Automatic Middleware Deployment Planning on Clusters

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

6 Oct 2005
GRAAL Group Meeting
Outline

1. Introduction
2. Deployment platform
3. Optimal deployment planning
4. DIET deployment planning
5. Experimental results
6. Discussion
Outline

1 Introduction
2 Deployment platform
3 Optimal deployment planning
4 DIET deployment planning
5 Experimental results
6 Discussion
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

1. Introduction
2. Deployment platform
3. Optimal deployment planning
4. DIET deployment planning
5. Experimental results
6. Discussion
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.
- $\rho$ is the throughput of the platform calculated as the completed requests per second.
Platform deployment architecture
Platform deployment architecture

1) Scheduling request
Platform deployment architecture

2) Forward request down
Platform deployment architecture

3) Request prediction & response generation
Platform deployment architecture

4) Response sorted & forwarded up
5) Scheduling response
(reference of selected server)
Platform deployment architecture

5) Scheduling response
   (reference of selected server)

1) Scheduling request

2) Forward request down

3) Request prediction
   & response generation

4) Response sorted
   & forwarded up

Scheduling Phase

Pushpinder Kaur Chouhan, Holly Dail, Eddy Caron, and Frédéric Vivien
Automatic Middleware Deployment Planning on Clusters
Platform deployment architecture
Platform deployment architecture

6) Service request
Platform deployment architecture

7) Run application & generate response

Pushpinder Kaur Chouhan, Holly Dail, Eddy Caron, and Frédéric Vivien
Automatic Middleware Deployment Planning on Clusters
Platform deployment architecture

8) Service response

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

Automatic Middleware Deployment Planning on Clusters
Platform deployment architecture

Service Phase

6) Service request

8) Service response

Run application & generate response
Outline

1. Introduction
2. Deployment platform
3. Optimal deployment planning
4. DIET deployment planning
5. Experimental results
6. Discussion
The completed request throughput $\rho$ of a deployment is given by the minimum of the scheduling request throughput $\rho_{\text{sched}}$ and the service request throughput $\rho_{\text{service}}$.

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

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

The scheduling throughput $\rho_{\text{sched}}$ is limited by the throughput of the agent with the highest degree.

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

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

Service requests are only sent to a single server.
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 $dMax$.

\[dMax=4\]

\[dMax=6\]
Optimal deployment - Theorems

The optimal throughput $\rho$ of any deployment with maximum degree $d_{\text{Max}}$ is obtained with a CSD tree.

- By Lemma 1, $\rho = \min(\rho_{\text{sched}}, \rho_{\text{service}})$
- By Lemma 2, $\rho_{\text{sched}}$ is limited by an agent with maximum degree
- By Lemma 3, $\rho_{\text{service}}$ increases with $|S|$
Optimal deployment - Theorems

**Theorem**

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

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

n=12  \quad d=4

n_{used}= 1
Optimal tree construction

\[ n=12 \quad d=4 \]
\[ n_{\text{used}}= 5 \]
Optimal tree construction

LA

MA

n=12   d=4
n_used= 5
Optimal tree construction

n=12      d=4
n_used=  9
Optimal tree construction

\[ n = 12 \quad d = 4 \]

\[ n_{\text{used}} = 9 \]
Optimal tree construction

n=12  d=4
n_used= 12
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.
\( S_{\text{req}} \) is the size in Mb of the message forwarded down the agent hierarchy for a scheduling request.

\( W_{\text{req}} \) is the amount of computation in MFlop needed by an agent to process one incoming request.

\( W_{\text{pre}} \) 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 = $W_{req} + W_{rep} \cdot (d \cdot w)$
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 = \( \frac{W_{req} + W_{rep}(d)}{B} \)
Deployment constraint for an Agent

- **Agent communication:**
  - \( \text{agent\_receive\_time} = \frac{S_{\text{req}} + d \cdot S_{\text{rep}}}{B} \)
  - \( \text{agent\_send\_time} = \frac{d \cdot S_{\text{req}} + S_{\text{rep}}}{B} \)

- **Agent computation:**
  - \( \text{agent\_comp\_time} = \frac{W_{\text{req}} + W_{\text{rep}}(d)}{\text{w}} \)
Deployment constraint for an Agent

- **Agent communication:**
  - \( \text{agent\_receive\_time} = \frac{S_{req} + d \cdot S_{rep}}{B} \)
  - \( \text{agent\_send\_time} = \frac{d \cdot S_{req} + S_{rep}}{B} \)

- **Agent computation:**
  - \( \text{agent\_comp\_time} = \frac{W_{req} + W_{rep}(d)}{w} \)
Deployment constraint for an Agent

- **Agent communication:**
  - \( \text{agent\_receive\_time} = \frac{S_{req} + d \cdot S_{rep}}{B} \)
  - \( \text{agent\_send\_time} = \frac{d \cdot S_{req} + S_{rep}}{B} \)

- **Agent computation:**
  - \( \text{agent\_comp\_time} = \frac{W_{req} + W_{rep}(d)}{w} \)
Deployment constraint for a Server

- **Server communication:**
  - \( \text{server\_receive\_time} = \frac{S_{\text{req}}}{B} \)
  - \( \text{server\_send\_time} = \frac{S_{\text{rep}}}{B} \)

- **Server computation:**
  - All server comp. time = \( W_{\text{pre}} |S| + W_{\text{app}} |S| \)
  - \( \text{server\ comp\ time} = \frac{W_{\text{pre}} + W_{\text{app}}}{w} \)
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 = \( W_{pre} + \sum W_{app} \)
- server_comp_time = \( W_{pre} + \frac{S_{rep}}{B} \)
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 = $W_{pre} \cdot |S| + W_{app}^w$
  - server\_comp\_time = $W_{pre} + \frac{W_{app}^w}{|S|}$
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 = $W_{pre} \cdot |S| + W_{app}$
- server_comp_time = $W_{pre} + \frac{W_{app}}{|S|}$
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 = \( \frac{W_{pre} \cdot |S| + W_{app}}{w} \)
  - server\_comp\_time = \( \frac{W_{pre} + \frac{W_{app}}{|S|}}{w} \)

---

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>......</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td>p1</td>
<td>p1</td>
<td>p1</td>
<td>......</td>
</tr>
<tr>
<td>p2</td>
<td>p2</td>
<td>p2</td>
<td>p2</td>
<td>......</td>
</tr>
<tr>
<td>p3</td>
<td>p3</td>
<td>p3</td>
<td>p3</td>
<td>......</td>
</tr>
<tr>
<td>p4</td>
<td>p4</td>
<td>p4</td>
<td>p4</td>
<td>......</td>
</tr>
<tr>
<td>p5</td>
<td>p5</td>
<td>p5</td>
<td>p5</td>
<td>......</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Wapp1</th>
<th>Wapp2</th>
<th>Wapp3</th>
<th>......</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

*Pushpinder Kaur Chouhan, Holly Dail, Eddy Caron, and Frédéric Vivien*  
*Automatic Middleware Deployment Planning on Clusters*
Send or receive or compute, single port

- **Scheduling throughput** $\rho_{\text{sched}}$:
  \[
  \min \left( \frac{W_{\text{pre}}}{w} + \frac{S_{\text{req}}}{B} + \frac{S_{\text{rep}}}{B}, \frac{S_{\text{req}} + d \cdot S_{\text{rep}}}{B} + \frac{d \cdot S_{\text{req}} + S_{\text{rep}}}{B} + \frac{W_{\text{req}} + W_{\text{rep}}(d)}{w} \right)
  \]

- **Service throughput** $\rho_{\text{service}}$:
  \[
  \frac{1}{S_{\text{req}} + S_{\text{rep}} + \frac{W_{\text{pre}} + W_{\text{app}}}{|S|}}
  \]
Scheduling throughput $\rho_{sched}$:

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

Service throughput $\rho_{service}$:

$$\frac{1}{\frac{S_{req}}{B} + \frac{S_{rep}}{B} + \frac{W_{pre} + W_{app}}{w}}$$
Scheduling throughput $\rho_{\text{sched}}$:

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

Service throughput $\rho_{\text{service}}$:

$$
\frac{1}{\max \left( \frac{S_{\text{req}}}{B}, \frac{S_{\text{rep}}}{B}, \frac{W_{\text{pre}} + W_{\text{app}}}{w} \right)}
$$
Scheduling throughput $\rho_{\text{sched}}$:

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

Service throughput $\rho_{\text{service}}$:

$$\frac{1}{\max \left( \frac{S_{\text{req}}}{B}, \frac{S_{\text{rep}}}{B}, \frac{W_{\text{pre}} + \frac{W_{\text{app}}}{\lvert S \rvert}}{w} \right)}$$
## Model Parametrization

<table>
<thead>
<tr>
<th>Components</th>
<th>Agent</th>
<th>SeD</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_{\text{req}}$ (Mflop)</td>
<td>1.4×$10^6$</td>
<td></td>
</tr>
<tr>
<td>$W_{\text{rep}}$ (Mflop)</td>
<td>$5.0 \times 10^7 + 5.3 \times 10^7 \cdot d$</td>
<td></td>
</tr>
<tr>
<td>$W_{\text{pre}}$ (Mflop)</td>
<td></td>
<td>7.1 ×$10^6$</td>
</tr>
<tr>
<td>$S_{\text{res}}$ (Mbit)</td>
<td>6.4×$10^{-5}$</td>
<td>6.4×$10^{-5}$</td>
</tr>
<tr>
<td>$S_{\text{req}}$ (Mbit)</td>
<td>5.3×$10^{-5}$</td>
<td>5.3×$10^{-5}$</td>
</tr>
</tbody>
</table>
Outline

1. Introduction
2. Deployment platform
3. Optimal deployment planning
4. DIET deployment planning
5. Experimental results
6. 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 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 (req/sec) vs. Agent degree for different models:
- Measured
- Model w/ 20 Mb/s
- Model w/ 190 Mb/s
- Model w/ 909 Mb/s

Pushpinder Kaur Chouhan, Holly Dail, Eddy Caron, and Frédéric Vivien
Automatic Middleware Deployment Planning on Clusters
Throughput validation - Parallel Model (DGEMM 10)

![Graph showing throughput validation for different agent degrees and bandwidth models.](image_url)
Throughput validation - DGEMM 1000, bandwidth 190Mb/s
Deployment selection validation - DGEMM 200, 25 Nodes

Throughput (requests/sec) vs. Degree of CSD tree

- Model prediction
- Measured throughput

Pushpinder Kaur Chouhan, Holly Dail, Eddy Caron, and Frédéric Vivien
Automatic Middleware Deployment Planning on Clusters
Deployment selection validation - DGEMM 200, 45 Nodes

Throughput (requests/sec) vs. Degree of CSD tree

- Model prediction
- Measured throughput
Deployment selection validation - DGEMM 310, 45 Nodes

Throughput (requests/sec) vs. Degree of CSD tree

- Model prediction
- Measured throughput
### Summary Table

<table>
<thead>
<tr>
<th>DGEMM Size</th>
<th>Nodes</th>
<th>Optimal</th>
<th>Selected</th>
<th>Model</th>
<th>Star</th>
<th>Tri-ary</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>21</td>
<td>1</td>
<td>1</td>
<td>100.0%</td>
<td>22.4%</td>
<td>50.5%</td>
</tr>
<tr>
<td>100</td>
<td>25</td>
<td>12</td>
<td>12</td>
<td>100.0%</td>
<td>84.4%</td>
<td>84.6%</td>
</tr>
<tr>
<td>200</td>
<td>45</td>
<td>3</td>
<td>8</td>
<td>86.1%</td>
<td>40.0%</td>
<td>100.0%</td>
</tr>
<tr>
<td>310</td>
<td>45</td>
<td>15</td>
<td>22</td>
<td>98.5%</td>
<td>73.8%</td>
<td>74.0%</td>
</tr>
<tr>
<td>1000</td>
<td>21</td>
<td>20</td>
<td>20</td>
<td>100.0%</td>
<td>100.0%</td>
<td>65.3%</td>
</tr>
</tbody>
</table>
Outline

1. Introduction
2. Deployment platform
3. Optimal deployment planning
4. DIET deployment planning
5. Experimental results
6. Discussion
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