Automatic Deployment for Hierarchical Network Enabled Server

Eddy Caron, Chouhan, Pushpinder Kaur, Arnaud Legrand

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n^o 5668

CENTRE NATIONAL
DE LA RECHERCHE
SCIENTIFIQUE

Automatic Deployment for Hierarchical Network Enabled Server

Eddy Caron,
Pushpinder Kaur Chouhan,
Arnaud Legrand

November 2003

Research Report N^o 2003-51

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`

INRIA

Automatic Deployment for Hierarchical Network Enabled Server

Eddy Caron, Pushpinder Kaur Chouhan, Arnaud Legrand

November 2003

Abstract

This paper focus on the deployment of grid infrastructures, more specifically Problem Solving Environments (PSE) for numerical applications on the grid. Even if the deployment of such an architecture is forced by physical constraints (firewall, access permission, security,. . .) its efficiency heavily depends on the quality of the mapping between its different components and the grid resources. This paper proposes a new model based on linear programming to estimate the performance of a deployment of a hierarchical PSE. The advantages of the modeling approach in this case are multiple: evaluate a virtual deployment before an actual deployment, provide a decision builder tool (i.e., designed to compare different architectures or buy new resource), take into account the platform scalability. Using this modeling, it is possible to determine the bottleneck of the platform and thus to know whether a given deployment can be improved or not. We illustrate this modeling by applying this results to an existing hierarchical PSE called DIET.

Keywords: Deployment, Grid de calcul, Network Enabled Server, Steady-state scheduling, Resource localization and selection.

Résumé

Ce papier porte sur le déploiement des infrastructures de grille, et plus précisément sur les Environnements de Résolution de Problèmes pour des applications numériques sur la grille. Même si le déploiement d'une telle architecture est influencé par des contraintes physiques (pare-feux, droits d'accès, sécurité, . . .), son efficacité dépend de l'adéquation entre le déploiement des différents composants et les ressources de grille. Cet article propose un nouveau modèle basé sur la programmation linéaire pour estimer la performance du déploiement hiérarchique d'un Environnements de Résolution de Problèmes. Les avantages de cette modélisation sont multiples. Elle permet d'évaluer la qualité d'un déploiement avant sa mise en œuvre, de comparer différentes architectures et de savoir s'il est intéressant d'acheter une nouvelle resource ou si un déploiement donne passe bien à l'échelle. Ce modèle nous permet de déterminer les goulots d'étranglement de la plate-forme et ainsi de savoir si une amêlioration pour un déploiement est envisageable. Nous appliquons cette modélisation à un Environnements de Résolution de Problèmes hiérarchique : DIET.

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

1 Introduction

Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed "autonomous" resources dynamically at runtime depending on their availability, capability, performance, cost, and user's quality-of-service requirements. Grid applications are special class of distributed applications that have high computing and resource requirements.

Such platforms are very promising but are very challenging to use because of their intrinsic heterogeneity in terms of hardware capacities, software environment and even system administrator orientations. That is why end-users have to rely on specialized middleware, like Problem Solving Environments (PSE), to run numerical applications on such platforms. Such middleware, like NetSolve [1], Ninf [11] or DIET [7], already exist and are commonly called Network Enabled Server (NES) environments [10]. They usually have five different components: *Clients* that submit problems to *Servers*, a *Database* that contains information about software and hardware resources, some *Monitors* that acquire information about the status of the network and of the computational resources, and a *Scheduler*. Depending on the client's request, the data location and the dynamic performance characteristics of the system, this *Scheduler* has to find the server that is the more suited to fulfill efficiently this request.

To use efficiently, grid resources require to organize the distributed NES components in some particular fashion. Most NES deployment are however very basic, yet neither theoretical nor practical framework exist, to find the best organization of these components. We propose a new model that enables to find bottlenecks in the organization and, if possible, can improve the overall performance of PSE by breaking these bottlenecks.

Section 2 gives an overview of DIET so as to apprehend the practical framework. Section 3 briefly presents some related work on steady-state scheduling. Section 4 use some ideas of this work to model the DIET architecture and present an algorithm that improve a deployment by analyzing its performance. Section 5 presents some early experiments on how to use this model to settle an automatic deployment of the DIET architecture.

2 DIET: Distributed Interactive Engineering Toolbox

DIET [7] is a hierarchical set of components to build NES applications in a Grid environment. This environment is built on top of different tools which are able to locate an appropriate server depending on the client's request, the data location (which can be anywhere on the system, because of previous computations) and the dynamic performance characteristics of the system. The aim of DIET is to provide a transparent access to a pool of computational servers at a very large scale. DIET architecture is shown in Figure 1. DIET mainly have following components:

Client is an application which uses DIET to solve problems. Many kinds of clients should be able to connect to DIET. Problems can be submitted from a web page, a PSE such as Scilab [6], or from a compiled program.

Master Agent (MA) receives computation requests from clients. These requests are generic descriptions of problem to be solved. Then the MA collects computation abilities from the servers and chooses the best one. The reference of chosen server is returned to the client. A client can be connected to an MA by a specific name server or a web page which stores the various MA locations.

Local Agent (LA) aims at transmitting requests and information between MAs and servers. The information stored on each LA is the list of requests and, for each of its subtrees, the number of servers that can solve a given problem and information about the data distributed in this subtree. Depending on the underlying network architecture, a hierarchy of LAs may be deployed between an MA and the servers it manages. No scheduling decision is made by an LA.

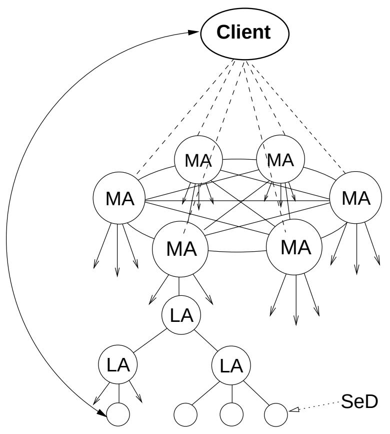


Figure 1: DIET Architecture

Server Daemon (SeD) encapsulates a computational server. The information stored on a 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,...). SeD declares the problems it can solve to its parent LA and provides an interface to clients for submitting their requests. A SeD can give performance prediction for a given problem with FAST [9] which is a dynamic performance forecasting tool.

Each server is thus handled by a SeD. A client wishing to solve a problem first has to obtain a reference to the server that is best suited for solving its problem. Clients connect to their nearest MA and send him their requests for computation. These requests are generic descriptions of problems to be solved. The MA then checks whether the request is correct (i.e., has all the parameters that a request should have), and if so, broadcasts the request to the neighboring nodes (LAs or MAs). These nodes forward the requests to the connected SeD. These servers send reply packets to their neighbor LAs. These packets contain the status (memory available, number of resources available, performance prediction, ...) of the server. LA compares the reply packet sent to it, by each of its connected server and selects the best server among them. Now the reply packet of the selected server is sent by the LAs to the neighboring LA or MA. Best server (or list of available servers, ranked in order of availability) is transmitted by the MA to the client. The client attempts to contact a server (from the list, starting with the first and moving down through the list). Then client sends the input data to the server. Finally the server executes the function on behalf of the client and returns the results.

Depending on the underlying network architecture, a hierarchy of LAs may be deployed between a MA and SeDs. The quality of a given deployment is its ability to handle a high number of request. On one hand, if the number of servers is very high and that no LAs are used, MAs clearly constitute the bottleneck. On the other hand, the DIET hierarchy may perfectly be able to handle the request whereas the servers cannot process quickly enough all the problems. In both cases, we need to evaluate the maximum number of requests per time-unit in a given deployment, so as to detect and handle which part of the platform is limiting. In the first situation, this information can be used to perform a redeployment of the DIET hierarchy and in the second one, this information could be used to warn users that the platform is overloaded.

3 Steady-state scheduling

A collection of heterogeneous resources (a processor, or a cluster, or whatever) and the communication links between them is naturally modeled as nodes and edges of an undirected tree-shaped graph. Each node is a computing resource capable of computing and communicating with its neighbors at different rates. We assume that one specific node, referred as client, initially generates requests and floods the MAs with these requests. The main problem is then to determine a steady state scheduling policy for each processor, i.e. the fraction of time spent in computing the request coming from client to server, fraction of time spent to select the best server, the fraction of time spent sending the request, and the fraction of time spent in receiving the reply packet (reply of the request), so that the (average) overall number of requests processed at each time-step can be maximized.

Beaumont et al. solve in [3] the steady-state master-slave scheduling problem for a tree-shaped heterogeneous platform. They explain how to allocate a large number of independent and equal size tasks on a heterogeneous grid computing platform. They first compute the maximum steady-state throughput of the platform using a linear program. Then, they show that this throughput can be reached if each node locally uses a bandwidth-centric strategy which states that, if enough bandwidth is available, then all children nodes are kept busy; if bandwidth is limited, then tasks should be allocated only to the children which have sufficiently small communication times, regardless of their computation power.

Some interesting points of this theoretical framework, like steady state scheduling strategy, equal size of requests, using linear constraints, etc. can be applied to our practical framework. We show in the next section how we model the DIET architecture in a particular situation.

4 Hierarchical Deployment Model

4.1 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 of 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). There is a client node, i.e. a node P_c, which generates the requests that are passed to the following nodes[1]. Each link $P_i \rightarrow P_j$ is labelled by the bandwidth value $c_{i,j}$ which represents the size of data sent per second between P_i and P_j. Measuring unit of bandwidth of link is Mb/second.

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 thus in Mb/request. The amount of computation needed by P_i to process one incoming request is denoted by $W_i^{(in)}$ and the amount of computation needed by P_i to merge the reply requests of its children is denoted by $W_i^{(out)}$. We denote by $W_i^{(DGEMM)}$ the amount of computation needed by P_i to process a generic problem (ie. BLAS [8] matrix multiplication called DGEMM)

The current architectural model does not consider data management, we focus on data in place applications (due to security problem, or parameter programming). Nevertheless, data management could be easily added like a new level of the modeling. The application target and the missing evaluate tools to estimate the data movement explain why we does not consider this aspect. In the same paradigm we consider the result that are very small can be neglected.

4.2 Steady State Operation

Our objective is to compare the maximum number of requests answered per second by a specific type of architecture so that best architecture can be selected. $\alpha_i^{(in)}$ denotes the number of incoming request (request coming from client) processed by P_i during one time-unit. Note that this number is not necessarily an integer and may be a rational. In a similar way, $\alpha_i^{(out)}$ is the number of outgoing requests (selection of the best server based on the reply packets) computed during one

[1] We use only one Client node for sake of simplicity but modeling many different clients with different problem types can be done easily.

time-unit by the node P_i. Servers are connected to the local agents at the last level of the graph. Therefore, $\alpha_i^{(DGEMM)}$ denote the number of problem solved by the node P_i if P_i is a server.

Number of requests replied in a time step depend on bandwidth of the link, size of the request, fraction of request being computed by a processor in a time step and the computing power of the processor. Therefore, we have the following constraints:

Computation resource for agents: $\forall P_i : \dfrac{\alpha_i^{(in)} \times W_i^{(in)} + \alpha_i^{(out)} \times W_i^{(out)}}{w_i} \leqslant 1$

Note that, it is necessary for each incoming request, that there should be a corresponding reply, and that each request is broadcasted along the whole hierarchy. Thus, there is no need to make a distinction between the $\alpha_i^{(in)}$ and the $\alpha_i^{(out)}$, that all are equal to the maximum throughput of the platform ρ. The previous equation can thus be simplified in the following equation:

$$\forall P_i : \rho \times \frac{W_i^{(in)} + W_i^{(out)}}{w_i} \leqslant 1 \tag{1}$$

Communication resources: ρ request for computations and ρ replies to these requests are transmitted per time-unit along each link $P_i \rightarrow P_j$. Therefore, we have:

$$\forall P_i \rightarrow P_j : \rho \times \frac{S_i^{(in)} + S_j^{(out)}}{c_{i,j}} \leqslant 1 \tag{2}$$

Server computation constraints: Each server P_i process $\alpha_i^{(DGEMM)}$ problem per time-unit. Therefore, we have

$$\forall P_i \text{ s.a } P_i \text{ is a server} : \frac{\alpha_i^{(DGEMM)} \times W_i^{(DGEMM)}}{w_i} \leqslant 1$$

All these values are linked to ρ by the equation $\rho = \sum_{P_i \text{ s.a } P_i \text{ is a server}} \alpha_i^{(DGEMM)}$. Therefore, we have

$$\rho \leqslant \sum_{P_i \text{ s.a } P_i \text{ is a server}} \frac{w_i}{W_i^{(DGEMM)}} \tag{3}$$

No internal parallelism: In this model, the computation and other operation performed by the node is done sequentially, so the summation of all operations performed by an agent should be less than the time step. Therefore, for all P_i, we have:

$$\underbrace{\rho\left(\frac{S_{father(i)}^{(in)}}{c_{father(i),i}} + \frac{S_i^{(out)}}{c_{father(i),i}}\right)}_{\text{Communications with the father}} + \underbrace{\rho\left(\sum_{P_i \rightarrow P_j} \frac{S_i^{(in)} + S_j^{(out)}}{c_{i,j}}\right)}_{\text{Communications with the slaves}} + \underbrace{\rho\left(\frac{W_i^{(in)} + W_i^{(out)}}{w_i}\right)}_{\text{Local computations}} \leqslant 1. \tag{4}$$

Note that if on P_i, computations can be performed in parallel with communications, the previous constraints should be changed as:

$$\underbrace{\rho\left(\frac{S_{father(i)}^{(in)}}{c_{father(i),i}} + \frac{S_i^{(out)}}{c_{father(i),i}}\right)}_{\text{Communications with the father}} + \underbrace{\rho\left(\sum_{P_i \rightarrow P_j} \frac{S_i^{(in)} + S_j^{(out)}}{c_{i,j}}\right)}_{\text{Communications with the slaves}} \leqslant 1. \tag{5}$$

Theorem 1. *The maximum number of requests that can be processed by the platform in steady state is given by the following formula:*

$$\rho = \min\left(\frac{w_i}{W_i^{(in)} + W_i^{(out)}}, \frac{c_{i,j}}{S_i^{(in)} + S_j^{(out)}}, \sum_{P_i \ s.a\ P_i\ is\ a\ server} \frac{w_i}{W_i^{(DGEMM)}}, \right. \tag{6}$$

$$\left. \frac{1}{\frac{S_{father(i)}^{(in)}}{c_{father(i),i}} + \frac{S_i^{(out)}}{c_{father(i),i}} + \sum_{P_i \to P_j} \frac{S_i^{(in)}+S_j^{(out)}}{c_{i,j}} + \frac{W_i^{(in)}+W_i^{(out)}}{w_i}} \right)$$

Theorem 2. *When maximizing the throughput, at least one of the constraints* (1)*,* (2)*,* (3)*,* (4) *and* (5) *is tight. This constraint represent the bottleneck of the platform.*

4.3 Automatic deployment and redeployment

Even when neglecting the servers constraints, finding the best topology is a hard problem since it amounts to find the best broadcast tree on a general graph, which is known to be NP-complete [4]. Note that even when neglecting the request mechanism, as soon as you take in account the communications of the problem's data, the problem of finding the best deployment becomes NP-complete too [2].

Even in real life, the topology of the underlying platform is particular and enforce some parts of the deployment. Therefore, we propose to improve the throughput of a given deployment by breaking its bottleneck. Using the previous theorems, we can find the bottlenecks and get rid of them by adding more LA to the parent of a loaded LA so as to divide the load of that particular LA. We add new LA according to the greedy algorithm 1.

```
1: while (number of available nodes > 0) do
2:   Calculate the throughput $\rho$ of structure.
3:   Find a node whose constraint is tight and that can be split
4:   if no such node exist then
5:     The deployment cannot be improved. Exit
6:   Split the load by adding new node to its parent
7:   Decrease the number of available nodes
```

Algorithm 1: Algorithm to add LA

In algorithm 1, line 3 checks whether it is possible to divide the load of a node or not. There may be many reasons for this condition to be false, for example, a node P_i having only one child cannot divide its load.

5 Case study

5.1 Parameter Measurement

To estimate the values of the different parameters, we did some experiments on a homogeneous cluster, composed of 16 processors (bi-PIII 1.4Ghz). To observe the effect on the computation time of each component, to process one request, we did different experiments by varying the number of LAs and SeDs with one MA and one client. We focus on linear approximation, as this approximation gives good result for our modeling.

The effect of adding LAs is shown in Figure 2, in the form of time taken to execute one request by each component. The time taken to compute a request by MA, increases with the addition of the LAs. MA take approximately 0.01 seconds for an incoming request and the time to compute an outgoing request is very very small, varies between 0.0001 to 0.00018 seconds. To compute an incoming and outgoing request by an LA is approximately between 0.0066 to 0.0076 and 0.0001125 to 0.000116 seconds respectively. The time taken by SeD for computation is very less effected by the increase in LAs.

In Figure 3, we have shown the effect of number of servers on the components computing capacity. For an incoming and outgoing request, computation time taken by MA, LA and SeD increases linearly. For incoming request MA and LA take approximately 0.009 to 0.01 and 0.008 to 0.022 seconds respectively. In case of outgoing request variation is very small for MA it is between 0.0001 to 0.0002 seconds and for LA 0.0002 to 0.00035 seconds. SeD computation time ranges between 0.00018 to 0.00019 seconds.

The time needed by each component to reply is so small that, to estimate which fraction should be incurred to computations and which fraction should be incurred to communications is very difficult. Therefore, we have considered it to be only computations (and therefore $S_i^{(out)}$ =0) as a very small amount of data is exchanged here.

From these experimental results, it is observed that if increasing the number of LAs increases accordingly the overall time needed to process one request on a MA or a LA, the fraction of time spent incurred to computations is almost constant. The behavior is similar when varying the number of SeDs.

From these measurements, we estimated the time taken by the components to communicate and compute the request. $S_i^{(in)}$ is then calculated by summing the communication time taken by each component and dividing it by the bandwidth of the local link. Similarly, the value of $W_i^{(in)}$ and $W_i^{(out)}$ is calculated by dividing the computation time of incoming request and the outgoing request by the processing power. Parameter values are summarized in the table below.

Components	$S_i^{(in)}$	$W_i^{(in)}$	$W_i^{(out)}$
Client	0.339	0.014	0
MA	0.010	0.159	0.78 e-3
LA	0.012	0.079	0.19 e-3

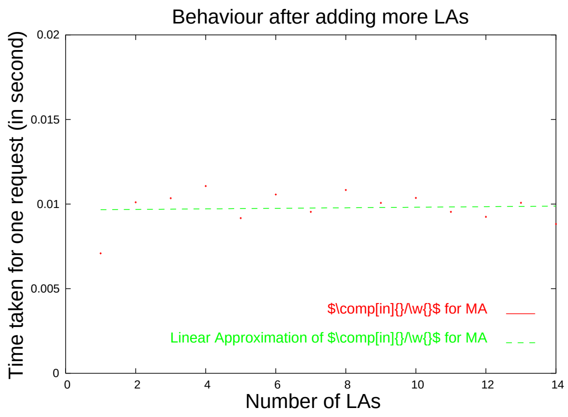
5.2 Test-bed

Distributed networks are all heterogeneous, i.e., every node has its own computing power(maybe different from other nodes) and bandwidth link between two nodes are also mostly different. We did simulation using a simple modeling of a real heterogeneous network shown in Figure 4. We have one client (`veloce`) and one MA at *Rocquencourt*. This MA is connected to two LAs: one at *Rennes* and another at *Grenoble*. 40 servers (`paraski` cluster) are connected to LA at Rennes. LA at Grenoble is connected with two LAs, LA at Grenoble has 200 servers (`icluster` cluster) and LA at *Sophia* has 14 servers (`galere` cluster). The power of the different nodes and the bandwidth of the different link between the nodes are depicted on Figure 4.

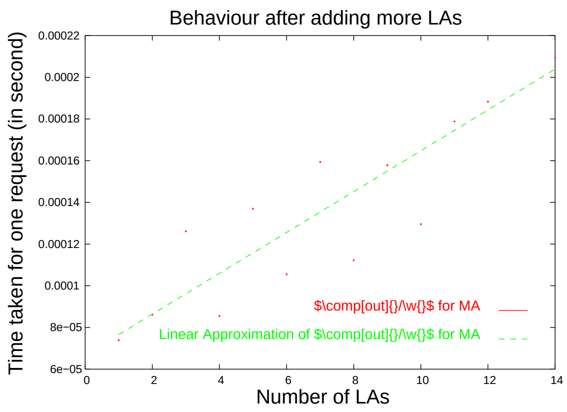
On this heterogeneous platform, the w_i's range from 15.6 Mflop/s to 292 Mflop/s and the bandwidth $c_{i,j}$ range from 10Mb/s to 2.5Gb/s.

5.3 Computing a good deployment

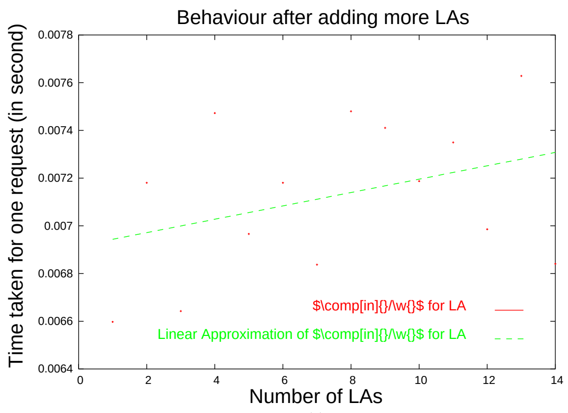
We calculated the throughput of this real heterogeneous network by using the formula mentioned before for calculating the throughput of structure. The performance of the original deployment (the natural one which is depicted on Figure 4) is rather low since it enable to process at most only 4 request per second. But we can improve the throughput of the network by breaking the first bottleneck located at `icluster` (see Figure 5). There is so much processors to handle on this cluster that broadcasting all the request and gathering the answer is very time-consuming. Seven more Local Agents have to be added, so that it will not be the bottleneck of the platform anymore. The eighth Local Agent has to be added at the Rennes's cluster. The gateway is very slow and, even if the number of servers is not as important as on the `icluster`, it has become an



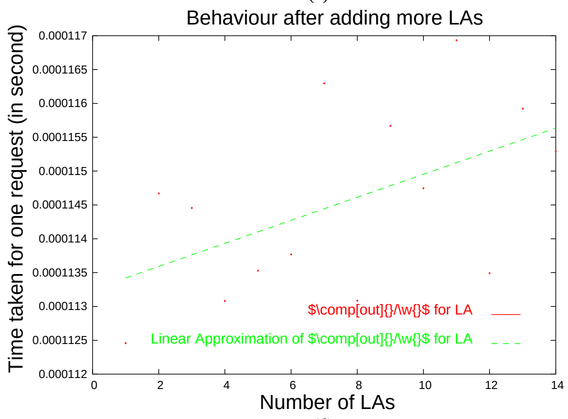
(a)



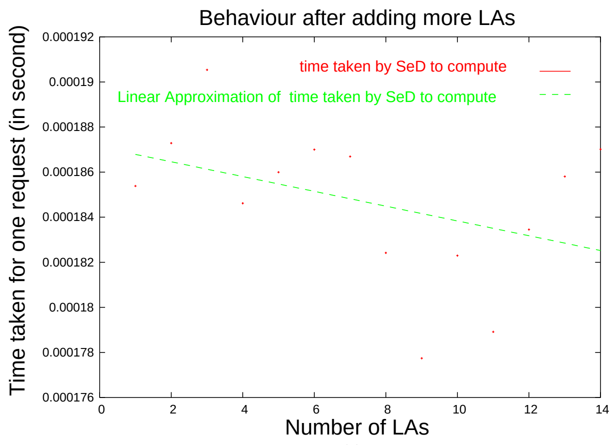
(b)



(c)

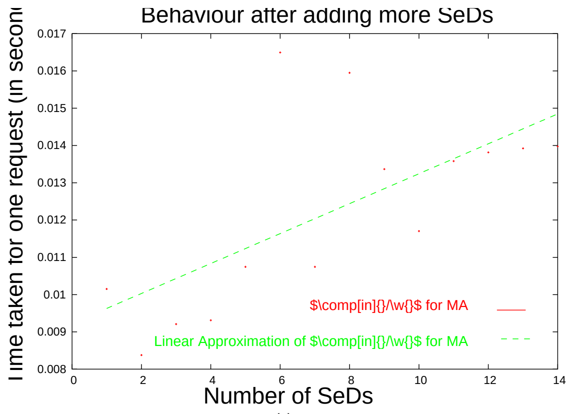


(d)

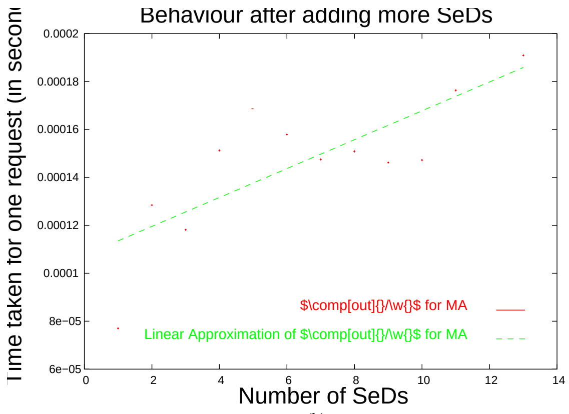


(e)

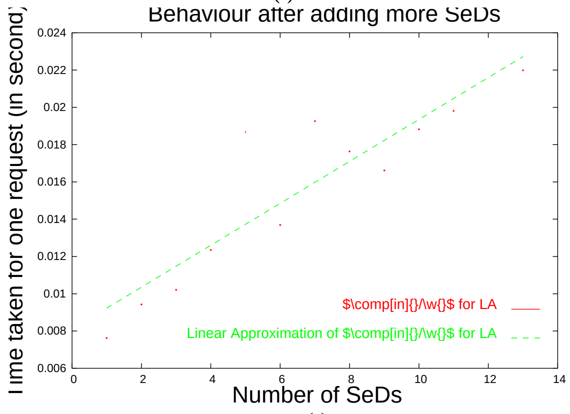
Figure 2: Performance calculation by adding LAs



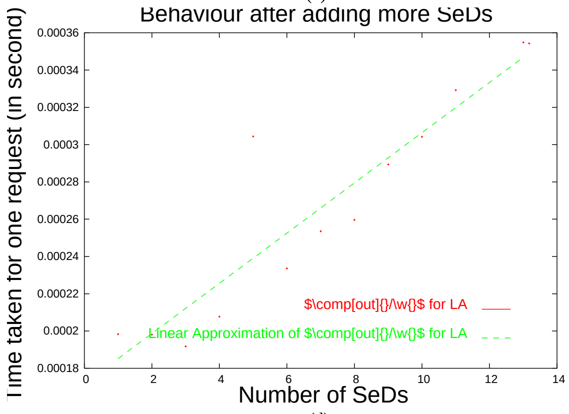
(a)



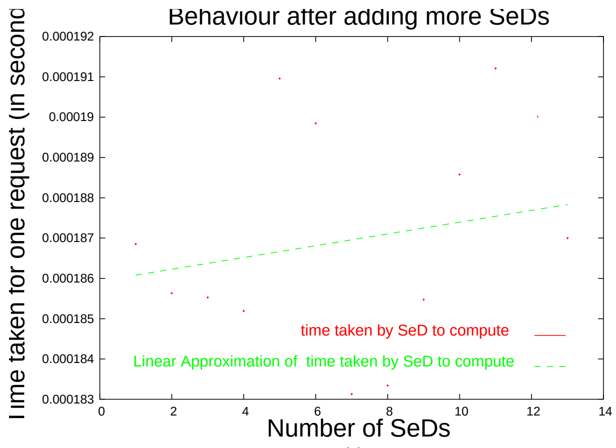
(b)



(c)



(d)



(e)

Figure 3: Performance calculation by adding SeDs

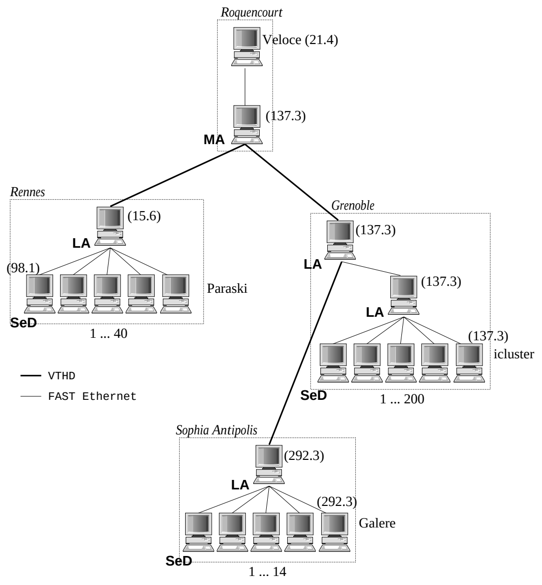


Figure 4: Heterogeneous network

issue. Nine more agents are then added on the `icluster` and then two again on Rennes. At this point, we have added a total of 18 agents and the tight equation is Equation (3), which means that all servers are working at full speed and that there is no hope of improving the throughput of the platform anymore.

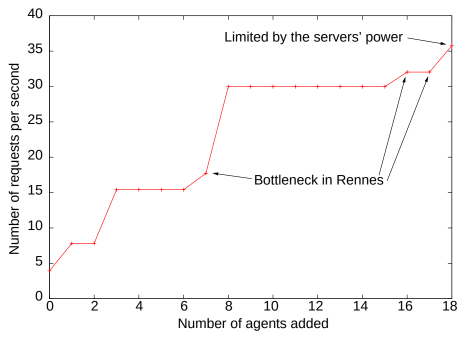


Figure 5: Throughput of heterogeneous network as more number of LA are added

6 Conclusion

An efficient deployment for NES environment is very important. However, yet neither theoretical nor practical framework exist, to find the best organization of these components. We use theoretical steady-state scheduling framework to propose a new model that enables to find bottlenecks in the organization of a hierarchical NES such as the DIET middleware. This model enables to improve the overall performance of PSE by breaking these bottlenecks and therefore to perform automatic deployment or redeployment. We plan to improve the presented algorithm to take into account the problem's data movement. Some theoretical work already exist to model this situation [5]. This model could also serve as a basis to an automatic deployment and redeployment tool. Thus, we plan to perform an automatic benchmarking of the different components coupled with the computation of a good deployment.

References

[1] 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. http://www.cs.utk.edu/netsolve/.

[2] Cyril Banino, Olivier Beaumont, Arnaud Legrand, and Yves Robert. Scheduling strategies for master-slave tasking on heterogeneous processor grids. In *PARA'02: International Conference on Applied Parallel Computing*, LNCS 2367. Springer Verlag, 2002.

[3] 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.

[4] Olivier Beaumont, Arnaud Legrand, Loris Marchal, and Yves Robert. Optimizing the steady-state throughput of broadcasts on heterogeneous platforms. Technical Report 2003-34, LIP, June 2003.

[5] Olivier Beaumont, Arnaud Legrand, Loris Marchal, and Yves Robert. Scheduling strategies for mixed data and task parallelism on heterogeneous clusters. *Parallel Processing Letters*, 13(2), 2003.

[6] E. Caron, S. Chaumette, S. Contassot-Vivier, F. Desprez, E. Fleury, C. Gomez, M. Goursat, E. Jeannot, D. Lazure, F. Lombard, J.M. Nicod, L. Philippe, M. Quinson, P. Ramet, J. Roman, F. Rubi, S. Steer, F. Suter, and G. Utard. Scilab to Scilab//, the OURAGAN Project. *Parallel Computing*, 11(27):1497–1519, Oct 2001.

[7] Eddy Caron, Frédéric Desprez, Frédéric Lombard, Jean-Marc Nicod, Martin Quinson, and Frédéric 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.

[8] Almadena Chtchelkanova, John Gunnels, Greg Morrow, James Overfelt, and Robert 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.

[9] F. Desprez, M. Quinson, and F. Suter. Dynamic Performance Forecasting for Network Enabled Servers in a Heterogeneous Environment. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2001)*, 2001.

[10] S. Matsuoka, H. Nakada, M. Sato, , and S. Sekiguchi. Design Issues of Network Enabled Server Systems for the Grid. http://www.eece.unm.edu/~dbader/grid/WhitePapers/satoshi.pdf, 2000. Grid Forum, Advanced Programming Models Working Group whitepaper.

[11] 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. http://ninf.apgrid.org/papers/papers.shtml.