Automatic Middleware Deployment Planning on Heterogeneous Platforms

Eddy Caron, Pushpinder Kaur Chouhan, Frédéric Desprez

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Automatic Middleware Deployment Planning on Heterogeneous Platforms

Eddy Caron — Pushpinder Kaur Chouhan — Frédéric Desprez

N° ????

Juin 2008

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--????--FR+ENG

Automatic Middleware Deployment Planning on Heterogeneous Platforms

Eddy Caron, Pushpinder Kaur Chouhan, Frédéric Desprez

Thème NUM — Systèmes numériques
Projet GRAAL

Rapport de recherche n° ???? — Juin 2008 — 20 pages

Abstract: The use of many distributed, heterogeneous resources as a large collective platform offers great potential. A key issue for these grid platforms is middleware scalability and how middleware services can be mapped on the available resources. Optimizing deployment is a difficult problem with no existing general solutions. In this paper, we address the following problem: how to perform out an adapted deployment for a hierarchy of servers and resource brokers on a heterogeneous system?

Our objective is to generate a best platform from the available nodes so as to fulfill the clients demands. However, finding the best deployment among heterogeneous resources is a hard problem since it is close to find the best broadcast tree in a general graph, which is known to be NP-complete.

Thus, in this paper, we present a heuristic for middleware deployment on heterogeneous resources. We apply our heuristic to automatically deploy a distributed Problem Solving Environment on a large scale grid. We present experiments comparing the automatically generated deployment against a number of other reasonable deployments.

Key-words: Deployment, Grid computing, Network Enabled Server, Scheduling, Resource localization and selection.

This text is also available as a research report of the Laboratoire de l'Informatique du Parallélisme `http://www.ens-lyon.fr/LIP`.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Planification de déploiement automatique pour intergiciel sur plateformes hétérogènes

Résumé :

L'utilisation de nombreuses ressources hétérogènes et distribuées comme une plate-forme collaborative à large échelle offre un grand potentiel. Un point clef de ces plates-formes de grille est l'extensibilité des intergiciels et comment associer les services de l'intergiciel aux ressources disponibles. L'optimisation du déploiement est un problème difficile qui ne dispose pas de solution générique. Dans ce rapport, nous nous intéressons au problème suivant: comment réaliser un déploiement adapté pour une hiérarchie de serveurs et d'ordonnanceurs sur un système hétérogène ?

Notre objectif est de générer la meilleure plate-forme à partir d'une ensemble de noeuds disponibles pour répondre aux demandes des clients. Cependant, trouver le meilleur déploiement parmi des ressources hétérogènes est un problème difficile proche de celui de trouver le meilleur arbre de diffusion pour un graphe général qui est un problème NP-difficile.

Dans ce rapport, nous présentons alors une heuristique dans ce cadre précis. Nous appliquons notre heuristique pour un déploiement automatique d'un environnement de résolution de problèmes à large échelle. Des expériences sont menées afin de comparer le déploiement généré par l'heuristique avec d'autres déploiements probables.

Mots-clés : Déploiement, Grille de calcul, Ordonnancement en régime permanent, Localisation et découverte de ressources

1 Introduction

Due to the scale of grid platforms, as well as the geography location of resources, middleware should be distributed to provide scalability and adaptability. Much work has focused on the design and implementation of distributed middleware [8, 14, 28, 29, 32]. 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 exists. In other words, questions such as **which (and 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 for heterogeneous resources.

Software engineering processes for deployment and configuration (software deployment) of distributed system service-level functionality have been extensively researched [1, 21, 24], however very little work has been done around system deployment [18, 22]. To the best of our knowledge, no deployment algorithm or model has been given for arranging the components of a problem solving environment (PSE) in such a way as to maximize the number of requests that can be treated in a time unit on heterogeneous resources. More related work is given in Section 2.

The goal of this paper is to provide an automated approach to deployment planning that provides a good deployment for client-server frameworks on heterogeneous platforms. We consider that a good deployment is the one that maximizes the throughput of the system in number of client requests. We define deployment planning as the process of assigning resources to middleware components with the goal of optimizing throughput. In this paper the terms "deployment planning" is often shortened to "deployment".

We use a distributed Problem Solving Environment called DIET (Distributed Interactive Engineering Toolbox) [8] as a test case for our approach. It uses a hierarchical arrangement of agents[1] and servers[2] to provide scalable, high-throughput scheduling and application provision services clients[3]. Figure 1 gives the execution scheme of a computation request. A hierarchy is a simple and effective distribution approach and has been chosen by a variety of middleware environments as their primary distribution approach [12, 17, 30].

We describe a hierarchy as follows. A server $s \in \mathbb{S}$ has exactly one parent that is always an agent $a \in \mathbb{A}$. A root agent $a \in \mathbb{A}$ has one or more child agents and/or servers and no parents. Non-root agents $a \in \mathbb{A}$ have exactly one parent and two or more child agents and/or servers. The arrangement of these elements is shown in Figure 1. We don't share resources between agents and servers. In a lot of case, it's important for application to be run alone without disruption. Moreover this model is close to real platform, thus a lot of batch scheduler are deployed on a gateway to access to computational servers.

To apply our approach to DIET, we have developed performance models for DIET components. We present real-world experiments demonstrating that the deployment chosen by our approach performs well when compared to other reasonable deployments.

[1] Agents select potential servers from a list of servers maintained in the database by frequent monitoring and maps client requests to one of the servers.

[2] Servers receive requests from clients and execute applications or services on their behalf.

[3] Users that require computations.

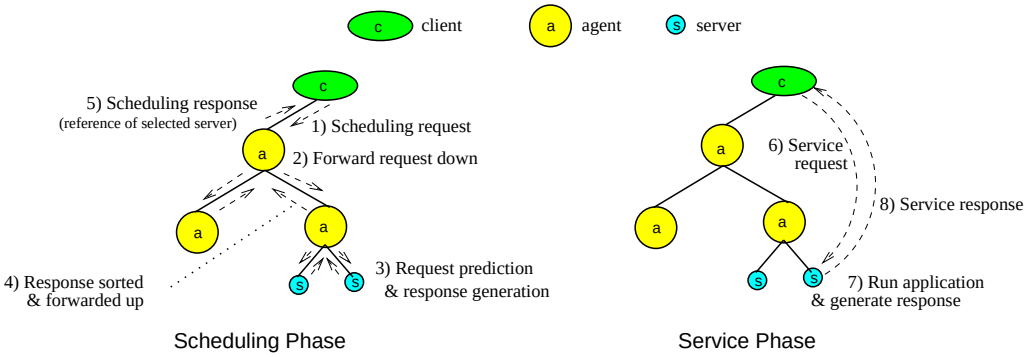

Figure 1: Platform deployment architecture and execution phases.

The rest of this paper is organized as follows. Section 2 presents some related work around deployment and scheduling. Section 3 first presents the request performance modeling and then the steady-state throughput modeling of our problem. A deployment heuristic for heterogeneous platforms is presented in Section 4. Section 5 presents the experimental results to test the ability of designed deployment heuristic and models to predict the good real-world deployments. Finally, Section 6 concludes the article and presents the future work.

2 Related work

A deployment is the mapping of a platform and middleware across many resources. Deployment can be broadly divided in two categories: software deployment and system deployment. *Software deployment* maps and distributes a collection of software components on a set of resources and can include activities such as configuring, installing, updating, and adapting a software system. Examples of tools that automate software deployment include SmartFrog [15] and Distributed Ant [16]. *System deployment* involves assembling physical hardware as well as organizing and naming whole nodes and assigning activities such as master and slave. Examples of tools that facilitate this process include the Deployment Toolkit [31] and Kadeploy [27].

ADAGE [25, 26] aims at automating applications deployments on a Grid environment. ADAGE relies on a generic description model for the applications: GADe. First, the user describes in its own formalism (the specific description) his/her application(s) he/she wants to deploy. Then, using a plugin ADAGE converts the specific description into the generic description GADe in XML (the plugin is related to the specific description, therefore it is the user's job to write it). Then, using the XML files containing the available resources description, the deployment control parameters (submission method: `ssh` or `rsh`, the scheduler, the placement constraints...) ADAGE computes a deployment plan containing the mapping of each process on resources (the schedulers are also plugins, so one can bring its own, currently only round-robin is implemented). Finally, ADAGE transfers the nec-

essary files (using `scp` or `rcp`), and launches the applications. While it offers a generic description model for the applications, which allows to use it for a wide variety of deployments. It does not have efficient algorithms to create the deployment plan (only round-robin), but its modular architecture allows to extend its possibilities.

Jade [4] is a middleware which purpose is to deploy and dynamically manage applications. The idea is to have a system which deals with the maintenance and management operations, instead of a physical person. Each application to deploy has first to be encapsulated in a Fractal[4] component which defines the incoming and outgoing interfaces, as well as controllers to control the behavior of the components (suspend, resume, or change its attributes). The user defines the application architecture in an XML file using an ADL (*Architecture Description Language*). This file contains the components description, their relations, the number of instances for the components, ...The *Software Installation Service* component is in charge of installing the components on the cluster's nodes, then the *Cluster Manager* maps the applications on the nodes. The *autonomic manager* executes and manages the components: it monitors the system's state and reacts according to the values given by a set of sensors (for example it can add or remove instances of a particular component in order to adapt to the system load) in order to keep the system in a stable state. Jade is more a dynamic autonomic management software than a deployment software, it takes into account the environment thanks to sensors within the components. Nevertheless, it still has an application deployment system based on an encapsulation in Fractal components. The resources selection is done among a group of available nodes on a cluster.

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 specialized middleware components to resources so that the best performance can be achieved from the system.

To the best of our knowledge, no deployment algorithm or model has been given for arranging the components of a PSE in such a way as to maximize the number of requests that can be treated in a time unit. In [23], 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, configured 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 complementary.

In [7] we presented a heuristic approach for improving deployments of hierarchical NES systems in heterogeneous Grid environments. The approach is iterative; in each iteration, mathematical models are used to analyze the existing deployment, identify the primary bottleneck, and remove the bottleneck by adding resources in the appropriate area of the system. The techniques given in [7] are heuristic and iterative in nature and can only be used to improve the throughput of a deployment that has been defined by other means; the current work provides an optimal solution to a more limited case and does not require a predefined deployment as input. In [6], we presented a decision builder tool to analyze existing hierarchical deployments, used mathematical models to identify bottleneck nodes, and removed the bottleneck by adding resources in the appropriate part of the system. The solution presented was iterative and heuristic in nature. In [10] we presented an approach for automatically determining optimal deployment for hierarchically distributed services

[4] `http://fractal.objectweb.org/`

on heterogeneous clusters. We proved that a complete spanning d-ary tree provides an optimal deployment and presented an algorithm to construct this optimal tree.

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 [3] frames workflow planning for the Grid as a planning problem. The approach is interesting for overall planning when one can consider that individual elements can communicate with no performance impact. 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.

Finally, this work is also somehow related to the design of efficient broadcast trees in grids [2, 13]. However, the big difference is that in our case, nodes also compute and this cost is taken into account.

3 Steady-state throughput modeling

In order to apply the heuristic presented in Section 4 to DIET, we require models for the scheduling and service throughput in DIET. Variables used in the formation of performance models to estimate the time required for various phases of request treatment are the following.

ρ is the throughput of the platform, ρ_{sched_i}, the scheduling request throughput of resource i, $\rho_{service}$, the service request throughput, S_{req}, the size of incoming requests, S_{rep}, the size of the reply, w_i, the computing power of resource i, d_in the children supported by resource i, W_{pre_i}, the amount of computation of resource i to merge the replies from d_i children, W_{sel_i}, the amount of computation of resource i needed to process the servers replies, W_{fix_i}, a fixed cost to process the server reply, W_{req_i}, the amount of computation needed by resource i to process one incoming request, W_{app_i}, the amount of computation needed by a server i to complete a service request for *app* service, B, the bandwidth of the link between resources, and N, the number of requests completed by all the servers in a time step. In our communication model, we assume that communication links are homogeneous, which is the case of our target platform.

Agent communication model: To treat a request, an agent *receives* the request from its parent, *sends* the request to each of its children, *receives* a reply from each of its children, and *sends* one reply to its parent. The time in seconds required by an agent i for receiving all messages associated with a request from its parent and d_i children is given by:

$$agent\_receive\_time = \frac{S_{req} + d_i \cdot S_{rep}}{B} \quad (1)$$

Similarly, the time in seconds required by an agent for sending all messages associated with a request to its d_i children and parent is given by:

$$agent\_send\_time = \frac{d_i \cdot S_{req} + S_{rep}}{B} \tag{2}$$

Server communication model: Servers have only one parent and no children, so the time in seconds required by a server for receiving messages associated with a scheduling request is given by:

$$server\_receive\_time = \frac{S_{req}}{B} \tag{3}$$

The time in seconds required by a server for sending messages associated with a request to its parent is given by:

$$server\_send\_time = \frac{S_{rep}}{B} \tag{4}$$

Agent computation model: Agents perform two activities involving computation: the processing of incoming requests and the selection of the best server amongst the replies returned by the agent's children. There are two activities in the treatment of replies: a fixed cost W_{fix_i} in MFlops and a cost W_{sel_i} that is the amount of computation in MFlops needed to process the server replies, and select the best server by agent i. Thus, the computation associated with the treatment of replies is given by

$$W_{rep_i}(d_i) = W_{fix_i} + W_{sel_i} \cdot d_i$$

The time in seconds required by the agent for the two activities is given by the following equation.

$$agent\_comp\_time = \frac{W_{req_i} + W_{rep_i}(d_i)}{w_i} \tag{5}$$

Server computation model: Servers also perform two activities involving computation: performance prediction as part of the scheduling phase and provision of application services as part of the service phase. We suppose that a deployment with a set of servers $\mathbb{S}$ completes N requests in a given time step. Then each server i will complete N_i requests, a fraction of N such that:

$$\sum_{i=1}^{N} N_i = N \tag{6}$$

On average each server i do prediction of N requests and complete N_i service requests in a time step. For example, the servers as a group require T seconds to complete N requests, then

$$T(N) = \frac{W_{pre_i} \cdot N + W_{app_i} \cdot N_i}{w_i} \tag{7}$$

From Equation 7, we can calculate the requests completed by each server i as:

$$N_i = \frac{T \cdot w_i - W_{pre_i} \cdot N}{W_{app_i}} \tag{8}$$

From Equations 6 and 8, we get time taken by the servers to process N requests as

$$T = N \cdot \frac{1 + \sum_{i=1}^{N} \frac{W_{pre_i}}{W_{app_i}}}{\sum_{i=1}^{N} \frac{w_i}{W_{app_i}}} \tag{9}$$

so, time taken by the servers to process one request is

$$server\_comp\_time = \frac{1 + \sum_{i=1}^{N} \frac{W_{pre_i}}{W_{app_i}}}{\sum_{i=1}^{N} \frac{w_i}{W_{app_i}}} \tag{10}$$

Now we use these constraints to calculate the throughput of each phase of the middleware deployment. We use a model with no internal parallelism, $M(r, s, w)$, [9] for the capability of a computing resource to do computation and communication. In this model, a computing resource has no capability for parallelism. It can either send a message, receive a message, or compute. Only a single port is assumed. Messages must be sent and received serially.

Servicing throughput of server i is then:

$$\rho_{service_i} = \frac{N_i}{T + \frac{S_{req}+S_{rep}}{B} \cdot N} \tag{11}$$

So according to Equations 6 and 11, servicing throughput of platform is given by:

$$\rho_{service} = \sum_{i=1}^{N} \frac{N_i}{T + \frac{S_{req}+S_{rep}}{B} \cdot N} \tag{12}$$

$$= \frac{1}{\frac{1+\sum_{i=1}^{N} \frac{W_{pre_i}}{W_{app_i}}}{\sum_{i=1}^{N} \frac{w_i}{W_{app_i}}} + \frac{S_{req}+S_{rep}}{B}} \tag{13}$$

The scheduling throughput ρ_{sched} in requests per second is given by the minimum of the throughput provided by the servers for prediction and by the agents for scheduling as shown below.

$$\rho_{sched} = min_{\forall i} \left(\frac{1}{\frac{W_{pre_i}}{w_i} + \frac{S_{req}}{B} + \frac{S_{rep}}{B}}, \frac{1}{\frac{W_{req_i}+W_{rep_i}(d_i)}{w_i} + \frac{S_{req}+d_i \cdot S_{rep}}{B} + \frac{d_i \cdot S_{req}+S_{rep}}{B}} \right) \tag{14}$$

In the servicing phase, only the server participate, as a result, the $\rho_{service}$ is calculated from server's computation constraint as shown below.

$$\rho_{service} = min_{\forall i} \left(\frac{1}{\frac{S_{req}}{B} + \frac{S_{rep}}{B} + \frac{1+\sum_{i=1}^{N} \frac{W_{pre_i}}{W_{app_i}}}{\sum_{i=1}^{N} \frac{w_i}{W_{app_i}}}} \right) \tag{15}$$

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 $\rho_{service}$ [10]. Thus using Equations 14 and 15, we generate the following equation that calculate the throughput of the platform.

$$\rho = min_{\forall i}\left(\frac{1}{\frac{W_{pre_i}}{w_i}+\frac{S_{req}}{B}+\frac{S_{rep}}{B}}, \frac{1}{\frac{W_{req_i}+W_{rep_i}(d_i)}{w_i}+\frac{S_{req}+d_i\cdot S_{rep}}{B}+\frac{d_i\cdot S_{req}+S_{rep}}{B}}, \frac{1}{\frac{S_{req}}{B}+\frac{S_{rep}}{B}+\frac{1+\sum_{i=1}^{N}\frac{W_{pre_i}}{W_{app_i}}}{\sum_{i=1}^{N}\frac{w_i}{W_{app_i}}}}\right) \quad (16)$$

We use this formula to calculate the throughput of the hierarchy in our experiments.

4 Heuristic for middleware deployment

Our objective is to *find a deployment that provides the maximum throughput (ρ) of completed requests per second*. A *completed request* is one that has completed both the scheduling and service request phases and for which a response has been returned to the client. When the maximum throughput can be achieved by multiple distinct deployments, the preferred deployment is the one using the least resources.

The platform architecture that we target is composed of heterogeneous resources that have homogeneous connectivity. The working phases of the architecture is shown in Figure 1.

`calc_hier_ser_pow`	is a function to calculate the servicing power provided by the hierarchy when load is equally divided among the servers of the hierarchy.
`calc_sch_pow`	is a function to calculate the scheduling power of a node according to the computing power of the node and number of its children.
`plot_hierarchy`	is a function to fill the adjacency matrix. Adjacency matrix is filled according to the number of children that each agent (from agent array) can support.
`shift_nodes`	is a function to shift up the node id in the server array if any server is converted as an agent.
`sort_nodes`	is a function to sort the available nodes according to there scheduling power calculated by function `calc_sch_pow`.
`write_xml`	is a function to generate an XML file according to the adjacency matrix, that is given as an input to deployment tool to deploy the hierarchical platform.

Table 1: Procedures used in Heuristic 1.

The throughput of the deployment platform is dependent on the scheduling throughput (ρ_{sched}) of each node and servicing throughput ($\rho_{service}$) of the deployment. Scheduling throughput and servicing throughput, as well as placement of the nodes in the hierarchy depend on the computing

diff	store minimum throughput among, ρ_{sched}, $\rho_{service}$ which is calculated in previous step, and client demand
min_ser_cv	store minimum values among servicing throughput of hierarchy and client demand
n_nodes	total nodes available for hierarchy construction
throughput_diff	store minimum throughput among, ρ_{sched}, $\rho_{service}$, and client demand
vir_max_sch_pow	scheduling power of top node in sorted_nodes array
vir_max_ser_pow	servicing power of the hierarchy
sorted_nodes	array to store the id of sorted nodes
supported_children	number of child nodes supported by an agent

Table 2: Variables used in Heuristic 1.

power of the nodes. It is very difficult to consider heterogeneity of computing resource and communication at the same time. In this primary work we focus on heterogeneous computing resource and consider homogeneous communication. In case of cluster it is not so far from the reality but the results will be different when we consider communications between clusters. We plan to deal with heterogeneous communication in future works.

For sake of simplicity we have defined some procedures (Table 1) for the middleware deployment heuristic 1. Variables used in the heuristic 1 are presented in Table 2. The heuristic is based on the exact calculation of number of children supported by each node. As scheduling power of any agent is limited by the number of children that it can support, to select a node that can act as agent, we calculate the scheduling power of each node with a number of children equal to the available nodes, n_nodes. Actually, it is not only the number of children that effects the scheduling power of the parent node but also the computing power of its children (that means the computing power dependency is taken into account). At this point we ignore which node will be agent, so as to remove from the total nodes, thus, initially we take children equal to $n\_nodes - 1$.

First, we sort the nodes according to scheduling power with n_nodes children in descending order (Step 1 and 2). Top node in the sorted list is most suitable to be an agent. Now we calculate the scheduling power of each node only with one child, so as to calculate the maximum scheduling power possible by the node (Step 3). Step 6 checks if the scheduling power of the node is less than the client demand then one level hierarchy is deployed with one agent and one server (top two nodes of the sorted list), because if more servers are added to the node, scheduling power will decrease. But if scheduling power is higher than client demand, then servicing power is increased by adding new nodes. For this new agents could be added.

In Step 14, we calculate the scheduling throughput of the top node in the sorted list by incrementing number of its children. If any other node can support more than one child, then that node is added in the hierarchy with the children that the node can support. Again servicing power is calculated considering the total number of servers in the hierarchy. Steps 15 to 39 are repeated until all the nodes are used or client demand is fulfilled or throughput of the hierarchy starts decreasing.

Then the connection between the nodes is presented in the form of adjacency matrix in Step 40. In Step 41 hierarchy is represented written in an XML file which is used by the deployment tool.

Algorithm 1 Pseudo-code describing heuristic to find the best hierarchy

```
Compute calc_sch_pow() for each node with n_nodes-1 children
Compute sorted_nodes[] by using sort_nodes()
Compute vir_max_sch_pow by using calc_sch_pow() with sorted_nodes[0] as an agent
Compute vir_max_ser_pow by using calc_hier_ser_pow() with sorted_nodes[1] as a server
Compute min_ser_cv
if (vir_max_sch_pow < min_ser_cv) then
   Deployment is just 1 agent and 1 server
else
   Compute throughput_diff and diff is a maximum arbitrary value
   while (diff>throughput_diff) do
      diff=throughput_diff;
      Compute vir_max_sch_pow with supported_children equal to 2
      Select next node from sorted_nodes[] as server
      Compute vir_max_ser_pow and throughput_diff
      while (∃ unused nodes and vir_max_sch_pow > vir_max_ser_pow) do
         if the number of children supported by the current server node >1) then
            Change server to agent using shift_nodes() and add next node from the sorted_nodes[] as a child to this new converted agent
            while the number of children < supported_children by the node do
               Compute vir_max_ser_pow with one extra child
               if ((vir_max_ser_pow<client_volume)&&(∃ unused nodes)&&(vir_max_ser_pow<vir_max_sch_pow)) then
                  Take next node from sorted_nodes[] as a server
                  Compute vir_max_ser_pow
               end if
            end while
         end if
         Compute new throughput_diff
      end while
      if (vir_max_sch_pow<vir_max_ser_pow) then
         if (diff<throughput_diff) then
            Remove 1 child from the last agent
         else
            diff = throughput_diff
         end if
      end if
      if (supported_children == n_nodes-1) then
         diff=throughput_diff;
      end if
   end while
end if
plot_hierarchy();
write_xml();
```

5 Experimental Results

In this section we present our experiments to test the ability of our deployment model to correctly identify good real-world deployments. The deployment method described in section focuses on maximizing steady-state throughput, as a result, we focus our experiments on testing the maximum sustained throughput provided by different deployments.

5.1 Experimental Environment

DIET 2.0 is used for all deployed agents and servers; GoDIET [5] version 2.0.0 is used to perform the actual software deployment. In general, at the time of deployment, one can know neither the exact job mix nor the order in which jobs will arrive. Instead, one has to assume a particular job mix, define a deployment, and eventually correct the deployment after launch if it was not well-chosen. For these tests, we use the DGEMM application, a simple matrix multiplication provided as part of the level 3 BLAS package [11].

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. A unit of load is introduced via a script that runs a single request at a time in a continual loop. We then introduce load gradually by launching one client script every second. We introduce new clients until the throughput of the platform stops improving; we then let the platform run with no addition of clients for 10 minutes.

Table 3 presents the parameter values we used for DIET in the model. To measure message sizes S_{req} and S_{rep}, we deployed an agent and a single DGEMM server on the Lyon cluster and then launched 100 clients serially from the same cluster. We collected all network traffic between the agent and the server machines using `tcpdump` and analyzed the traffic to measure message sizes using the Ethereal Network Protocol analyzer[5]. This approach provides a measurement of the entire message size including headers. Using the same agent-server deployment, 100 client repetitions, and the statistics collection functionality in DIET. Detailed measurements of the time required to process each message at the agent and server level was also recorded. The parameter W_{rep} depends on the number of children attached to an agent. We measured the time required to process responses for a variety of star deployments including an agent and different numbers of servers. A linear data fit provided a very accurate model for the time required to process responses versus the degree of the agent with a correlation coefficient of 0.97. Thus, this linear model is used for the parameter W_{rep}. Finally, we measured the capacity of our test machines in MFlops using a mini-benchmark extracted from Linpack and this value is used to convert all measured times to estimates of the MFlops required.

[5] `http://www.ethereal.com`

DIET elements	W_{req} (MFlop)	W_{rep} (MFlop)	W_{pre} (MFlop)	S_{rep} (Mb)	S_{req} (Mb)
Agent	1.7×10^{-1}	$4.0 \times 10^{-3} + 5.4 \times 10^{-3} \cdot d$	-	5.4×10^{-3}	5.3×10^{-3}
Server	-	-	6.4×10^{-3}	6.4×10^{-5}	5.3×10^{-5}

Table 3: Parameter values for middleware deployment on Lyon site of Grid'5000

5.2 Deployment approach validation on homogeneous platform

The usefulness of our deployment heuristic depends heavily on the performance model of the middleware. This section presents experiments designed to show the correctness of the performance model presented in previous section.

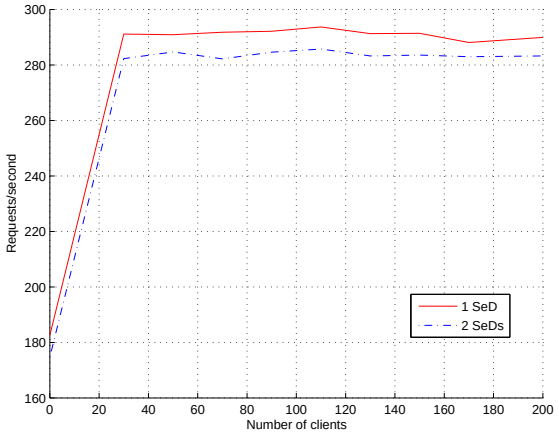

Figure 2: Star hierarchies with one or two servers for DGEMM 10x10 requests. Measured throughput for different load levels.

Experimental results shown in Figures 2 and 3 uses a workload of DGEMM 10x10 to compare the performance of two hierarchies: an agent with one server versus an agent with two servers. 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. More important is the correct prediction by our model to judge the effect of adding servers than to correctly predict the throughput of the platform (which is tough using a small computation grain for which cache effects improve performance).

Experimental results shown in Figures 4 and 5 use a workload of DGEMM 200x200 to compare the performance of the same one and two servers hierarchies. In this scenario, the model predicts that both hierarchies are limited by server performance and therefore, performance will roughly double

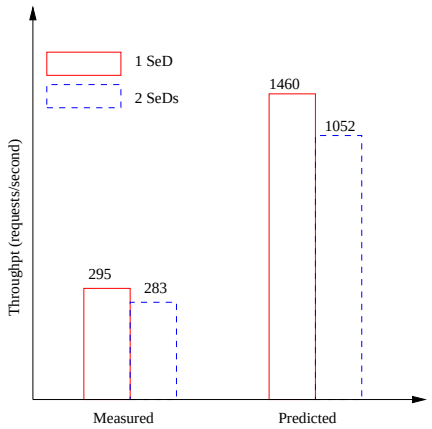

Figure 3: Star hierarchies with one or two servers for DGEMM 10x10 requests. Comparison of predicted and measured maximum throughput.

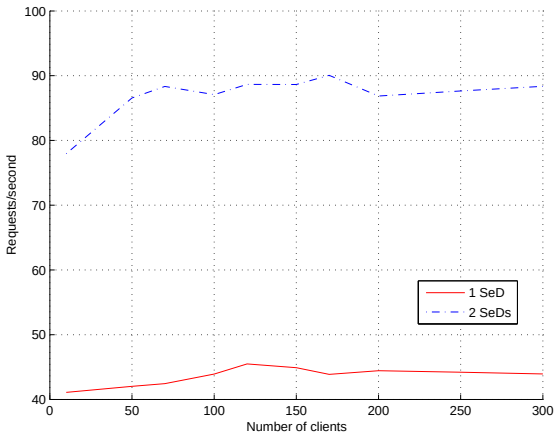

Figure 4: Star hierarchies with one or two servers for DGEMM 200x200 requests. Measured throughput for different load levels.

with the addition of the second server. The model correctly predicts that the two-server deployment will be the better choice.

In summary, our deployment performance model is able to accurately predict the impact of adding servers to a server-limited or agent-limited deployment.

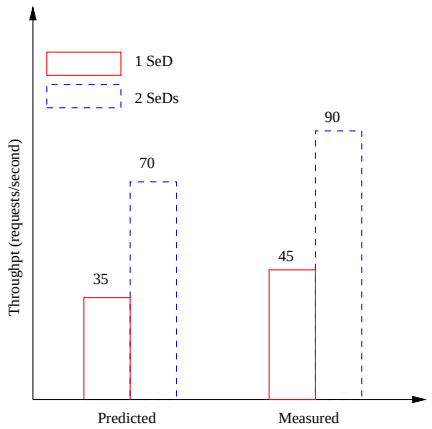

Figure 5: Star hierarchies with one or two servers for DGEMM 200x200 requests. Comparison of predicted and measured maximum throughput.

To verify our heuristic we compared the predicted deployment given by the heuristic with the experimental results presented in [10]. Table 4 presents the comparison by reporting the percentage of optimal throughput achieved by the deployments selected by different means.

DGEMM Size	Total Nodes	Opt. Deg.	Homo. Deg.	Heur. Deg.	Heur. Perf.
10	21	1	1	1	100.0%
100	25	2	2	2	100.0%
310	45	15	22	33	89.0%
1000	21	20	20	20	100.0%

Table 4: A summary of the percentage of optimal achieved by the deployment selected by our heterogeneous heuristic, optimal homogeneous model, and optimal degree.

5.3 Heuristic validation on heterogeneous cluster

To validate our heuristic we did experiments using two sites, Lyon and Orsay of the french experimental Grid called Grid'5000, a set of distributed computational resources deployed in France. We used 200 nodes of Orsay for the deployment of middleware elements and 30 nodes of Lyon for submitting the requests to the deployed platform.

To convert the homogeneous cluster into heterogeneous cluster, we changed the workload of the reserved nodes by launching different size of matrix multiplication as the background program on

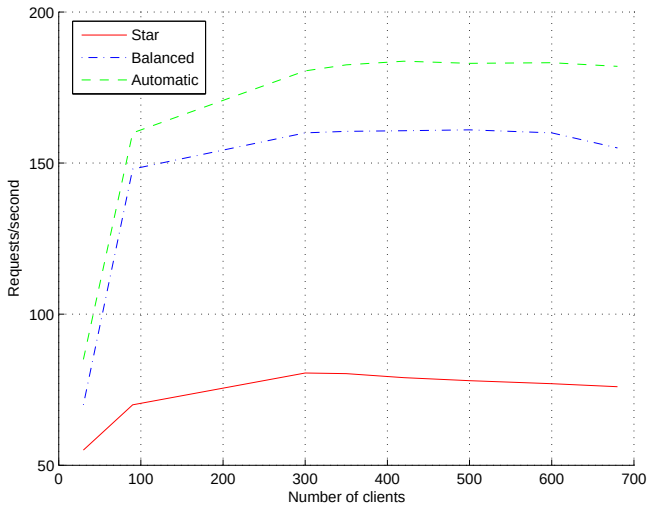

Figure 6: Comparison of automatically-generated hierarchy for DGEMM 310x310 with intuitive alternative hierarchies.

some of the nodes. After launching the matrix program in the background on the machines we used Linpack mini-benchmark to measure the capacity of the nodes in MFlops.

We compared two different deployments with the automatically generated deployment by our heuristic. The first deployment is a simple star type, where one node acts as an agent and all the rest are directly connected to the agent node (and these act as servers). In the second deployment, we deployed a balanced graph, one top agent connected to 14 agents and each agent connected to 14 servers with the exception of one agent with only 3 servers.

Clients submitted DGEMM problems of two different sizes. First we tested the deployments with DGEMM using 310x310 matrices. The heuristic generated deployment used only 156 nodes and deployment is organized as: top agent connected with 9 agents and each agent again connected to 9 agents. Two agents are connected with 9 servers, 6 agents are connected with 7 servers and one with 5 servers. Automatically generated deployment performed better than the two compared deployments. Results of the experimental results are shown in Figure 6.

Second experiment is done with DGEMM with matrices of size 1000x1000. Heuristic generated a star deployment for this problem size. Results in Figure 7 shows that star performed better than the second compared deployment.

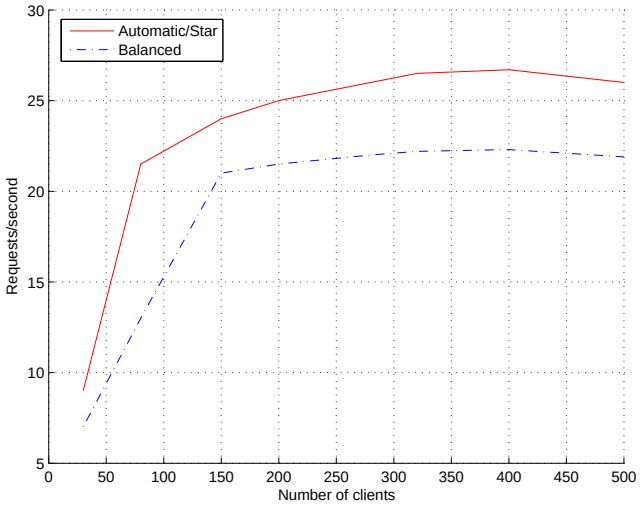

Figure 7: Comparison of automatically-generated hierarchy for DGEMM 1000* 1000 with intuitive alternative hierarchy.

6 Conclusion and Future Work

We presented a deployment heuristic that accurately predicts the maximum throughput that can be achieved by the use of available nodes. The heuristic predicts a good deployment hierarchies for both homogeneous and heterogeneous resources. A comparison is made to test the heuristic for homogeneous resources and the heuristic performed up to 90% as compared to the homogeneous optimal algorithm presented in [10]. To validate the heuristic, experiments are performed on the Grid'5000 platform. Experiments have shown that automatically generated deployment by the heuristic performs better than some intuitive deployments for heterogeneous platforms.

In the near future one of our principal objectives is to implement the theoretical deployment planning techniques as Automatic Deployment Planning Tool (ADePT). It will be interesting to validate our theoretical concept of deployment planning by further experimentation with other hierarchical middlewares. We would also like to implement deployment planning for *arbitrary arrangements* of distributed resources.

In this model we consider that we have a function to know the execution time but we should study another approach with statistical mathematical function to forecast the execution time. Finally, we are interested to find a modelization to deploy several middlewares and/or applications on grid.

References

[1] A. Akkerman, A. Totok, and V. Karamcheti. Infrastructure for automatic dynamic deployment of j2ee applications in distributed environments. In *Component Deployment*, pages 17–32, 2005.

[2] O. Beaumont, L. Marchal, and Y. Robert. Broadcast trees for heterogeneous platforms. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Papers*, page 80.2, Washington, DC, USA, 2005. IEEE Computer Society.

[3] 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*, June 2003.

[4] Sara Bouchenak, Noël De Palma, Daniel Hagimont, and Christophe Taton. Autonomic Management of Clustered Applications. *IEEE International Conference on Cluster Computing*, pages 1–11, September 25th-28th 2006.

[5] E. Caron, P. K. Chouhan, and H. Dail. GoDIET: A Deployment Tool for Distributed Middleware on Grid 5000. In *EXPEGRID workshop at HPDC2006*, Paris, June 2006.

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

[7] E. Caron, P.K. Chouhan, and A. Legrand. Automatic deployment for hierarchical network enabled server. In *The 13th Heterogeneous Computing Workshop*, Apr. 2004.

[8] E. Caron and F. Desprez. DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid. *International Journal of High Performance Computing Applications*, 20(3):335–352, 2006.

[9] P. K. Chouhan. *Automatic Deployment for Application Service Provider Environments*. PhD thesis, Ecole Normale Supérieure de Lyon, Sept 2006.

[10] P. K. Chouhan, H. Dail, E. Caron, and F. Vivien. Automatic middleware deployment planning on clusters. *International Journal of High Performance Computing Applications*, 20(4):517–530, November 2006.

[11] 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, Oct. 1995.

[12] S. Dandamudi and S. Ayachi. Performance of Hierarchical Processor Scheduling in Shared-Memory Multiprocessor Systems. *IEEE Trans. on Computers*, 48(11):1202–1213, 1999.

[13] Mathijs den Burger and Thilo Kielmann. Mob: zero-configuration high-throughput multicasting for grid applications. In *HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing*, pages 159–168, New York, NY, USA, 2007. ACM.

[14] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[15] P. Goldsack and P. Toft. Smartfrog: a framework for configuration. In *Large Scale System Configuration Workshop*. National e-Science Centre UK, 2001. http://www.hpl.hp.com/research/smartfrog/.

[16] W. Goscinski and D. Abramson. Distributed Ant: A system to support application deployment in the Grid. In *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, Nov. 2004.

[17] A.W. Halderen, B.J. Overeinder, and P.M.A. Sloot. Hierarchical Resource Management in the Polder Metacomputing Initiative. *Parallel Computing*, 24:1807–1825, 1998.

[18] T. Kichkaylo, A. Ivan, and V. Karamcheti. Constrained Component Deployment in Wide-Area Networks Using AI Planning Techniques. In *IPDPS*, 2003.

[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*, Apr. 2003.

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

[21] G. Kirby, S. Walker, S. Norcross, and A. Dearle. A methodology for developing and deploying distributed applications. In *Component Deployment*, pages 37–51, 2005.

[22] S. Lacour, C. Pérez, and T. Priol. Deploying corba components on a computational grid: General principles and early experiments using the globus toolkit. In *Component Deployment*, pages 35–49, 2004.

[23] S. Lacour, C. Pérez, and T. Priol. Deploying CORBA components on a Computational Grid: General principles and early experiments using the Globus Toolkit. In *2nd International Working Conference on Component Deployment*, May 2004.

[24] S. Lacour, C. Perez, and T. Priol. Generic application description model: Toward automatic deployment of applications on computational grids. In *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, pages 284–287, Washington, DC, USA, 2005. IEEE Computer Society.

[25] Sébastien Lacour, Christian Pérez, and Thierry Priol. A network topology description model for grid application deployment. In Rajkumar Buyya, editor, *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID 2004)*, pages 61–68, Pittsburgh, PA, USA, November 2004. Held in conjunction with Supercomputing 2004 (SC2004).

[26] Sébastien Lacour, Christian Pérez, and Thierry Priol. Generic application description model: Toward automatic deployment of applications on computational grids. In *6th IEEE/ACM International Workshop on Grid Computing (Grid2005)*, Seattle, WA, USA, November 2005. Springer-Verlag.

[27] C. Martin and O. Richard. Parallel launcher for cluster of PC. In *Parallel Computing, Proceedings of the International Conference*, Sep. 2001.

[28] J. P. Morrison, B. Clayton, D. A. Power, and A. Patil. Webcom-G: grid enabled metacomputing. *Neural, Parallel Sci. Comput.*, 12(3):419–438, 2004.

[29] H. Nakada, M. Sato, and S. Sekiguchi. Design and implementations of NINF: towards a global computing infrastructure. *Future Gener. Comput. Syst.*, 15(5-6):649–658, 1999.

[30] J. Santoso, G.D. van Albada, B.A.A. Nazief, and P.M.A. Sloot. Simulation of Hierarchical Job Management for Meta-Computing Systems. *Int. Journal of Foundations of Computer Science*, 12(5):629–643, 2001.

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

[32] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399