Automatic Middleware Deployment Planning on Clusters

Eddy Caron, Pushpinder Kaur Chouhan, Holly Dail

27 May 2005
GRAAL Group Meeting

Outline

1. Introduction
2. Modeling for hierarchical systems
3. Deployment
4. Experimental results
5. Discussion

Outline

1 Introduction

2 Modeling for hierarchical systems

3 Deployment

4 Experimental results

5 Discussion

Whats Deployment

A deployment is the distribution 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.

Whats Deployment

A deployment is the distribution 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?

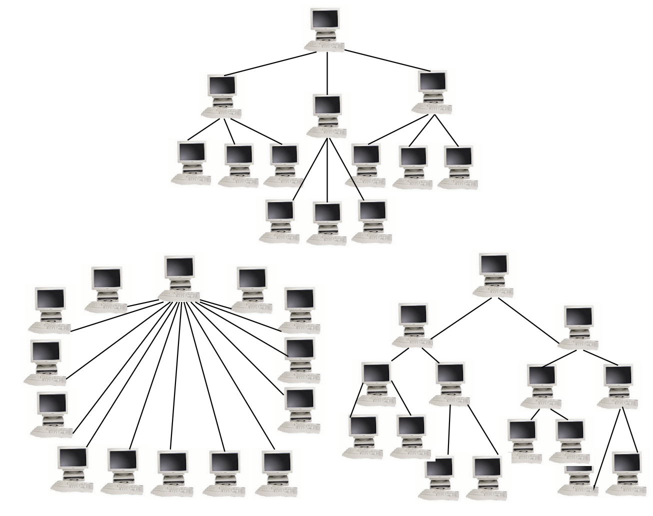
Problem Statement

- 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?

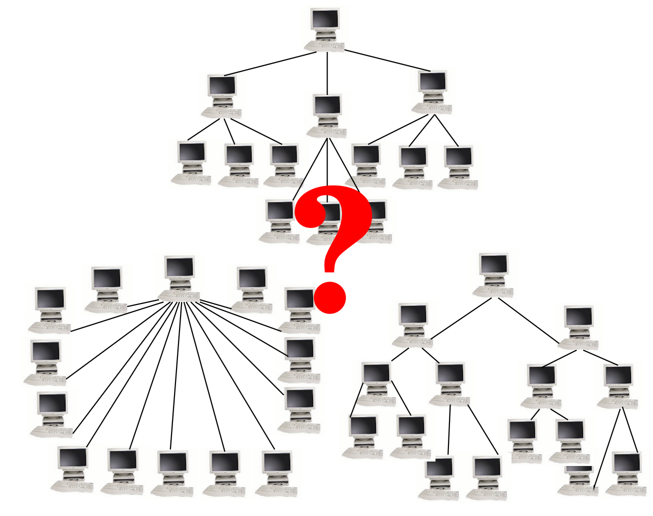
Problem Statemant - Diagramatic

Problem Statemant - Diagramatic

Problem Statemant - Diagramatic



Problem Statemant - Diagramatic



Outline

1 Introduction

2 Modeling for hierarchical systems

3 Deployment

4 Experimental results

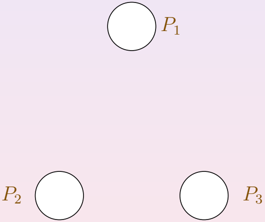
5 Discussion

Notation

- $G = (V, E, w, c)$
- $P_i \in V$: computing resource ($\mathbb{A}$ U $\mathbb{S}$)
- 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

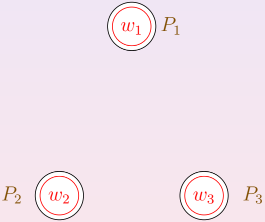
Notation

- $G = (V, E, w, c)$
- $P_i \in V$: computing resource ($\mathbb{A}$ U $\mathbb{S}$)
- 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



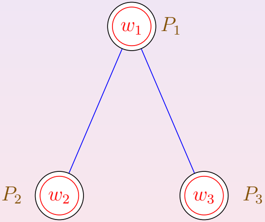
Notation

- $G = (V, E, w, c)$
- $P_i \in V$: computing resource ($\mathbb{A}$ U $\mathbb{S}$)
- 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



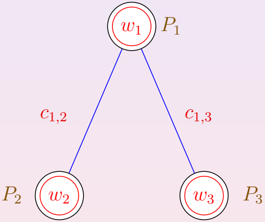
Notation

- $G = (V, E, w, c)$
- $P_i \in V$: computing resource ($\mathbb{A}$ U $\mathbb{S}$)
- 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



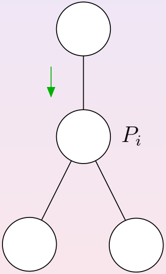
Notation

- $G = (V, E, w, c)$
- $P_i \in V$: computing resource ($\mathbb{A}$ U $\mathbb{S}$)
- 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



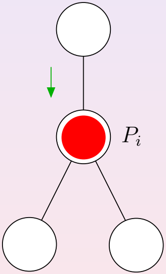
Notation

- $S_i^{(in)}$: 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
- $W_i^{(X_{ser})}$: computation amount needed by server P_i to process a generic problem



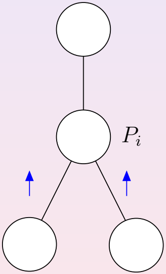
Notation

- $S_i^{(in)}$: 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
- $W_i^{(X_{ser})}$: computation amount needed by server P_i to process a generic problem



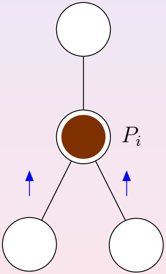
Notation

- $S_i^{(in)}$: 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
- $W_i^{(X_{ser})}$: computation amount needed by server P_i to process a generic problem



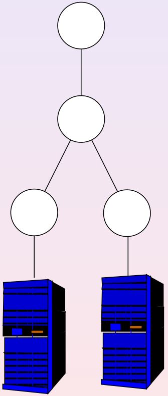
Notation

- $S_i^{(in)}$: 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
- $W_i^{(X_{ser})}$: 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
- $W_i^{(X_{ser})}$: computation amount needed by server P_i to process a generic problem



Outline

1. Introduction
2. Modeling for hierarchical systems
3. Deployment
   - Steady State Approach
   - Deployment construction example
   - Model implementation
4. Experimental results
5. Discussion

Deployment constraints

- Server computation:

$$\forall P_i \in \mathbb{S} : Server_{comp} \leq \frac{w_i}{W_i^{X_{ser}}}$$

Deployment constraints

- Agents computation:

$$\forall P_i \in \mathbb{A} : Node_{comp_i} \leq \frac{w_i}{W_i^{in} + W_i^{out}}$$

Deployment constraints

- Agents Communication:

$$\forall P_i \rightarrow P_j : Comm_{req} \leq \frac{C_{i,j}}{S_i^{in} + S_j^{out}}$$

Deployment constraints

- Agents Communication:

$$\forall P_i \rightarrow P_j : Comm_{req} = min(\frac{C_{i,j}}{S_i^{in}}, \frac{C_{j,i}}{S_j^{out}})$$

Deployment construction

- Number of servers supported by an agent:MSPA

$$= \frac{Node_{comp_i}}{min(Server_{comp}, Comm_{req})}$$

Deployment construction

- Number of agents supported by another agent: $MAPA_l$

$$= \frac{Node_{comp_l}}{min(Node_{comp_{l+1}}, subtree_{comp_l}, Comm_{req})}, P_l, P_{l+1} \in \mathbb{A}$$

where,

$$subtree_{comp_l} = Server_{comp} \times MSPA, if l == h$$

else

$$min(subtree_{comp_{l+1}}, Node_{comp_{l+1}}) \times MAPA_{l+1}$$

- **n_req** $= \sum_{k=0}^{l}(MAPA_k \times MAPA_{k-1}) + MAPA_l \times MSPA$

where,

$$MAPA_k = 1, if k \leq 0$$

Deployment construction

- Number of agents supported by another agent: $MAPA_l$

$$= \frac{Node_{comp_l}}{min(Node_{comp_{l+1}}, subtree_{comp_l}, Comm_{req})}, P_l, P_{l+1} \in \mathbb{A}$$

where,

$$subtree_{comp_l} = Server_{comp} \times MSPA, if l == h$$

else

$$min(subtree_{comp_{l+1}}, Node_{comp_{l+1}}) \times MAPA_{l+1}$$

- **n_req** $= \sum_{k=0}^{l}(MAPA_k \times MAPA_{k-1}) + MAPA_l \times MSPA$

where,

$$MAPA_k = 1, if k \leq 0$$

Deployment construction Algorithm

```
calculate MSPA
l=0, n_req=0
while n_req < n do
    calculate MAPA_l
    n_req_low = n_req
    calculate n_req
    l++
end while
l − − , n_req_high = n_req
if (n − n_req_low) > (n_req_high − n) then
    call Make_Hierarchy
    call Node Removal Algorithm
else
    l − −
    call Make_Hierarchy
    call Node Addition Algorithm
end if
```

Procedure:
Make_Hierarchy

```
while l > 0 do
    add MAPA_l agents to
    lowest level agent
    l − −
end while
add MSPA servers to lowest
level agent
```

Example to explain algorithm

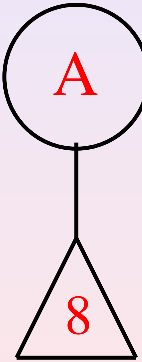
MAPA_1 = 2

MAPA_2 = 3

MSPA=8

1) N=15
2) N=25

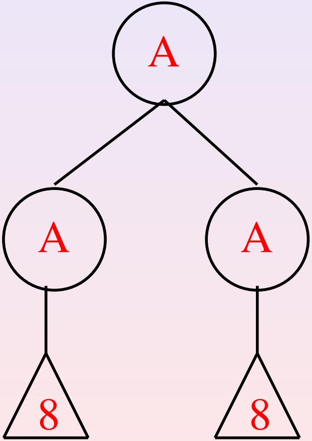
Example to explain algorithm



MAPA_1 = 2
MAPA_2 = 3
MSPA=8

N_req=9

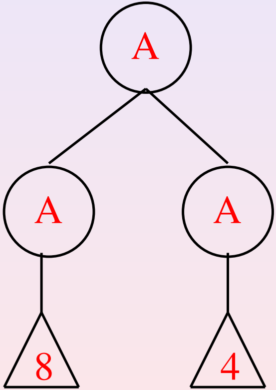
Example to explain algorithm



MAPA_1 = 2
MAPA_2 = 3
MSPA=8

N_req=19

Example to explain algorithm



MAPA_1 = 2
MAPA_2 = 3
MSPA=8

N=15

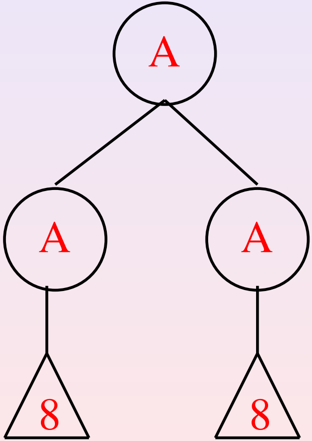
Remove 4 servers

Example to explain algorithm

MAPA_1 = 2
MAPA_2 = 3
MSPA=8

1) N=15
2) N=25

Example to explain algorithm



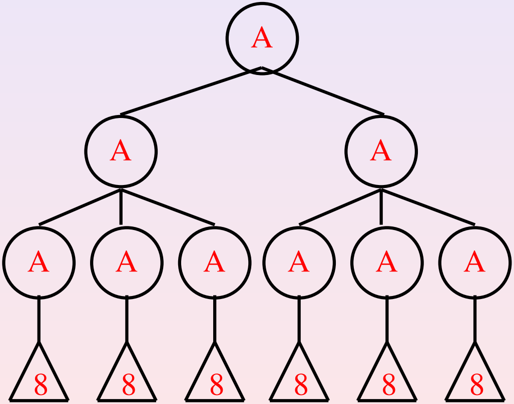
MAPA_1 = 2

MAPA_2 = 3

MSPA=8

N_req=19

Example to explain algorithm



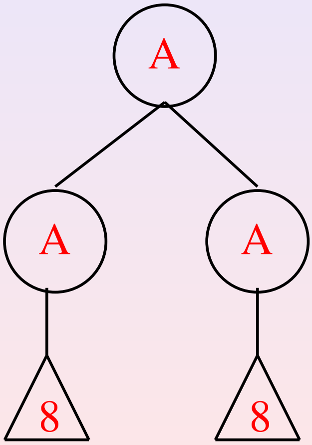
MAPA_1 = 2

MAPA_2 = 3

MSPA=8

N_req=57

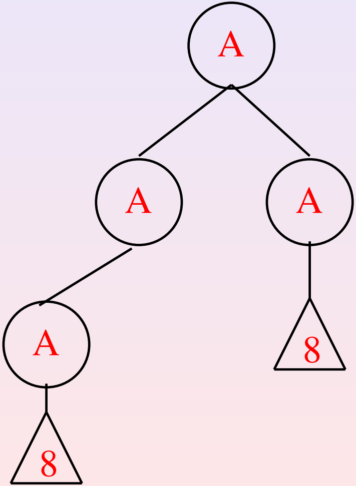
Example to explain algorithm



MAPA_1 = 2
MAPA_2 = 3
MSPA=8

N_req=19

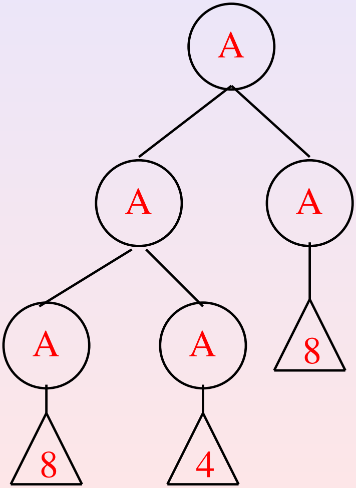
Example to explain algorithm



MAPA_1 = 2
MAPA_2 = 3
MSPA=8

N_req=20

Example to explain algorithm



MAPA_1 = 2
MAPA_2 = 3
MSPA=8

N=25

Model with DIET

- The MA and LA are considered as having the same performance
- An agent can connect either agents or servers but not both
- Root of the tree is always an MA
- All clients will submit their request to the DIET hierarchy through one MA
- Large data items are already in-place on the server

Model with DIET

- The MA and LA are considered as having the same performance
- An agent can connect either agents or servers but not both
- Root of the tree is always an MA
- All clients will submit their request to the DIET hierarchy through one MA
- Large data items are already in-place on the server

Model with DIET

- The MA and LA are considered as having the same performance
- An agent can connect either agents or servers but not both
- Root of the tree is always an MA
- All clients will submit their request to the DIET hierarchy through one MA
- Large data items are already in-place on the server

Model with DIET

- The MA and LA are considered as having the same performance
- An agent can connect either agents or servers but not both
- Root of the tree is always an MA
- All clients will submit their request to the DIET hierarchy through one MA
- Large data items are already in-place on the server

Model with DIET

- The MA and LA are considered as having the same performance
- An agent can connect either agents or servers but not both
- Root of the tree is always an MA
- All clients will submit their request to the DIET hierarchy through one MA
- Large data items are already in-place on the server

Outline

1 Introduction

2 Modeling for hierarchical systems

3 Deployment

4 Experimental results
- Performance model validation
- Deployment selection validation

5 Discussion

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 clients (each client launch one request, wait for the response, and then sleep 0.05 seconds)
- Resources: 55-nodes,246 processors dual AMD Opteron @ 2GHz, each with cache size of 1024KB, and 2GB of main memory and a 1Gb/s
- Model parameterization: we measure the performance for a benchmark task (X_{ser})

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 clients (each client launch one request, wait for the response, and then sleep 0.05 seconds)
- Resources: 55-nodes,246 processors dual AMD Opteron @ 2GHz, each with cache size of 1024KB, and 2GB of main memory and a 1Gb/s
- Model parameterization: we measure the performance for a benchmark task (X_{ser})

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 clients (each client launch one request, wait for the response, and then sleep 0.05 seconds)
- Resources: 55-nodes,246 processors dual AMD Opteron @ 2GHz, each with cache size of 1024KB, and 2GB of main memory and a 1Gb/s
- Model parameterization: we measure the performance for a benchmark task (X_{ser})

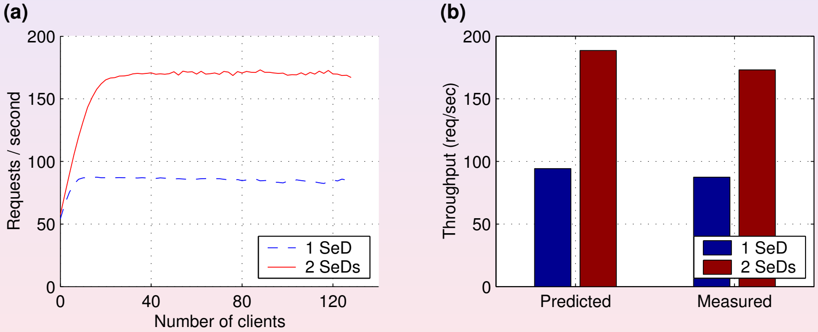
Experimental design

- Software: GoDIET is used to deploy DIET.
- Job types: DGEMM, a simple matrix multiplication (BLAS package).
- Workload: steady-state load with 1 - 200 clients (each client launch one request, wait for the response, and then sleep 0.05 seconds)
- Resources: 55-nodes,246 processors dual AMD Opteron @ 2GHz, each with cache size of 1024KB, and 2GB of main memory and a 1Gb/s
- Model parameterization: we measure the performance for a benchmark task (X_{ser})

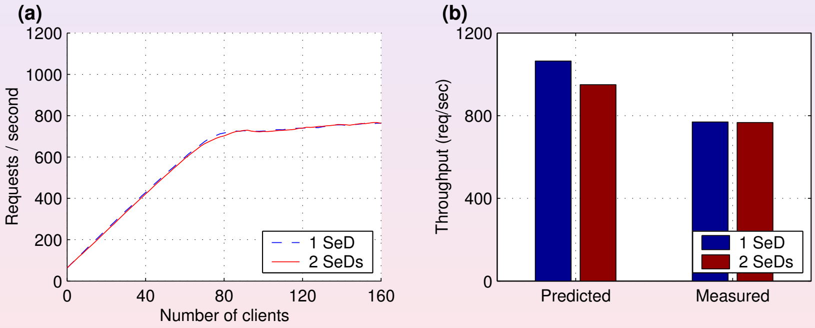
Experimental design

- Software: GoDIET is used to deploy DIET.
- Job types: DGEMM, a simple matrix multiplication (BLAS package).
- Workload: steady-state load with 1 - 200 clients (each client launch one request, wait for the response, and then sleep 0.05 seconds)
- Resources: 55-nodes,246 processors dual AMD Opteron @ 2GHz, each with cache size of 1024KB, and 2GB of main memory and a 1Gb/s
- Model parameterization: we measure the performance for a benchmark task (X_{ser})

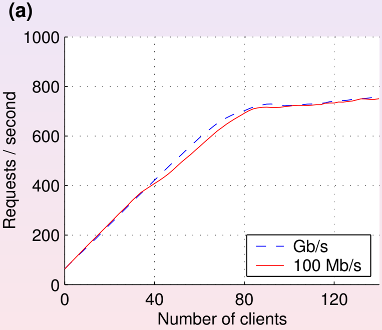
DGEMM 150x150

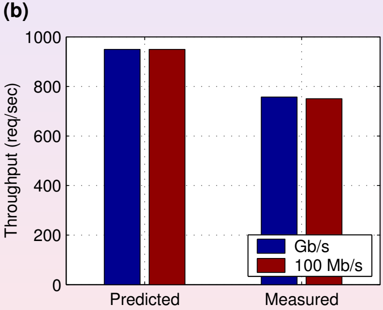


DGEMM 10x10

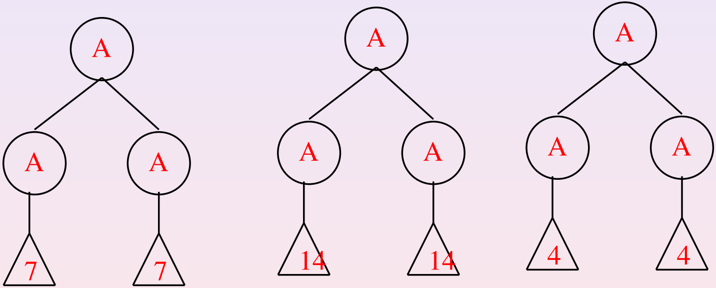

(a)
Requests / second
0
200
400
600
800
1000
1200
0
40
80
120
160
Number of clients
1 SeD
2 SeDs
(b)
Throughput (req/sec)
0
400
800
1200
Predicted
Measured
1 SeD
2 SeDs

Gb/s versus 100 Mb/s network





Types of platforms compared - 150x150

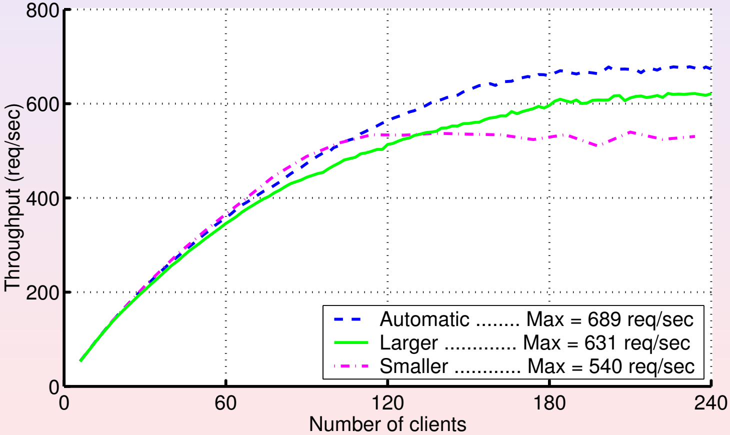


Automatically–defined by model

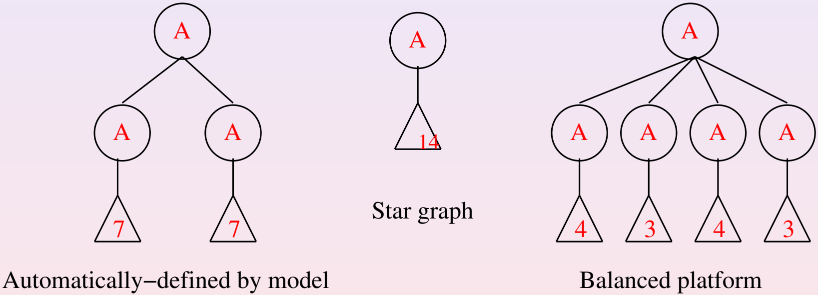
Larger

Smaller

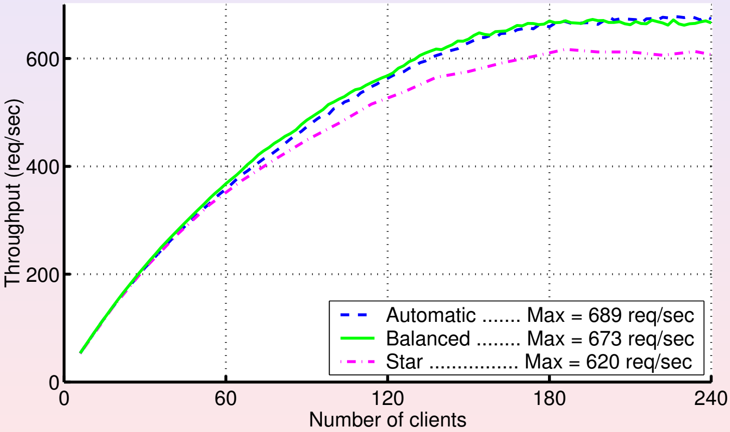
Comparison of automatically-generated hierarchy according to number of servers



Types of platforms compared - 150x150



Comparison of automatically-generated hierarchy



Outline

1 Introduction

2 Modeling for hierarchical systems

3 Deployment

4 Experimental results

5 Discussion
- Conclusion
- Future Work

Summary

- Determines how many nodes should be used and in what hierarchical organization
- Provides an optimal real-valued solution before rounding and without resource constraints
- Provide algorithms to modify the obtained hierarchy to limit the number of resources used to the number available
- Experiments validated the hierarchy and throughput performance model

Future work

- Test our approach with experiments on large clusters using bigger (and 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