Dynamic Performance Forecasting for Network-Enabled Servers in a Heterogeneous Environment

Frédéric Desprez, Martin Quinson, Frédéric Suter

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Dynamic Performance Forecasting for Network-Enabled Servers in a Heterogeneous Environment

F. Desprez, INRIA
M. Quinson, LIP
F. Suter, LIP

No 4320

Novembre 2001

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4320--FR+ENG

INRIA
RHÔNE-ALPES

Dynamic Performance Forecasting for Network-Enabled Servers in a Heterogeneous Environment

F. Desprez, INRIA
M. Quinson, LIP
F. Suter, LIP

Thème 1 — Réseaux et systèmes
Projet ReMaP

Rapport de recherche n ° 4320 — Novembre 2001 — 11 pages

Abstract: This paper presents a tool for dynamic forecasting of Network-Enabled Servers performance. FAST (*Fast Agent's System Timer*) is a software package allowing client applications to get an accurate forecast of communication and computation times and memory use in a heterogeneous environment. It relies on low level software packages, i.e., network and host monitoring tools, and some of our developments in computation routines modeling. The FAST internals and user interface are presented and a comparison between the execution time predicted by FAST and the measured time of complex matrix multiplication executed on an heterogeneous platform is given.

Key-words: Performance forecasting, Computation modeling, Resources monitoring, heterogeneous environments, Network-Enabled Servers.

(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`).

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN (France)
Téléphone : 04 76 61 52 00 - International: +33 4 76 61 52 00
Télécopie : 04 76 61 52 52 - International: +33 4 76 61 52 52

Prédiction dynamique de performance pour des serveurs de calcul dans un environement de metacomputing

Résumé : Cet article présente un outil pour la prédiction dynamique de performances dans un environnement de type *Client-Agent-Serveurs*. La bibliothèque FAST (*Fast Agent's System Timer*) permet aux applications clientes de prédire les besoins en temps d'exécution et de communication, ainsi qu'en espace mémoire de sous-programmes dans un environnement hétérogène. Elle est basée sur des logiciels existants de plus bas niveau pour la surveillance du réseau et des machines et sur des développements propres portant sur la modélisation des routines. Le fonctionnement du système et son interface sont présentés, ainsi qu'une comparaison entre le temps prédit par FAST et le temps mesuré pour une multiplication de matrices complexes exécutée sur une plate-forme hétérogène.

Mots-clé : Prédiction de performances, Modélisation de calcul, Surveillance de ressources, Environement hétérogène, Client-Agent-Serveurs.

1 Introduction

One of the main goals of computer systems is to provide more computational power to scientists. To achieve this goal, after trying to design bigger and bigger computers (until 9000 Pentiums for one of the ASCI machines), we now aggregate smaller machines into clusters. A new trend, called metacomputing (or grid computing) [4, 5, 9], consists in trying to use world wide available computers with more or less transparency. This is the promise of a huge computational power because most of interactive machines are under-used. Moreover, computations could be performed on the most appropriate computer. More ambitious projects aim at building a system considering the whole Internet as a metacomputer. But such solutions are difficult to achieve while keeping a good efficiency. Users would be able to use computational power distributed across the world as easily as electrical power to form the *Computational Grid*. In [9], Foster and Kesselman give a classification of grid applications. In this paper, we will focus on *On-demand Computing* applications, and more particularly on numerical Network-Enabled Servers (NES). These tools allow users to automatically (and sometimes transparently) access remote hardware and software resources [7, 8, 12] through computational servers. One client submits a problem to an agent which has information about each server. The agent is supposed to return the address of the most appropriate server to the client, according to parameters such as its capacity to solve the problem, both in terms of performance and software availability, and the speed of the communication link between the client and the server. Then the client sends its data to the server which in turn computes and sends back the result to the client. The agent is thus one the most important entities because it is where all scheduling and placement decisions are taken.

Collecting knowledge about available servers at runtime is a very important problem. Indeed, in a metacomputing context, we deal with hierarchical networks of heterogeneous computers which furthermore have hierarchical memories. Even if software packages exist to acquire the informations needed about both computer and network using monitoring techniques [14], they often use a flat and homogeneous view of the system, which is clearly not the case of our heterogeneous target platforms. So we have to model computation routines in relation to the computer which will execute them.

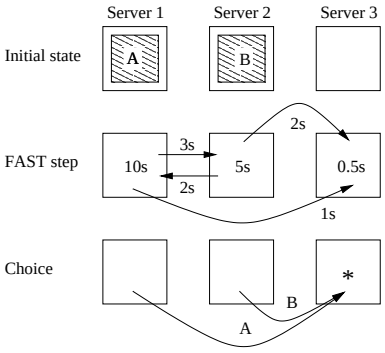
In this paper, we describe *FAST*, a Fast Agent's System Timer, which is designed to handle these important issues. It allows an agent (or every other software entity that needs this kind of information) to get informations about heterogeneous servers, networks, and computational routines. In Section 2, we present more precisely the problem by giving two motivating examples. In Section 3, we give an overview of existing software packages for performance forecasting. In Section 4, the architecture and internals of *FAST* are described, and before concluding, experimental results are given in Section 5.

2 Motivating Examples and Problem Presentation

The main problem in heterogeneous *Client-Agent-Servers* systems is a scheduling one. We need to find the most appropriate server for a set of computations and the most appropriate mapping according to communication links. In this section, we present two motivating examples to illustrate these issues. We consider the average completion time as a quality criterion.

We assume that we can not easily aggregate several servers to solve one problem, which is generally the case for Network-Enabled Servers. Let consider three geographically distant servers, and two matrices A and B. Because of previous computations, A is located on the first server and B on the second one. Now if we want to compute $A * B$, there are three solutions. We can move B to server 1 and then compute, move A to server 2 and compute, or finally move A and B to server 3 and compute. If the communication and computation times are those given in Figure 1(top), the final execution costs are the ones shown in Figure 1(bottom). We can see that even if the third solution is not the most natural one, this is the one that will lead to the faster execution.

The quality of the schedule, and thus the performance of the whole system, depends on the accuracy of the communication and computation times forecasts. They depend on several parameters, which



$$
\begin{aligned}
T_1 &= T_{B\to 1} + T_{comp}(1) \\
&= 2s + 10s \\
&= 12s \\
T_2 &= T_{A\to 2} + T_{comp}(2) \\
&= 3s + 5s \\
&= 8s \\
T_3 &= T_{A\to 3} + T_{B\to 3} + T_{comp}(3) \\
&= 1s + 2s + 0.5s \\
&= 3.5s
\end{aligned}
$$

Figure 1: Motivating example.

can be splitted into three main categories. First, we have the computer parameters such as CPU speed, memory size, and use of a batch system. Then, we have the network parameters, like topology, bandwidth and latency for every link. For all these parameters, both static (theoretical) and dynamic (measured) values have to be obtained. Finally, there are computational parameters relative to a given problem which depend on the architecture of the target machine, the size of the input data and the software package used to solve the problem.

For the second example, the heterogeneous platform considered is composed of a client connected to two servers. If we want compute a complex matrix multiplication, it is important to have a good knowledge of communication times to move data across the platform. Indeed, if the link between client and servers is fast (resp. slow) and the one between servers is slow (resp. fast), we have to minimize data transfers on the slowest link.

Figure 2(a) gives the formulation of the computation that we want to execute. One server will compute C_r and the other will compute C_i. All data are initially located on the client. To execute this computation, we have two options, shown respectivelly in Figure 2(b) and (c). Either we send both matrices A and B to servers, or we send only one matrix to each server, and then, servers exchange data to continue the computation. In both cases, results are sent back to the client.

The purpose of the *FAST* library is to allow the user to access as fast as possible the most accurate estimations of these parameters to obtain better scheduling results.

3 Related Work

3.1 Network Weather Service (NWS)

The Network Weather Service (NWS) [14] is a project initiated at the University of California San Diego and now located at the University of Tennessee Knoxville. It is a distributed system that periodically monitors and dynamically forecast performance of various network and computational resources. It is used by many grid projects like AppLeS [3], Globus [10], NetSolve [7], and Ninf [12]. NWS can

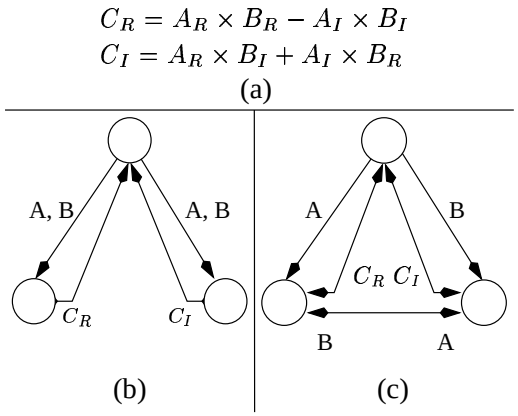


Figure 2: Complex matrix multiplication formulation (a). Computation without communication between servers (b). Computation with data exchange between servers (c).

monitor several resources including communications links, CPU, and memory. For any communication link between two points, NWS gives the bandwidth and latency. Concerning the CPU availability, NWS gives two kind of information: calls to `top` or `vmstat` commands give the CPU load implied by the existing jobs, and NWS is also able to guess the percentage of the CPU power that can be expected by an incoming process at given priority level. Moreover, NWS is not only able to get measurements, it can also forecast the evolution of the monitored system, using basic statistic functions like average or median.

3.2 Bricks and Ninf

Bricks [1] is a tool developed to simulate a computational grid, using tools like NWS, to compare scheduling algorithms and software. Bricks has been developed for Ninf [12] which is a Network-Enabled Servers environment. Ninf uses Bricks for the evaluation of performance of grid based applications. This work is close to our developments, because the simulation of the grid implies to have a model of it. In this model, everything is seen as a queue. A communication link between two hosts is a queue which processing speed is the bandwidth. A server is a queue where incoming tasks are enqueued and processed in a FCFS way. A task is modeled by two values: the number of operations needed, and the amount of data transfer it implies.

We see two main problems to this approach. First, it lacks of accuracy because the amount of flops that can be obtained from the same machine is not constant. For example, on modern pipelined processors, level-3 BLAS are more efficient than level-2 or level-1 BLAS because of memory hierarchies and dual operations. Then, such a tool needs a lot of computation at runtime and does not seem well suited for an interactive tool.

3.3 Remote Computation Service (RCS)

RCS [2] is a RPC system to unify the interface to remote resources. It has its own NWS-like monitor, communication layer and modeling methods. But the monitor lacks some important features like forecasting and non-intrusive CPU monitoring. The communication layers are implemented on top of PVM.

The model of routines provided by RCS is based on source code analysis to determine the computation complexity of each routine. For example, the result the LAPACK *LU* factorization function is $\frac{2}{3}n^3 - \frac{1}{2}n^2 + \frac{5}{6}n$ where n is the size of the matrix. But this is only the theoretical complexity of the LAPACK *LU* routine. Even if this formulation is asymptotically true, it does not take the com-

puter architecture into account, therefore loosing accuracy for medium range values. Indeed, between two versions of the same routine, a generic and a vendor optimized one, performance is affected by a significant multiplicative factor.

4 *FAST* Architecture and Internals

In this section, we detail the different parts of *FAST*. Then, we present the interface between *FAST* and client applications.

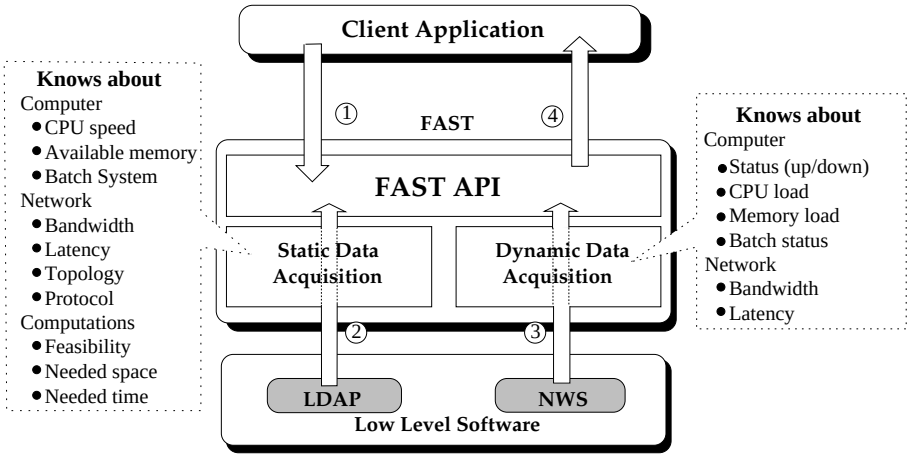


Figure 3: Overview of the *FAST* architecture.

As shown in Figure 3, *FAST* is composed of two main modules offering an user API: the *static data acquisition* module and the *dynamic data acquisition* module. The former models the time and space needs of a computation on a given machine for a given set of parameters, while the later forecast the behavior of dynamically changing resources, e.g., workload or bandwidth. The system relies on low level software packages. First, a database system is used to store the static data. The Lightweight Directory Access Protocol (LDAP) [11] was chosen because it can be distributed over the system and for its performance for reading or searching data. Furthermore, LDAP is widely used in the grid community. Then, the dynamic data are acquired from a network and host monitoring software. As NWS can monitor both network and host, we chose it for the beta version of our tool. But, *FAST* is not tightly linked to these two software packages. If they are not available, *FAST* could include mechanisms and interfaces to acquire dynamic data and store static data from and to more basic tools. This would imply a lost of accuracy but a gain of portability.

In the next two sections, we present the main modules of *FAST*. We also give an idea of what kind of routines *FAST* uses to interact with low level software packages.

4.1 Static Data Acquisition

This module is one of the main contributions of *FAST*. Its goal is to predict the time and space needs of routines. We see several approaches, depending on the type of routine that has to be evaluated.

Some routines can be easily measured. This kind of routines are usually sequential and highly optimized for the processor, as the matrix-matrix product routine `dgemm()` from the level 3 BLAS library. Each computer hardware vendor ships this library optimized for its material or at least automatically tuned version are available like ATLAS [13]. This is why it is not always possible to study the source code to compute manually its time and space complexity. In this case, we benchmark the routine in time and space, fit the resulting data by a polynomial regression, and store the result in

a LDAP tree. This phase can be time-consuming, but it has only to be done once when a new server registers to an agent. Furthermore, it is of course possible to share the results between all identical machines or clusters.

Some other routines can not be evaluated experimentaly as easily, but are simpler to study. For example, the parallel matrix-matrix product routine `pdgemm` from the ScaLAPACK library is based on the concurrent execution of `dgemm` on sub-matrices. In [6], the authors show that timing of such functions are harder to evaluate because many parameters have to be taken into account (such as data distribution or the shape of the processor grid). For these routines, *FAST* allows to specify the computation time by a complex expression that aggregates the computation time of other routines (obtained with the first method) to communication time of the parallel version of the algorithm given by a careful evaluation of the algorithms.

Finally, there is a last kind of routines that are much difficult to deal with because their performance depend on characteristics which are hard to extract (like the shape of the matrices involved or the values of their elements). It is often the case of sparse matrix operations. The processing of such routines in *FAST* is an open problem, and will need further work. Two approaches seem possible. First, we can simulate the resolution of the problem without actually compute it, or of course we can choose the most powerful machine that can solve the problem without trying to predict its execution time.

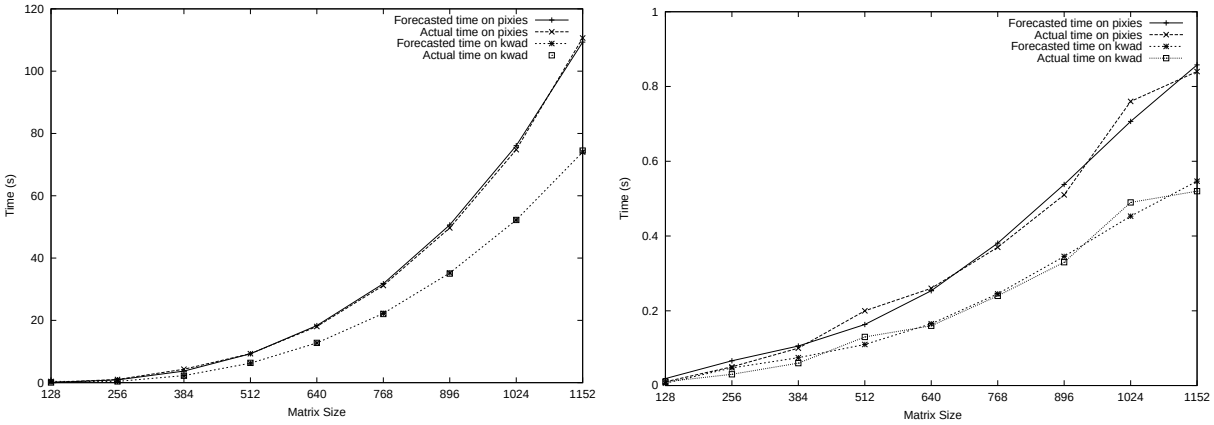
4.2 Dynamic Data Acquisition

This module can acquire measured values at runtime. We developed a set of routines based on NWS, even if other similar libraries could also be used. For every pair of machines in the system, we are able to report the bandwidth and the latency, and for each computer, we can give the workload and the free available memory at a given time. This module allows *FAST* to get the network values between two machines even if there is no direct NWS monitoring between them. *FAST* searches automatically for the shortest path between these two machines in the graph of monitored links. In this case, the estimated bandwidth is the minimum of those in the given path. For the latency, we take the sum of the different latencies.

Queries to NWS can be time consuming because the client requests are sent to the forecaster. This component contact the name server to locate the memory server. Then, it can ask for the data needed to this memory server, compute its forecast and send it back to *FAST*. For example, as there is only one active test every two minutes on a TCP link, it is useless to send the request to NWS each time. That's why we implemented a cache of queries to NWS to speed up the response time of *FAST*. If a given query has been sent twice to *FAST* in less that 10 seconds, the data in the cache are sent back to the client. Ten seconds is the default interval between two CPU tests.

4.3 Interface between *FAST* and Client Applications

We designed a very simple API to combine the static and dynamic data acquired by the system and to produce values which can be directly used by a client application. Indeed, only a few values are needed in a scheduling context: the time needed for the computation on a given host, if its space needs can be fulfilled on this host, and the time needed to transfer the data from their location to the host on which the computation will be done. This is the reason why there are only four functions in the API. First, the `fast_comm_time(data_desc, source, dest)` function gives the communication time to transfer the data described by `data_desc` from the host `source` to the host `dest`, taking the current network characteristics into account. Then, the `fast_comp_time(host, problem, data_desc)` function gives an estimation of the time needed to execute the routine `problem` on the given `host` for the parameters `data_desc`, taking into account both the theoretical performance of this host for this problem, and the actual load of the host. The `fast_comp_size(host, problem, data_desc)` function gives, for the same arguments, the memory space needed by the routine. Finally, the `fast_get_time(host,`



(a) dgemm (b) addition

Figure 4: Comparison between *FAST* forecasts and actual computation time for one operation.

`problem, data_desc, localizations)` function aggregates the results of others functions and forecast the time needed to execute the given `problem` on `host` for the data described by `data_desc`, taking into account the prior `localizations` of data and the time to get them on `host`. This function returns -1 if the computation is impossible, for example if there is not enough memory available.

With this API, the pseudo-code corresponding to the example of Figure 1 would be to choose the server s minimizing the expression:

`get_time(`s`,"MatMult",{{"A",dense,M,N}, {"B",dense,N,K}},{"`s_1`","`s_2`"})`

where s can be either s_1, s_2, or s_3.

5 Experimental Results

To validate our approach, we conducted experiments to evaluate the accuracy of *FAST* forecasts. We wanted to show if *FAST* was able to forecast accurately the time and space need of basic matrix operations and if it was possible to combine these forecasted values to estimate the time need of a sequence of operations.

As we wanted to evaluate *FAST*'s accuracy and not its impact on scheduling performances, the test bed used was heterogeneous, but relatively simple. We used the two machines: *Pixies*, a PII desktop computer with 128 Mb of memory, and *Kwad*, a server doted of 4 PIII processors, each with one Mb of cache, and 256 Mb of memory for the system.

5.1 Quality of the data fitting

For each host, we used *FAST* to forecast the time and space needs of some matrix operations and we compared the obtained values to the measured ones (not taking in account the actual load of the resources). The used operations where the matrix multiplication using the `dgemm` function of the BLAS library and matrix addition using a simple double loop.

Figure 4(a) shows the time values on both hosts for `dgemm` while Figure 4(b) shows the result for the plus operation. *FAST* used a polynomial function of maximal order 4 in the first case and 6 in the second case. In all cases, the error is negligible. The error is about 1% in most case, and bigger when the estimated times becomes too small (less than one second). In this case, the relative error is very big (up to 700%), but the absolute error is then smaller than 0.1 second. Given the coarse grain of target applications, this is not seen as critical. Theses errors when the measured times are very

small seem to come from the use of the `getrusage(2)` function to measure CPU timing, which have a resolution of 0.01 second.

Concerning space needs, *FAST* succeeded to find a polynomial function estimating almost perfectly the data (maximal error smaller than 0.1%). This is because the size used by a program performing a matrix multiplication is a constant term for its code size, plus the size of allocated matrices, which is obviously a polynomial function.

5.2 Using *FAST* for a sequence of operations

On Client:

- Send A_r, A_i and B_r to Server 1 ;
- Send A_r, A_i and B_i to Server 2.

On Server 1:

- $C_{r_1} = A_r \times B_r$;
- $C_{i_2} = A_i \times B_r$;
- Send C_{i_2} to Server 2 ;
- $C_r = C_{r_1} - C_{r_2}$;
- Send C_r to the Client.

On Server 2:

- $C_{r_2} = A_i \times B_i$;
- $C_{i_1} = A_r \times B_i$;
- Send C_{r_2} to Server 1 ;
- $C_i = C_{i_1} + C_{i_2}$;
- Send C_i to the Client.

Figure 5: Complex matrices multiplication algorithm.

Then, we evaluated the ability of *FAST* to forecast the needs of a sequence of operations which could possibly be done in parallel, taking in account the load of the resources. The example chosen is the complex matrices multiplication because it can be easily done in parallel on two hosts (as shown on Figure 2), and because it combines basic matrix operations such as multiplication and addition.

The test-bed was composed of the same two machines *Pixies* and *Kwad*. The LDAP server, the NWS name, memory and forecast servers and the client were all located on *Pixies* while the two agents were on separate nodes of *Kwad*.

We implemented the complex matrices multiplication following the algorithm described in Figure 5, and compared the observed execution time to the combination of *FAST* forecast of each individual operation composing the sequence.

Figure 6 shows the result of this experiment. Even if the system had to deal with the errors of both static and dynamic data acquisition, cumulated on each operation, *FAST* managed to forecast the time of the sequence with an accuracy of 23% in the worst case and about 10% in average, which is a reasonable result given the facts that the operation is composed of several steps and the platform is heterogeneous.

6 Conclusion and Future Work

In this paper, we presented *FAST*, a tool for performance forecasting in the scope of *Client-Agent-Servers* systems. After giving a motivating example of the need of accurate models of both communications and computations and the related issues, we have presented some related work in this domain. We then presented the internals of *FAST*, and the software packages it relies on (like NWS) to monitor host and network resources. We also developed a benchmarking tool coupled with an data fitting method to accurately model computation routines. We presented some experiments, showing that *FAST* models very accurately simple operations, and that time of operation sequences can be

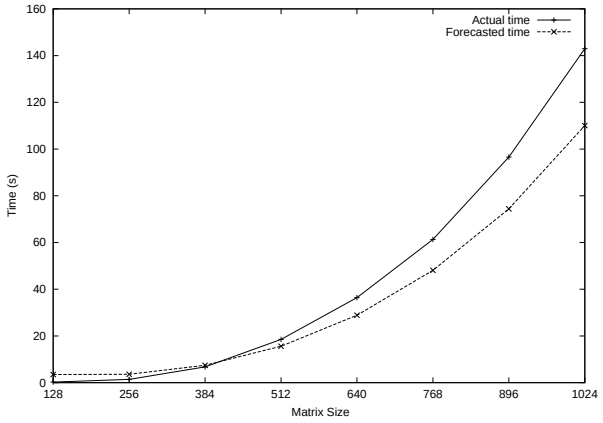


Figure 6: Comparison between *FAST* forecasts and actual computation time for a complex matrix multiplication.

forecasted with an error lower than 23% in the worst case and about 10% in average by combining the obtained values.

In the current version of *FAST*, only the matrix multiplication and addition operation were tested. So as a first future work, we would like to model other numerical operations such as other BLAS or LAPACK routines and their parallel counterparts. We also plan to develop other *FAST* client applications into numerical interactive environments such as Matlab or Scilab. Then, we would like to give the ability to *FAST* to use other networks than TCP, like myrinet or VIA. Finally, we have contacts with the NWS and NetSolve teams to achieve a better integration of our programs. We are of course interested by other grid applications that need accurate performance estimations.

References

[1] K. Aida, A. Takefusa, H. Nakada, S. Matsuoka, S. Sekiguchi, and U. Nagashima. Performance evaluation model for scheduling in a global computing system. *International Journal of High-Performance Computing Applications*, (to appear).

[2] P. Arbenz, W. Gander, and M. Oettli. The Remote Computation System. In H. Liddell, A. Colbrook, B. Hertzberger, and P. Sloot, editors, *High Performance Computing and Networking*, volume 1067 of *Lecture Notes in Computer Science*, pages 820–825. Springer-Verlag, Berlin, 1996.

[3] F. Berman and R. Wolski. The AppLeS Project: A Status Report. In *Proceedings of the 8th NEC Research Symposium*, Berlin, Germany, May 1997.

[4] R. Buyya, editor. *High Performance Cluster Computing, Vol. 1: Architectures and Systems.* Prentice Hall, 1999.

[5] R. Buyya, editor. *High Performance Cluster Computing, Vol. 2: Programming and Applications.* Prentice Hall, 1999.

[6] E. Caron, D. Lazure, and G. Utard. Performance Prediction and Analysis of Parallel Out-of-Core Matrix Factorization. In *Proceedings of the 7th International Conference on High Performance Computing (HiPC'00)*, Dec 2000.

[7] H. Casanova and J. Dongarra. Using Agent-Based Software for Scientific Computing in the NetSolve System. *Parallel Computing*, 24:1777–1790, 1998.

[8] M. Ferris, M. Mesnier, and J. Moré. NEOS and Condor: Solving Optimization Problems Over the Internet. *ACM Transactions on Mathematical Software*, 26(1):1–18, March 2000.

[9] I. Foster and C. Kesselman (Eds.). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[10] I. Foster and C. Kesselman. The Globus Project: A Status Report. In *Proceedings of the Heterogeneous Computing Workshop*, pages 4–18, 1998.

[11] I. A. Howes, M. C. Smith, and G. S. Good. *Understanding and deploying LDAP directory services*. Macmillian Technical Publishing, 1999. ISBN: 1-57870-070-1.

[12] 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, Oct. 1999.

[13] R. Whaley and J. Dongarra. LAPACK Working Note 131: Automatically Tuned Linear Algebra Software. Technical Report CS-97-366, The University of Tennessee, 1997.

[14] 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 scientifique,
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