Deadline Scheduling with Priority for Client-Server Systems on the Grid

Eddy Caron, Pushpinder Kaur Chouhan, Frédéric Desprez

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Deadline Scheduling with Priority for Client-Server Systems on the Grid

Eddy Caron, Pushpinder Kaur Chouhan, Frederic Desprez

N° 5335

Octobre 2004

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--5335--FR+ENG

Deadline Scheduling with Priority for Client-Server Systems on the Grid

Eddy Caron, Pushpinder Kaur Chouhan, Frederic Desprez

Thème 1 — Réseaux et systèmes
Projet ReMaP

Rapport de recherche n° 5335 — Octobre 2004 — 13 pages

Abstract: We present algorithms for the scheduling sequential tasks on a Network Enabled Server (NES) environment. We have implemented the non-preemptive scheduling, since at the user level we cannot interrupt a running process and block it in order to allow a new process to run. This article is an extension of the paper: "A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid" by Takefusa et al. We mainly discuss a deadline scheduling with priority strategy that is more appropriate for multi-client, multi-server case. Importance is first given to the task's priority and then the task is allocated to the server that can meet the task's deadline. This may cause that some already allocated tasks on the server miss their deadline. We augment the benefits of scheduling algorithms with load measurements (which is done with the use of a forecasting tool called FAST) and fallback mechanisms. The experimental results shows that the deadline scheduling with priority along with fallback mechanism can increase the overall number of tasks executed by the NES.

Key-words: Scheduling, Problem Solving Environment

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

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

Ordonnancement à échéance avec priorité pour les systèmes client-serveur sur la Grille

Résumé : Dans ce rapport, nous proposons des algorithmes pour l'ordonnancement de tâches séquentielles dans un environnement distribué de type NES (Network Enabled Server). Nous nous plaçons dans un cas non préemtif, il est donc impossible d'interrompre un processus en cours d'exécution pour en exécuter un autre. Ce rapport est une extension à l'article "A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid" de Takefusa et al. une approche similaire à laquelle nous ajoutons un mécanisme de prédiction de performances. De plus, nous proposons d'inclure la notion de priorité qui est appropriée dans le cas des NES. L'ordonnanceur tient alors compte de la priorité de la tâche dans un premier temps, puis on vérifie son échéance (date à laquelle on souhaite que la tâche soit finie). Le respect de la priorité à un impact sur le respect des échéances, ce qui nous conduit à appliquer des mécanismes de *fallback*. Notons que nos algorithmes seront évalués par simulations.

Mots-clés : Ordonnancement, Environnement de résolution de problèmes

1 Introduction

The Grid provides a solution to compute huge applications over the Internet [7]. Several approaches co-exist like object-oriented languages, message passing environments, infrastructure toolkits, Web-based, and global computing environments. The Remote Procedure Call (RPC) [9] paradigm seems also to be a good candidate to build Problem Solving Environments (PSE) for different applications on the Grid. Indeed, it allows one to hide complicated resource allocation and job distribution strategies from the user. Moreover, parallelism can be hidden using parallel servers. Several environments following this approach exist, like NetSolve [1], Ninf [10] or DIET [2]. They are commonly called Network Enabled Server (NES) environments [9].

Currently, NES systems use the Minimum Completion Time (MCT) on-line scheduling algorithm. The deadline scheduling model uses the expected response time of each execution as a scheduling parameter called a "deadline". We assume that each task, when it arrives in the scheduler, has a given static deadline (possibly given by the client). If a task completes its execution before its chosen relative deadline, it meets the deadline. Otherwise the tasks fails.

We divide our contribution in two parts. The main idea of the first part is to introduce the mechanism of performance prediction for tasks. The aim of the second part is to give an expanded version of deadline scheduling with priority mechanism. The presented scheduling algorithms aim at scheduling sequential tasks on a NES environment. This kind of environment is usually composed of an agent receiving tasks and finding the most efficient server that will be able to execute a given task on behalf of the client.

2 Related Work

While a large number of papers describe priority algorithms for classical operating systems, little research exists around scheduling algorithms using priority and deadline for grid applications.

Scheduling problems that combine tails and deadlines is discussed in [12]. Lower bounds are given for the shop scheduling problems and an algorithm is presented with an improved complexity to solve two parallel machine problem with unit execution time operations. An algorithm in [14] finds minimum-lateness schedules for different classes of DAGS when each task has to be executed in a non-uniform interval. Both papers present interesting theoretical results that can not be directly implemented in a grid platform due to some limitations of models.

A deadline scheduling strategy is given in [13] for multi-client multi-server case on a grid platform. The authors assume that each task (tasks sent to the scheduling agent) receive a deadline from the client. The algorithm presented aims at minimizing deadline misses. It is also augmented with "Load Correction" and "fallback" mechanisms to improve its performance. The first optimization is done by taking into account load changes (due to previous scheduling decisions) as soon as possible. The fallback mechanisms makes some corrections to the schedule at the server level. If the server finds that a task will not meet the deadline due to prediction errors, the task is re-submitted to the system. Simulation has been provided using the Bricks simulation framework. A background load is simulated and real traces are injected in the model to simulate the extra load of the system. Optimized algorithms are compared to a simple greedy algorithm that does not take deadlines into account. This algorithm is indeed less efficient than the optimized ones. The simulation shows that while the load correction mechanism does not improve the overall execution time significantly, the fallback optimization leads to important reductions of the failure rates without increasing the cost of scheduling.

3 FAST's Overview

Our first contribution consist to do performance prediction using a forecasting tool Fast Agent's System Timer (FAST) [5, 11]. FAST is a dynamic performance forecasting tool for Grid environments designed to handle these important issues. Information acquired by FAST concerns sequential computation routines and their execution platforms. For example it can be useful to a scheduler to optimize task mapping.

As shown in Figure 1, FAST is composed of two mainmodules offering a user API: the *static data acquisition* module and the *dynamic data acquisition* module. The former model forecasts the time and space needs of a sequential computation routine on a given machine for a given set of parameters, while the latter forecasts the behavior of dynamically changing resources, *e.g.,* workload, bandwidth or memory use. FAST relies on low–level software packages. First, LDAP [8] is used to store static data. Then to acquire dynamic data. FAST is essentially based on the NWS (Network Weather Service) [16], a project initiated at the University of California San Diego and now being developed at the University of Santa Barbara. It is a distributed system that periodically monitors and dynamically forecasts performance of various network and computational resources. NWS can monitor several resources including communications links, CPU, disk space, and memory. Moreover, NWS is not only able to obtain measurements, it can also forecast the evolution of the monitored system.

FAST extends NWS as it allows to determine theoretical needs of computation routines in terms of execution time and memory. The current version of FAST only handles regular sequential routines like those of the dense linear algebra library BLAS [6]. However BLAS kernels represent the heart of many applications, especially in numerical simulation. The approach chosen by FAST to forecast execution times of such routines is an extensive benchmark followed by a polynomial regression. Indeed this kind of routines is often strongly optimized with regard to the platform, either by vendors or by automatic code generation [15]. FAST is also able to forecast the use of parallel routines [3]. However, only the sequential part of FAST will be used in this article.

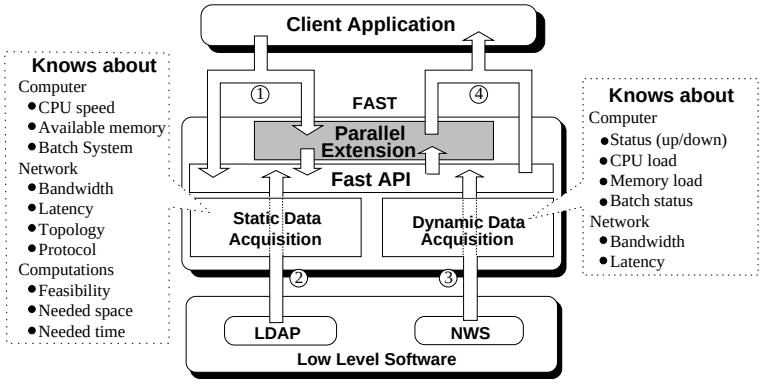


Figure 1: Overview of the FAST architecture.

In this paper FAST and NWS will be used to give the availability of CPU and network, and to know the execution time of a specific task on a specific server.

4 Client-Server Scheduling algorithms

4.1 Client-Server Scheduler with load measurements

As we target heterogeneous architectures, each task can have a different execution time on each server. Let T_{aS_i} be the execution time for the task a on server i. This time includes the time to send the data, the time to receive the result of the computation and the execution time:

$$T_{aS_i} = \frac{W_{send}}{P_{send}} + \frac{W_{recv}}{P_{recv}} + \frac{W_{aS_i}}{P_S}$$

where W_{send} is the size of the data transmitted from the client to the server, W_{recv} denotes to the data size transmitted from the server to the client, W_{aS_i} is the number of floating point operations of the task, P_{send} denotes the predicted network throughputs from the client to the server, P_{recv} denotes the predicted network throughputs from the server to the client and P_S is the server performance (in floating point operations per second).

P_{send} should be replaced by the network throughput value measured just before the task. This value is returned by one call to FAST (cf. Section 3). P_{recv} is estimated using previous measurements. The CPU performance is also dynamic and depends on other tasks running on the target processors. Thus, FAST can be used to provide a forecast of CPU performance, so as to take into account the actual CPU workload.

Algorithm 1 Straightforward algorithm: Client-Server Scheduler with load measurements.

repeat
  for all server S_i **do**
    if can_do(S_i, T_a) **then**
      $T_{aS_i} = \frac{W_{send}}{F_{Bd}} + \frac{W_{recv}}{P_{recv}} + \frac{W_{aS_i}}{F_{S_i}}$
      List=sort_insert(List,T_{aS_i}, T_a, S_i)
    end if
  end for
  num_submit=task_ack($List, \frac{2 \times W_{send}}{F_{Bd}}$)
  task_submit($List[num\_submit]$)
until the end

Algorithm 1 gives a straightforward algorithm to get a sorted list of servers that are able to compute the client's task. It assumes that the client takes the first available server, which is most efficient, from the list. However, a loop can be added at the end of the algorithm between the task_ack and task_submit calls.

For sake of simplicity we define four functions for Algorithm 1:

`can_do` This function returns `true` if server S_i have the resource required to compute task T_a. This function takes into account the availability of memory and disk storage, the computational library etc.

`sort_insert` This function sorts servers by efficiency. As an input, we have the current $List$ of servers, the time predicted T_{aS_i}, the task name T_a and the server name S_i. Its output is the $List$ of ordered servers.

`task_ack` This function sends the data and the computation task. To avoid a dead-lock due to network problems, the function chooses the next server in the list if the time

to send the data is greater than the time given in the second parameter. The output of this function is the server where the data are sent (index number in the array $List$).

`task_submit` This function performs the remote execution on the server given by `task_ack`.

4.2 Client-Server Scheduler with a Forecast Correction Mechanism

The previous algorithm assumes that FAST always returns an accurate forecast. In this section, we take into account the gap between the performance prediction and the actual execution time of each task.

Algorithm 2 Scheduling Algorithm with forecast correction mechanism.

$CorrecFAST = 100$
$nb\_exec = 0$
for all server S_i **do**
  if can_do(S_i, T_a) **then**
    $T_{aS_i} = \frac{W_{send}}{F_{Bd}} + \frac{W_{recv}}{P_{recv}} + \frac{W_{aS_i}}{F_{S_i}}$
    $T_{aS_i} = \frac{T_{aS_i} \times CorrecFAST}{100}$
    $List$=sort_insert($List, T_{aS_i}, T_a, S_i$)
  end if
end for
num_submit=task_ack($List, \frac{2 \times W_{send}}{F_{Bd}}$)
T_r=task_submit($List[num\_submit]$)
$CorrecFAST = \frac{nb\_exec \times CorrecFAST + \frac{100 \times T_r}{T_{aS_i}}}{nb\_exec+1}$
nb_exec++

The function `task_submit` is upgraded to return the time of the remote execution. Thus, we can modify the next predictions from the monitoring system as follows to obtain the corrected load values:

$$T_{aS_i} = \frac{T_{aS_i} \times CorrecFAST}{100}$$

where $CorrecFAST$ is an error average between the prediction time and the actual execution time. This value is updated at each execution as follows:

$$CorrecFAST = \frac{nb\_exec \times CorrecFAST + \frac{100 \times T_r}{T_{aS_i}}}{nb\_exec + 1}$$

where T_r is the actual execution time and T_{aS_i} the time predicted. Algorithm 2 includes this correction mechanism.

4.3 Client-Server Scheduler with a Priority Mechanism

Until now, for client-server system, either the deadline scheduling or the priority based scheduling are considered. Here we give an algorithm that utilizes both criteria to select a server for the task.

In Algorithm 3, tasks have a predefined priority and deadline. A task is allocated on the server task queue according to its priority. If a task can meet its deadline then it is sent to the server for execution. For Algorithm 3, we define some new variables. TD_a is the deadline and TP_a is the priority of task T_a. TF_{aS_i} is the changed execution time of task T_a on server S_i after placing it on the server task queue. To simplify the explanation of the algorithm, we define five functions:

Algorithm 3 Client-Server Scheduler with priority mechanism

```
repeat
  for all server S_i do
    if can_do(S_i, T_a) then
      T_aSi = W_send/F_Bd + W_recv/P_recv + W_aSi/F_Si
    end if
    if T_aSi < TD_a then
      count_fallback_tasks(T_a, T_aSi, TP_a, TD_a)
      if TF_aSi < TD_a then
        best_server(S_i,best_server_name)
      end if
    end if
  end for
  task_submit(best_server_name,task_name)
  Re-submission(task_name)
until the end
```

`can_do` This function works like for Algorithm 1.

`Count_fallback_tasks` This function counts the fallbacked tasks. Tasks that cannot meet their deadline after the insertion of the new task are called fallbacked tasks. Task T_a is placed according to its priority TP_a on the server task queue, which may change the execution time of the tasks on the queue.

`best_server` This function selects the best server among the servers that can execute the task within the deadline time. The best server is selected by comparing the number of fallbacked tasks. Server with less fallbacked tasks is selected for the task execution. If the servers have same number of fallbacked tasks, then the time to compute the task is compared and the server that takes less time is selected.

`task_submit` This function works the like in Algorithm 1. It performs the remote execution on the server given by the `best_server` but in this case the argument of the function one server and not a list of servers.

`Re-submission` This function submits the fallbacked task to the servers, for recomputing the execution time. If any server can meet the task's deadline then the task is allocated to that server.

			Exec. time on server		
Task	Priority	Deadline	S_1	S_2	S_3
1	3	15	3	5	6
2	5	10	5	12	9
3	2	30	11	20	15
4	4	20	10	np	17
5	5	15	12	14	np

Table 1: Priority, deadline and computation time of each task.

Figure 2 shows an example to explain the behavior of Algorithm 3. Lets consider 3 servers with different capacities and 5 tasks. The priority, deadline, and computation time of each task on each server is shown in Table 1. Here computation time is the time taken by the dedicated server to compute the task when the server is free. Computation value np denotes that the task cannot be executed on the server, which maybe due to the type of the task, memory requirement etc. A task is allocated to the server while checking the execution time on each server, its priority and deadline. Task T_1 is allocated to server S_1. Task T_2 is also allocated to server S_1. As its priority is higher it shifts the task T_1, so the execution time of T_1 is changed. If task T_3 is placed on server S_1, it will take less execution time but due to its priority the execution time will be changed. So task T_3 is placed on server S_3. Task T_4 is placed on server S_1, but while doing so task T_1 is fallbacked. Re-submission of task T_1 is done and it is allocated to server S_2. Task T_5 is placed on server S_2. As its

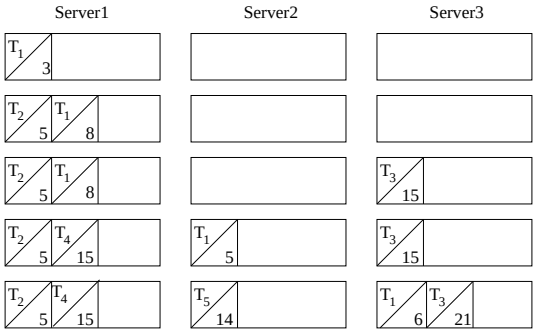


Figure 2: Example for priority scheduling algorithm with fallback mechanism. Task id and execution time is written diagonally in each box.

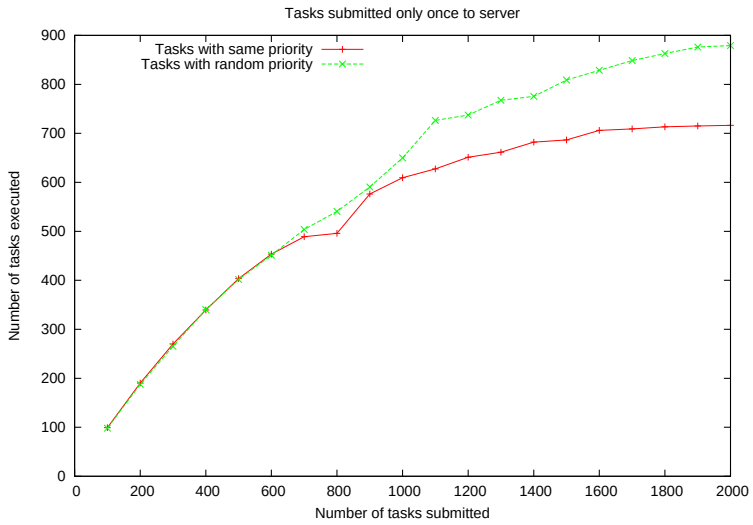


Figure 3: Priority based tasks are executed without fallback mechanism.

priority is higher than task T_1, execution time of T_1 changed and the task is then fallbacked. Again re-submission is done and task T_1 is placed on server S_3.

5 Simulation Results

To simulate the deadline algorithm with priority and the impact of fallback mechanism with this model we use a simulation toolkit called Simgrid [4]. Simgrid provides an excellent framework for setting up a simulation where decisions are taken by a single scheduling process.

We took 100 servers to execute the tasks in our experiments. Each task is associated with a priority and deadline. We randomly generated the priority between 1 and 10 and considered tasks deadline to be 5 times of the computation amount needed by the task on dedicated server.

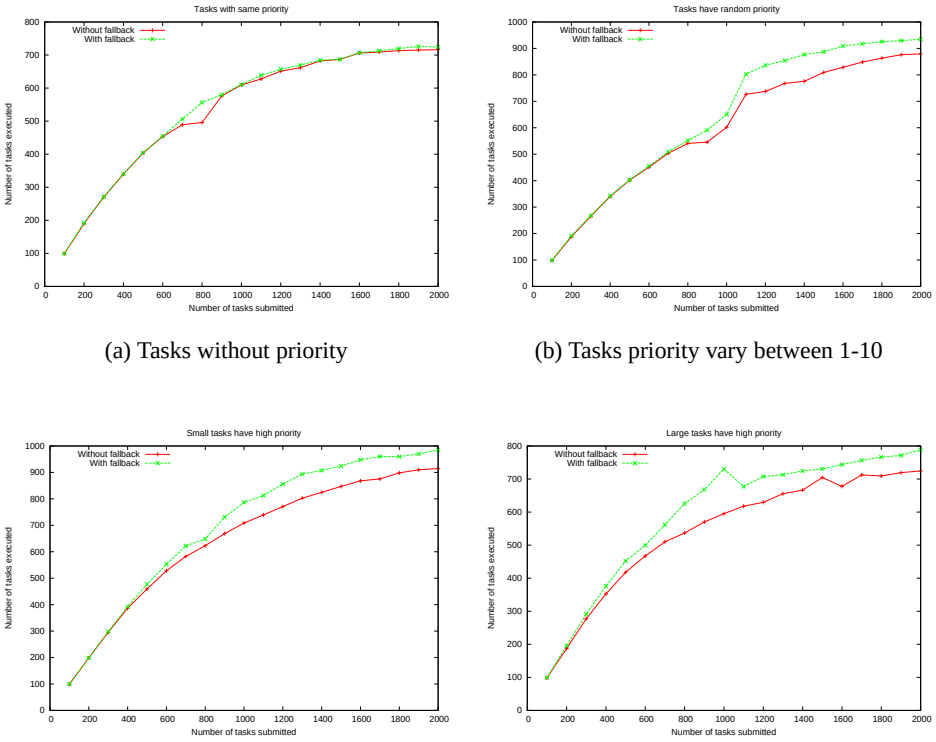
We did experiments by fixing the priority of the tasks and varying the priority depending on tasks' size. Figure 3 shows when tasks with same priority is submitted the number of executed tasks is less than the tasks executed with random priority. When the number of tasks is less the impact of task priority is negligible. But as the number of tasks increases, tasks' priority plays an important role for increasing the number of tasks executed.

We used Algorithm 3 to check the impact of fallback mechanism on the number of tasks executed under different criteria in Figure 4. Figure 4(a) shows that the fallback mechanism has no effect if the tasks have the same priority. But in Figure 4(b), 4(c), and 4(d), it can be seen that the fallback mechanism is very useful as the number of submitted tasks (with priority and different sizes) is increased.

6 Conclusions and Future Work

We have presented algorithms for scheduling of sequential tasks for client-server systems on the grid. This article is the an extension of the paper: "A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid" [13]. We gave the introduction of the performance prediction tool FAST [5, 11] and mainly focused on the management of tasks with respect to their priorities and deadlines. Load correction mechanism using FAST and fallback mechanism are used to increase the number of executed tasks. We presented an algorithm that considers both priority and deadline of the tasks to select a server. We showed through experiments that the number of tasks that can meet their deadlines can be increased by 1) using task priorities and by 2) using a fallback mechanism to reschedule tasks that were not able to meet their deadline on the selected servers.

Our future work consists in implementing these scheduling algorithms in the DIET platform [2].



(a) Tasks without priority

(b) Tasks priority vary between 1-10

(c) Tasks with execution time less than 15 minutes on dedicated servers

(d) Tasks with execution time greater than 4 hours on dedicated server

Figure 4: Comparison of tasks executed with and without fallback.

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] E. Caron, F. Desprez, F. Lombard, J.-M. Nicod, M. Quinson, and F. Suter. A Scalable Approach to Network Enabled Servers. In *Proceedings of EuroPar 2002*, Paderborn, Germany, 2002.

[3] E. Caron and F. Suter. Parallel Extension of a Dynamic Performance Forecasting Tool. In *Proceedings of the International Symposium on Parallel and Distributed Computing*, pages

80–93, Iasi, Romania, Jul 2002.

[4] H. Casanova. SIMGRID: A toolkit for the simulation of application scheduling. In *Proceedings of the IEEE Symposium on Cluster Computing and the Grid (CCGrid'01)*. IEEE Computer Society, May 2001.

[5] 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)*, June 2001.

[6] J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson. An Extended Set of Fortran Basic Linear Algebra Subroutines. 14(1):1–17, 1988.

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

[8] T. Howes, M. Smith, and G. Good. *Understanding and deploying LDAP directory services*. Macmillian Technical Publishing, 1999. ISBN: 1-57870-070-1.

[9] 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.

[10] 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`.

[11] M. Quinson. Dynamic Performance Forecasting for Network-Enabled Servers in a Metacomputing Environment. In *Proceedings of the International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS'02)*, Apr 2002.

[12] F. Sourd and W. Nuijten. Scheduling with Tails and Dead-lines. *Journal of Scheduling*, 4:105–121, 2001.

[13] A. Takefusa, H. Casanova, S. Matsuoka, and F. Berman. A study of deadline scheduling for client-server systems on the computational grid. In *the 10th IEEE Symposium on High Performance and Distributed Computing (HPDC'01)*, San Francisco, California., Aug. 2001.

[14] J. Verriet. Scheduling UET, UCT DAGS with Release Dates and Deadlines. Technical report, Utrecht University, Department of Computer Science, 1995.

[15] R. Whaley and J. Dongarra. Automatically Tuned Linear Algebra Software (ATLAS). In *Proceedings of IEEE SC'98*, 1998. Best Paper award.

[16] R. Wolski, N. 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.

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

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

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