HAL
open science

Automatic Middleware Deployment Planning on Clusters

Eddy Caron, Pushpinder Kaur Chouhan, Holly Dail

To cite this version:

Eddy Caron, Pushpinder Kaur Chouhan, Holly Dail. Automatic Middleware Deployment Planning on Clusters. [Research Report] LIP RR-2005-26, Laboratoire de l'informatique du parallélisme. 2005, 2+13p. hal-02102561

HAL Id: hal-02102561

https://hal-lara.archives-ouvertes.fr/hal-02102561

Submitted on 17 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

Automatic Middleware Deployment Planning on Clusters

Eddy Caron,
Pushpinder Kaur Chouhan,
Holly Dail

May 2005

Research Report N° 2005-26

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

Automatic Middleware Deployment Planning on Clusters

Eddy Caron, Pushpinder Kaur Chouhan, Holly Dail

May 2005

Abstract

The use of many distributed, heterogeneous resources as a large collective resource offers great potential and has become an increasingly popular idea. A key issue for these Grid platforms is middleware scalability and how middleware services can best be mapped to the resource platform structure. Optimizing deployment is a difficult problem with no existing, general solutions. In this paper we address a simpler sub-problem: how to carry out an adapted deployment on a cluster with hundreds of nodes? Efficient use of clusters alone or as part of the Grid is an important issue.

We present a deployment model that predicts the maximum throughput of each element of a deployment. Our deployment construction algorithm uses this model to automatically create a mapping of middleware elements onto resources with the goal of maximizing throughput. We apply our approach to automatically deploy a distributed Problem Solving Environment (PSE) on a homogeneous cluster environment. We present experiments comparing the automatically-generated deployment against a number of other reasonable deployments.

Keywords: Deployment, Cluster, Middleware, Modeling

Résumé

L'utilisation de plusieurs ressources hétérogènes distribuées comme une seule et même ressource offre un grand potentiel et est une idée de plus en plus répandue. L'approche la plus célèbre répondant à cette idée sont les plates-formes de type Grille. Un des problèmes pour la gestion de ces grilles reste l'extensibilité des intergiciels sous-jacent mais également comment associer les services rendus par l'intergiciel avec la structure matérielle de la plate-forme. L'optimisation du déploiement est un problème difficile pour lequel il n'existe pas de solution générique. Dans ce papier nous nous intéressons à un sous-problème: comment trouver un déploiement performant sur une grappe de plusieurs centaines de nœuds ? On vise une utilisation efficace de l'ensemble de la grappe seule ou en tant qu'élément d'une grille.

À ces fins, nous proposons un modèle pour le déploiement qui prédit le débit maximum de chaque élément déployé. Notre algorithme de construction de déploiement utilise ce modèle pour créer automatiquement la projection des éléments de l'intergiciel sur les ressources avec pour objectif de maximiser le débit total. Nous appliquons notre approche pour automatiser le déploiement d'un environnement de résolution de problème (PSE) sur un environnement de type grappe. Enfin, nous présentons les expériences qui comparent le déploiement généré automatiquement avec quelques déploiements qu'un utilisateur pourrait envisager.

Mots-clés: Déploiement, Grappes, Intergiciel, Modélisation

1 Introduction

Due to the scale of Grid platforms, as well as the geographical localization of resources, middleware approaches should be distributed to provide scalability and adaptability. Much work has focused on the design and implementation of distributed middleware. To benefit most from such approaches, an appropriate mapping of middleware components to the distributed resource environment is needed. However, while middleware designers often note that this problem of deployment planning is an important problem, few solutions for efficient and automatic deployment exist. In other words, questions such as "which resources should be used?", "how many resources should be used" and "should the fastest and best-connected resource be used for middleware or as a computational resource?" remain difficult to answer.

Before deploying on the scale of the Grid, the first problem encountered by users is "how to manage an adapted deployment on a cluster with tens to hundreds of nodes?" While the homogeneous properties of such a platform tend to simplify many aspects of the problem, this article will show that the task is not as simple as it appears. Moreover, given the popularity of resource reservation mechanisms (Condor, PBS, OAR, etc) on clusters, the task can be dynamic even within a cluster.

The main goal of this paper is to provide an automated approach to deployment planning that provides a good deployment on homogeneous cluster environments. We focus on hierarchically distributed middleware approaches and we consider that a "good" deployment is one that maximizes throughput. We present a model for the throughput of each element of the hierarchy. This model is used by our deployment planning algorithm that defines an optimal hierarchy whereby the number of children at each level is given by a real-valued solution. We then apply an integer approximation to obtain a realisable hierarchy. We also present several algorithms that allow us to obtain a deployment under limited resource conditions.

We use a distributed Problem Solving Environment (PSE) as a test case for our approach. This PSE, the Distributed Interactive Engineering Toolbox (DIET) [13], uses a hierarchical arrangement of agents and servers to provide a scalable, high-throughput scheduling service. In [13], the authors showed that if a simple star deployment is used for DIET the centralized agent can become a bottleneck. We apply our approach to find an appropriate hierarchical deployment of this middleware.

The rest of the article is organized as follows. Section 2 presents related work on the subject of deployment and deployment planning. In Section 3 the architectural model is presented and a deployment performance model is developed based on steady-state operation. Section 4 describes our algorithm for deployment construction and provides two algorithms that can be used to utilize the exact number of nodes available in the cluster. Section 5 gives an overview of DIET and describes how we applied the given model to the deployment of DIET. Section 6 presents some experiments that validate this work. Finally, Section 7 concludes the paper and describes future work.

2 Related Work

A deployment is the distribution of a common platform and middleware across many resources. Deployment can be broadly divided in two categories: system deployment and software deployment. 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. Modern software systems are increasing the complexity of these tasks as more sophisticated architectural models, such as systems of systems and coordinated distributed systems, become commonplace. Many tools have been developed for software deployment; examples include ManageSoft [1], FootPrints Software Deployment [2], Software Dock [17], SmartFrog [3], and Ant [16].

ManageSoft [1] is a dynamic software deployment and management solution for desktops, servers, and mobile devices. The ManageSoft client-centric architecture enables rapid and reliable software distribution across any network including the Internet. With automatic installation and self-healing, unlimited scalability, and web-based status reporting, ManageSoft provides ongoing savings long after initial deployment. It's the way to deploy, update, and manage software on Windows, Linux, UNIX, and Macintosh devices.

FootPrints Software Deployment [2] distributes software to every computer in an organization. It outfits and configures workstations, laptops and servers to organizations' software standards and policies.

Utilizing New Boundary's Prism Deploy [4] technology, set up computers the way they should be to ensure maximum control and security of the network environment quickly and easily. The Prism Deploy helps to keep them configured that way automatically no matter what accidental or unauthorized changes end users may make.

The Software Dock [18] is a system of loosely coupled, cooperating, distributed components. The Software Dock support software producers by providing the release dock that acts as a repository of software system releases. The Software Dock employs agents that travel from release docks to field docs in order to perform specific software deployment tasks while docked at a field dock.

SmartFrog [3] is a technology for describing distributed software systems as collections of cooperating components, and then activating and managing them. It was developed at HP Labs in Bristol, in the UK. The core SmartFrog framework is released under LGPL.

Distributed Ant [16] extends the Ant [26] build file environment to provide a flexible procedural deployment description and implements a set of deployment services. It fills a gap in end user support in Grid middleware and provides a dedicated secure mechanism for decentralized application deployment.

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. As deploying systems can be a time consuming and cumbersome task, tools such as Deployment Toolkit [6] and Kadeploy [22] have been developed to facilitate this process.

DELL Deployment Toolkit [6] provides a suite of enabling technologies for PowerEdge servers designed to help with pre-operating system configuration, remote server deployment or re-provisioning. The Deployment Toolkit provides the flexibility and control to leverage organization's remote deployment policies and practices.

Kadeploy [22] is a deploying tool to allow the deployment of multiple computing environment on every node, without compromising the original system/boot sector. The environment to be deployed can either already exist and be registered in the database or already exist but not be registered or neither exist nor be registered. The deployment system is composed of several tools or commands whose aim is the execution of all the needed actions to carry out deployments and other administrative tasks.

Although these toolkits can automate many of the tasks associated with deployment, they do not automate the decision process of finding an appropriate mapping of task to resource so that the best performance can be achieved from the system.

In [11] we presented a decision builder tool to analyze existing hierarchical deployments, use mathematical models to identify the bottleneck node, and remove the bottleneck by adding resources in the appropriate area of the system. The solution presented was iterative and heuristic in nature; the current work provides a direct analytical method that provides an optimal real-valued solution with an integer approximation.

In [21], software components based on the CORBA component model are automatically deployed on the computational Grid. The CORBA component model contains a deployment model that specifies how a particular component can be installed, configure and launched on a machine. The authors note a strong need for deployment planning algorithms, but to date they have focused on other aspects of the system. Our work is thus perfectly complementary; a collaboration has already been planned to test our algorithms with their deployment approach.

Optimizing deployment is an evolving field. In [19], the authors propose an algorithm called Sikitei to address the Component Placement Problem (CPP). This work leverages existing AI planning techniques and the specific characteristics of CPP. In [20] the Sikitei approach is extended to allow optimization of resource consumption and consideration of plan costs. The Sikitei approach focuses on satisfying component constraints for effective placement, but does not consider detailed but sometimes important performance issues such as the effect of the number of connections on a component's performance.

The Pegasus System [10] frames workflow planning for the Grid as a planning problem. The approach is interesting for overall planning when one can consider the individual elements as composable with no performance consideration. Our work is more narrowly focused on a specific style of assembly and interaction between components and has a correspondingly more accurate view of performance to guide the deployment decision process.

The middleware deployment model that we present in this article is different from above mentioned tools and systems. Our model deploy middleware on PCs like software deployment and specify the connectivity between the nodes like logical system deployment. Our model specify the way to construct a platform from homogeneous distributed systems and also tells which type of hierarchy will be good for specific type of computation intensive problems.

3 Steady-state modeling for hierarchical systems

We consider the steady-state scheduling techniques [9, 8] for our model because we are interested in maximizing steady-state throughput. In steady-state techniques, the performance of the startup and shutdown phases are not considered. The initial integer formulation is replaced by a continuous or rational formulation. The precise ordering and allocation of tasks and messages are not required, at least in the first step. The main idea is to characterize the activity of each resource during each time-unit.

3.1 Architectural Model

The target architectural framework is represented by a weighted graph $G = (V, E, w, C)$. Each $P_i \in V$ represents a computing resource with computing power w_i, meaning that node P_i execute w_i MFlop/second (so bigger the w_i, the faster the computing resource P_i and $w_i = 0$ is not possible since it would permit node P_i to have no computing power). There are one or more client nodes P_c that generate requests for the system. Each link $P_i \rightarrow P_j$ is labelled by the bandwidth value $C_{i,j}$ in Mb/s.

Nodes can be divided in two types, agents and servers. Agents are the nodes that coordinate incoming requests and communicate the requests to the appropriate servers; examples could include scheduling computation requests in a remote procedure call system or providing load balancing between servers in a web server farm. Servers provide services to clients; examples could include providing computational services in a remote procedure call system or responding to web requests as part of a web server farm. If responses are generated by the servers the agents may coordinate the response.

The size of the request generated by the client is S_i^{in} and the size of the reply request created by each node is S_i^{out}. The measuring unit of these quantities is MB/request. The amount of computation needed by $P_i \in \mathbb{A}$ ($\mathbb{A}$ is a set of agents) to process one incoming request is denoted by W_i^{in} and the amount of computation needed by $P_i \in \mathbb{A}$ to merge the reply requests of its children is denoted by W_i^{out}. In short W_i^{out} is the time needed for sorting the servers. $W_i^{X_{Ser}}$ is the amount of computation needed by $P_i \in \mathbb{S}$ ($\mathbb{S}$ is a set of servers) for X_{ser} service.

3.2 Steady State Operation

Our objective is to generate a best hierarchy from the available number of nodes so as to get maximum throughput. Throughput is the maximum number of requests that can be completed in a time step. The main focus for constructing the hierarchy is the maximum throughput of each node, where this throughput may depend on the number of children a node is supporting in the hierarchy. The overall throughput of the system depends on the bandwidth of the links, message size of requests, time spent computing by each node on each request, and the computing power of the nodes. Therefore, we have the following constraints:

Computation resource for agents: α_i^{in} denotes the number of incoming requests (requests from clients) processed by P_i during one time-unit (we take one time-unit to be one second). Note that α_i^{in} is not necessarily an integer, it may be a rational. In a similar way, α_i^{out} is the number of outgoing requests (e.g. selection of the best server based on the reply packets) computed during one time-unit by the node P_i. In steady-state, there is a corresponding reply for each incoming request and α_i^{in} must equal α_i^{out}. Thus, we can consider only α_i, the overall throughput of the sub-tree rooted at P_i. The following equation states that the sub-tree throughput can not be larger than the throughput of the agent.

$$\forall P_i \in \mathbb{A} : \alpha_i \leq \frac{w_i}{W_i^{in} + W_i^{out}} \tag{1}$$

So we re-organize Equation (1) to represent the number of requests computed by an agent as $Node_{comp_i}$ in Equation (2).

$$Node_{comp_i} \leq \frac{w_i}{W_i^{in} + W_i^{out}}, \forall P_i \in \mathbb{A} \tag{2}$$

Communication resources: Depending on the physical network, network bandwidth on a single link is either shared by incoming and outgoing data or bandwidth is fully duplex. Depending on the type of bandwidth utilization either of the two equations represented as Equation (3) will be used to calculated the number of requests transmitted per time-unit along each link $P_i \leftrightarrow P_j$.

$$\forall P_i \rightarrow P_j : Comm_{req} \leq \frac{C_{i,j}}{S_i^{in} + S_j^{out}} \;\; || \;\; Comm_{req} = min(\frac{C_{i,j}}{S_i^{in}}, \frac{C_{i,j}}{S_j^{out}}) \tag{3}$$

Server computation constraints: The number of requests that can be calculated by a $P_i \in \mathbb{S}$ in a time step is given by Equation (4).

$$Server_{comp} \leq \frac{w_i}{W_i^{X_{ser}}}, \forall P_i \in \mathbb{S} \tag{4}$$

4 Deployment construction

The maximum nodes that can be connected to another node without making it bottleneck can be calculated from the above constraints.

Number of servers supported by an agent: MSPA (Maximum Servers Per Agent) represents the maximum number of servers that can be supported by a node $P_i \in \mathbb{A}$.

$$\textbf{MSPA} = \frac{Node_{comp_i}}{min(Server_{comp}, Comm_{req})} \tag{5}$$

Number of agents supported by another agent: The maximum number of agents of that can be added to another agent is given as **MAPA** (Maximum Agents Per Agent). The MAPA value for agents at a particular level in the hierarchy l depends on number of requests processed per second by each agent at level $l+1$. The calculation of MAPA must therefore be done for each level of the hierarchy; no more levels can be added when MAPA < 3 since there can be no benefit to adding levels.

$$MAPA_l = \frac{Node_{comp_l}}{min(Node_{comp_{l+1}}, Comm_{req})}, P_l, P_{l+1} \in \mathbb{A} \tag{6}$$

As mentioned earlier the throughput of the platform can be a rational, so the values of MAPA and MSPA are not necessarily integers, they could be rational too. To make these values integers we use either round up or round down. The simple method that we consider is according to the available number of nodes. If we have enough available nodes we round up, otherwise we round down.

We consider l to be the number of levels in the hierarchy between the top-level agent and the servers. For example, $l = 0$ is the minimal hierarchy of an agent with servers. The total number of nodes required for the best hierarchy, **n_req**, can be calculated as follows.

$$\textbf{n_req} = \sum_{k=0}^{l} MAPA^k + MAPA_l \times MSPA \tag{7}$$

The hierarchy is constructed using Algorithm 1. This algorithm uses Algorithm 2 and Algorithm 3 to use no more than the number of available nodes.

5 Model implementation

To organize the nodes in an efficient manner and use the nodes' power efficiently is out of the scope of end users. That is why end-users have to rely on specialized middleware, like Problem Solving Environments (PSE), for their applications on such platforms. Such middleware, like NetSolve [7], Ninf [24] or DIET [13], already exist and are commonly called Network Enabled Server (NES) environments [23]. We illustrate our model by applying the results to an existing hierarchical PSE called DIET.

5.1 DIET overview

The Distributive Interactive Engineering Toolbox is developed at the Ecole Normale Supérieure de Lyon and is available at http://graal.ens-lyon.fr/DIET. Figure 1 gives a general view of DIET. DIET is built around five main components. The **Client** is an application that uses DIET to solve problems in a RPCS mode. The hierarchy of scheduling agents is composed of a single **Master Agent** and zero or more **Local Agents** (LA). The MA is the entry point of the DIET environment and thus receives all computation requests from clients. The MA forwards service requests onto servers, collects their computation abilities from their responses, selects the best server, and forwards the server reference back to the client according to some scheduling heuristics (dead-line scheduling, shortest completion time first, minimization of the requests throughput, ...). The LAs transmit requests and information between the MA and the servers. The hierarchy of LAs can provide scalability and adaptation to diverse network environments. For additional flexibility, peer-to-peer technology is used in DIET to provide

calculate MSPA using Equation (5)
l=0, n_req=0
while n_req < n **do**
  calculate $MAPA_l$ using Equation (6)
  $n_req_{low} = n_req$
  calculate n_req using Equation (7)
  l++
$l--$, $n_req_{high} = n_req$
if $(n - n_req_{low}) > (n_req_{high} - n)$ **then**
  call Make_Hierarchy
  call Node Removal Algorithm 2 with $extra_nodes = n_req_{high} - n$
else
  $l--$
  call Make_Hierarchy
  call Node Addition Algorithm 3 with $extra_nodes = n - n_req_{low}$

Procedure: Make_Hierarchy

while l > 0 **do**
  add $MAPA_l$ agents to lowest level agent
  $l--$
add MSPA servers to lowest level agent

Algorithm 1: Construction Algorithm

clients with access to multiple hierarchies. In this paper we consider the deployment of a single DIET hierarchy. The information stored on an LA is the list of requests and the number of servers that can solve a given problem and information about the data distributed in this subtree. Depending on the underlying network topology, a hierarchy of LAs may be deployed between an MA and the bottom LAs. The scheduling and the gathering of information is distributed in the tree. **Server Daemons (SeD)** provide a variety of computational services. These services are declared by the SeD to its parent LA. For instance it can be located on the entry point of a parallel computer.

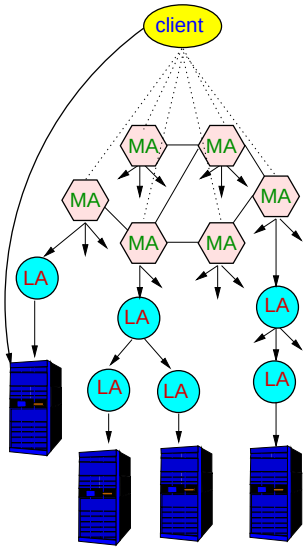

Figure 1: General view of DIET.

The information stored on an SeD is a list of the data available on its server (with their distribution and the way to access them), the list of problems that can be solved on it, and all information concerning its load (memory available, number of resources available, ...). A SeD can give performance prediction for a given problem using the performance evaluation module FAST [25, 15]. Finally, computation in interactive mode are executed by a federation of **Computational Resource Daemons** located on the

```
while (extra_nodes > 0) do
  if extra_nodes < MSPA then
    remove extra_nodes from the one of the bottom agents. Exit
  else if (extra_nodes == MSPA || extra_nodes == MSPA + 1 ) then
    remove one bottom agent with all its servers. Exit
  else
    remove_agent = extra_nodes mod MSPA
    if remove_agent > (MAPA_l × MSPA) + MAPA_l then
      remove MAPA_l agents with all its servers.
      add MSPA /* servers to the agent that now have neither agent nor server connected to it*/
      left_nodes = extra_nodes − ((MAPA_l × MSPA) + MAPA_l) + MSPA
      extra_nodes = left_nodes
    else
      remove remove_agent agents with all its servers
      left_nodes = extra_nodes − ((remove_agent × MSPA) + remove_agent)
      extra_nodes = left_nodes
```

Algorithm 2: Node Removal Algorithm

```
1:  while (extra_nodes > 1) do
2:    if all bottom-level agents have maximum number of servers then
3:      add one node to the bottom agent
4:      add parent servers as his servers
5:      extra_nodes − −
6:    else
7:      if extra_nodes ≤ MSPA then
8:        add one agent to the agent having child agents < MAPA
9:        add extra_nodes − 1 servers to this newly added agent. Exit
10:     else
11:       add_agent = extra_nodes mod MSPA
12:       if extra_nodes ≥ add_agent + add_agent × MSPA then
13:         while (add_agent > 0) do
14:           add one agent to the agent having child agents < MAPA
15:           add MSPA servers to this new added agent
16:           add_agent − −
17:         extra_nodes = extra_nodes − add_agent + add_agent × MSPA
18:       else
19:         add_agent − −
20:         go to line 13
```

Algorithm 3: Node Addition Algorithm

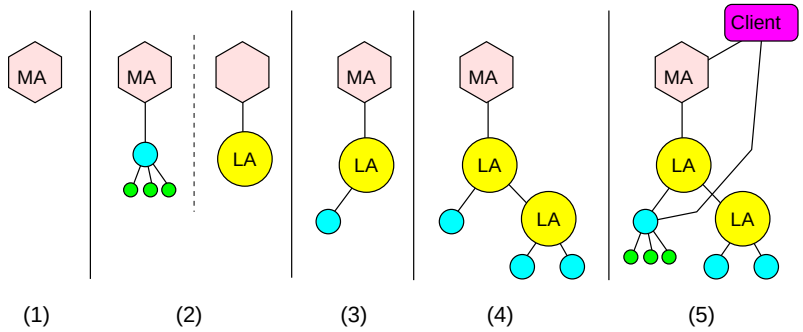

Figure 2: Deployment ordering for a DIET system.

different processors of a parallel server. A batch approach can also be used and the computation requests are sent to the batch system directly by the Server Daemon. Figure 2 shows each step of initialization in the DIET system. The architecture is built in hierarchical order with each component connecting to its parent.

5.2 Model with DIET

We make some assumptions to use DIET for model validation. The MA and LA are considered as having the same performance because we observed only negligible differences in their performance. We assume that an agent can connect either agents or servers but not both and that the root of the tree is always an MA. Thus all clients will submit their request to the DIET hierarchy through one MA.

When client requests are sent to the agent hierarchy, DIET is optimized such that large data items like matrices are not included in the problem parameter descriptions (only their sizes are included). These large data items are included only in the final request for computation from client to server. Since we are primarily interested in modeling throughput for the agent hierarchy as an example of a hierarchical middleware system, we assume that large data items are already in-place on the server.

6 Experimental results

In this section we present experiments designed to test the ability of our deployment model to correctly identify good real-world deployments. In Section 6.1 the experimental design and the corresponding parameterization of our deployment model are presented. Section 6.2 presents experiments testing the accuracy of our deployment performance model; the accuracy of this model is key to providing confidence in the deployment algorithms. These algorithms are tested in Section 6.3 with experiments comparing the performance of the best deployment identified by our deployment algorithms against the performance of other intuitive deployments.

In this section we present experiments designed to test the accuracy of our performance models for hierarchical deployments (Section 6.2) and the ability of our deployment optimization algorithms to identify deployments that are appropriate to a given resource environment and workload (Section 6.3). The next section describes the experimental design.

6.1 Experimental design

Software: DIET is used for all deployed agents and servers. The deployment arrangements are defined according to experimental goals and will be described in the following sections. Once a deployment has been chosen, GoDIET [12] is used to perform the actual software deployment. The desired hierarchical arrangement of agents and servers is defined in an XML file; GoDIET reads this file, generates and distributes all needed DIET configuration files, launches first agents and then servers in an appropriate hierarchical order, and manages cleanup of all processes after the experiment is finished. DIET is written in C++ and CORBA and GoDIET is written in Java. We used development versions of both packages for these experiments; however, stable versions of both software packages are publicly available [5].

Job types: Since our performance model and deployment approach focus on maximizing steady-state throughput, our experiments focus on testing the maximum sustained throughput provided by different

deployments. As an initial study we consider DGEMM, a simple matrix multiplication provided as part of the Basic Linear Algebra Subprograms (BLAS) package [14]. All servers in our test deployments provide the DGEMM service and all clients request this service. Furthermore, for each specific throughput test we use a single problem size.

Workload: Measuring the maximum throughput of a system is non-trivial: if too little load is introduced the maximum performance may not be achieved, if too much load is introduced the performance may suffer as well. Thus we use small quantities of steady-state load in the form of a client script that uses a continual loop to launch one request, wait for the response, and then sleep 0.05 seconds. With one client script there is at most a single request in the system. We then introduce load gradually by launching one client script every five seconds; four scripts are launched on each of 35 client machines. A full throughput test thus takes 700 seconds.

Resources: For these experiments we used a 55-node cluster at the Ecole Normale Supérieure in Lyon, France. Each node provided dual AMD Opteron 246 processors at 2 GHz, a cache size of 1024 KB, and 2 GB of memory. All nodes are connected by both a Gb ethernet and a 100 Mb/s ethernet; unless otherwise noted all communicates were sent over the Gigbit network. As measured with the Network Weather Service [27], available bandwidth on this network is 909.5 Mb/s and latency is 0.08 msec.

Model parameterization: To collect the values needed to calculate MSPA and MAPA, we measure the performance for a benchmark task (X_{ser}) using a small hierarchy of one agent and one server. We assume that the maximum throughput of an agent or a server can be calculated as the inverse of the time required by that element to treat a single request. Thus we measured the time required to treat a request at the server level and at the agent level. At the agent-level the processing time depends on the number of children attached to the agent so we collected benchmarks for a variety of sizes of star-based hierarchies. We then used a linear fit of the results to generate a simple model for agent-level throughput as a function of number of children. To measure data transfer within the hierarchy, we used `tcpdump` to monitor all data transferred between elements and `ethereal` to analyze the data. With these benchmarks we obtained predictions of all the variables that were not known such as $W_i^{X_{ser}}, W_i^{in} and W_i^{out}$.

The benefit of this estimation approach is that measurements of the time required to treat a request are fairly easy to collect and each benchmark can be run in a few minutes with only a single client script. However, this approach assumes that there is no overhead to running requests in parallel and that all resources can be perfectly shared. Estimates generated in this fashion will therefore tend to overestimate the throughput of servers and agents.

6.2 Performance model validation

The usefulness of our deployment approach depends heavily on the ability to predict the maximum steady-state throughput of each element in a deployment. This section presents experiments designed to answer this question.

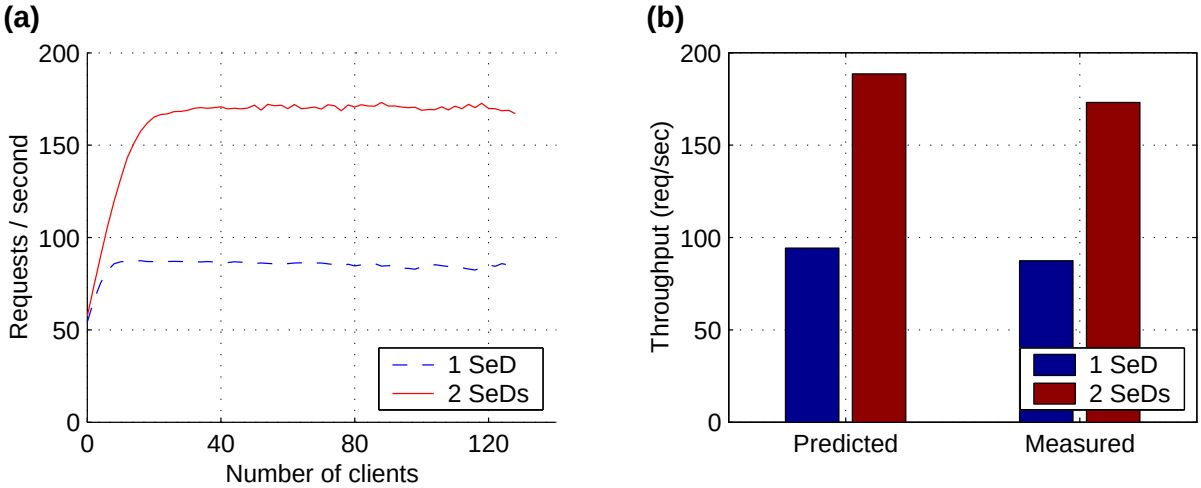

Figure 3: Star hierarchies with one or two servers for DGEMM 150x150 requests. (a) Real-world platform throughput for different load levels. (b) Comparison of predicted and actual maximum throughput.

The first test, shown in Figure 3, uses a workload of DGEMM150x150 to compare the performance of two hierarchies: an agent with one server versus an agent with two servers. For this scenario, the model predicts that both hierarchies are limited by server performance and that, therefore, performance will roughly double with the addition of the second server. These predictions are accurate: the model correctly predicts the absolute performance of these deployments and also predicts that the two-server deployment will be the better choice.

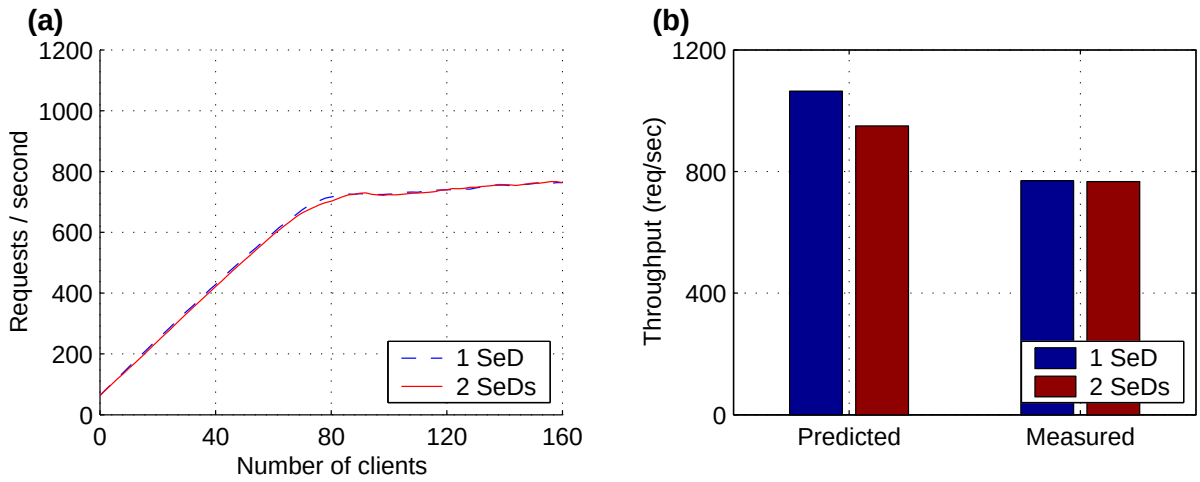

Figure 4: Star hierarchies with one or two servers for DGEMM 10x10 requests. (a) Throughput at different load levels. (b) Comparison of predicted and actual maximum throughput.

Figure 4 uses a workload of DGEMM10x10 to compare the performance of the same one and two server hierarchies. The model correctly predicts that both deployments are limited by agent performance and that the addition of the second server will in fact hurt performance. The error in the magnitude of the performance prediction is about 20-30%. Agent throughput is predicted based on easy-to-collect measurements of time spent by requests at the agent-level; We believe that the error in these predictions arises from the fact that some overhead is introduced by trying to run so many requests in parallel on the agent machine. Our model also over-estimates the negative effect of adding servers on agent performance. Thus, for small numbers of children-per-agent the model over-estimates agent performance and for large numbers of children the model under-estimates performance.

Figure 5 compares performance using a Gb/s network versus a 100 Mb/s network; two-server hierarchies are used in both cases with a workload of DGEMM10x10. The model correctly predicts that the network is not the limiting factor on steady-state performance. Given the message sizes transferred by DIET and the networks available to us for these experiments, we were not able to test a configuration where the network performance was the limiting factor. In future work we plan to test the network model more thoroughly.

In summary, our deployment performance model is able to accurately predict server throughput and can predict the impact of adding servers to a server-limited or agent-limited deployment. Some readers may wonder at this point if we use the results of this section to improve the model so that our deployment construction phase can be more effective. One of the goals of this work is that our approach could be applied to optimize the deployment of other hierarchical systems. The model parameterization described in Section 6.1 is straightforward and can be done rapidly for other systems; the tests described in this section are more difficult to design and run. Thus, we do not modify the model to provide an accurate example of the results one could obtain in applying this approach to other systems. The best deployment algorithm accuracy could be obtained by adjusting our model parameterization to account for the results of this section. However, adding this adjustment phase to model parameterization greatly complicates the parameterization phase and we prefer to maintain an approach that could be applied rapidly to other software and resource environments. Thus, for the following section we maintain the easy-to-obtain original model parameterization.

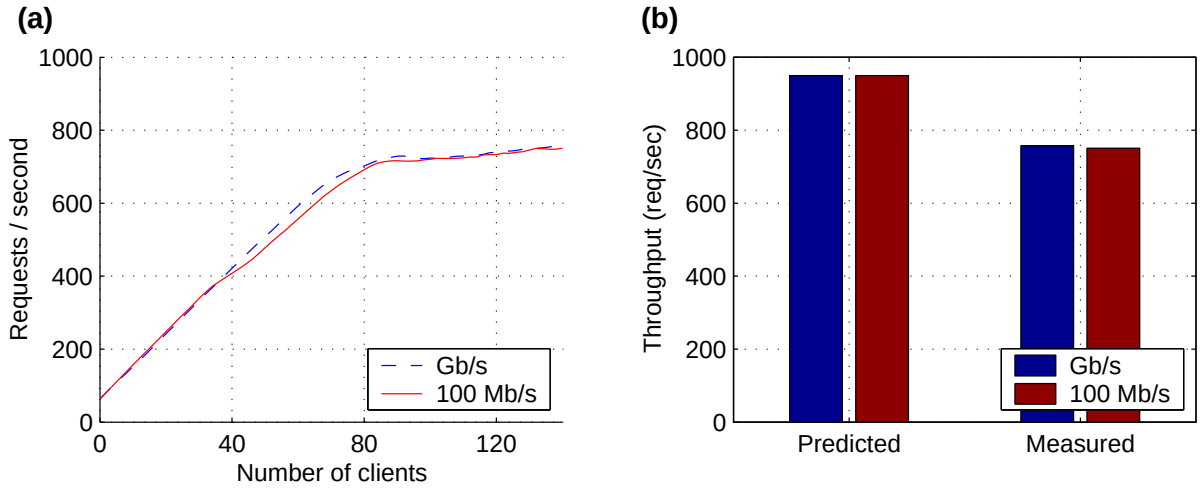

Figure 5: Star hierarchies with two servers using a Gb/s or a 100 Mb/s network. Workload was DGEMM 10x10. (a) Throughput at different load levels. (b) Comparison of predicted and actual maximum throughput.

6.3 Deployment selection validation

In this section we test the ability of the deployment algorithm to select an appropriate arrangement of agents and servers for deployment. We were not able to find alternative deployment algorithms that could be applied to our scenario as a comparison; in our experience, given the lack of automatic tools users typically define deployments by hand using intuition about the scenario. Thus we compare the model-defined deployment against several alternatives that, in our experience, would be reasonable intuitive choices with users.

We wish to find the best deployment on our 50-machine cluster for a workload of DGEMM150x150. Our deployment algorithm predicts that the best deployment uses a top-level agent and two middle-level agents where each middle-level agent can support 7 servers. This deployment thus contains 14 servers and 17 machines in total. To check whether the algorithm selected the correct number of servers, Figure 6 compares this `Automatic` deployment against deployments with the same two-level hierarchy of agents but with different numbers of servers. The `Larger` deployment contains twice as many servers (14 on each middle-level agent so 28 in total) while the `Smaller` deployment contains roughly half as many servers (4 on each middle-level agent so 8 in total). The `Automatic` deployment provides a significantly higher maximum throughput than the others.

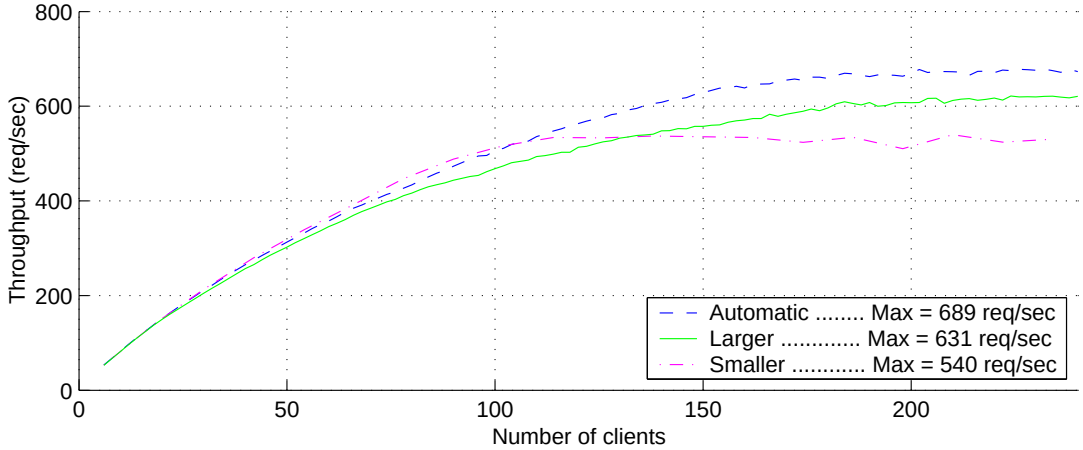

Figure 6: Comparison of automatically-generated hierarchy with hierarchies containing twice as many and half as many servers.

Although it is important that the deployment approach can select an appropriate number of resources, it is also important that it select an appropriate hierarchy. We therefore design two alternative deployments that we have found to be typical hierarchical styles for users. To remove the effect of the number of servers, we use exactly 14 servers, the number chosen by our approach, for these deployments. The `Star` deployment uses 14 servers attached directly to the top-level agent. The `Balanced` tries to balance work amongst the agents by using the same branching factor at all levels of the hierarchy; for 14 servers the most balanced hierarchy uses a top-level agent with 4 middle-level agents and 3 or 4 servers attached to each mid-level agent. Figure 7 shows these three platforms.

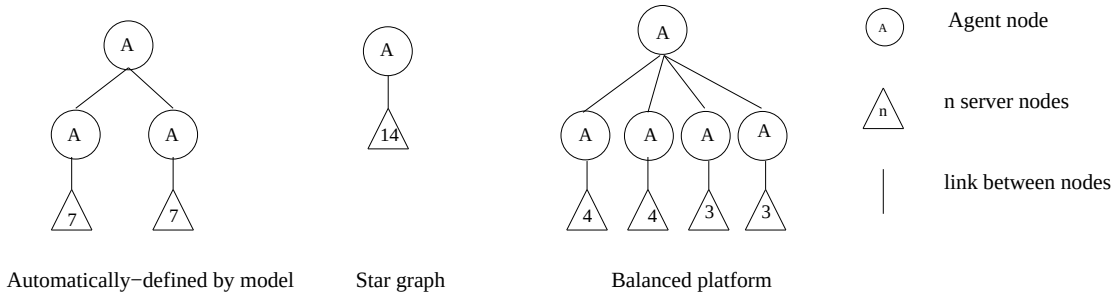

Figure 7: Types of platforms compared

Figure 8 compares the performance achieved by these three deployments. The `Automatic` approach performs the best and provides a significant advantage over the `Star` topology. However, the `Balanced` approach performs almost as well as the `Automatic` approach. This result is not surprising: the two hierarchies are in fact fairly similar in structure.

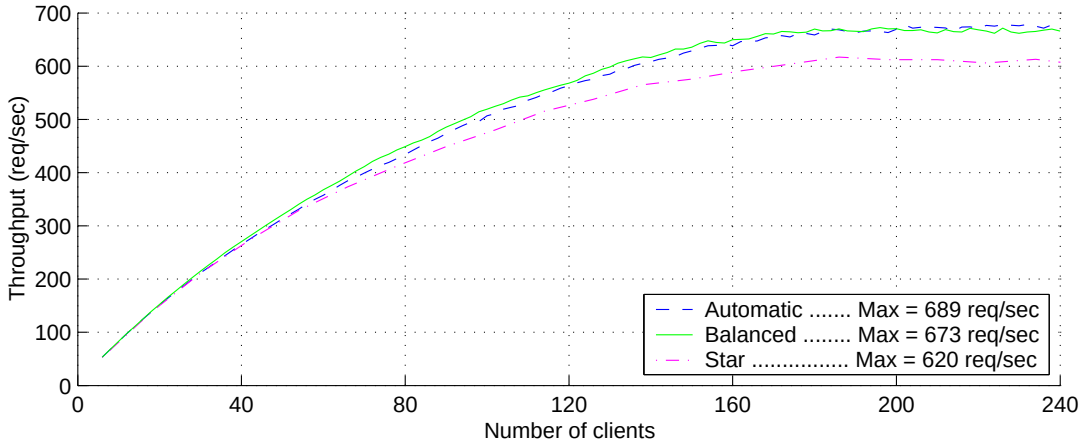

Figure 8: Comparison of automatically-generated hierarchy with intuitive alternative hierarchies.

7 Conclusion and Future Work

This paper has presented an approach for determining an appropriate hierarchical middleware deployment for a homogeneous resource platform of a given size. The approach determines how many nodes should be used and in what hierarchical organization; the goal is to maximize steady-state throughput. The model provides an optimal real-valued solution without resource constraints; we then apply round-up or round-down to obtain integer factors for the hierarchy definition. We also provide algorithms to modify the obtained hierarchy to limit the number of resources used to the number available. We instantiate the model for the hierarchical scheduling system used in the DIET Network Enabled Server environment. Our experiments validated the throughput performance model used for servers and agents and demonstrated that the automatic deployments performed well as compared to other intuitive deployments.

This article provides only the first step for automatic middleware deployment planning. We plan to test our approach with experiments on large clusters using bigger (and a variety of) problem sizes. While our current approach depends on a predicted workload, it will be interesting to develop re-deployment approaches that can dynamically adapt the deployment to workload levels after the initial deployment. We also plan to extend the model to consider heterogeneous computation abilities for agents and test our approach on heterogeneous clusters. Our final goal is to develop deployment planning and re-deployment algorithms for middleware on heterogeneous clusters and Grids.

Acknowledgement

This work has been supported by INRIA, CNRS, ENS Lyon, UCBL, Grid5000 from the French Department of Research, and the INRIA associated team I-Arthur. The authors would like to thank Frédéric Desprez for his insightful ideas and guidance, Stephane D'Alu for technical support on the Grid'5000 platform, and Ashish Meena (LIFL, France) for helpful and stimulating discussions.

References

[1] http://www.managesoft.com/solution/distribution/index.xml.

[2] http://www.unipress.com/footprints/deploy.html.

[3] http://www.hpl.hp.com/research/smartfrog/.

[4] http://www.newboundary.com/products/prismdeploy/.

[5] http://graal.ens-lyon.fr/DIET/.

[6] Simplifyng system deployment using the Dell OpenManage Deployment Toolkit, October 2004. Dell Power Solutions.

[7] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Sagi, Z. Shi, and S. Vadhiyar. Users' Guide to NetSolve V1.4. Computer Science Dept. Technical Report CS-01-467, University of Tennessee, Knoxville, TN, July 2001.

[8] O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Bandwidth-centric allocation of independent tasks on heterogeneous platforms. In *International Parallel and Distributed Processing Symposium IPDPS'2002*. IEEE Computer Society Press, 2002.

[9] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert. Steady-state scheduling on heterogeneous clusters: why and how? In *6th Workshop on Advances in Parallel and Distributed Computational Models APDCM 2004*. IEEE Computer Society Press, 2004.

[10] J. Blythe, E. Deelman, Y. Gil, C. Kesselman, A. Agarwal, G. Mehta, and K. Vahi. The role of planning in grid computing. In *The International Conference on Automated Planning & Scheduling (ICAPS)*, Trento, Italy, June 2003.

[11] E. Caron, P.K. Chouhan, and A. Legrand. Automatic Deployment for Hierarchical Network Enabled Server. In *The 13th Heterogeneous Computing Workshop (HCW 2004)*, page 109b (10 pages), Santa Fe. New Mexico, April 2004.

[12] E. Caron and H. Dail. GoDIET: a tool for managing distributed hierarchies of DIET agents and servers. Research report RR-2005-06, Laboratoire de l'Informatique du Parallélisme (LIP), February 2005. Also available as INRIA Research Report RR-5520.

[13] E. Caron, F. Desprez, F. Lombard, J. Nicod, M. Quinson, and F. Suter. A Scalable Approach to Network Enabled Servers. In B. Monien and R. Feldmann, editors, *Proceedings of the 8th International EuroPar Conference*, volume 2400 of *LNCS*, pages 907–910, Paderborn, Germany, August 2002. Springer-Verlag.

[14] A. Chtchelkanova, J. Gunnels, G. Morrow, J. Overfelt, and R. Van de Geijn. Parallel implementation of BLAS: General techniques for level 3 BLAS. Technical Report CS-TR-95-40, University of Texas, Austin, October 1995.

[15] Frédéric Desprez, Martin Quinson, and Frédéric Suter. Dynamic performance forecasting for network enabled servers in an heterogeneous environment. In *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2001)*. CSREA Press, June 25-28 2001.

[16] W. Goscinski and D. Abramson. Distributed Ant: A system to support application deployment in the Grid. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, pages 436–443, Pittsburgh, PA, USA, November 2004. IEEE Computer Society.

[17] R.S. Hall, D. Heimbigner, and A.L. Wolf. A cooperative approach to support software deployment using the Software Dock. In *Proceedings of the 1999 International Conference on Software Engineering (ICSE'99)*, pages 174–183, New York, May 1999. Association for Computing Machinery.

[18] Andr'e Van Der Hoek, Dennis Heimbigner, Er L. Wolf, and Richard S. Hall. The software dock: A distributed, agent-based software deployment system, October 05 1997.

[19] T. Kichkaylo, A. Ivan, and V. Karamcheti. Constrained component deployment in wide area networks using AI planning techniques. In *International Parallel and Distributed Processing Symposium IPDPS'2003*, Nice, France, 2003.

[20] T. Kichkaylo and V. Karamcheti. Optimal resource aware deployment planning for component based distributed applications. In *The 13th High Performance Distributed Computing (HPDC 2004)*, Honolulu, HI, June 2004.

[21] S. Lacour, C. Pérez, and T. Priol. Deploying CORBA components on a Computational Grid: General principles and early experiments using the Globus Toolkit. Technical Report PI-1611, IRISA, March 2004.

[22] C. Martin and O. Richard. Parallel launcher for cluster of PC. In G. R. Joubert, A. Murli, F. J. Peters, and M. Vanneschi, editors, *Parallel Computing, Advances and Current Issues. Proceedings of the International Conference ParCo2001*, pages 473–480, Naples, Italy, September 2001. Imperial College Press, London.

[23] S. Matsuoka, H. Nakada, M. Sato, and S. Sekiguchi. Design Issues of Network Enabled Server Systems for the Grid, 2000. Grid Forum, Advanced Programming Models Working Group whitepaper.

[24] H. Nakada, M. Sato, and S. Sekiguchi. Design and Implementations of Ninf: towards a Global Computing Infrastructure. *Future Generation Computing Systems, Metacomputing Issue*, 15(5-6):649–658, 1999.

[25] Martin Quinson. Dynamic performance forecasting for network-enabled servers in a metacomputing environment. In *International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS'02), in conjunction with IPDPS'02*, April 15-19 2002.

[26] Eric D. Taillard and Istituto Dalle Molle. Ant systems. Technical report, February 24 1999.

[27] R. Wolski, N.T. Spring, and J. Hayes. The Network Weather Service: A distributed resource performance forecasting service for metacomputing. *The Journal of Future Generation Computing Systems*, 15(5-6):757–768, 1999.