Immediate Mode Scheduling of Independent Jobs in Computational Grids

Fatos Xhafa*
Polytechnic University of Catalonia
Dept. of Languages and Informatics Systems
Campus Nord, C/Jordi Girona 1-3, 08034 Barcelona, Spain
fatos@lsi.upc.edu

Leonard Barolli
Dept. of Information and Communication Engineering
Fukuoka Institute of Technology (FIT)
3-30-1 Wajiro-higashi, Higashi-ku, Fukuoka 811-0295, Japan
barolli@fit.ac.jp

Arjan Durresi
Dept. of Computer Science
Louisiana State University
298 Coates Hall, Baton Rouge, LA 70803
durresi@csc.lsu.edu

Abstract

With the emerging paradigm of grid computing and the development of grid infrastructures, grid-based applications are becoming a common approach for solving many complex, large-scale problems in science and engineering. In order to benefit from the large computing power of grid systems, efficient allocation of jobs to resources is necessary. In this work, we consider the allocation problem in immediate mode, in which jobs are allocated as soon as they arrive in the system. We implemented several methods and measured four parameters of the system: makespan, flowtime, resource utilization and matching proximity. The immediate methods are especially interesting when good quality allocations are necessary in very short time. The considered methods have been tested using the most difficult benchmark in the literature for the problem. The computational results allowed us to identify which of considered methods perform better for makespan, flowtime, resource utilization and matching proximity. Also, we were able to evaluate the usefulness of such methods if we knew in advance certain grid characteristics such as degree of consistency of computing, heterogeneity of jobs and resources.

*Research supported by ASCE Project TIN2005-09198-C02-02, Project FP6-2004-IST-FETPI (AEOLUS) and MEC TIN2005-25859-E.

1. Introduction

Computational grids were introduced by Foster and other researchers in late '90s [10, 11] as new computational frameworks by which geographically distributed resources are logically unified as a computational unit. Grid computing motivated the development of large scale applications that benefited from the large computing capacity offered by the grid. Thus, several projects such as NetSolve [8], MetaNeos Project[1], and applications for stochastic programming and optimization [14, 19] used computational grids.

Grid systems are parallel in nature; the large computing capacity provided by grid systems is beneficial for solving complex problems by using many nodes of the grid at the same time. The usefulness of a grid system largely depends, among other factors, on the efficiency of the system regarding the allocation of jobs to grid resources. The resource allocation problem is known to be computationally hard and much more difficult than its standard version for sequential or LAN computation environments. Indeed, not only the grid systems are heterogenous and dynamic, but also usually a large number of jobs must be allocated to grid nodes. For instance, researchers from MetaNeos Project reported that for solving the difficult instance Nug30 of the Quadratic Assignment Problem, a queue of thousands of

[1] http://www-unix.mcs.anl.gov/metaneos/

21st International Conference on Advanced Networking and Applications(AINA'07)
0-7695-2846-5/07 $20.00 © 2007 IEEE

pending jobs had to be managed for a grid of roughly 1000 machines.

Another important function of computational grids is to provide load balancing of the resources. Indeed, an organization or company could occasionally have unexpected peaks of activity, which require a larger capacity of computation resources. If their applications were grid-enabled, they would be able to be emigrated to machines with low utilization during those peaks.

Due to its importance, scheduling in Grids is being studied from different perspectives such as queueing systems and resource management (e.g. Condor-G, Nimrod/G) [13, 1], optimization approaches (e.g. genetic algorithms, metaheuristics) [1, 4, 16, 7, 6, 21] and economic models [3, 5].

Given the dynamic nature of the grid systems, any scheduler should provide allocations of jobs to resources as fast as possible. Therefore, schedulers based on very efficient methods are very important especially in presence of time restrictions on job executions on the grid. Immediate mode methods fall into this type of methods since they distinguish for their efficiency in contrast to more sophisticated schedulers that could need larger execution times to provide the allocation (e.g. a genetic-based scheduler would require more time for the population of individuals to converge). These methods have been proposed in the literature [15, 1, 2, 20] mainly with the objective to use them as part of other approaches. Our main focus here is to adapt and implement these methods for the case of grid systems as well as to conduct a complete comparative study on their performance. To the best of our knowledge, this is the first time these methods are empirically studied altogether. Furthermore, in previous work the performance of these methods is measured using only the makespan of the system while we study their performance with regard to four parameters: *makespan*, *flowtime*, *resource utilization* and *matching proximity*.

In the immediate mode, a job is scheduled as soon as it enters in the system without waiting for the next time interval when the scheduler will get activated or the job arrival rate is small having thus available resources to execute jobs immediately. We consider the following five immediate mode methods: *Opportunistic Load Balancing* (OLB), *Minimum Completion Time* (MCT), *Minimum Execution Time* (MET), *Switching Algorithm* (SA) and k-Percent Best (kPB). Once implemented, these methods have been tested using a benchmark of instances proposed by Braun et al. [2], which is obtained from the Expected Time to Compute model that simulates distributed heterogenous systems.

The experimental study revealed the performance of these methods with regard to the four considered parameters (makespan, flowtime, resource utilization and matching proximity). Furthermore, based on the computational results, we are able to evaluate the usefulness of the considered methods if we knew in advance certain grid characteristics such as the degree of consistency of computing, heterogeneity of jobs and heterogeneity of resources.

We also discuss some issues related to the integration of immediate methods in grid scheduling services. The recent grid standards and architectural schemes such as the Open Grid Services Architecture[2] (OGSA), developed by the Globus Alliance and based on standard XML-based web services technology, require the identification and integration of efficient dynamic schedulers as part of global grid service delivery. We observe that immediate mode schedulers are appropriate for decentralized scheduling services, that is, scheduling services that will integrate local schedulers, which will be in charge for the scheduling of user jobs to idle computers in a LAN. Immediate mode-based schedulers are especially efficient on LANs due to their small or moderate size and for sites with high throughput.

The paper is organized as follows. We give in Section 2 a description of the job scheduling in computational grids. The immediate mode methods considered in this work are given in Section 3. We give in Section 5 some computational results, which are evaluated in Section 6. We end in Section 8 with some conclusions.

2. Problem description

The job scheduling in grids consists in efficiently allocating jobs to resources in a global, heterogenous and dynamic environment. The efficiency means that we are interested to allocate jobs as fast as possible and optimizing several conflicting criteria such as *makespan*, *flowtime* and *resource utilization*.

Jobs have the following characteristics: are originated from different users/applications, have to be completed in unique resource (*preemptive*), are independent and could have their requirements over resources. On the other hand, resources could dynamically be added/dropped from the grid, can process one job at a time and have their own computing characteristics.

In order to formalize the instance definition of the problem, a benchmark simulation model is used. The main reason behind this choice is the difficulty of using at present real distributed grid systems to evaluate different grid scenarios, e.g. different degree of heterogeneity of resources, which would require modifying the configuration of grid nodes. Instead, we use the ETC (Expected Time to Compute) matrix model by Braun et al. [2], which is able to capture most important characteristics of the scheduling problem. Thus, in this model, an instance of the problem at a certain instant consists of:

[2] http://www.globus.org/ogsa/

– A *number* of independent (user/application) *jobs* to be scheduled.

– A *number* of heterogeneous *machines* candidates to participate in the planning.

– The *workload* of each job.

– The *computing capacity* of each machine (in *mips*).

– Ready time $ready_m$ –when machine m will have finished the previously assigned jobs. (Measures the previous workload of a machine.)

– The Expected Time to Compute matrix, ETC, $(nb\_jobs \times nb\_machines)$. $ETC[i][j] =$ the expected execution time of job i in machine j.

Note that this version of the problem does not include *local policies* of resources and possible *job dependencies*. However, it is possible to express incompatibilities among jobs and resources and penalties could be used to include in the *ETC* values the cost of data transmission.

Optimization criteria. Several parameters could be measured for a given schedule. Among these, there are:

- *Makespan* (finishing time of latest job) defined as $\min_{schedule} \max\{F_j : j \in Jobs\}$,
- $Flowtime$ (sum of finishing times of jobs), that is, $\min_{schedule} \sum_{j \in Jobs} F_j$, and
- *Resource utilization*, in fact, we consider the *average resource utilization*.

The last parameter is defined using the *completion time* of a machine, which indicates the time in which the machine will finalize the processing of the previous assigned jobs as well as of those already planned for the machine. Formally, for a machine m, it is defined as follows:

$$completion[m] = ready[m] + \sum_{j \in schedule^{-1}(m)} ETC[j][m]. \quad (1)$$

Having the values of the completion time for the machines, we can define the *local_makespan*, which is the makespan by considering only the machines involved in the current schedule:

$$local\_makespan = \max\{completion[i] \mid i \in Machines'\}. \quad (2)$$

Then, we define:

$$local\_avg\_utilization = \frac{\sum_{\{i \in Machines\}} completion[i]}{local\_makespan \cdot nb\_machines}. \quad (3)$$

It should be noted that these parameters are among the most important parameters of a grid system. Makespan measures the productivity (throughput) of the grid system, the flowtime measures the QoS of the grid system and resource utilization indicates the quality of a schedule with respect to the utilization of resources involved in the schedule aiming to reduce the idle time of resources.

We consider also a fourth parameter, called *matching proximity*, which is very useful for measuring the performance of the presented methods. Matching proximity indicates the degree of proximity of a given schedule with regard to the schedule produced by Minimun Execution Time (MET) method (see later). A large value of matching proximity means that a large number of jobs is assigned to the machine that executes them faster. Formally, this parameter is defined as follows:

$$matching\_proximity = \frac{\sum_{i \in Jobs} ETC[i][schedule[i]]}{\sum_{i \in Jobs} ETC[i][MET[i]]}. \quad (4)$$

3. Immediate mode methods

In this work five immediate mode methods, namely, *Opportunistic Load Balancing* (OLB), *Minimum Completion Time* (MCT), *Minimum Execution Time* (MET), *Switching Algorithm* (SA) and k-Percent Best (kPB) are implemented and empirically evaluated.

OLB: This method assigns a job to the earliest idle machine without taking into account the execution time of the job in the machine. If two or more machines are available at the same time, one of them is arbitrarily chosen. Usually this method is used in *scavenging grids*. One advantage of this method is that it tries to keep the machines as loaded as possible; however, the method is not aware of the execution times of jobs in machines, which is, certainly, a disadvantage regarding the makespan, flowtime and matching proximity parameters.

MCT: This method assigns a job to the machine yielding the earliest completion time (the ready times of the machines are used). When a job arrives in the system, all available resources are examined to determine the resource that yields the smallest completion time for the job. Note that a job could be assigned to a machine that does not have the smallest execution time for that job. This method is also

known as Fast Greedy, originally proposed for SmartNet system [12].

MET: This method assigns a job to the machine having the smallest execution time for that job. Unlike MCT method, MET does not take into account the ready times of machines. Clearly, in grid systems having resources of different computing capacity, this method could produce an unbalance by assigning jobs to fastest resources. However, the advantage is that jobs are allocated to resources that best fit them considering the execution time. This method is also known in the literature as LBA (*Limited Best Assignment*) and UDA (*User Directed Assignment*).

SA: This method tries to overcome some limitations of MET and MCT methods by combining their best features. More precisely, MET is not good for load balancing while MCT does not take into account execution times of jobs into machines. Essentially, the idea is to use MET till a threshold is reached and then use MCT to achieve a good load balancing. SA method combines MET and MCT cyclically based on the workload of resources.

In order to implement the method, let $r_{\max}$ be the maximum ready time and $r_{\min}$ the minimum ready time; the load balancing factor is then $r_{\min}/r_{\max}$, which takes values in $[0, 1]$. Note that for $r = 1.0$ we have a perfect load balancing and if $r = 0.0$ then there exists at least one idle machine. Further, we use two threshold values r_l (low) and r_h (high) for r, $0 \leq r_l < r_h \leq 1$. Initially, $r = 0.0$ so that SA starts allocating jobs according to MCT until r becomes greater than r_h; after that, MET is activated so that r becomes smaller than r_l and a new cycle starts again until all jobs are allocated.

kPB: For a given job, this method considers a subset of candidate resources from which the resource to allocate the job is chosen. The candidate set consists of $nb\_machines \cdot k/100$ best resources (with respect to execution times) for the given job, for k, $nb\_machines/100 \leq k \leq 100$. The machine where to allocate the job is the one from the candidate set yielding the earliest completion time. Note that for $k = 100$, kPB behaves as MCT and for $k = 100/nb\_machines$ it behaves as MET. It should be noted that this method could perform poorly if the subset of resources is not within $k\%$ best resources for none of jobs implying thus a large idle time.

Notice that both kPB and SA combine the best features of MCT and MET, however, only kPB tries to simultaneously optimize the objectives of MCT and MET while assigning a job to a machine. Nonetheless, kPB has an important drawback in case a subset of machines is not among the $k\%$ best for none of the jobs, implying thus a large idle time.

4 A simple working example

One characteristic that can be observed from the description of all considered methods is that they use concrete strategies, which could work good for some grid systems instances and poorly for some others. To illustrate this, let us consider the following example [15]. Suppose we have three resources with ready times (current workload) given in Table 1.

Table 1. Ready times of three machines (in arbitrary time units).

m_0	m_1	m_2
70	110	200

Further, suppose that three jobs j_0, j_1 and j_2 arrive in the systems in this order whose expected time to compute are given in Table 2.

Table 2. ETC values for 3 machines and 3 jobs (in arbitrary time units).

	m_0	m_1	m_2
j_0	50	20	15
j_1	20	60	15
j_2	20	50	15

By applying the immediate mode methods, we obtain the following results.

OLB: This method assigns j_0 to m_0 because m_0 will be the earliest idle machine. Similarly, it assigns j_1 to m_1 and j_2 to m_0. The resulting makespan is 170 time units (see Eq. 2).

MCT: This methods finds out that the earliest completion time for j_0 is achieved in machine m_0 and thus assigns j_0 to m_0 despite that its expected time to compute is almost three times the expected time to compute of j_0 in m_2. Next, MCT assigns j_1 to m_0 and j_2 to m_1. The resulting makespan is 160 time units.

MET: This method finds out that all jobs have the least expected time to compute in m_2, but it does not take into account that m_2 is the most overloaded resource and assigns all three jobs to this machine. The resulting makespan is 245 time units!

SA: This methods first computes the load balancing factor $r = 75/200 = 0.38$. By letting $r_l = 0.40$ and $r_h = 0.70$, and since $r < r_l$, SA chooses MCT to obtain

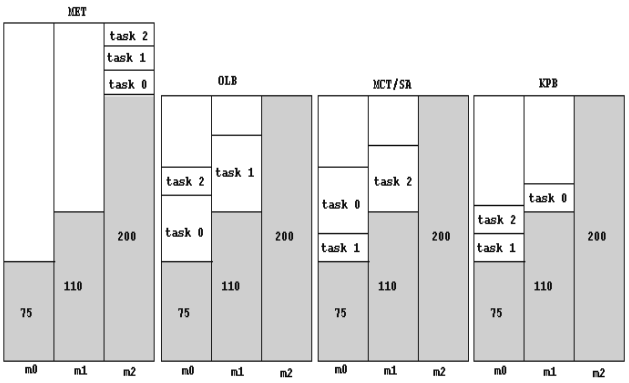

Figure 1. An example of immediate mode methods.

the first assignment, namely, it assigns j_0 to m_0. After this assignment, r becomes $110/200 = 0.55 < r_h$, hence SA continues to use MCT for the second assignment. It is after the third assignment when r becomes $145/200 = 0.725 > r_h$ that SA will use MET method for the assignment of the forth job (still to be arrived in the system). The resulting makespan is 160 time units, the same as in the case of MCT.

kPB: By letting $k = 2/3$, for each job the method considers two resources having the smallest ETC value. Thus, for j_0, these machines are m_1 and m_2, from which m_1 is chosen to assign j_0. Note that this allows the machine m_0 to be available for other jobs (e.g. j_2) that would take longer execution times in other machines. The resulting makespan is 130 time units, the best among all considered methods.

A graphical illustration of the example is shown in Figure 1.

Implementation. We have efficiently implemented the presented methods in C++. It can be shown that for immediate mode methods, their time complexity is $O(N \cdot M)$, except kPB, which has time complexity $O(N \cdot M \cdot \log M)$; N denotes the number of scheduled jobs and M the number of machines. As can be observed from this time complexity, the presented immediate mode methods have low computational costs, being thus very appropriate for dynamic heterogenous systems, such as computational grids.

5. Computational results

In this section we present some computational results obtained with the implementations of the immediate mode methods using a benchmark of instances by Braun et al. [2] for distributed heterogenous systems. The simulation model of Braun et al. allows us to establish a fair comparison of the presented methods. Moreover, different grid scenarios can be considered by combining different characteristics of the grid systems such as computing consistency, heterogeneity of resources and jobs.

The instances of this benchmark are classified into 12 different types of ETC matrices, each of them consisting of 100 instances, according to three parameters: job heterogeneity, machine heterogeneity and consistency. Instances are labelled as $u\_x\_yyzz.k$ where:

- u means uniform distribution (used in generating the matrix).

- x means the type of consistency (c–consistent, i–inconsistent and s means semi-consistent). An ETC matrix is considered consistent when, if a machine m_i executes job j faster than machine m_j, then m_i executes all the jobs faster than m_j. Inconsistency means that a machine is faster for some jobs and slower for some others. An ETC matrix is considered semi-consistent if it contains a consistent sub-matrix.

- yy indicates the heterogeneity of the jobs (hi means high, and lo means low).

- zz indicates the heterogeneity of the resources (hi means high, and lo means low).

Note that all instances consist of 512 jobs and 16 machines. For each method we compute the makespan, flowtime, resource utilization and matching proximity.

We give in Tables 3, 4, 5 and 6 the computational results obtained from immediate mode methods for makespan, flowtime, resource utilization and matching proximity, respectively, for a set of 12 instances of the Braun et al. benchmark. In order to have a representative set of instances, three groups of four instances having consistent, semi-consistent and inconsistent *ETC* matrices are chosen. For each group, we have chosen instances having different types of heterogeneity of jobs and heterogeneity of resources.

6. Evaluation

In this section we evaluate the computational results[3] obtained from immediate mode methods regarding four parameters: makespan, flowtime, resource utilization and matching proximity. As can be seen from the description of the considered methods, they use concrete strategies therefore the objective is to evaluate their performance, from which we could deduce which method to use for certain

[3]Note that the presented methods are deterministic. Their execution times are given by their time complexity.

grid systems characteristics (configurations), especially if we knew such characteristics in advance.

From Tables 3, 4, 5 and 6 we can easily draw the following conclusions.

- *Makespan*: The methods that best perform regarding the makespan are MCT and kPB. Moreover, a careful examination of the results show that MCT obtains better results for consistent matrices while kPB performs better for semi-consistent and inconsistent matrices. SA shows a good performance, except for inconsistent matrices. MET shows a good performance for inconsistent matrices and poor solutions for semi-consistent matrices (see Table 3).

- *Flowtime*: The performance of these methods regarding this parameter are not as good as for makespan; rather small reductions of flowtime are obtained. For consistent matrices SA performs better followed by MCT. For semi-consistent matrices the best performance is shown by SA and kPB while for inconsistent matrices MET outperforms all the other methods. OLB shows a poor performance (see Table 4).

- *Average resource utilization*: The MET method obtains the worse resource utilization overall (the resource utilization is 1.0 for consistent matrices, as expected). kPB performs better for consistent matrices, however, MCT and OLB achieve a better load balancing for inconsistent matrices. Overall, OLB is the method that performs best, in particular, for semi-consistent matrices. Nonetheless, MCT and kPB behave quite similarly. Observe also that though OLB achieves very good resource utilization, its makespan values are not good. Thus, we could say that for very heterogenous systems, resource utilization is not a good indicator for makespan. Finally, SA performs rather poorly (see Table 5).

- *Matching proximity*: Except for MET (which shows perfect proximity matching), the best matching proximity is obtained by kPB and SA. In particular, kPB performs better for consistent matrices and SA performs better for semi and inconsistent matrices. MCT performs poorly and OLB shows the worst matching proximity values (see Table 6).

7. Immediate scheduling and Grid services

Grid systems are expected to provide a large variety of complex services [18] whose interactions require scheduling and resource allocation policies. At present, coordinated scheduling of services in grid systems is not yet available despite the existence of several middleware (e.g. Globus Toolkit [4]) or brokerage examples (e.g. Nimrod/G Resource Broker[5] [4]).

One approach to scheduling services is that of a decentralized schedulers or brokers that will integrate local schedulers, also known as distributed scheduling components, which will be in charge for the scheduling of user jobs to idle computers in a LAN (for instance, Nimrod/G Resource Broker system incorporates such a component). In this context, immediate mode schedulers presented here are appropriate for local schedulers and could be thus integrated in a more complex scheduler system as well as with higher-level scheduling services. Indeed, as shown by the study of this work, immediate mode methods explore the characteristics of the grid system and thus if we knew in advance some of these characteristics, for instance the degree of heterogeneity of jobs and resources or the degree of consistency we could adaptively choose the appropriate immediate method. This is possible for LANs since we could know in advance the characteristics of the LAN at the institution or enterprise. Grid scheduling architectures then should support cooperation between different scheduling instances. We could think also of job submission service responsible for managing jobs: it receives a job request from users/applications, requests scheduling to a grid scheduling service[6] and requests job execution to local scheduler; this last could be immediate mode-based scheduler since this sort of schedulers are especially efficient on LANs due to their small or moderate size and for sites with high throughput.

Immediate mode schedulers are dynamic and hence they could be also useful in scheduling of workflow applications [22] since in such applications the workflow scheduler must be able to adapt and update the schedule based on resource dynamics.

All in all, given that for a large family of grid applications requests for resources could be immediate, the presented immediate mode methods are potentially useful in scheduling jobs originated by such applications.

8. Conclusions and future work

In this work we have presented a family of methods for dynamic scheduling of independent jobs in computational grids, known as immediate mode methods. The main characteristic of these methods is that they schedule the jobs as soon as the jobs arrive in the system. Five methods (*Opportunistic Load Balancing*, *Minimum Completion Time*, *Minimum Execution Time*, *Switching Algorithm* and k*-Percent Best*) have been presented and studied.

[4] http://www.globus.org/toolkit/

[5] http://www.csse.monash.edu.au/∼davida/nimrod/

[6] Examples of integrating batch mode schedulers with grid services already exit in the literature. Condor system [9] is one such example.

The implementations of these methods have been tested using the most difficult benchmark in the literature for the problem, by Braun et al. [2]. The computational results show that none of the methods performs best; rather, their performance depends on the grid scenarios based on heterogeneity of the jobs (high, low), heterogeneity of the resources (high, low) and consistency (consistent, inconsistent and semi-consistent). It should be noted that consistency allow to simulate real grid environments in which restrictions job-resource could exist.

Thus it is very interesting to dispose implementations of such a variety of methods so that we could choose and apply the appropriate method if we knew in advance some of the grid characteristics, and also if we were interested in a concrete parameter (makespan, flowtime, resource utilization or matching proximity). Also, these methods are very interesting if there are time restrictions because they are very fast. Finally, our implementations are ready to use as part of more sophisticated heuristics.

We are currently testing a discrete-event grid simulator based on HyperSim package [17, 7] that we will use to study the performance of the presented methods and address the issue of experimenting in a dynamic grid environment. Additionally, we plan to extend this work in designing and implementing some hyper-heuristic using the methods presented here so that the appropriate immediate methods will be adaptively chosen according to the grid characteristics.

References

[1] A. Abraham, R. Buyya, and B. Nath. Nature's heuristics for scheduling jobs on computational grids. In *The 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000) India*, 2000.

[2] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, and B. Yao. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, 2001.

[3] R. Buyya. *Economic-based Distributed Resource Management and Scheduling for Grid Computing*. PhD thesis, Monash University, Melbourne, Australia, 2002.

[4] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid. In *The 4th Int. Conf. on High Performance Computing, Asia-Pacific Region, China*, 2000.

[5] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1507–1542, 2002.

[6] J. Carretero and F. Xhafa. Using genetic algorithms for scheduling jobs in large scale grid applications. *Journal of Technological and Economic Development, Vilnius Gediminas Technical University*, 12(1):11–17, 2006.

[7] J. Carretero. A Comprative Study of Heurisric Methods for Job Scheduling on the Grid (in Spanish). Master Thesis, Faculty of Informatics of Barcelona, Polytecnic University of Catalonia, 2005.

[8] H. Casanova and J. Dongarra. Netsolve: Network enabled solvers. *IEEE Computational Science and Engineering*, 5(3):57–67, 1998.

[9] C. Chapman, P. Wilson, T. Tannenbaum, M. Farrellee, M. Livny, J. Brodholt, and W. Emmerich. Condor services for the global grid: interoperability between Condor and OGSA. In *Proceedings of 2004 UK e-Science All Hands Meeting*, pages 870–877, Nottingham, UK, August 2004.

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

[11] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid. *Int. J. of Supercomputer Applications*, 15(3), 2001.

[12] R. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. Lima, F. Mirabile, L. Moore, B. Rust, and H. Siegel. Scheduling resources in multi-user, heterogeneous, computingenvironments with smartnet. In *Seventh Heterogeneous Computing Workshop*, pages 184–199, 1998.

[13] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. In *Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing*, 2001.

[14] L. Linderoth and S. Wright. Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications*, 24:207–250, 2003.

[15] M. Maheswaran, S. Ali, H. Siegel, D. Hensgen, and R. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 59(2):107–131, 1999.

[16] V. D. Martino and M. Mililotti. Sub optimal scheduling in a grid using genetic algorithms. *Parallel Computing*, 30:553–565, 2004.

[17] S. Phatanapherom and V. Kachitvichyanukul. Fast simulation model for grid scheduling using hypersim. In *Proceedings of the 2003 Winter Simulation Conference*, USA.

[18] S. Venugopal, R. Buyya, and L. J. Winton. A grid service broker for scheduling distributed data-oriented applications on global grids. In *Middleware for Grid Computing: Proceedings of the 2nd Workshop on Middleware for Grid Computing, Toronto, Ontario, Canada, October 18-22, 2004*, pages 75–80. ACM, 2004.

[19] S. Wright. Solving optimization problems on computational grids. *Optima*, 65, 2001.

[20] M.-Y. Wu and W. Shu. A high-performance mapping algorithm for heterogeneous computing systems. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium*, page 74, 2001.

[21] F. Xhafa. An experimental study on GA replacement operators for scheduling on grids. In *The 2nd International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2006)*, pp. 121-130, Ljubljana, Slovenia, 2006.

[22] J. Yu, R. Buyya, and C. K. Tham. Qos-based scheduling. of workflow applications on service grids. In *Proceedings of the 1st IEEE International Conference on e-Science and Grid Computing,Melbourne, Australia.* IEEE, 2005.

Table 3. Makespan values obtained with immediate mode methods (in arbitrary time units).

Instance	OLB	MCT	MET	SA (r_l=0.6, r_h=0.9)	kPB (k=20%)
u_c_hihi.0	14376662.175	11422624.494	47472299.429	12613221.101	12496863.706
u_c_hilo.0	221051.823	185887.404	1185092.968	194549.794	201153.956
u_c_lohi.0	477357.019	378303.624	1453098.003	426271.390	400291.050
u_c_lolo.0	7306.595	6360.054	39582.297	8167.052	6846.273
u_i_hihi.0	26102017.618	4413582.982	4508506.791	4692192.006	4508655.928
u_i_hilo.0	272785.200	94855.913	96610.481	102980.982	93005.897
u_i_lohi.0	833605.654	143816.093	185694.594	143905.246	143816.093
u_i_lolo.0	89380.269	3137.350	3399.284	3485.290	3122.956
u_s_hihi.0	19464875.910	6693923.896	25162058.136	7127729.951	6514162.148
u_s_hilo.0	250362.113	126587.591	605363.772	149050.289	123543.792
u_s_lohi.0	603231.467	186151.286	674689.535	194318.366	187955.955
u_s_lolo.0	8938.389	4436.117	21042.413	5836.962	4405.247

Table 4. Flowtime values obtained with immediate mode methods (in arbitrary time units).

Instance	OLB	MCT	MET	SA (r_l=0.6, r_h=0.9)	kPB (k=20%)
u_c_hihi.0	1995375910.590	1815882056.432	4844329725.992	1846422579.779	2002538268.654
u_c_hilo.0	35915700.026	35394474.558	195784905.697	34901832.621	37220460.128
u_c_lohi.0	66414109.547	65625471.648	156597212.675	63250093.468	66422481.844
u_c_lolo.0	1189657.523	1193304.135	6399549.478	1181075.902	1255064.070
u_i_hihi.0	3578545420.915	591703697.305	369819381.995	557457024.518	596857288.709
u_i_hilo.0	40168914.127	16309961.615	12778823.342	15027765.500	16093172.399
u_i_lohi.0	112209075.410	18954137.704	12686655.759	18900796.619	18552145.175
u_i_lolo.0	1380271.657	545326.248	443305.108	498180.120	548765.218
u_s_hihi.0	2711172017.693	964163417.887	1559122316.924	949405775.550	907398445.237
u_s_hilo.0	36996922.147	22357619.927	57134246.084	21369485.252	22056448.264
u_s_lohi.0	83583290.767	27064495.376	43054529.545	25883799.205	27266722.451
u_s_lolo.0	1395727.226	797285.039	1973399.444	791072.307	780609.157

Table 5. Average resource utilization values obtained with immediate mode methods (see Eq. 3).

Instance	OLB	MCT	MET	SA r_l=0.6 r_h=0.9	kPB (k=20%)
u_c_hihi.0	0.946	0.953	1.000	0.890	0.972
u_c_hilo.0	0.920	0.970	1.000	0.920	0.974
u_c_lohi.0	0.928	0.969	1.000	0.832	0.969
u_c_lolo.0	0.923	0.951	1.000	0.727	0.960
u_i_hihi.0	0.951	0.932	0.628	0.846	0.929
u_i_hilo.0	0.955	0.959	0.750	0.816	0.954
u_i_lohi.0	0.934	0.949	0.536	0.948	0.937
u_i_lolo.0	0.979	0.965	0.740	0.797	0.968
u_s_hihi.0	0.967	0.928	0.197	0.863	0.928
u_s_hilo.0	0.924	0.938	0.214	0.781	0.951
u_s_lohi.0	0.961	0.953	0.216	0.891	0.946
u_s_lolo.0	0.951	0.951	0.221	0.708	0.950

Table 6. Matching proximity values obtained with immediate mode methods (see Eq. 4).

Instance	OLB	MCT	MET	SA r_l=0.6 r_h=0.9	kPB (k=20%)
u_c_hihi.0	0.217	0.272	1.000	0.264	0.300
u_c_hilo.0	0.364	0.410	1.000	0.413	0.464
u_c_lohi.0	0.204	0.247	1.000	0.255	0.287
u_c_lolo.0	0.366	0.408	1.000	0.416	0.463
u_i_hihi.0	0.114	0.688	1.000	0.713	0.675
u_i_hilo.0	0.278	0.796	1.000	0.862	0.816
u_i_lohi.0	0.127	0.729	1.000	0.730	0.739
u_i_lolo.0	0.287	0.830	1.000	0.905	0.831
u_s_hihi.0	0.148	0.448	1.000	0.453	0.461
u_s_hilo.0	0.315	0.614	1.000	0.626	0.620
u_s_lohi.0	0.141	0.463	1.000	0.474	0.462
u_s_lolo.0	0.308	0.620	1.000	0.633	0.625