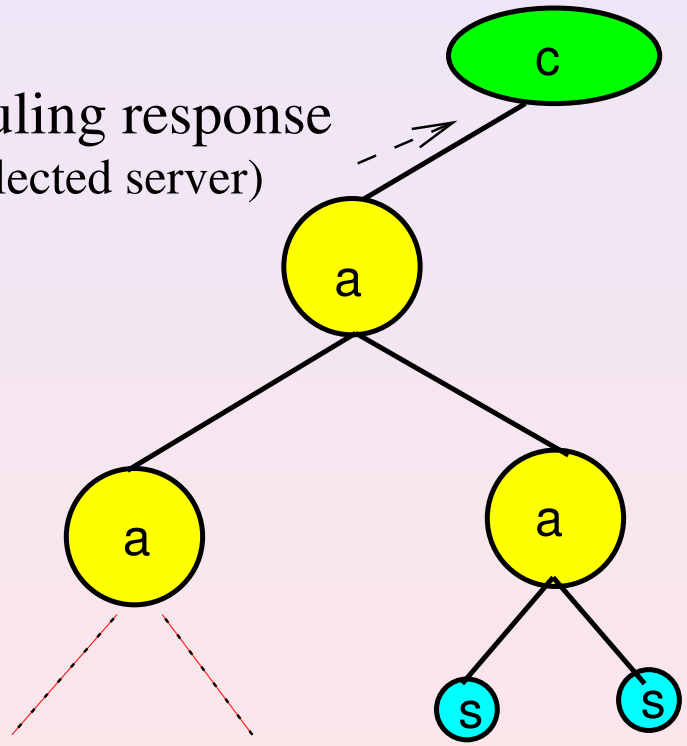
Platform deployment architecture



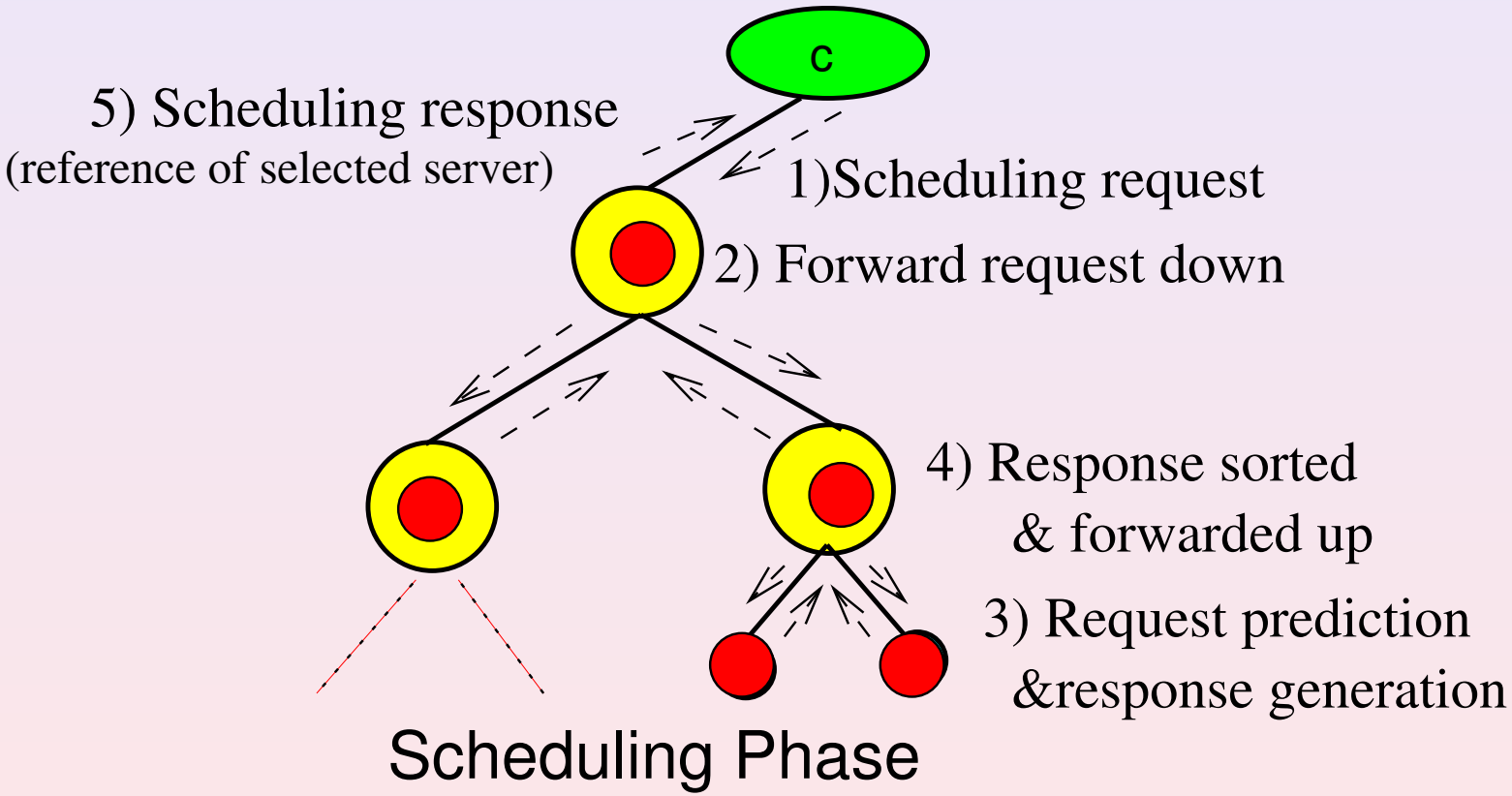
4) Response sorted & forwarded up

Platform deployment architecture

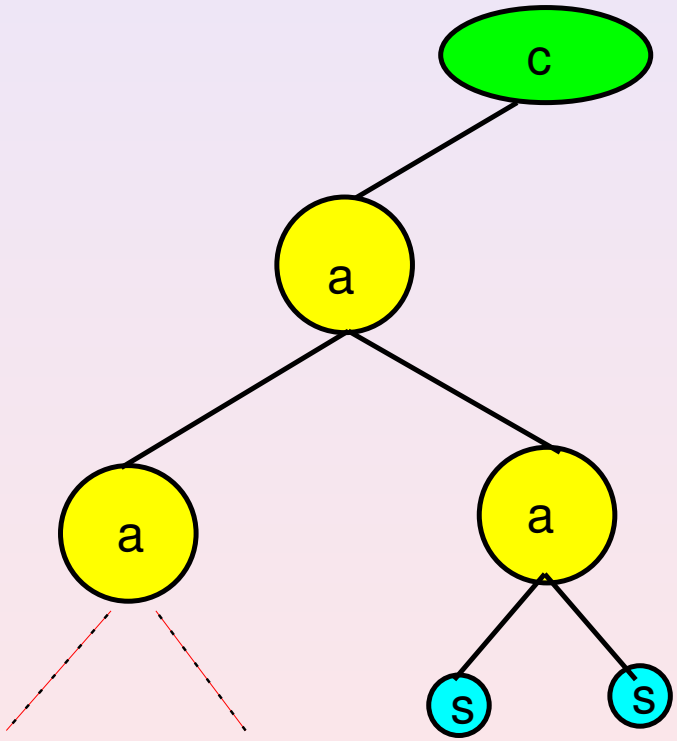
5) Scheduling response
(reference of selected server)



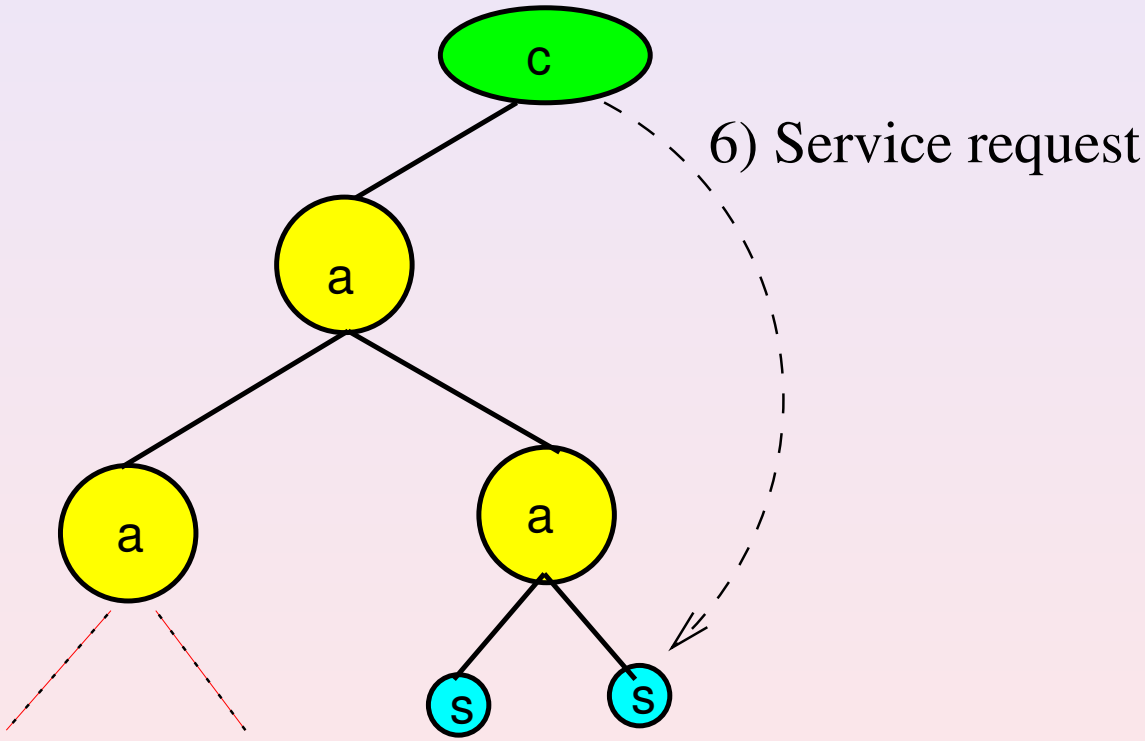
Platform deployment architecture



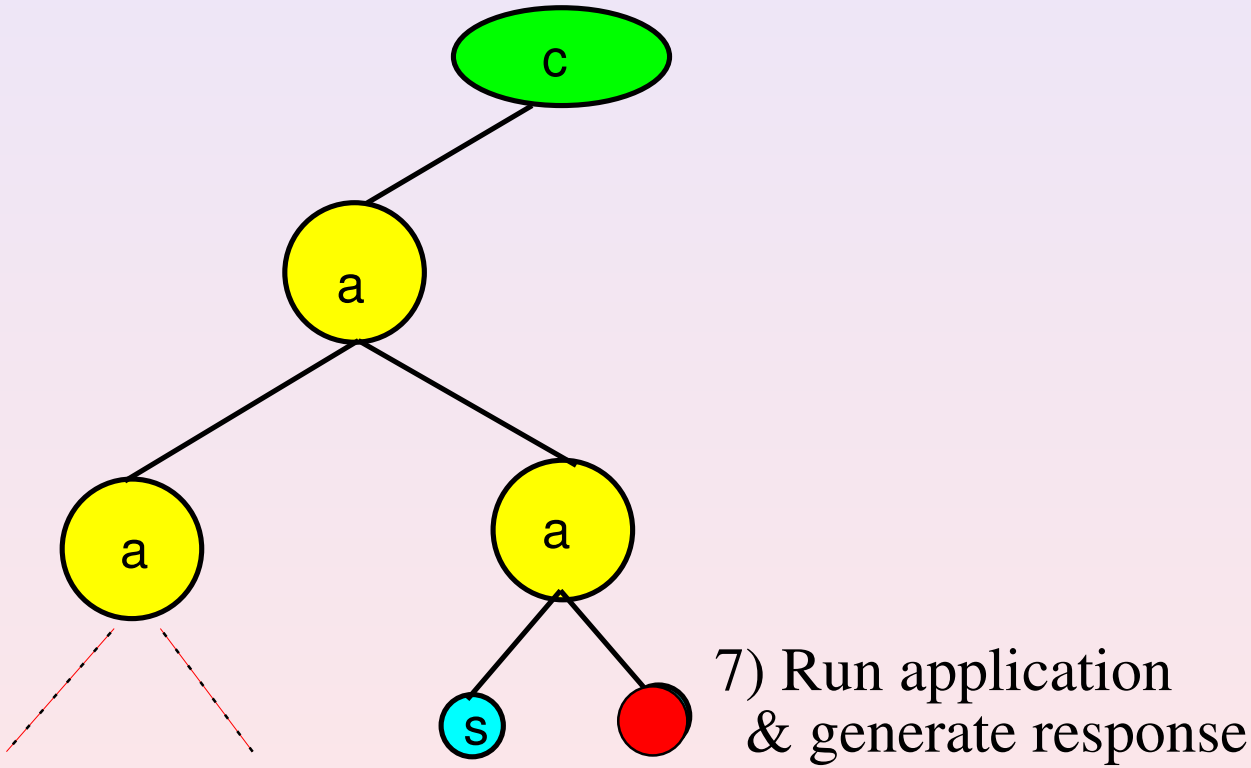
Platform deployment architecture



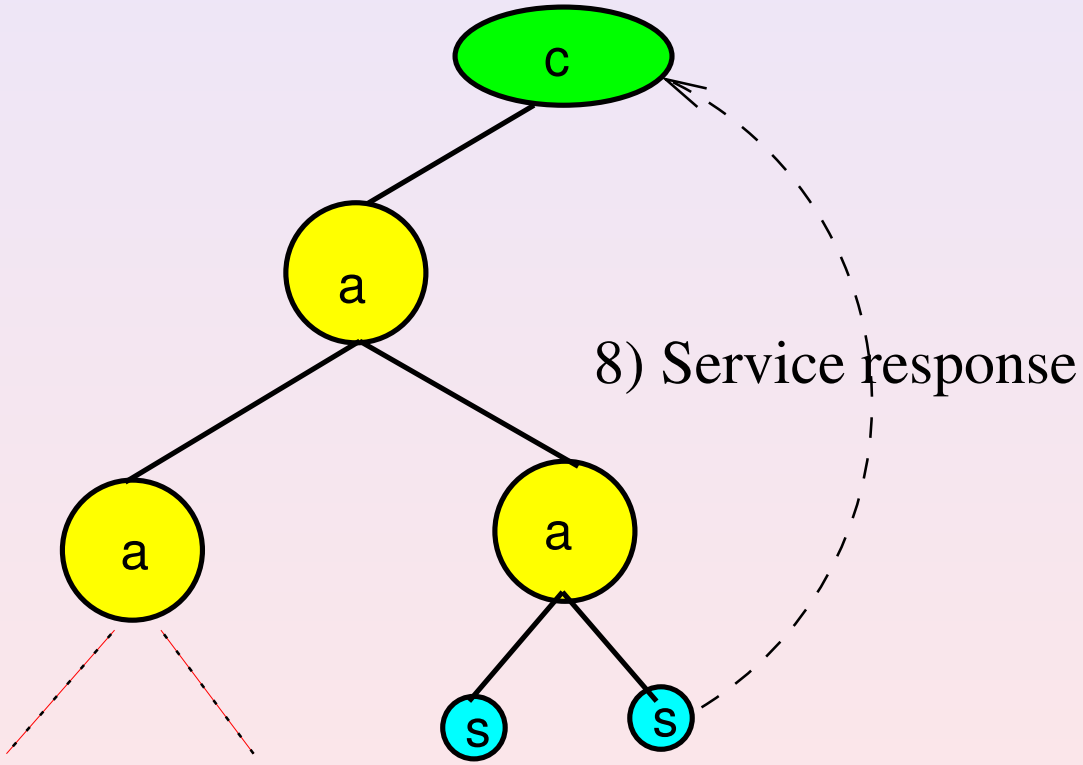
Platform deployment architecture



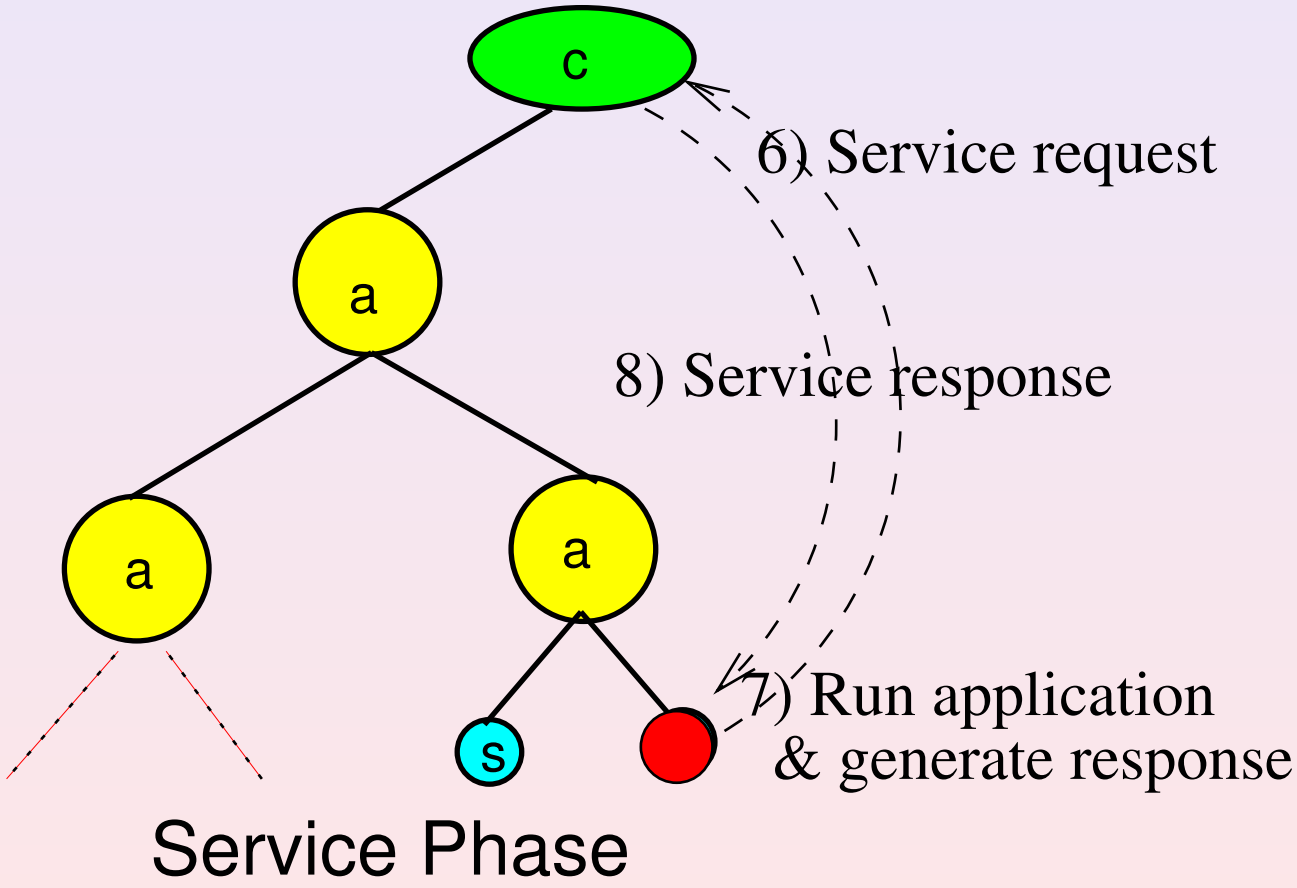
Platform deployment architecture



Platform deployment architecture



Platform deployment architecture



Outline

Deployment planning

Lemma

The completed request throughput ρ of a deployment is given by the minimum of the scheduling request throughput ρ_{sched} and the service request throughput $\rho_{service}$.

$$\rho = min(\rho_{sched}, \rho_{service})$$

- ρ_{sched} the scheduling throughput in requests per second, is defined as the rate at which requests are processed by the scheduling phase.
- $\rho_{service}$ the service throughput in requests per second, is defined as the rate at which requests finish the service response phase.

Deployment planning

Lemma

The scheduling throughput ρ_{sched} is limited by the throughput of the agent with the highest degree.

- Scheduling throughput is controlled bu slowest agent
- Slowest agent is the one with highest degree

Deployment planning

Lemma

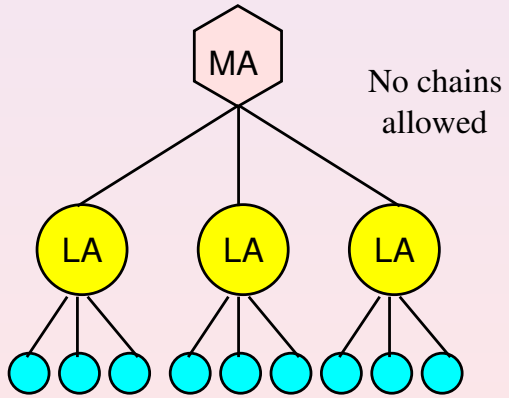
The service request throughput $\rho_{service}$ increases as the number of servers included in a deployment increases.

Service requests are only sent to a single server.

Complete Spanning D-ary tree

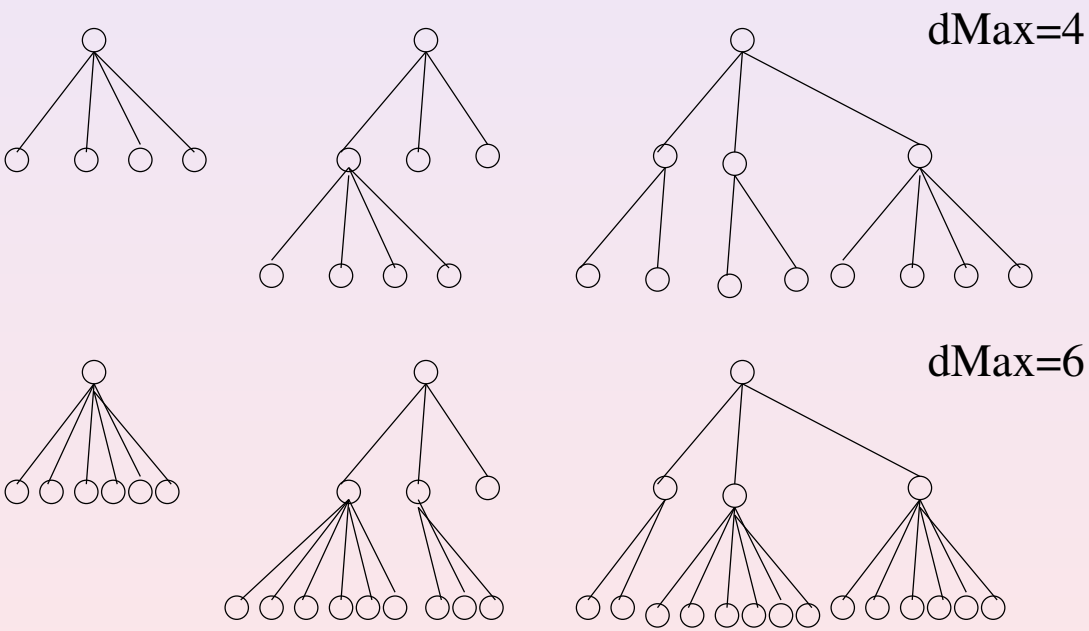
A complete d-ary tree is a tree in which every level, except possibly the deepest, is completely filled. All internal nodes except one have a degree, or number of children, equal to d; the remaining internal node is at depth $n-1$ and may have any degree from 1 to d.
A spanning tree is a connected, acyclic subgraph containing all the vertices of a graph.

A **complete spanning d-ary tree** (CSD tree) is a tree that is both a complete d-ary tree and a spanning tree.



dMax Set

The set of all trees for which maximum degree is equal to $dMax$.



Optimal deployment - Theorems

Theorem

The optimal throughput ρ of any deployment with maximum degree $dMax$ is obtained with a CSD tree.

- By Lemma1 $\rho = min(\rho_{sched}, \rho_{service})$
- By lemma2 ρ_{sched} is limited by agent with maximum degree
- By Lemma3 $\rho_{service}$ increases with $|S|$

Optimal deployment - Theorems

Theorem

The complete spanning d-ary tree with degree $d \in [1, |\mathbb{V}| - 1]$ that maximizes the minimum of the scheduling request and service request throughputs is an optimal deployment.

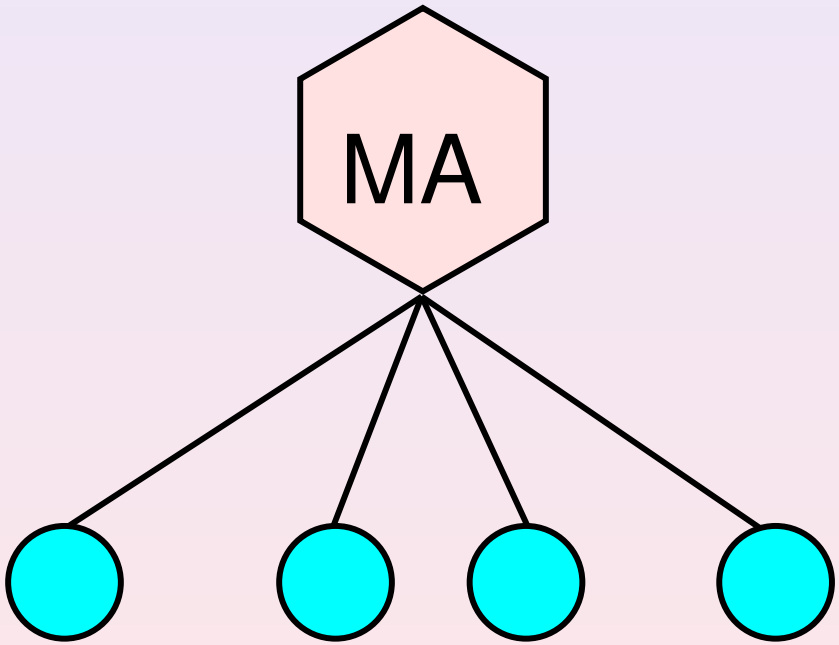
- Test all possible degrees $d \in [1, |\mathbb{V}| - 1]$
- Select MAX min$(\rho_{sched}, \rho_{service})$

Optimal tree construction

n=12 d=4

n_used= 1

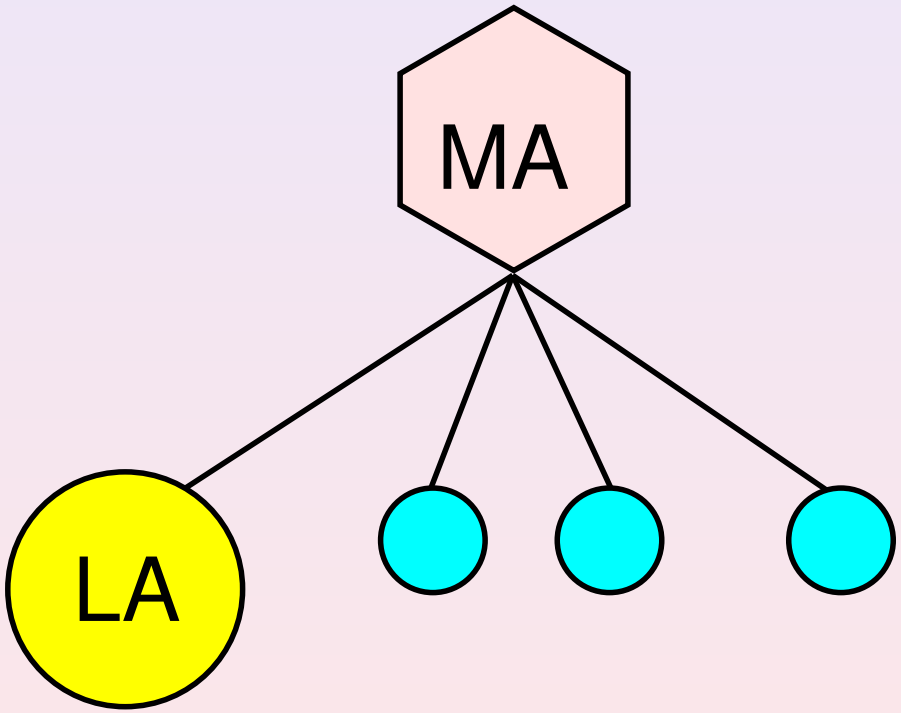
Optimal tree construction



n=12 d=4

n_used= 5

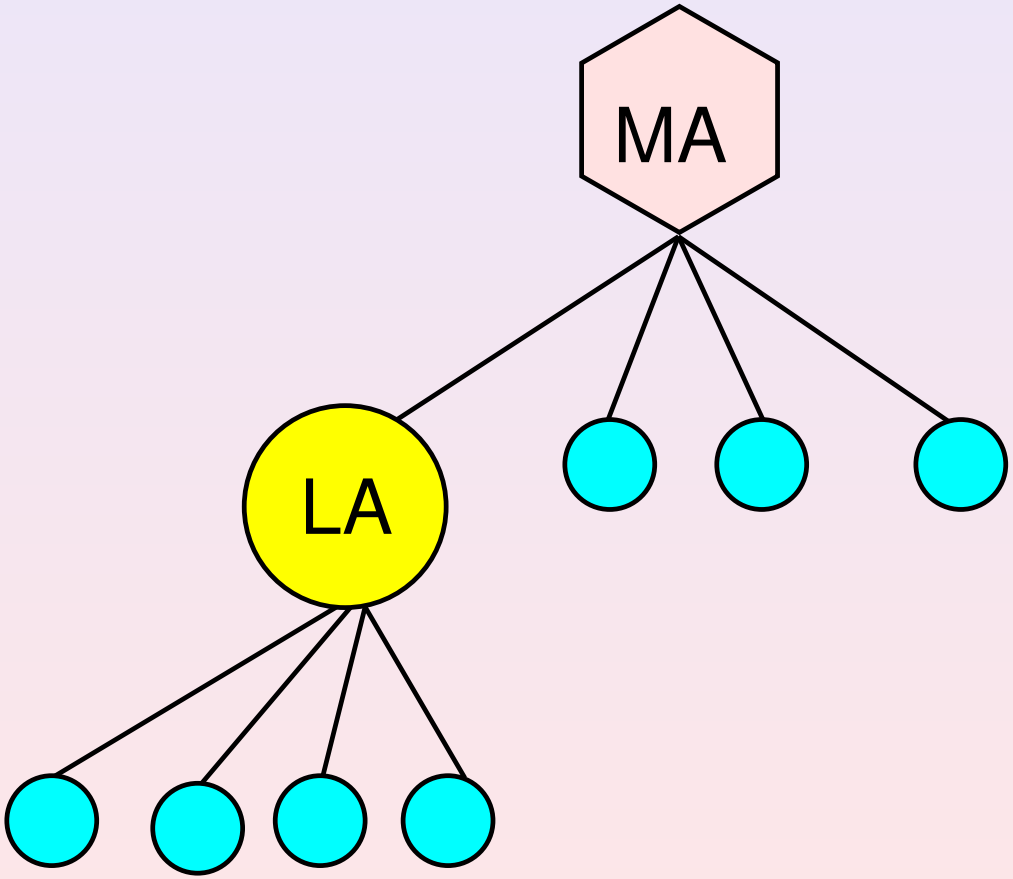
Optimal tree construction



n=12 d=4

n_used= 5

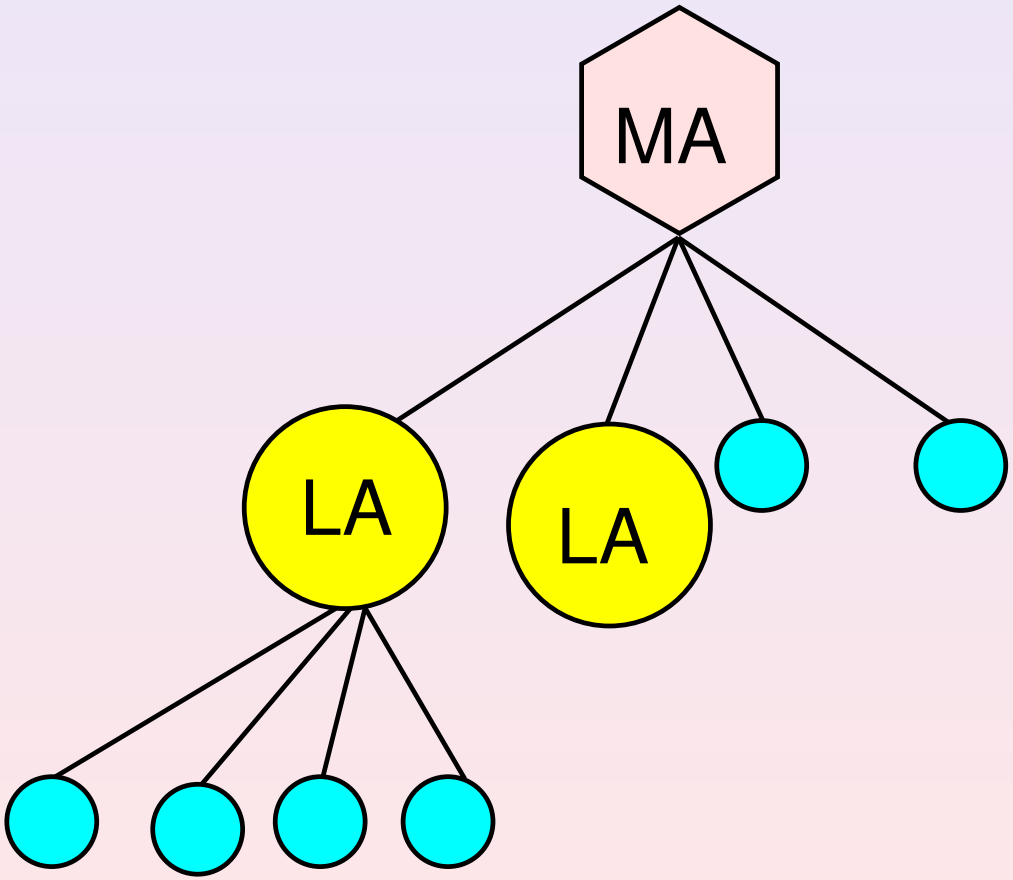
Optimal tree construction



n=12 d=4

n_used= 9

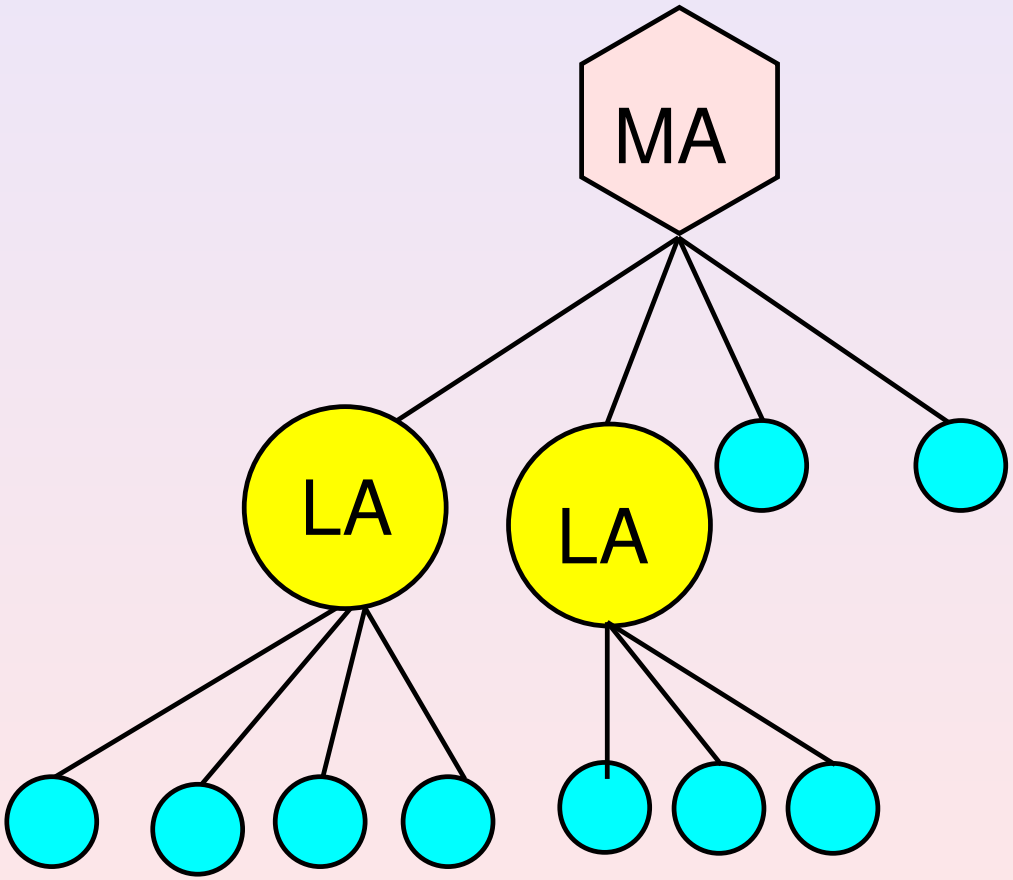
Optimal tree construction



n=12 d=4

n_used= 9

Optimal tree construction



n=12 d=4

n_used= 12

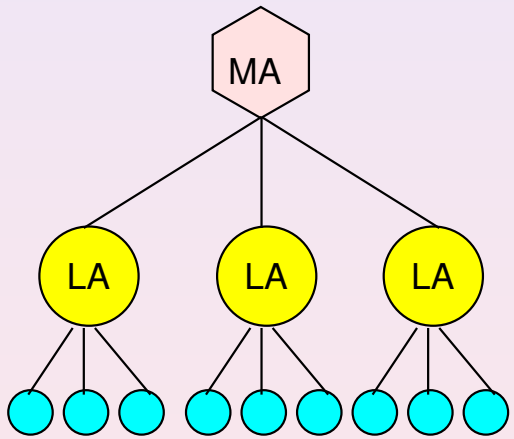
Outline

Model with DIET

- Root of the tree is always an MA
- The MA and LA are considered as having the same performance
- An MA can connect either agents or servers or both
- All clients will submit their request to the DIET hierarchy through one MA

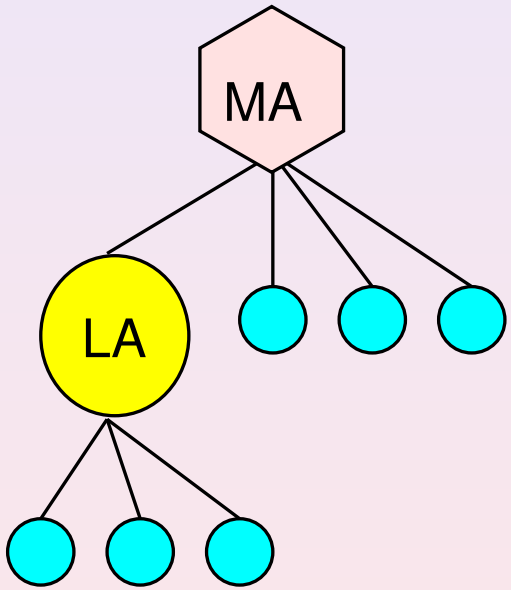
Model with DIET

- Root of the tree is always an MA
- The MA and LA are considered as having the same performance
- An MA can connect either agents or servers or both
- All clients will submit their request to the DIET hierarchy through one MA



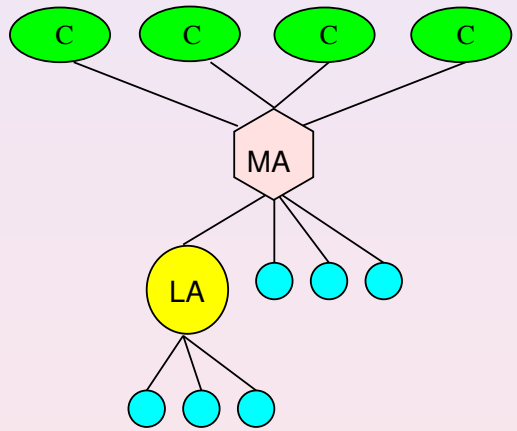
Model with DIET

- Root of the tree is always an MA
- The MA and LA are considered as having the same performance
- An MA can connect either agents or servers or both
- All clients will submit their request to the DIET hierarchy through one MA



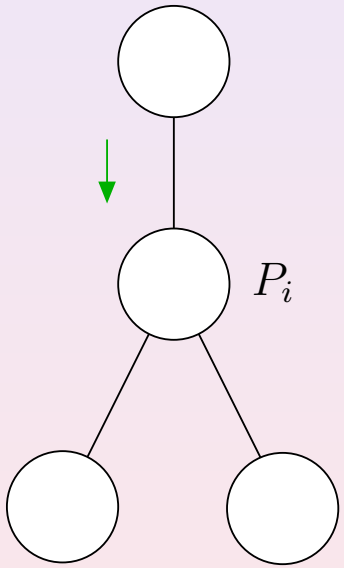
Model with DIET

- Root of the tree is always an MA
- The MA and LA are considered as having the same performance
- An MA can connect either agents or servers or both
- All clients will submit their request to the DIET hierarchy through one MA



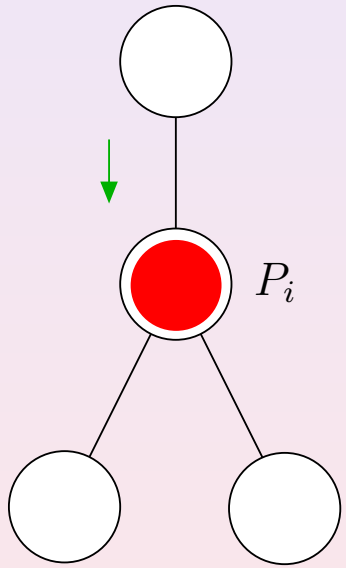
Deployment construction for DIET

- S_{req} is the size in Mb of the message forwarded down the agent hierarchy for a scheduling request.
- W_{req} is the amount of computation in MFlop needed by an agent to process one incoming request.
- W_{pre} is the amount of computation in Mflop needed for a server to predict its own performance for a request.



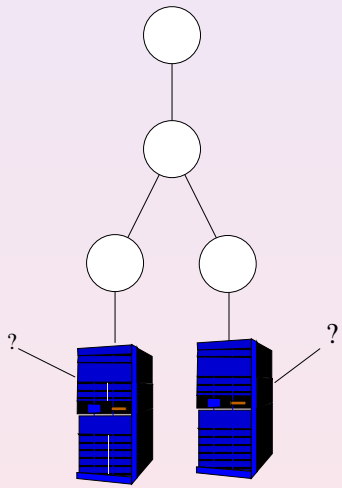
Deployment construction for DIET

- S_{req} is the size in Mb of the message forwarded down the agent hierarchy for a scheduling request.
- W_{req} is the amount of computation in MFlop needed by an agent to process one incoming request.
- W_{pre} is the amount of computation in Mflop needed for a server to predict its own performance for a request.



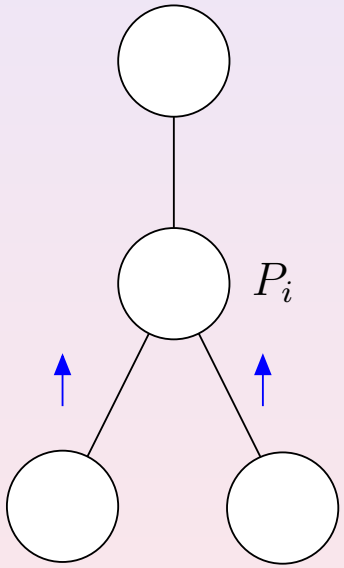
Deployment construction for DIET

- S_{req} is the size in Mb of the message forwarded down the agent hierarchy for a scheduling request.
- W_{req} is the amount of computation in MFlop needed by an agent to process one incoming request.
- W_{pre} is the amount of computation in Mflop needed for a server to predict its own performance for a request.



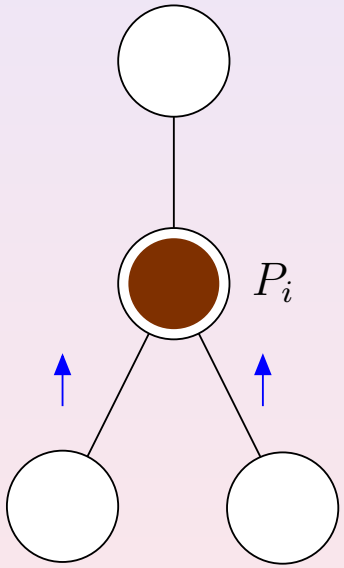
Deployment construction for DIET

- S_{rep} is the size in Mb of the reply forwarded back up the agent hierarchy.
- W_{rep} is the amount of computation in MFlop needed by an agent to merge the replies from its children.
- W_{app} is the amount of computation in Mflop needed by a server to complete a service request for **app** service.



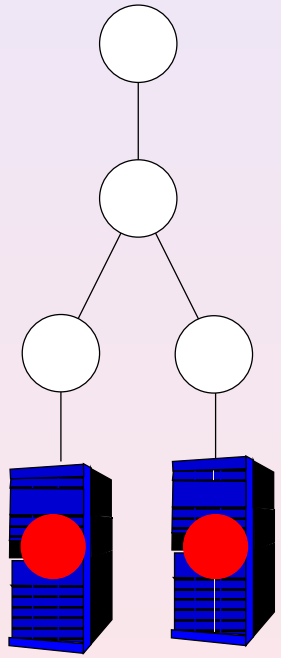
Deployment construction for DIET

- S_{rep} is the size in Mb of the reply forwarded back up the agent hierarchy.
- W_{rep} is the amount of computation in MFlop needed by an agent to merge the replies from its children.
- W_{app} is the amount of computation in Mflop needed by a server to complete a service request for **app** service.



Deployment construction for DIET

- S_{rep} is the size in Mb of the reply forwarded back up the agent hierarchy.
- W_{rep} is the amount of computation in MFlop needed by an agent to merge the replies from its children.
- W_{app} is the amount of computation in Mflop needed by a server to complete a service request for **app** service.

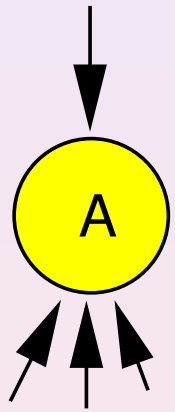


Deployment constraint for an Agent

- Agent communication:
  - agent_receive_time $= \frac{S_{req}+d \cdot S_{rep}}{B}$
  - agent_send_time $= \frac{d \cdot S_{req}+S_{rep}}{B}$
- Agent computation:
  - agent_comp_time $= \frac{W_{sel}+W_{aga}(d)}{w}$

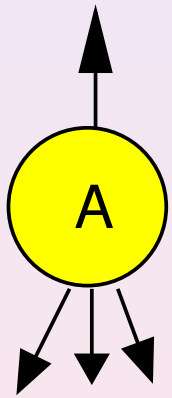
Deployment constraint for an Agent

- Agent communication:
  - agent_receive_time $= \frac{S_{req}+d \cdot S_{rep}}{B}$
  - agent_send_time $= \frac{d \cdot S_{req}+S_{rep}}{B}$
- Agent computation:
  - agent_comp_time $= \frac{W_{req}+W_{rep}(d)}{w}$



Deployment constraint for an Agent

- Agent communication:
  - agent_receive_time $= \frac{S_{req}+d \cdot S_{rep}}{B}$
  - agent_send_time $= \frac{d \cdot S_{req}+S_{rep}}{B}$
- Agent computation:
  - agent_comp_time$= \frac{W_{req}+W_{rep}(d)}{w}$

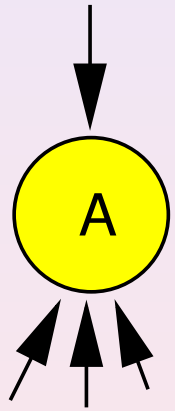


Deployment constraint for an Agent

- Agent communication:
  - agent_receive_time $= \frac{S_{req}+d \cdot S_{rep}}{B}$
  - agent_send_time $= \frac{d \cdot S_{req}+S_{rep}}{B}$
- Agent computation:
  - agent_comp_time$= \frac{W_{req}+W_{rep}(d)}{w}$

Deployment constraint for an Agent

- Agent communication:
  - agent_receive_time $= \frac{S_{req}+d \cdot S_{rep}}{B}$
  - agent_send_time $= \frac{d \cdot S_{req}+S_{rep}}{B}$
- Agent computation:
  - agent_comp_time$= \frac{W_{req}+W_{rep}(d)}{w}$

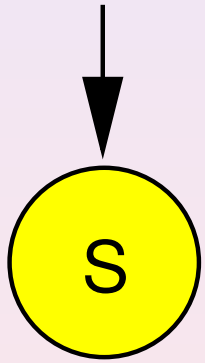


Deployment constraint for a Server

- Server communication:
  - server_receive_time $= \frac{S_{req}}{B}$
  - server_send_time $= \frac{S_{rep}}{B}$
- Server computation:
  - all_servers_comp_time $= \frac{W_{rep}(|S|) + W_{app}}{w}$
  - server_comp_time $= \frac{W_{pre} + \frac{W_{app}}{|S|}}{w}$

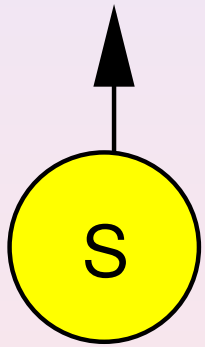
Deployment constraint for a Server

- Server communication:
  - server_receive_time $= \frac{S_{req}}{B}$
  - server_send_time $= \frac{S_{rep}}{B}$
- Server computation:
  - all_servers_comp_time $= \frac{W_{pre} \cdot |S| + W_{app}}{w}$
  - server_comp_time $= \frac{W_{pre} + \frac{W_{app}}{|S|}}{w}$



Deployment constraint for a Server

- Server communication:
  - server_receive_time $= \frac{S_{req}}{B}$
  - server_send_time $= \frac{S_{rep}}{B}$
- Server computation:
  - all_servers_comp_time $= \frac{W_{pre} \cdot |\mathbb{S}| + W_{app}}{w}$
  - server_comp_time $= \frac{W_{pre} + \frac{W_{app}}{|\mathbb{S}|}}{w}$

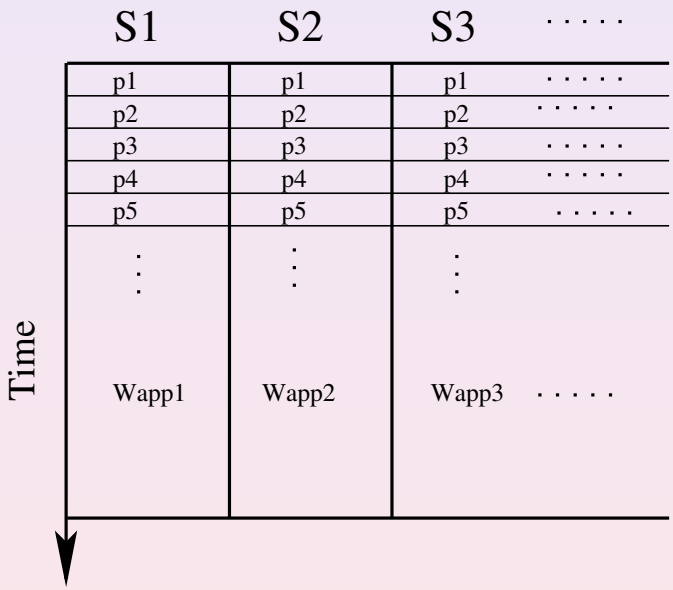


Deployment constraint for a Server

- Server communication:
  - server_receive_time $= \frac{S_{req}}{B}$
  - server_send_time $= \frac{S_{rep}}{B}$
- Server computation:
  - all_servers_comp_time $= \frac{W_{pre} \cdot |\mathbb{S}| + W_{app}}{w}$
  - server_comp_time $= \frac{W_{pre} + \frac{W_{app}}{|\mathbb{S}|}}{w}$

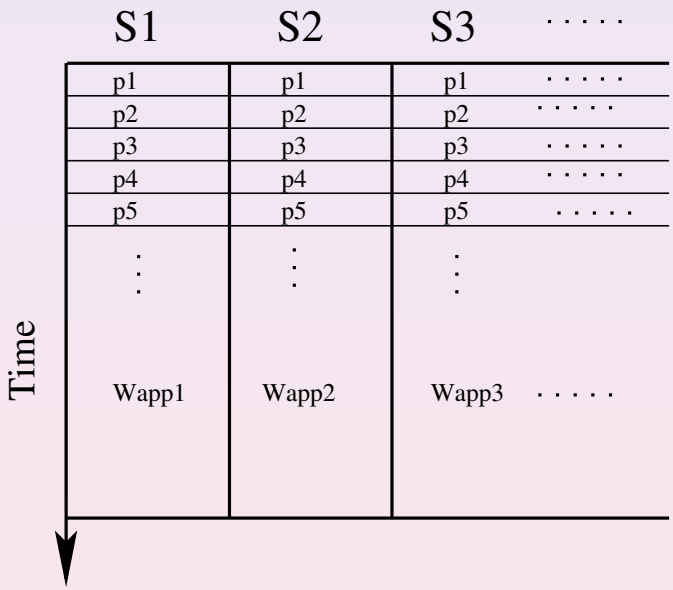
Deployment constraint for a Server

- Server communication:
  - server_receive_time $= \frac{S_{req}}{B}$
  - server_send_time $= \frac{S_{rep}}{B}$
- Server computation:
  - all_servers_comp_time $= \frac{W_{pre} \cdot |\mathbb{S}| + W_{app}}{w}$
  - server_comp_time $= \frac{W_{pre} + \frac{W_{app}}{|\mathbb{S}|}}{w}$



Deployment constraint for a Server

- Server communication:
  - server_receive_time $= \frac{S_{req}}{B}$
  - server_send_time $= \frac{S_{rep}}{B}$
- Server computation:
  - all_servers_comp_time $= \frac{W_{pre} \cdot |\mathbb{S}| + W_{app}}{w}$
  - server_comp_time $= \frac{W_{pre} + \frac{W_{app}}{|\mathbb{S}|}}{w}$



send or receive or compute, single port

- Scheduling throughput ρ_{sched}:

$$min\left(\frac{1}{\frac{W_{pre}}{w}+\frac{S_{req}}{B}+\frac{S_{rep}}{B}}, \frac{1}{\frac{S_{req}+d\cdot S_{rep}}{B}+\frac{d\cdot S_{req}+S_{rep}}{B}+\frac{W_{req}+W_{rep}(d)}{w}}\right)$$

- Service throughput $\rho_{service}$:

$$\frac{1}{\frac{S_{req}}{B}+\frac{S_{rep}}{B}+\frac{W_{pre}+\frac{W_{app}}{|\mathbb{S}|}}{w}}$$

send or receive or compute, single port

- Scheduling throughput ρ_{sched}:

$$min\left(\frac{1}{\frac{W_{pre}}{w}+\frac{S_{req}}{B}+\frac{S_{rep}}{B}}, \frac{1}{\frac{S_{req}+d\cdot S_{rep}}{B}+\frac{d\cdot S_{req}+S_{rep}}{B}+\frac{W_{req}+W_{rep}(d)}{w}}\right)$$

- Service throughput $\rho_{service}$:

$$\frac{1}{\frac{S_{req}}{B}+\frac{S_{rep}}{B}+\frac{W_{pre}+\frac{W_{app}}{|\mathbb{S}|}}{w}}$$

send || receive || compute, single port

- Scheduling throughput ρ_{sched}:

$$min\left(\frac{1}{max\left(\frac{W_{pre}}{w}, \frac{S_{req}}{B}, \frac{S_{rep}}{B}\right)}, \frac{1}{max\left(\frac{S_{req}+d\cdot S_{rep}}{B}, \frac{d\cdot S_{req}+S_{rep}}{B}, \frac{W_{req}+W_{rep}(d)}{w}\right)}\right)$$

- Service throughput $\rho_{service}$:

$$\frac{1}{max\left(\frac{S_{req}}{B}, \frac{S_{rep}}{B}, \frac{W_{pre}+\frac{W_{app}}{|\mathbb{S}|}}{w}\right)}$$

send || receive || compute, single port

- Scheduling throughput ρ_{sched}:

$$min\left(\frac{1}{max\left(\frac{W_{pre}}{w}, \frac{S_{req}}{B}, \frac{S_{rep}}{B}\right)}, \frac{1}{max\left(\frac{S_{req}+d\cdot S_{rep}}{B}, \frac{d\cdot S_{req}+S_{rep}}{B}, \frac{W_{req}+W_{rep}(d)}{w}\right)}\right)$$

- Service throughput $\rho_{service}$:

$$\frac{1}{max\left(\frac{S_{req}}{B}, \frac{S_{rep}}{B}, \frac{W_{pre}+\frac{W_{app}}{|\mathbb{S}|}}{w}\right)}$$

Model Parametrization

Components	Agent	SeD
W_{req} (Mflop)	1.4×10^6	
W_{rep} (Mflop)	$5.0 \times 10^7 + 5.3 \times 10^7 \cdot d$	
W_{pre} (Mflop)		7.1×10^6
S_{res} (Mbit)	6.4×10^{-5}	6.4×10^{-5}
S_{req} (Mbit)	5.3×10^{-5}	5.3×10^{-5}

Outline

Experimental design

- Software: GoDIET is used to deploy DIET.
- Job types: DGEMM, a simple matrix multiplication (BLAS package).
- Workload: steady-state load with 1 - 200 client scripts (each script launches requests serially)
- Resources: dual AMD Opteron 246 processors @ 2GHz, each with cache size of 1024KB, 2GB of main memory and a 1Gb/s Ethernet
  - Lyon cluster - 55 nodes
  - Sophia cluster - 140 nodes

Experimental design

- Software: GoDIET is used to deploy DIET.
- Job types: DGEMM, a simple matrix multiplication (BLAS package).
- Workload: steady-state load with 1 - 200 client scripts (each script launches requests serially)
- Resources: dual AMD Opteron 246 processors @ 2GHz, each with cache size of 1024KB, 2GB of main memory and a 1Gb/s Ethernet
  - Lyon cluster - 55 nodes
  - Sophia cluster - 140 nodes

Experimental design

- Software: GoDIET is used to deploy DIET.
- Job types: DGEMM, a simple matrix multiplication (BLAS package).
- Workload: steady-state load with 1 - 200 client scripts (each script launches requests serially)
- Resources: dual AMD Opteron 246 processors @ 2GHz, each with cache size of 1024KB, 2GB of main memory and a 1Gb/s Ethernet
  - Lyon cluster - 55 nodes
  - Sophia cluster - 140 nodes

Experimental design

- Software: GoDIET is used to deploy DIET.
- Job types: DGEMM, a simple matrix multiplication (BLAS package).
- Workload: steady-state load with 1 - 200 client scripts (each script launches requests serially)
- Resources: dual AMD Opteron 246 processors @ 2GHz, each with cache size of 1024KB, 2GB of main memory and a 1Gb/s Ethernet
  - Lyon cluster - 55 nodes
  - Sophia cluster - 140 nodes

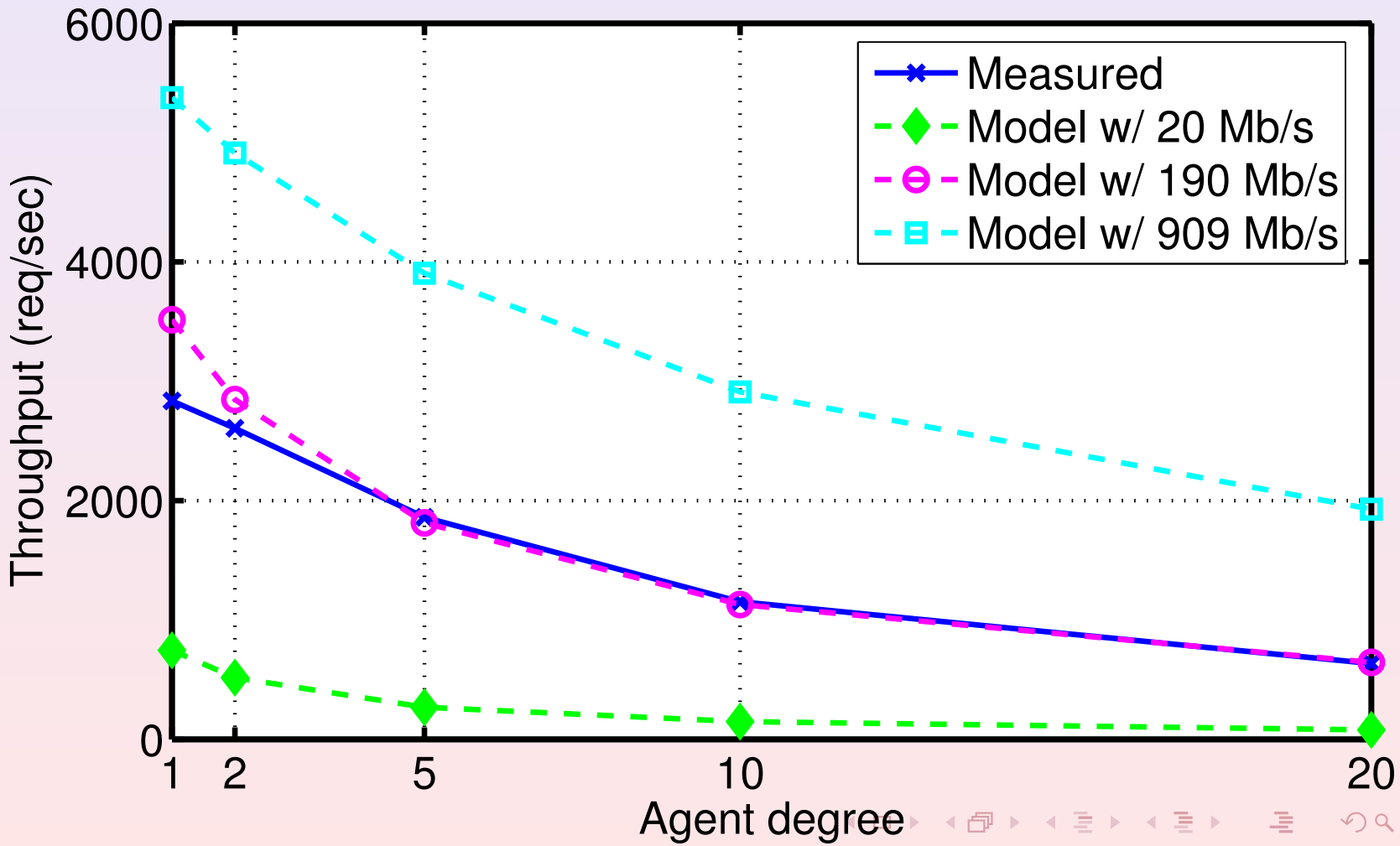
Experimental design

- Software: GoDIET is used to deploy DIET.
- Job types: DGEMM, a simple matrix multiplication (BLAS package).
- Workload: steady-state load with 1 - 200 client scripts (each script launches requests serially)
- Resources: dual AMD Opteron 246 processors @ 2GHz, each with cache size of 1024KB, 2GB of main memory and a 1Gb/s Ethernet
  - Lyon cluster - 55 nodes
  - Sophia cluster - 140 nodes

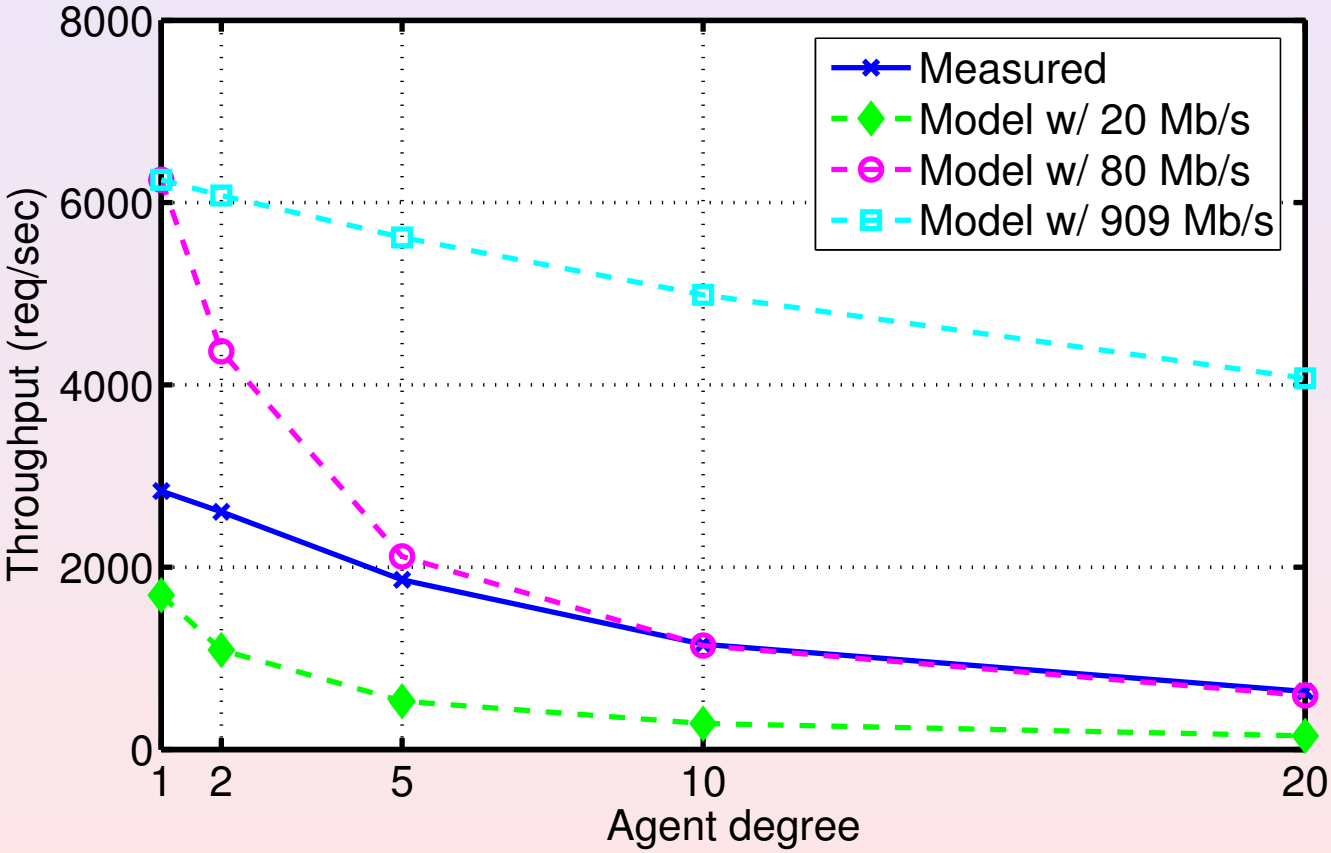
Experimental design

- Software: GoDIET is used to deploy DIET.
- Job types: DGEMM, a simple matrix multiplication (BLAS package).
- Workload: steady-state load with 1 - 200 client scripts (each script launches requests serially)
- Resources: dual AMD Opteron 246 processors @ 2GHz, each with cache size of 1024KB, 2GB of main memory and a 1Gb/s Ethernet
  - Lyon cluster - 55 nodes
  - Sophia cluster - 140 nodes

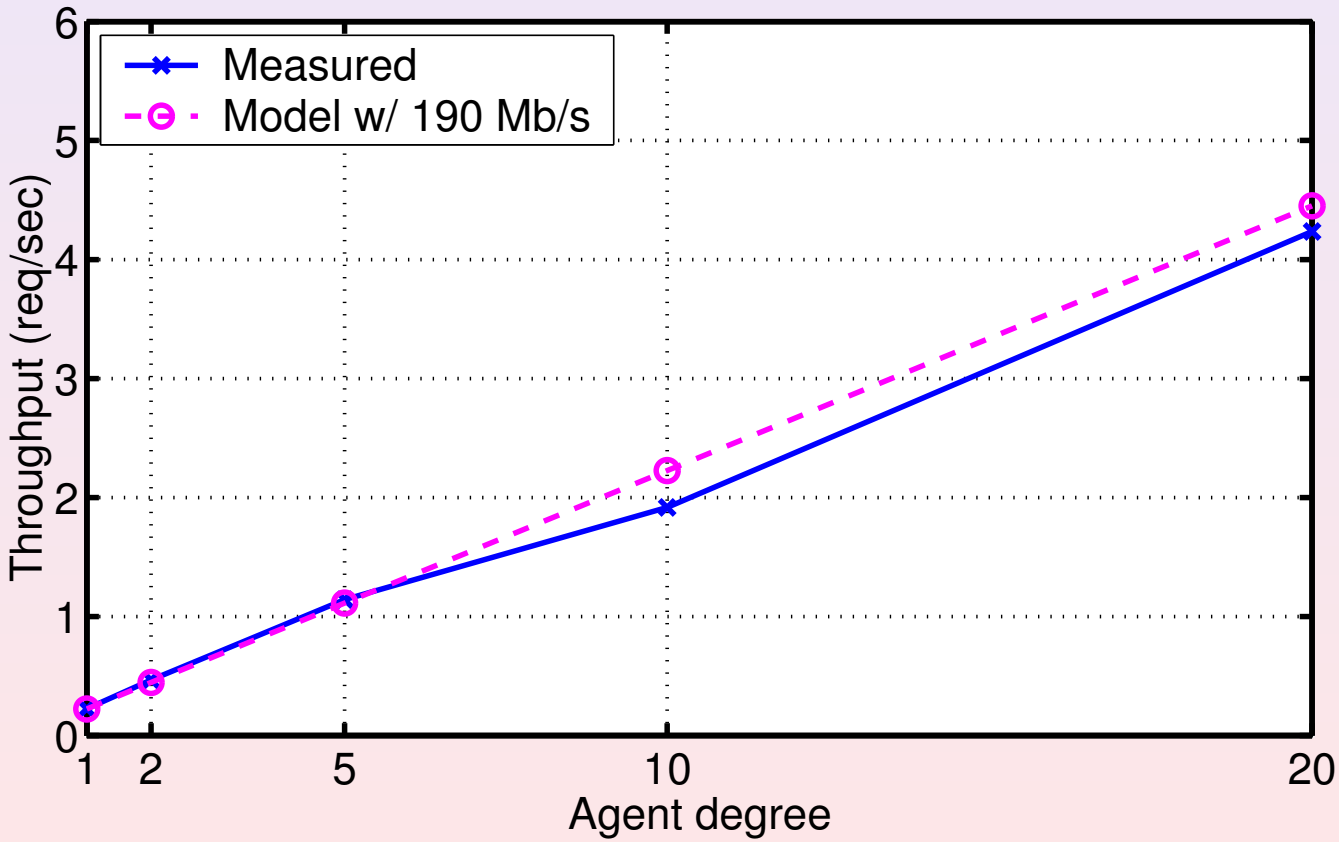
Throughput validation - Serial Model (DGEMM 10)



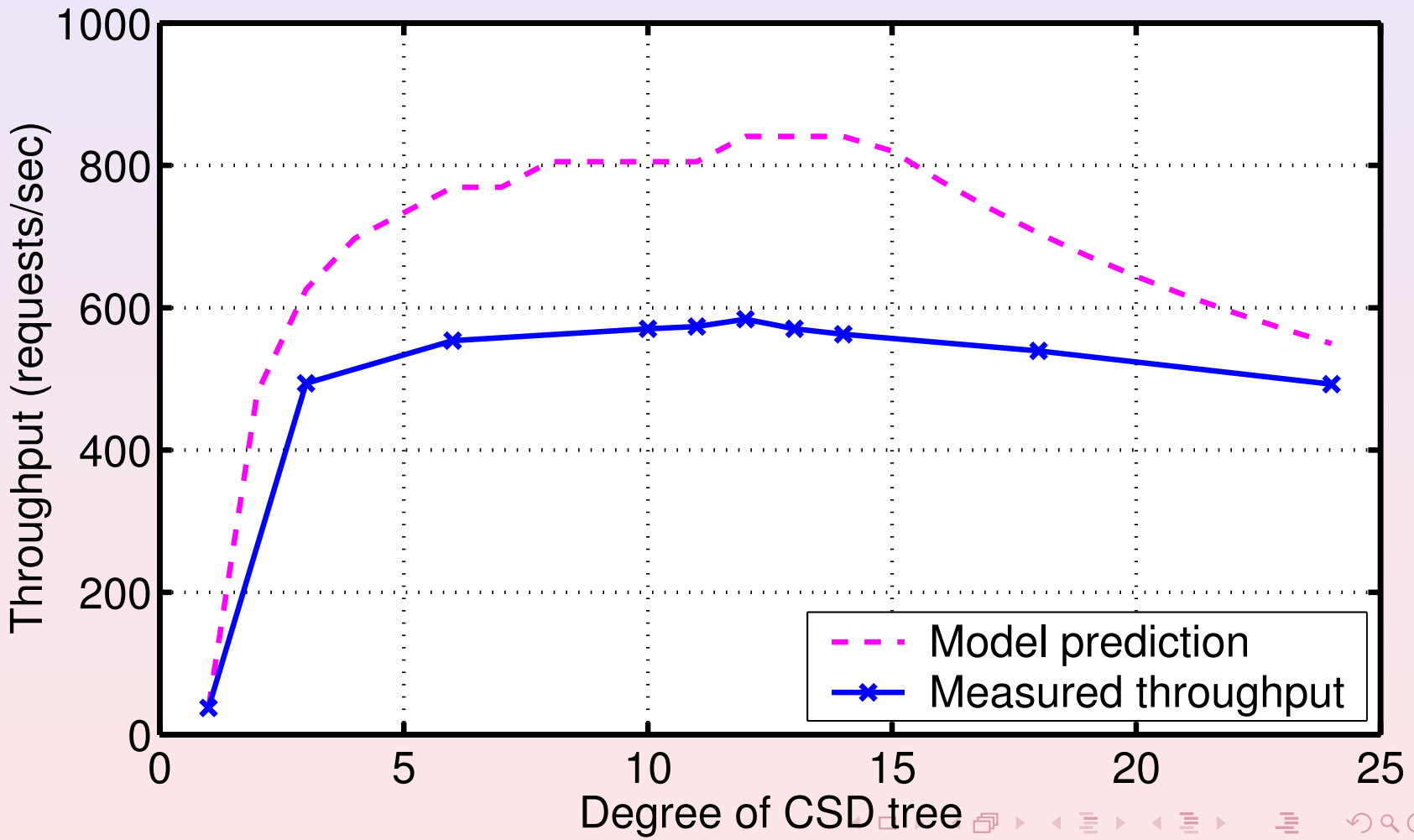
Throughput validation - Parallel Model (DGEMM 10)



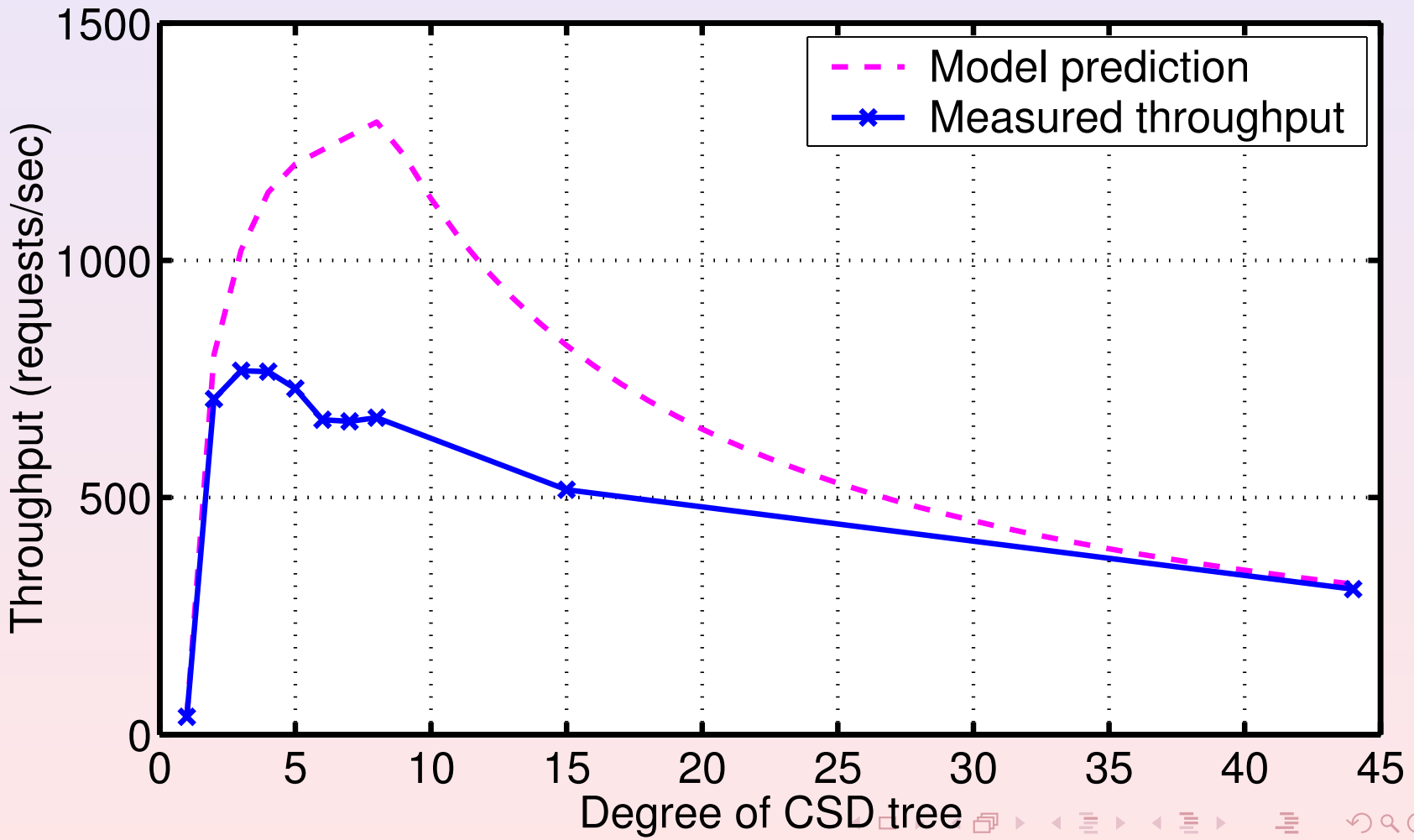
Throughput validation - DGEMM 1000, bandwidth 190Mb/s



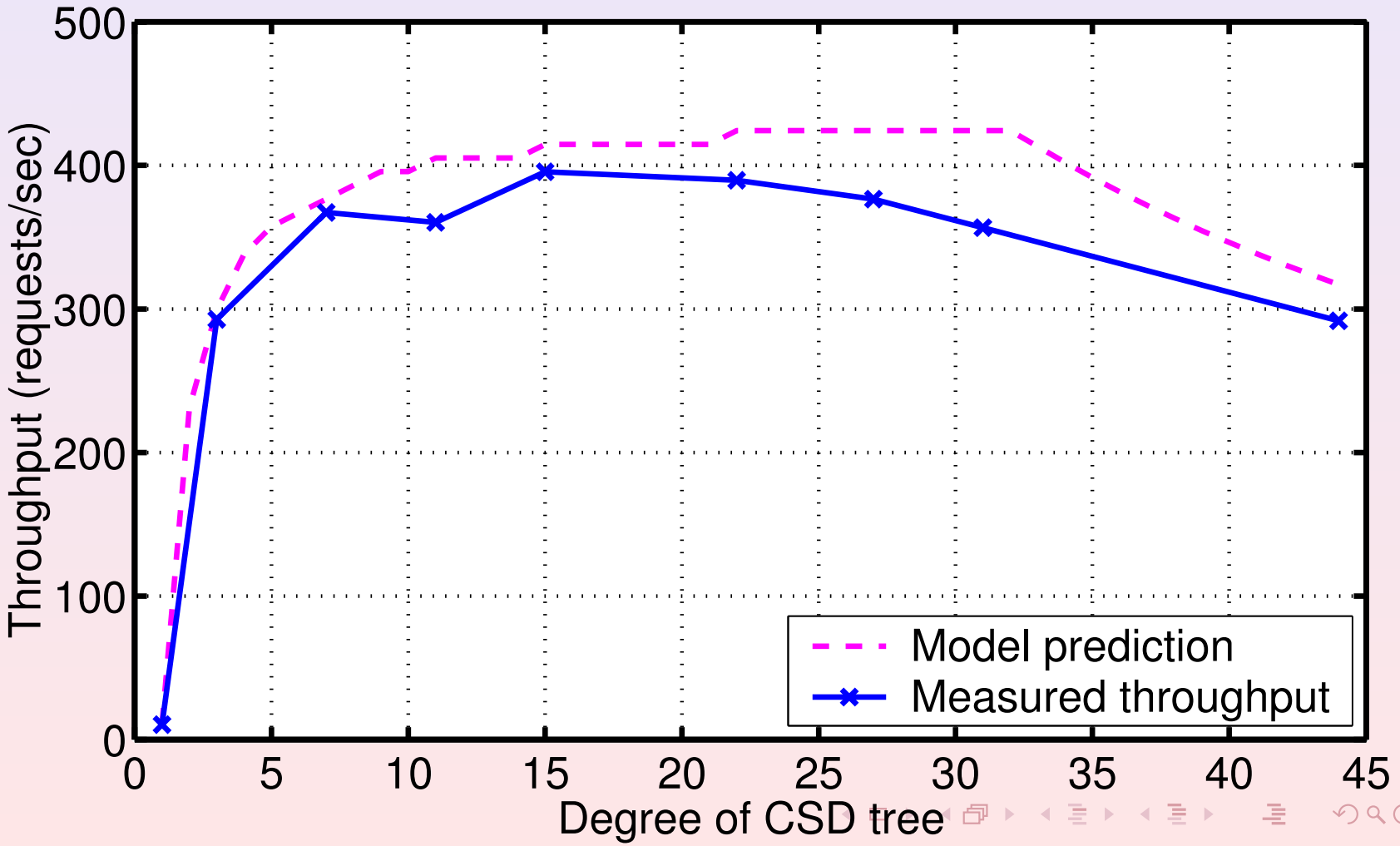
Deployment selection validation - DGEMM 200, 25 Nodes



Deployment selection validation - DGEMM 200, 45 Nodes



Deployment selection validation - DGEMM 310, 45 Nodes



Summary Table

DGEMM Size	Nodes	Optimal	Selected	Model	Star	Tri-ary
10	21	1	1	100.0%	22.4%	50.5%
100	25	12	12	100.0%	84.4%	84.6%
200	45	3	8	86.1%	40.0%	100.0%
310	45	15	22	98.5%	73.8%	74.0%
1000	21	20	20	100.0%	100.0%	65.3%

Outline

Conclusion

- Determines how many nodes should be used and in what hierarchical organization
- Proved an optimal deployment as a CSD tree
- Algorithm to construct optimal tree
- Deployment prediction is easy,fast and scalable
- Experiments validated the model

Future work

- Test our approach with experiments on large clusters using a variety of problem sizes
- Develop re-deployment approaches
  - Dynamically adapt the deployment to workload levels
- Final goal is to develop deployment planning and re-deployment algorithms for middleware on heterogeneous clusters and Grids