INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Scalable Approach to Network Enabled Servers

Eddy Caron, Philippe Combes, Sylvain Contassot-Vivier, Frédéric Desprez,

Frédéric Lombard, Jean-Marc Nicod, Laurent Philippe, Martin Quinson,

Frédéric Suter

No 4501

July 2002

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4501--FR+ENG

A Scalable Approach to Network Enabled Servers

Eddy Caron, Philippe Combes, Sylvain Contassot-Vivier, Frédéric Desprez,
Frédéric Lombard, Jean-Marc Nicod, Laurent Philippe, Martin Quinson,
Frédéric Suter

Thème 1 — Réseaux et systèmes
Projet ReMaP

Rapport de recherche n˚4501 — July 2002 — 18 pages

Abstract: This report presents the architecture and the algorithms used in DIET (Distributed Interactive Engineering Toolbox), a hierarchical set of components to build Network Enabled Server applications in a Grid environment. This environment is built on top of different tools which are able to locate an appropriate server depending of 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. Some experiments are related at the end of this report, that exhibit the low cost of adding branches in the hierarchical tree of components and the performance increase induced.

Key-words: Metacomputing, Computational servers, Agent hierarchy.

(Résumé : tsvp)

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

Unit´e de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN (France)
T´el´ephone : 04 76 61 52 00 - International : +33 4 76 61 52 00
T´el´ecopie : 04 76 61 52 52 - International : +33 4 76 61 52 52

Une approche extensible des serveurs de calculs

Résumé : Ce rapport présente l'architecture et les algorithmes employés dans DIET (Distributed Interactive Engineering Toolbox), un ensemble hiérarchique de composants pour l'élaboration d'applications utilisant des serveurs de calcul dans un environnement de type Grille. Cet environnement est basé sur différents outils capables de localiser un serveur approprié en fonction de la requête d'un client, de la localisation des données et des caractéristiques dynamiques, en termes de performance, du système. Quelques expériences sont présentées à la fin de ce rapport qui exhibe le faible coût de l'addition de branches dans l'arbre hiérarchique des composants et l'amélioration des performances induite.

Mots-clé : Metacomputing, Serveurs de calcul, Hiérachie d'agents.

1 Introduction

Huge problems can now be computed over the Internet thanks to Grid Computing Environments [10]. Because most the current applications on the Grid are numerical, the use of specialized libraries like BLAS, LAPACK, ScaLAPACK, PETSc or FFTW is mandatory. But the integration of such libraries in high-level applications using languages such as Fortran or C is far from easy. Moreover the computational power and memory needs of such applications obviously may not be available on every workstation. Thus the RPC paradigm [13, 14] seems to be a good candidate to build Problem Solving Environments (PSE) for numerical applications on the Grid [11]. Several tools following this approach exist, such as NetSolve [3], NINF [15], NEOS [9], or RCS [2]. They are most commonly referred to as Network Enabled Server (NES) environments [14]. Such environments usually have five different components: *Clients* that submit problems they have to solve to *Servers*, a *Database* that contains information about software and hardware resources, a *Scheduler* that chooses an appropriate server depending on the problem sent and the information contained in the database, and finally *Monitors* that acquire information about the status of the computational resources.

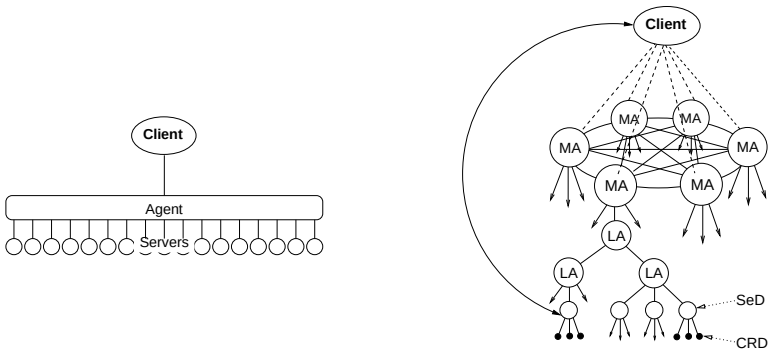
For instance in the architecture of NetSolve, which is a NES environment developed at the University of Tennessee, Knoxville, we find these components in the form of the client, server, and agent, with the agent containing the database and scheduler. Figure 1(a) shows how these components are organized. A NetSolve session works as follows. First the agent (which has to be unique) is launched. Then servers register to it by sending a list of problems that they are able to solve as well as the speed and the workload of the machine on which they are running and the network's speed (latency and bandwidth) between them and the agent. Once this initialization step is performed, a client can call the agent to solve a problem. The scheduler selects a set of most suitable servers to this problem and sends back this list to the client. The latter sends input objects to the first of the servers it can reach. The requested tasked are then run on this computational resource and the output objects are returned to the client.

But NetSolve and the other environments previously cited have a centralized scheduler which can become a bottleneck when many clients try to access several servers. Moreover as networks are highly hierarchical, the location of the scheduler has a great impact on the performance of the overall platform. This paper presents the architecture of DIET (Distributed Interactive Engineering Toolbox), a hierarchical set of components to build NES applications.

This document is organized as follows: Section 2 presents the overall architecture of the DIET platform and its main components and how distributed objects are used to connect these components. In Section 3, we give the algorithms used to discover software and hardware resources that are able to solve a problem submitted by a client. An experimental evaluation of these algorithms is given in Section 4, just before our conclusion and presentation of future work.

2 DIET architecture and related tools

In this section we give some details about the DIET architecture and present the different components involved in its hierarchy. The aim of an NES environment such as DIET is to provide a transparent access to a pool of computational servers. DIET focuses on offering such a service at a very large



(a) NetSolve overview (b) DIET overview

Figure 1: Comparison between two NES environments: NetSolve and DIET.

scale. A client which has a problem to solve should be able to obtain a reference to the server that is best suited for it. DIET is designed to take into account the data location when scheduling jobs. Data are kept as long as possible on (or near to) the computational servers in order to minimize transfer times. This kind of optimization is mandatory when performing job scheduling on a wide-area network.

DIET is built upon *Computational Resource Daemons* and *Server Daemons*. The scheduler is scattered across a hierarchy of *Local Agents* and *Master Agents*. NWS [17] sensors are placed on every node of the hierarchy to collect resource availabilities, which are used by an application-centric performance prediction tool named FAST [8, 16]. The database service is provided by the Scientific Libraries Metaserver (SLiM). Figure 1(b) shows the hierarchical organization of DIET.

2.1 DIET components

The different components of our software architecture are the following:

Client
A client is an application which uses DIET to solve problems. Many kinds of clients should be able to connect to DIET. A problem can be submitted from a web page, a PSE such as Scilab [5] or Matlab, or from a compiled program.

Master Agent (MA)
An MA receives computation requests from clients. These requests are generic descriptions of problems to be solved. The SLiM module (see Section 2.2) is used to find all the implementations of these generic problems. Then the MA collects computation abilities from the

servers and chooses the best one. The reference of the 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)
An LA aims at transmitting requests and information between MAs and servers. The information stored on an 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.

Server Daemon (SeD)
A 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, ...). A 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 thanks to the FAST module (see Section 2.3).

Computational Resources Daemon (CRD)
A computational resource represents a set of hardware and software components that can perform sequential or parallel computations on data sent by a client (or another server). For instance a CRD can be the entry point of a parallel computer. It usually provides a set of libraries and is managed by an SeD.

2.2 SLiM: Scientific Libraries Metaserver

SLiM's goal is to make the connection between problems submitted by clients and implementations available on servers. In most cases there is no one-to-one mapping. A single problem can be solved by many implementations from several libraries, while another problem may need more than one computational step to be solved. For example, if a user wants to solve a system of linear equations with a sparse matrix, depending of the data themselves, this problem can be solved by a direct solver or by a preconditioner followed by an iterative solver. Sequential and/or parallel versions of the routines may be available.

The main issue of this approach is to find a unified way to express the problems and data descriptions. One could use the description problem language from NetSolve, but it is not standard and lacks of a way to express parallel functions.

Since our first client interface was Scilab, we decided to use the name of the built-in functions as first problem description meta-language. Even if this approach is satisfying in this context, it lacks generality, and so we are currently working on defining a better solution based on the GAMS [4] problem taxonomy.

All needed information is stored in a LDAP [12] tree. LDAP is a distributed database protocol which was chosen for its read and search optimizations.

2.3 FAST: Fast Agent's System Timer

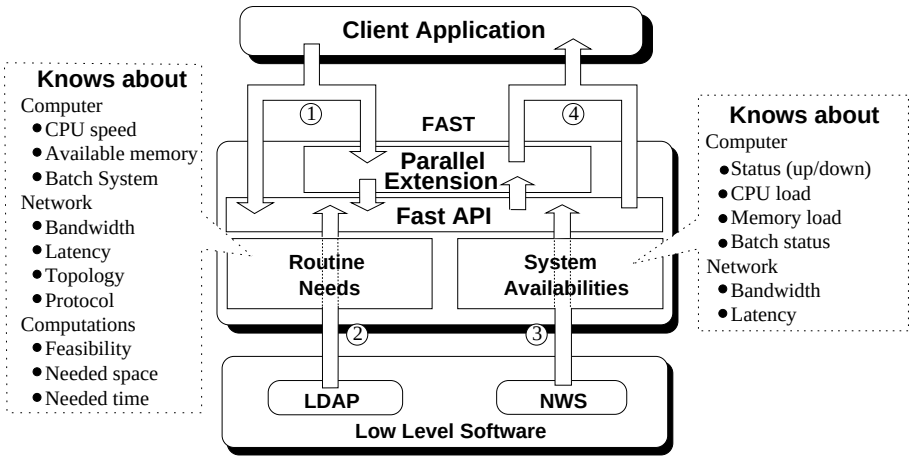


Figure 2: FAST overview.

FAST [8, 16] is a tool for dynamic performance forecasting in a Grid environment. As shown in Figure 2, FAST is composed of several layers and relies on low-level software. First it uses a network and CPU monitoring software to handle dynamically changing resources, like workload or bandwidth. FAST uses the Network Weather Service (NWS) [17], a tool developed at the University of California, Santa Barbara. This is a distributed system that periodically monitors and dynamically forecasts the performance of various network and computational resources. The resource availabilities acquisition module of FAST uses and enhances NWS. Indeed, if there is no direct NWS monitoring between two machines, FAST automatically searches for the shortest path between them in the graph of monitored links. It estimates the bandwidth as the minimum of those in the path and the latency as the sum of those measured. This allows for more accurate predictions when DIET is deployed over a hierarchical network.

In addition to the system availabilities, FAST can also forecast the time and space needs of computational routines, depending on both the parameter set and the machine where the computation would take place. For this, FAST benchmarks the routines at installation time on each machine for a representative set of parameters. After polynomial data fitting, the results are stored in the same LDAP-tree as SLiM. The user API of FAST is composed of a small set of functions that combine resource availabilities and routine needs from low-level software to produce ready-to-use values. These results about sequential routines and system availabilities can be combined into analytical models by the parallel extension [6] to forecast execution times of parallel routines.

Thus DIET components, as FAST clients, can access information like the time needed to move a given amount of data between two SeDs, the time to solve a problem with a given set of CRDs managed by an SeD, or the addition of these two quantities.

2.4 Interactions between SLiM, FAST and DIET

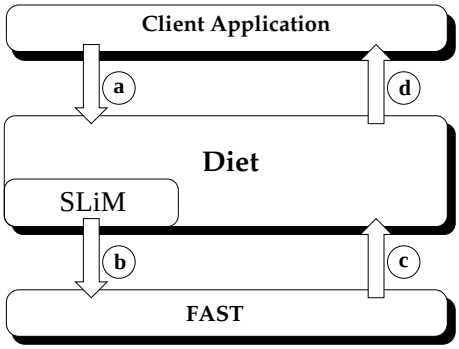


Figure 3: SLiM, FAST and DIET interactions.

To give a first overview of interactions between SLiM, FAST and DIET, let us consider a simplified case of the problem submission, as shown on Figure 3. The complete algorithm used by several agents organized in a DIET hierarchy will be given in Section 3.2.

First the client submits its problem to DIET (Fig 3.a). DIET invokes SLiM to know which implementations can solve this problem. For instance if this is a dense matrix multiplication problem, the `dgemm` function of the BLAS library would be a candidate. For each matching implementation, and for each machine providing this implementation, FAST has to forecast the computation time (Fig 3.b). The answer returned to DIET is therefore a list { scheduling possibility ; estimated time } (Fig 3.c). Given this, the MA finally makes a scheduling decision and informs the client of the best suited server to solve its problem (Fig 3.d).

2.5 Using Corba in DIET

NES environments can be implemented using a classic socket communication layer. NINF and NetSolve are implemented that way. Several problems to this approach have been pointed out such as the lack of portability or the limitation of opened sockets. Our aim is to implement and then deploy a distributed NES environment that works at a wider scale. Distributed object environments, such as *Java*, *DCOM* or Corba have proven to be a good base for building applications that manage access to distributed services. They not only provide transparent communications in heterogeneous networks, but they also offer a framework for the large scale deployment of distributed applications. Being open and language independent, Corba was chosen for the communication layer in DIET.

Corba systems provide a remote method invocation facility with a high level of transparency. This transparency should not dramatically affect the performance, communication layers being well optimized in most Corba implementations [7]. Indeed, the communication time is that of with sockets plus a constant value [1]. Moreover, the time to select a server using Corba should be short compared to the computation time.

Furthermore, Grid computing is a very active research domain. Existing platforms are usually subject to frequent experimental modifications and feature add-ons. Object oriented development platforms allow an easier development and a greater maintainability of the code. Corba is thus well suited to support distributed resources and applications in a large scale Grid environment. New dedicated services can be easily published and existing services can also be used. Thus we can conclude that Corba systems are one of the alternatives of choice for the development of Grid specific services. Our first DIET prototype is based upon *OmniORB*, a free Corba implementation which provides good communication performance.

3 DIET initialization and operation

In this section we study how to specify the order in which components should be started, giving as an example a DIET platform involving only one MA. This example is actually simpler to discuss and the algorithms presented here are easily extendable to the general case by broadcasting computation requests to the other MAs. Then, we discuss the way a server is chosen to solve a given problem, taking the communication and computation times into account.

3.1 DIET initialization

Figure 4 shows each step of the initialization of a simple Grid system. The architecture is built in the hierarchical order, each component contacting its father. The MA is the first entity to be started (1). It waits for connections from LAs or requests from clients.

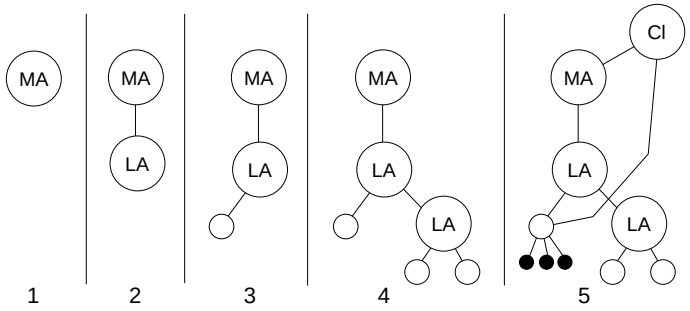


Figure 4: Initialization of a DIET system.

Then when an LA is launched, it subscribes to the MA (2). At this step of the system initialization, two kinds of components can connect to the LA: an SeD (3), which manages some computational resource, or another LA (4), to add a hierarchical level in this branch. Finally, any client can access the registered resource through the platform: it can contact an MA (5) to get a reference to the best server known and then directly connect to it.

The architecture of the hierarchy is described in configuration files and each component transmits the local configuration to its father. Thus, the system administration can also be hierarchical. For instance, an MA can manage a domain like a university, providing priority access to users of this domain. Then each laboratory can run an LA, while each team of the laboratory can run some other child-LAs to administrate its own servers. This hierarchical administration of the system allows local changes in the configuration without interfering with the whole platform.

3.2 Solving a problem

Let us assume that the architecture described in section 2 includes several servers able to solve the same problem, and that all data needed for the computation are available on one single server. The example presented in Figure 5 considers the submission of the problem `F()` involving data `A` and `B`.

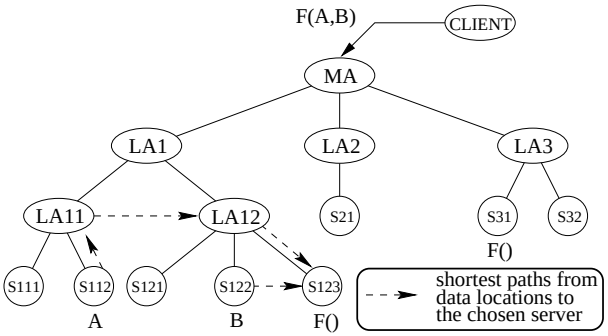


Figure 5: Problem submission example.

The algorithm presented below lets an MA choose among the servers it manages the one which will perform the computation. This decision is made in four steps:

- the MA finds the implementations matching the generic problem using SLiM;
- to locate the data involved and the capable servers it propagates a request through its subtrees down to the computational servers;
- then FAST estimates the computation time on all candidate servers, and they send this estimation back to the MA, as an answer to the request;
- once the MA has collected all the answers, it chooses the “fastest” server, sends its reference to the client, and performs the transfer of persistent data involved in the computation.

For the problem solving itself, the client connects to the server chosen: it sends its local data and specifies if the results should be kept in-place for further computation or if it should be brought back.

3.3 Data structures and algorithms

In order to choose a server, the MA first has to locate the servers that are able to solve the problem submitted and the data involved in the computation. This is why a request structure is sent to the target servers. This structure contains two fields: the `problemNickname`, and a list of `attributes` of the data involved, including details about their size and properties in order to evaluate computation and communication times. When the MA receives a request from a client, it builds a request structure and sends it to all its children which either own some of the needed data or are able to solve the problem. The request is transmitted from father to child in the tree following this scheme down to SeDs. Each LA labels its children reached by the request and waits for their responses.

Once the request structure reaches the SeDs concerned, they initiate a response structure and it back to their father. This structure contains three fields:

`myName` is the unique id of the component that sends back this response;

`data` is an array with an entry for each variable involved in the computation, each one containing two fields:

- `location` is the name of the component that owns the data,
- `timeToMe` is the estimation of the communication time to bring the variable from the component sending the structure, if its location is known;

`comp` contains an entry for every server (able to satisfy the request) known at this point of the tree. Three pieces of data are kept for each server:

- `name` is the id of the server,
- `tComp` is the estimated computation time to satisfy the request,
- `tComm` is an array containing the estimated time to bring each variable involved to that server;

Transfer times for data are computed dynamically while sending back the answers to the MA. Our idea is that all agents of the hierarchy should be deployed on the underlying network nodes. That is why we just sum the estimated transfer times between the various nodes of the tree. The algorithm is divided in three steps:

1. **Initialization**
   When an SeD receives a request, it sends a response structure to its father. It fills the `data` field for the variables it owns, leaving a null value for the others. If the server can solve the problem, it also fills a (`tComp`) (in a one-element `comp` array).

2. **Aggregation**
   Every LA gathers the responses coming from its children and aggregates them into one single

structure. The fields related to communication times are gradually filled as the structures come back up to the MA. FAST computes the transfer time of data to the capable servers, combining information from monitoring and data attributes. This transfer will use the shortest path among those that are monitored. This path only uses links between a father and its children or between brothers as shown in Figure 4. Figure 6 gives the complete algorithm used by LAs to aggregate responses coming from their children.

3. **Use**
   When the responses come back to the MA, it uses them to make a decision. The evaluated computation and communication times are used to find the server with the lowest response time to perform the computation.

However, we have to consider the case when a server is chosen twice and the first computation has not already started when the second problem was submitted. It could happen that the penalty of the first computation problem is not considered in the estimation of the computation time for the second one. This is a classical problem in dynamic performance evaluation, and we are presently working on an algorithm, based on contracts between servers and clients, to check that the estimation is still meaningful at the time of client/server connection.

for each data `D` **do**
  if none of my descendants own `D` **then**
    `timeToMe` = `0`
  else if one of my children references `D` **then**
    `timeToMe` = `timeToMe` for this child + time to send `D` from this child to me
  else if `D` is not known by any of my children **then**
    if a server `S` of my sub-tree can solve the problem **then**
      `D` will be sent through me to `S` if it's selected.
      ⇒ Increase `D`'s `tComm` for each server
    else
      `D` will be sent to a capable server `S` following a path in which I am not involved.
      ⇒ End `tComm`'s computation for my descendants.
    end if
  end if
end for

Figure 6: Complete result aggregation algorithm for a LA.

4 DIET architecture evaluation

In this section we exhibit the first experiments made with the DIET prototype we develop. These experiments only involve one MA and a hierarchy of LAs. Our goal is to validate our architecture and evaluate the performance of the request broadcast algorithm proposed.

In section 4.1, we examine the cost of adding servers in a DIET architecture with a single agent, then we compare a linear architecture to a binary tree in order to show how parallelism can be introduced in the request processing.

The impact of the use of LAs on the request broadcasting on a low bandwidth network is then investigated in section 4.2. This experiment shows that using a hierarchy of LAs can increase notably the speed of the server lookup.

We finally show in section 4.3 how the tree structure implies parallel request diffusion and performance predictions. Two experiments are lead in this section. The first one consists in adding a new branch to an existing DIET tree. The second one evaluates different ways to add servers to an existing architecture. These experiments show that servers can be added to a DIET tree without additional lookup cost if the tree architecture is chosen wisely.

4.1 Running DIET on a local network

Experiments in this section have been performed on a local Fast Ethernet network with several switches. This network is dedicated, so we do not take contention into account. For each experiment, we build a DIET tree. SeDs are only able to register and answer requests but not to perform computations. The problem submited by the clients is known by every SeDs. Thus, all SeDs are contacted at each request. The value measured is the average submission time (i.e. the time elapsed from the submission to the MA to the reception of a server reference).

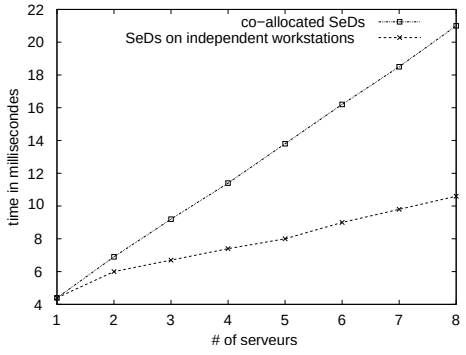


Figure 7: Time to process a request.

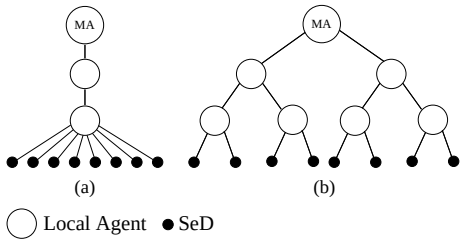


Figure 8: Comparison between two DIET trees.

4.1.1 Adding a server to DIET

Here we aim at evaluating the cost of adding an SeD to an existing DIET architecture by connecting it to the only launched LA. This introduces no parallelism in the broadcast of requests and thus should give the worst performance. Eight experiments have been done with a number of SeDs ranging from one to eight directly connected to the MA. The available number of workstations and the impossibility to co-allocate SeDs on a same machine without changing the results prevented us from running experiments on more servers. When an SeD is added to an existing architecture without adding an LA, the performance loss should be equivalent to the one observed in this experiment. The results of this experiment is given in the following table and Figure 7.

number of SeDs	1	2	3	4	5	6	7	8
request time (ms)	4.4	6.0	6.7	7.4	8.0	9.0	9.8	10.6

Results for a similar experiment with all SeDs co-allocated on the same workstation are also given Figure 7 to provide evidence that this has a strong impact on the system behavior. When SeDs are co-allocated, requests processing on all SeDs are run sequentially on the same processor. The curves show that the request processing is less time consuming when all SeDs are running on different computers thanks to parallel processing of those requests. The amount of parallelism in this processing actually depends on the structure of the tree: section 4.3 focuses on trying to build trees that maximize it. For instance, this experiment shows that adding four servers to a system that already contains four adds 3.2 ms to the request processing time. Experiments have been conducted to look for the lowest overhead when adding a server.

4.1.2 Comparing two architectures of similar depth

Figure 8 shows two DIET architectures that have the same depth and the same number of SeDs. Although a linear architecture such as (a) would never be deployed in practice, two LAs are used to obtain the same tree depth as for architecture (b). Thus, the cost of a communication on an branch of the DIET tree is exactly the cost of the communication through the Ethernet network. This prevents us from comparing the performance of architectures that have different depths, that is with different costs of a communication between the MA and an SeD.

The average request processing time is 52.2 ms on architecture (a) and only 33.5 ms for (b). The administrators have to carefully build their LA hierarchy to improve the performance. Experiments conducted in section 4.3 aim at providing some simple rules for this task.

4.2 Evaluation of the broadcast algorithm

We aim here at showing the benefits of the DIET hierarchical approach when a LA is used to optimize communication delays over a slow network link. In such a case, LAs are used to perform an efficient request broadcast from the MA to the SeDs. Figure 9 shows the two configurations used in that experiment. The right part of the figure shows that the introduction of a LA reduces to one the

number of messages sent across the slow link. In this section we examine the evolution of the request submission time depending on the number of SeDs with the two configurations shown in Figure 9.

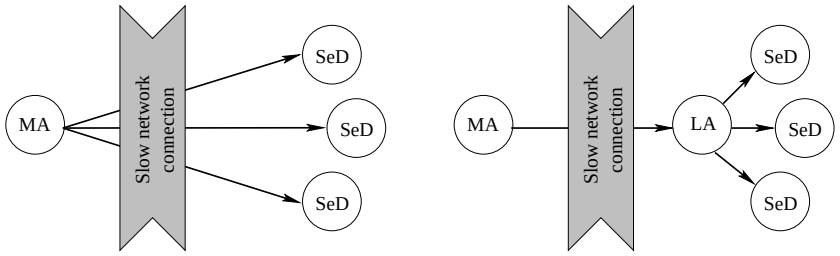


Figure 9: The DIET broadcasting algorithm validation experiment.

The slow network link has a bandwidth of 2MB/s. This bandwidth is shared with other links. The local network that supports the SeDs (and the LA in configuration shown on Figure 9(b)) is a Fast Ethernet network with several switches. The request processing time being much lower than the communication times on the slow link, up to 25 SeDs have been co-allocated on the same workstations. The MA and the client run on the same workstation on the other end of the slow link. Figure 10 shows the result of our tests. Experiments have been done with both configurations shown on Figure 9. For each number of SeDs, 50 clients were run sequentially and the average client execution time has been used has the submission time.

In both cases, results are nearly linear. With the configuration shown on Figure 9(b), the request submission time appears to be 3 (with 42 servers) up to 4 (with 72 servers) times less than without any LA. A linear regression shows that the slope is 4 times greater when no LA is deployed. This corroborates our idea that network bottlenecks are an important issue in NES environments.

4.3 Evaluating the architecture's cost

By evaluating the cost of additional servers in the architecture using several strategies, we show here how the request processing can be done in parallel in DIET. The experimental results show that new servers can be added to the system nearly without overhead to the request processing. The experimental conditions in this section are the same as in section 4.1.

4.3.1 Adding a branch to a DIET tree

Figure 11 shows how it is possible to have twice as many SeDs just by adding one son to the MA. This son heads a hierarchy similar to the existing branch, which consists of a binary tree. The

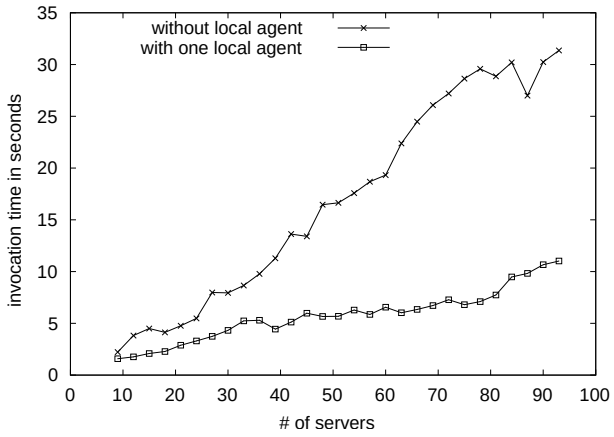


Figure 10: Time to process a request as a function of the number of servers with and without LA.

experimental conditions are the same as in section 4.1, in which an overhead of 3 ms has been shown when we increase the number of SeDs in the same proportion. The average submission time is 32.3 ms for the first architecture (a) and of 33.5 ms for the second one (b). This results in an increase of 1.2 ms, which is nearly three times less than when the servers are directly added in an existing branch. This simple experiment shows how the requests are processed in parallel when we add servers on an independent branch. Thus, the strategy to add new servers has to be carefully examined. In the next experiment, we test different ways to add new servers on a simple DIET tree and try to figure out some basic rules for establishing a DIET hierarchy.

4.3.2 Trying several architectures to add servers

In this experiment, we only consider trees that have a depth of one LA. These configurations can involve more LAs as long as there is one and only one LA between the MA and an SeD. This allows us to make comparisons between them when the number of SeDs is the same. Figure 12 shows the configurations used in this experiment. The name of each configuration depends on its number of LAs and SeDs (e.g., configuration (1/4) involves one LA and four SeDs). Every time we create a new configuration, we add four servers to an existing configuration. Several configurations can be generated this way from one given configuration. We used six of them during this experiment. The results are:

confi guration	1/4	1/8	2/8	1/12	2/12	3/12
request time (ms)	58.8	69.6	60.9	74.4	62.6	62.9

The average submission time is 58.8 ms for the first configuration. When four new servers directly subscribe to the existing LA (configuration 1/8), this leads to an increase of 10.8 ms. If they

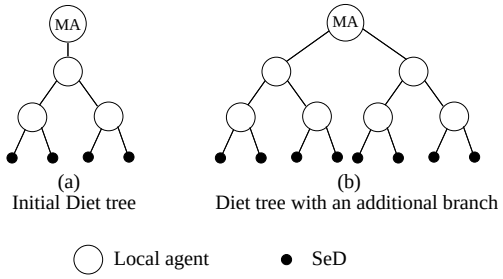


Figure 11: Adding a branch to a DIET tree.

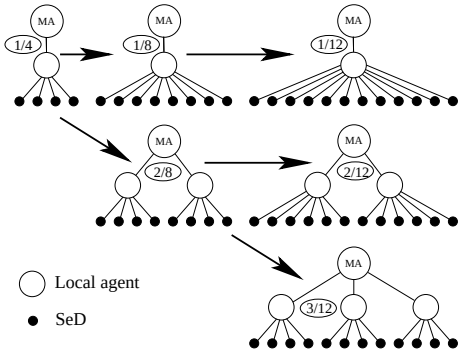


Figure 12: Tested configurations.

subscribe to a new LA (configuration 2/8), the overhead cost is only 2.1 ms. This demonstrates the benefits of using two LAs to send requests simultaneously to several servers.

The last column of Figure 12 shows configurations with twelve SeDs. The first one (1/12) is obtained by adding four more SeDs on the only LA of configuration 1/8. This leads to a request process time of 74.4 ms. The two other configurations with twelve SeDs are obtained with two LAs holding six SeDs each and three LAs holding four SeDs each. These two configurations give similar results. This shows that adding a new LA is only necessary for a certain amount of new SeDs. With twelve SeDs, the time difference between the best and the worst architecture is 11.8 ms. This difference may increase with a higher number of SeDs.

This experiment shows the impact of the architecture on the performance of the system. Obviously, architectures that place LAs in parallel perform better than the others. Tuning such an architecture is a matter of knowledge of the underlying network. Some branches may contain more SeDs than others for various reasons (technical or administrative ones) but these results should be kept in mind when building a DIET tree.

5 Conclusion and future work

In this paper we have presented our view of a scalable Network Enabled Server system. We believe that hierarchy is mandatory when building such environments for the Grid. When thinking about Grid Computing, scalability should be one of the main concerns of developers. We propose a hierarchical approach to Network Enabled Servers using existing software and standards such as NWS, LDAP or Corba.

Our architecture was also validated experimentally, and it seems that performance of the DIET platform is closely linked to the structure of the tree. Thus, a well suited DIET tree can significantly improve the performance of the system, as long as two main constraints are taken into account: the DIET tree should be mapped onto the physical network architecture for faster request broadcasting

and more accurate performance predictions; the use of an LA hierarchy can have an impact on the performance, as soon as a large amount of SeDs is connected, by introducing parallelism in the request processing.

Our future work will first focus on testing this approach on real applications from our project partners. These applications arise from various scientific fields: tridimensional model of Earth ground from two 2D satellite pictures, simulation of electronic components, simulation of the atoms' trajectory in molecular interactions, as well as an application computing the points on a hypersurface of potential energy in quantum chemistry. As one of our target platforms allows 2.5 Gb/s communications between several INRIA research centers in France, connecting several clusters of PCs and parallel machines, such applications written in an RPC mode could benefit from DIET and the whole platform. Thus the powerful computational resources needed for such application can be utilized that could not otherwise be obtained.

After we have implemented a solution to the duplicate choice of a server (see section 3), we would like to tackle the issue of fault tolerance. We are currently specifying the behavior of the system in case of failure. A problem we would also like to address is the optimization of data distribution for parallel library calls using a mixed data and task parallel approach. We also would like to connect our developments to infrastructure toolkits like Globus to benefit from their development as regards security, accounting, and interoperability of services.

Acknowledgements

This work was supported in part by the projects ACI GRID–GRID ASP and RNTL GASP funded by the French department of research.

References

[1] J.-L. Anthoine, P. Chatonnay, D. Laiymani, J.-M. Nicod, and L. Philippe. Parallel Numerical Computing Using Corba. In H. R. Arabnia, editor, *Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*, volume III, pages 1221 – 1228. CSREA Press, 1998.

[2] P. Arbenz, W. Gander, and J. Mori. The Remote Computational System. *Parallel Computing*, 23(10):1421–1428, 1997.

[3] 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.

[4] R.F. Boisvert. The Architecture of an Intelligent Virtual Mathematical Software Repository. *Mathematics and Computers in Simulation*, 36:269–279, 1994.

[5] 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.

[6] E. Caron and F. Suter. Parallel Extension of a Dynamic Performance Forecasting Tool. In *Proc. of the International Symposium on Parallel and Distributed Computing*, Iasi, Romania, 2002.

[7] A. Denis, C. Pérez, and T. Priol. Towards high performance CORBA and MPI middlewares for grid computing. In Craig Lee, editor, *Proc. of the 2nd Int. Workshop on Grid Computing*, number 2242 in LNCS, pages 14–25, Denver, 2001. Springer-Verlag.

[8] M. Quinson F. Desprez and F. Suter. Dynamic Performance Forecasting for Network Enabled Servers in a Metacomputing Environment. In *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2001)*. CSREA Press, 2001.

[9] M.C. Ferris, M.P. Mesnier, and J.J. Moré. NEOS and Condor: Solving Optimization Problems Over the Internet. *ACM Transaction on Mathematical Sofware*, 26(1):1–18, 2000.

[10] I. Foster and C. Kesselman (Eds.). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1998.

[11] E. N. Houstis and J. R. Rice. On the future of problem solving environments. `http://www.cs.purdue.edu/homes/jrr/pubs/kozo.pdf`, 2000.

[12] I. A. Howes, M. C. Smith, and G. S. Good. *Understanding and deploying LDAP directory services*. Macmillian Technical Publishing, 1999.

[13] S. Matsuoka and H. Casanova. Network-Enabled Server Systems and the Computational Grid. `http://www.eece.unm.edu/~dbader/grid/WhitePapers/GF4-WG3-NES-whitepaper%-draft-000705.pdf`, July 2000. Grid Forum, Advanced Programming Models Working Group whitepaper (draft).

[14] 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.

[15] 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.

[16] M. 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)*, 2002.

[17] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computing Systems, Metacomputing Issue*, 15(5–6):757–768, Oct. 1999.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifi que,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

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