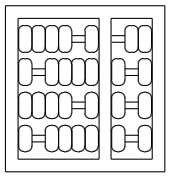


INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Robust hybrid mechanisms for scheduling of grid tasks

R. L. Curi *D. M. Batista*
N. L. S. da Fonseca

Technical Report - IC-12-12 - Relatório Técnico

May - 2012 - Maio

Robust hybrid mechanisms for scheduling of grid tasks

Rafael Lima Curi Daniel Macedo Batista
Nelson Luis Saldanha da Fonseca*

Abstract

This paper proposes three new hybrid mechanisms for the scheduling of grid tasks, which integrate reactive and proactive approaches. They differ by the scheduler used to define the initial schedule of an application and by the scheduler used to reschedule the application. The mechanisms are compared to reactive and proactive mechanisms. Results show that hybrid approach produces performance close to that of the reactive mechanism, but demanding less migrations.

1 Introduction

In order to provide the Quality of Service (QoS) required by applications, grids must employ efficient mechanisms for the scheduling of tasks [13] (grid applications are composed by a set of tasks), assuring that the mapping between tasks and hosts will minimize the execution time of grid applications, also known as makespan. Scheduling mechanisms must also cope with the uncertainties of the demands of grid applications as well as those of the resource availability [2, 3].

In general, two opposite approaches can be adopted for grid scheduling: the reactive and the proactive approaches. In reactive mechanisms [4, 14, 10, 15], resource monitoring is used to update information about the grid state during the execution of applications, for the verification of the maintenance of the effectiveness of the initial schedule. Migrations of tasks can be proposed to improve performance.

In the proactive approach, the degree of uncertainty on the values of application demands and resource availability are provided as input to the scheduler. One way to deal with such uncertainties is to formulate the scheduling problem as a fuzzy optimization problem [7, 9, 6, 5]. The proactive approach avoids the overhead caused by the re-computation of the schedules and by task migration. Besides, it does not need frequent monitoring and

*Institute of Computing, University of Campinas, Campinas, SP, Brazil, 13089-971.

Curi, R.L.; Batista, D.M.; da Fonseca, N.L.S., Robust hybrid mechanisms for scheduling of grid tasks In Proc. of 2011 IEEE Latin-American Conference on Communications (LATINCOM), pg. 1-6, 2011, doi:10.1109/LatinCOM.2011.6107411

checkpointing system as in the reactive approach. One disadvantage is that opportunities for improving the schedule are not taken.

This paper introduces three new mechanisms, denominated as hybrid mechanisms, that capitalize on the benefits of both reactive and proactive approaches. These mechanisms are oriented to grid applications composed of a set of dependent tasks. They are evaluated and compared to those based on reactive approach [4] and on proactive approach [6, 5]. Moreover, two new grid schedulers are proposed to be used in these mechanisms.

In [10], both proactive and reactive mechanisms in grid scheduling were used for building fault tolerant grids. A scheduler based on an evolutionary algorithm as well as Anycast were employed. The difference of this proposal to those in this paper is that it aims at handling failures only, and it ignores the variation of resource availability. In [15], a self-healing approach is proposed to perform proactive task migration in High-Performance Computing environments, which is also oriented to provide fault tolerance. Moreover, the works in [10] and [15] ignore the entry of new resources when evaluating the need of task migration. In contrast, the hybrid mechanisms proposed in this paper considers changes in resource availability of both types (failures and increase or decrease of availability) and it prevents unnecessary migrations.

The remainder of this paper is organized as follows: Section 2 introduces three new hybrid mechanisms and two new schedulers. Section 3 presents the experiments conducted to analyze the mechanisms and Section 4 concludes the paper.

2 Hybrid mechanisms

If on one hand, proactive mechanisms do not imply overhead due to monitoring, rescheduling and task migration, they do not react to changes in resource availability. Conversely, reactive mechanisms are capable of reacting, but imply on the mentioned overhead, and schedules produced do not consider uncertainties on the information provided to the schedulers.

The proposed hybrid mechanisms follows the reactive mechanisms cycle, but schedules are given by proactive schedulers. In this way, react to changes can be as immediate as in the reactive mechanisms and the need for changes can be reduced by the consideration of uncertainties in demands and resource availability. Variations of hybrid mechanisms are analyzed by considering combinations of the use of proactive schedulers either at the initial scheduling step and at the rescheduling steps.

In addition to the comparison between the proactive and the reactive approaches, this paper attempts to answer several questions, such as: which of the two approaches decreases the number of migrations more significantly; what is the gain given by the employment of the proactive approach when adopted at different steps of the reactive approach cycle; and which of these approaches leads to shorter makespan values.

Since the performance of the mechanisms depends heavily on the schedulers used, two new schedulers, which are revised versions of existing schedulers in [4, 6], are proposed in Subsection 2.1, and used by the three mechanisms presented in Subsection 2.2.

2.1 Schedulers

The schedulers proposed are based in two existing schedulers: the IPDT [4] and the IP-FULL-FUZZY [6]. The first is commonly used in reactive mechanisms, while the second in proactive mechanisms. The requirements of the application given as input are represented by Directed Acyclic Graphs (DAGs), in which nodes correspond to tasks and arcs to the dependencies among tasks, i.e., if task i has an arc directed to task j, then task j depends on data computed by task i. The weights of the nodes represent the number of instructions of each task and the weights of the arcs represent the amount of bits that must be transferred from one task to the other. The grid given as input is represented by a graph, in which the nodes correspond to the hosts and the edges correspond to the links. The weights of the nodes give the available processing capacity, in minutes/instruction, and the weights of the edges the inverse of the available bandwidth, given in minutes/bit.

The IPDT solves an integer linear programming problem. Time is discretized and defined as $\mathcal{T} = \{1, \ldots, T_{max}\}$, where T_{max} corresponds to the ceiling of the makespan reached when all the tasks are executed sequentially in the fastest host. The objective function of the problem is the minimization of the makespan, which must be at most equal to T_{max}. The set of tasks is represented by $\mathcal{J} = (V_J, E_J)$, where $E_J = \mathcal{D}$ is the set of arcs $\{ij : i < j$, and there exists an arc from vertex i to vertex j in the DAG$\}$, i.e., the dependencies of the application. The set of hosts is represented by $\mathcal{H} = (V_H, E_H)$. $\delta(l)$ is the set of hosts connected with the host l, including l itself. I_i corresponds to the weight of the task i, $B_{i,j}$ corresponds to the weight of the arc between the tasks i and j, TI_l corresponds to the weight of the host l and $TB_{k,l}$ corresponds to the weight of the link between the hosts k and l.

$x_{i,t,k}$ is a binary variable that has value 1 if task i execute in the host k finishing at time t, and 0 otherwise.

The formulation adopted in the experiments, which is called IPDT-2, is showed next:

$$\text{Minimize} \sum_{t \in \mathcal{T}} \sum_{k \in \mathcal{H}} t \, x_{j,t,k}$$

subject to

$$\sum_{t \in \mathcal{T}} \sum_{k \in \mathcal{H}} x_{j,t,k} = 1 \quad \text{for } j \in \mathcal{J}; \tag{D1}$$

$$x_{j,t,k} = 0$$
$$\text{for } j \in \mathcal{J}, \quad k \in \mathcal{H}, \quad t \in \{1, \ldots, \lceil MC + I_j \, TI_k \rceil\}; \tag{D2}$$

$$\sum_{k \in \delta(l)} \sum_{s=1}^{t - \lceil I_j \, TI_l + B_{i,j} \, TB_{k,l} \rceil} x_{i,s,k} \geq \sum_{s=1}^{t} x_{j,s,l}$$
$$\text{for } i, j \in \mathcal{J}, \quad ij \in \mathcal{D}, \quad l \in \mathcal{H}, \quad t \in \mathcal{T}; \tag{D3}$$

$$\sum_{j \in \mathcal{J}} \sum_{s=t}^{t+\lceil I_j TI_k \rceil - 1} x_{j,s,k} \leq 1$$
$$\text{for } k \in \mathcal{H}, \quad t \in \mathcal{T}, \quad t \leq T_{max} - \lceil I_j TI_k \rceil; \quad \text{(D4)}$$
$$x_{j,t,l} \in \{0,1\} \quad \text{for } j \in \mathcal{J}, \quad l \in \mathcal{H}, \quad t \in \mathcal{T}. \quad \text{(D5)}$$

The IPDT-2 differs from the IPDT by the constraints D2, D3 and D4. Constraint D2 must be applied to $t \in \{1, \ldots, \lceil MC + I_j TI_k \rceil\}$, where MC is related to the fact that a task can start only after all the dependencies to the specific task on the longest dependency path have finished. Constraints D3 and D4 put t out of the ceiling operation. When compared to the original formulation [4], these changes allow tasks to be started earlier than they would be in IPDT, and as a consequence there is a decrease of the makespan.

Constraint D1 establishes that each task j must be executed in a single host k, finishing at time t. Constraint D2 establishes that a task j executing on host k can not finish before the end of the last task in the longest dependency path starting in the initial task 0 and ending on task j, when these tasks are executed sequentially on the fastest host (MC). Constraint D3 establishes that task j can start only after it has received the required data from all the tasks it depends on. Constraint D4 establishes that only a single task can be executed at a time on each host. Constraint D5 establishes the domains of the variables x.

The IP-FULL-FUZZY is based on a fuzzy optimization problem. It uses triangular fuzzy numbers to model the values of the computational and the communication demands, processing capacity of the hosts and available bandwidth of network links. These values are given as intervals $[min, max]$ that represent a fuzzy set. Each point $x \in \mathbb{R}$ is associated to a value of the membership function, $\mu(x) \in [0,1]$. Let $c = \frac{min+max}{2}$, then the fuzzy numbers used are of the type $[min, c, max]$, where:

$$\mu(x) = \begin{cases} 0 & \text{, for } x < min \\ \frac{x-min}{c-min} & \text{, for } min \leq x \leq c \\ \frac{x-max}{c-max} & \text{, for } c < x \leq max \\ 0 & \text{, for } x > max \end{cases}$$

The triangular fuzzy numbers are obtained considering the uncertainty levels given as input to the scheduler: σ is the uncertainty level associated to the tasks, ρ is the uncertainty level associated to the dependencies, χ is the uncertainty level associated to the hosts and ω is the uncertainty level associated to the links. Therefore, the amount of instructions of task i is given by $\widetilde{I_i} = [\underline{I_i}, I_i, \overline{I_i}]$, where $\underline{I_i} = I_i(1 - \frac{\sigma}{100})$ and $\overline{I_i} = I_i(1 + \frac{\sigma}{100})$. The amount of bits of dependency ij is given by $\widetilde{B_{i,j}} = [\underline{B_{i,j}}, B_{i,j}, \overline{B_{i,j}}]$, which is computed in the same way. The processing capacity of host k is given by $\widetilde{TI_k} = [\underline{TI_k}, TI_k, \overline{TI_k}]$ and the available bandwidth between the hosts k and l is given by $\widetilde{TB_{k,l}} = [\underline{TB_{k,l}}, TB_{k,l}, \overline{TB_{k,l}}]$.

The fuzzy optimization problem is also modeled as an integer linear programming problem, which is obtained by transforming the fuzzy constraints to the corresponding crisp constraints [17]. In this transformation, T''_{max} is the maximum time value and it is given

by $T_{max}^{''} = T_{max}(1 + \frac{\sigma}{100})(1 + \frac{\chi}{100})$. In the formulation adopted in the experiments, called IP-FULL-FUZZY-2, a new heuristic is used to find the value of $T_{max}^{''}$, aiming at decreasing the complexity of the integer linear programming problem. It makes the solution faster and allows lower discretization values. Therefore, the new $T_{max}^{''}$ is given by $T_{max}^{''} = T_{max} + T_f$, where T_f is the larger ending time of the tasks already executed and it is computed in the rescheduling process. If the value of $T_{max}^{''}$ leads to an unfeasible problem, then, let $T_{max}^{'} = T_{max}^{''}$, the reschedule is produced again with the new value of $T_{max}^{''}$ given by $T_{max}^{'} + T_f$. It is repeated until a feasible solution is reached. It does not cause a significant increase in the time spent to produce the schedule at the rescheduling step, because the solution given by the optimization library is faster when the problem is unfeasible than when it is not. $T_{min}^{''}$ corresponds to the lowest execution time of the application, and it is equal to the time spent executing, in the fastest host, the instructions of the tasks belonging to the shortest path starting from the initial task 0 and finishing at task $n-1$ (SP). Therefore, $T_{min}^{''}$ is given by $T_{min}^{''} = min(\{TI_{k|k \in V_H}\}) \times \sum_{i \in SP} I_i$. $\lambda \in [0, 1]$ corresponds to the degree of satisfaction of the schedule proposed and it is inversely proportional to the reached makespan f_n. Due to the limited space in the paper, only the integer linear programming formulation is presented. Actually, only the constraints that differ from those of the IPDT-2 are presented.

The formulation adopted considers that the capacity of hosts and links can only decrease, and never increase. This is realistic since it aims to produce a schedule robust in the worst case. Moreover, the consideration of greater availability of resources can be dangerous, since it leads to poor performance when the assumption is false. The formulation differs from that of the IPDT-2 in: the objective function (maximization of λ); the addition of the constraint F1, which establishes a relation between λ and f_n (the ending time of the last task); and constraint F4, which consider the maximum time of processing and transmission of data, preventing a task to be executed before its predecessors.

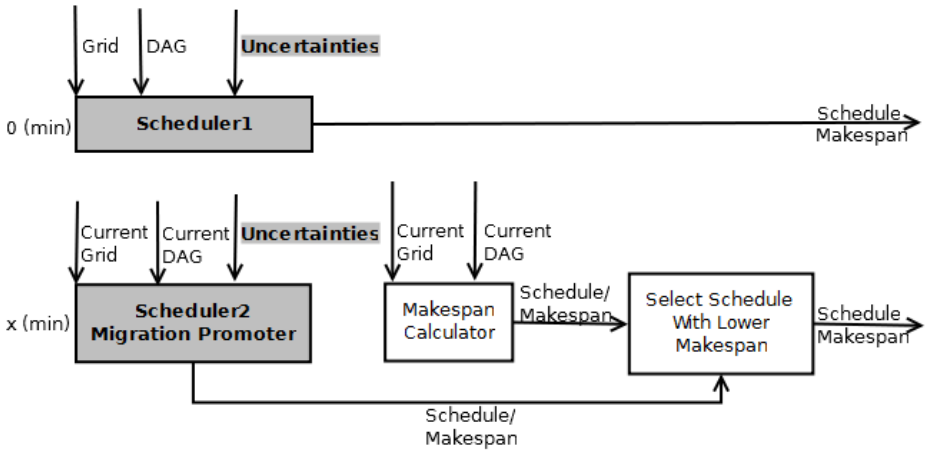
The new constraint and the changed one is showed next:

$$f_n \leq (1 - \lambda)T_{max}^{''} + \lambda T_{min}^{''}; \qquad \text{(F1)}$$

$$\sum_{k \in \delta(l)} \sum_{s=1}^{t - \lceil \overline{I_j TI_l} + \overline{B_{i,j} TB_{k,l}} \rceil} x_{i,s,k} \geq \sum_{s=1}^{t} x_{j,s,l}$$
$$\text{for } i, j \in \mathcal{J}, \quad ij \in \mathcal{D}, \quad l \in \mathcal{H}, \quad t \in \mathcal{T}; \qquad \text{(F4)}$$

2.2 Mechanisms

Three new mechanisms are proposed. They differ by the schedulers used in each phase of their operation. The hybrid1 mechanism uses the IP-FULL-FUZZY-2 scheduler to produce the schedules at the initial scheduling and at rescheduling steps. The hybrid2 mechanism uses the IP-FULL-FUZZY-2 to produce the initial schedule and the IPDT-2 at the rescheduling steps. The hybrid3 mechanism uses the IPDT-2 to produce the initial schedule and the IP-FULL-FUZZY-2 at the rescheduling steps.



Mechanism	Scheduler1	Scheduler2
Hybrid1	IP-FULL-FUZZY-2	IP-FULL-FUZZY-2
Hybrid2	IP-FULL-FUZZY-2	IPDT-2
Hybrid3	IPDT-2	IP-FULL-FUZZY-2

Figure 1: Hybrid mechanisms.

The operation of the hybrid mechanisms is illustrated in Figure 1. The difference between the mechanisms is the scheduler used in the initial scheduling and the one used in the rescheduling steps. The initial schedule is produced and, periodically, the rescheduling process is executed. At rescheduling steps, each task that have already started its execution, and have not finished yet, is split into two: one containing the instructions already executed, and the other containing the remaining instructions. This operation is illustrated in Figure 2, where a task T5, that is executing in the host h1, is split into tasks $T5^{'}$ and $T5^{''}$, with $T5^{'}$ containing the instructions already executed and being fixed to run in h1, and $T5^{''}$ containing the instructions to be executed in the new host if it is scheduled to a host different from h1. The arc directed from $T5$ to $T8$, now is directed from $T5^{''}$ to $T8$. The arc between $T5^{'}$ and $T5^{''}$ has weight corresponding to the amount of data to be transferred from h1 to the new host in case migration occurs. A schedule is produced to the new DAG originated from this operation. The new tasks generated by the splitting are migrated only if migrations lead to a makespan lower than that of the actual schedule.

The makespan of the actual schedule is computed by the "Makespan Calculator" component, considering the remaining instructions, of all tasks, to be executed on the new grid state. It is important to observe that the calculation of the makespan when migration occurs considers the time spent with data transfer and task migration.

Besides the three hybrid mechanisms proposed, three other are evaluated: the reactive,

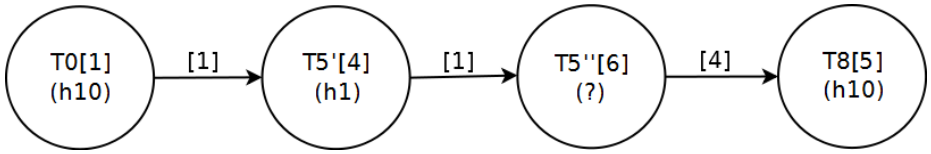


Figure 2: Splitting of a task for the rescheduling.

which uses the IPDT-2 to produce the initial schedule and the schedules at rescheduling steps; the proactive, which uses the IP-FULL-FUZZY-2 to produce a single schedule; and the non-proactive, which produces a single schedule given by the IPDT-2. The comparison of the six mechanisms considers different levels of fluctuation. The following metrics are evaluated: makespan, average number of migrations and computational demand, that includes the total processing time spent on the scheduling and rescheduling processing. The makespan is useful to analyze the quality of the schedules and migrations produced. The average number of migrations is useful to analyze the network interference caused by the mechanisms. The computational demand is useful to analyze if the schedules and migrations produced can be obtained in a feasible time in relation to the time required to the execution of the application.

3 Performance Evaluation

Simulations were conducted to compare the performance of the hybrid mechanisms with the proactive, reactive and non-proactive mechanisms. A program was written in C to simulate the operation of the six mechanisms. The schedulers (IPDT-2 and IP-FULL-FUZZY-2) were implemented using the libraries provided by the Fico Xpress Optimization Suite 7 [8] with support to multi-core. All the experiments were executed in a computer equipped with an Intel Xeon 2.27GHz 64-bit Quad-core, 6GB of RAM and running the Debian squeeze operating system.

The same inputs were provided to the six mechanisms. The DAG of the simulated application is shown in Figure 3. It is based on an application of quantum chemistry called WIEN2K [12], developed at the Vienna University of Technology [16]. The weights of the tasks in Figure 3 are in the range [112,252], on a scale of 10^5 millions of instructions, and the weights of data dependencies are in the range [112,308], on a scale of 10^6 bits. The topologies of the simulated grids were generated by the Barabasi-Albert model [1], a model used to generate network topologies similar to those found in the Internet, by the software BRITE [11]. The values of the available bandwidth were obtained randomly from a uniform distribution in the interval [10,1024]Mbps and the available processing capacities of the computers were obtained randomly from a uniform distribution in the interval [840,3300]MIPS. One grid with 75 hosts were simulated. The projected uncertainties given as input to the IP-FULL-FUZZY-2 scheduler varied from 0 to 200% and the same range was employed to simulate the fluctuations in resource availability during the execution of the application. Fluctuations in both processing capacity of hosts and available bandwidth of the links were simulated. The weights values of the grid graph were increased, corresponding to a decrease

in resource availability.

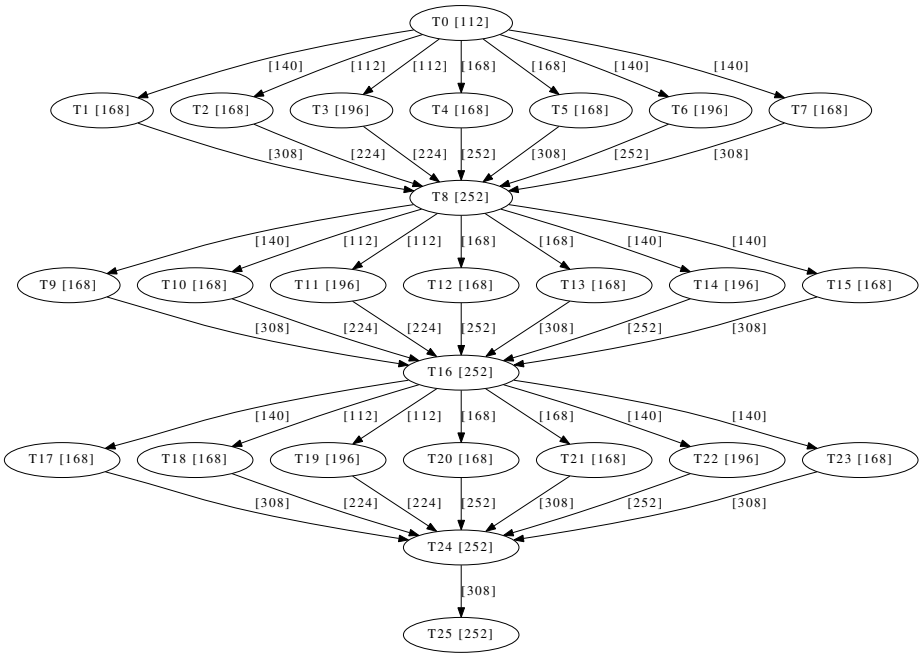


Figure 3: Simulated application.

Three metrics were evaluated: the makespan of the application, the number of migrations and the computational demands. Each point of the graphs presented in the next subsections is the average of 20 executions derived for the production of confidence interval with 95% confidence level using the replication method. Each value in the horizontal axis represents the increase of the weight values of the grid graph from a uniform distribution in the interval [0,x].

Subsection 3.1 presents results for the makespan, Subsection 3.2 presents results for the average number of migrations and Subsection 3.3 presents results for the computational demands of the mechanisms.

3.1 Makespan

Figure 4 plots the makespan reached by the mechanisms. It is possible to see that there is an increasing trend of the makespan with the increase of fluctuations. In general, the proactive mechanism did not perform better than the hybrid and reactive mechanisms. The exceptions happened with the fluctuation levels of 75%, 125% and 200% (but only fluctuation level of 200% is showed in Figure 4), in which the proactive mechanism performed better than the other mechanisms. Besides, the comparison between the proactive and the non-proactive mechanisms shows that the effectiveness of the proactive mechanism depends on the projected uncertainties given as input. This makes possible that the non-proactive scheduler IPDT-2 gives makespans lower than the proactive mechanism when no migration

occurs. This can be observed, for instance, for fluctuation levels of 50% and 100% in Figure 4, in which the projected uncertainties of processing capacity and bandwidth were 50% and 100%, respectively.

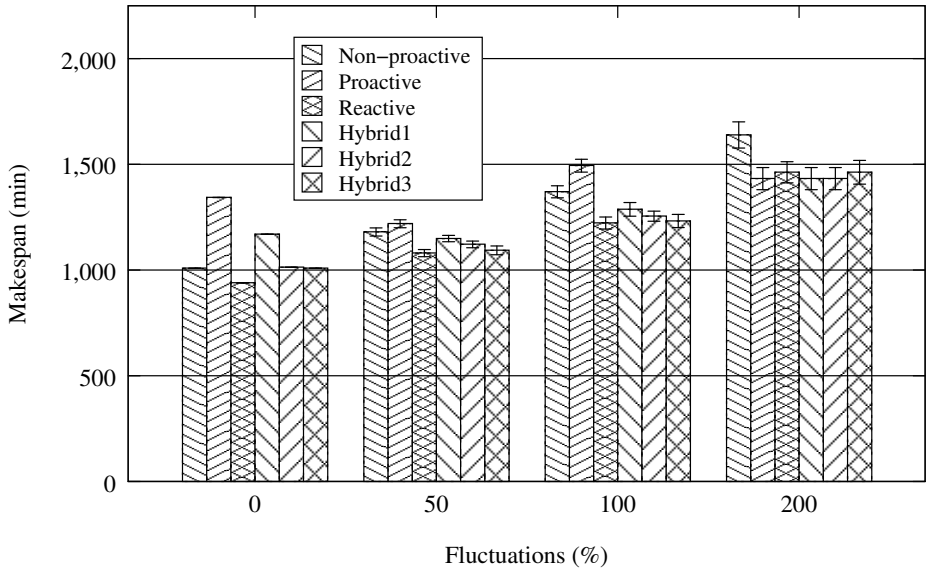


Figure 4: Makespans reached by the mechanisms when there were fluctuations in the processing capacity and available bandwidth, for the grid with 75 hosts.

The hybrid and the reactive mechanisms showed, in general, similar makespan. By comparing the schedules given by the hybrid1 and the hybrid2 mechanisms (both use the IP-FULL-FUZZY-2 as initial scheduler) to those given by the proactive mechanism, it is possible to see the importance of migrations and rescheduling. For instance, it can be seen that for fluctuation levels of 50% and 100% (Figure 4), migrations triggered by these mechanisms lowered the makespan. Results given by the hybrid1 and hybrid2 mechanisms were quite similar. The exceptions happened when there was no fluctuation (where the hybrid2 produced better result than did the hybrid1). In a similar way, results given by the hybrid3 and the reactive mechanisms were quite similar, except when it does not occur fluctuations.

3.2 Average number of migrations

Table 1 shows the average number of migrations produced by the mechanisms. The numbers in bold represent the lowest average number of migrations for each fluctuation level. The non-proactive and the proactive mechanisms are not presented in the table because they do not realize task migration. In general, no trend can be observed in relation to the number of migrations produced by mechanisms with the same initial scheduler or to the number of migrations produced by the same scheduler. However, it can be observed that the IP-FULL-FUZZY-2 produced a good initial schedule when the projected uncertainties was 200% of both processing capacity and available bandwidth, leading to a low number of

migrations. The average number is lowest when using the hybrid1 and hybrid2 mechanisms for level of fluctuations of 200%, as can be seen in the Table 1. This suggests that a way of dealing with the unstable behaviour of the IP-FULL-FUZZY-2 is by giving as input the most suitable projected uncertainties.

Mechanisms	0%	50%	100%	125%	150%	175%	200%
Hybrid1	7.00	4.00	4.75	**1.75**	4.80	4.60	0.10
Hybrid2	1.00	4.30	**3.45**	4.40	4.85	5.45	**0.00**
Hybrid3	**0.00**	2.90	4.90	2.90	**3.15**	**2.95**	2.80
Reactive	1.00	**1.85**	4.35	4.70	4.85	4.05	5.00

Table 1: Average number of migrations produced by the mechanisms when there were fluctuations in the processing capacity and available bandwidth, for the grid with 75 hosts.

3.3 Computational demand

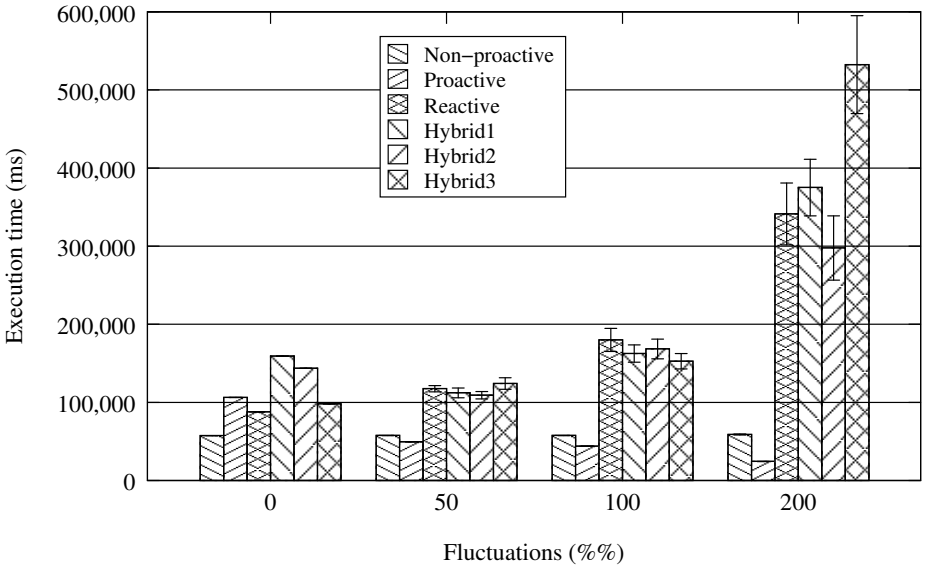


Figure 5: Computational demand of the mechanisms when there were projected uncertainties to the processing capacity and available bandwidth, for the grid with 75 hosts.

Figure 5 plots the computational demand (the time spent to produce the initial schedule and those at the rescheduling steps) of the mechanisms. It is possible to observe that the mechanisms that do not produce migrations demands less time, as it would be expected, since they run the scheduler just once. Among the mechanisms that produce migrations, the reactive mechanism showed the most stable behaviour, with a tendency to increase the demand as the level of fluctuation increases. The other mechanisms, that use the IP-FULL-FUZZY-2 in one of their steps, has also showed a tendency to increase the demand as the

level of fluctuations increases. However, the hybrid1 and hybrid2 mechanisms showed an opposite trend up to fluctuation levels of 50%. This can be explained by observing the performance of the proactive mechanism, since the higher the projected uncertainty given as input to the IP-FULL-FUZZY-2, the lower is the computational demand.

Results given by the hybrid mechanisms were quite similar, except for fluctuation levels of 200%. In this, the hybrid3 was the most demanding, followed by the hybrid1 and by the hybrid2. This can be explained by the fact that, in general, the IP-FULL-FUZZY-2 needs higher values of T_{max} to produce feasible solutions than do the IPDT-2. Therefore, the complexity of the problems solved by the hybrid3 and by the hybrid1 were higher than that demanded by the hybrid2. Besides, the difference between the hybrid1 and the hybrid3 can be explained by the fact that the initial schedule produced by the IP-FULL-FUZZY-2, with projected uncertainties of 200%, had a shorter makespan than that produced by the IPDT-2, which resulted in a lower number of rescheduling steps given by the hybrid1 than that given by the hybrid3.

4 Conclusion

Mechanisms that adopt hybrid approaches for the scheduling of application tasks on grids can benefit from the advantage of both reactive and proactive approaches. In this paper, it was proposed three new hybrid mechanisms that differ by the scheduler used to produce the initial schedule and to produce the schedules at rescheduling steps. In addition, two new schedulers were proposed, one proactive and one non-proactive, used by the mechanisms.

Results presented in previous section showed that the efficiency of the IP-FULL-FUZZY-2 can be strongly associated to the projected uncertainties given as input. It seems appropriate to consider 200% of projected uncertainties, since it led to the best performance when this value was used. Results are consistent with those found in [6]. Therefore, it is a good approach to use hybrid mechanisms that have the IP-FULL-FUZZY-2 as initial scheduler. The use of hybrid2 can be a good choice if the time to produce schedules at the rescheduling steps is important, since the IPDT-2 demands less time than does the IP-FULL-FUZZY-2 scheduler.

As future work, it is recommended the identification of the feasibility criterion of the integer linear programming problem implemented by the IP-FULL-FUZZY-2, aiming to avoid the need for re-computation of the schedules when the IP-FULL-FUZZY-2 gives unfeasible solutions.

References

[1] A. L. Barabasi and R. Albert.

[2] D. M. Batista and N. L. S da Fonseca. A Survey of Self-adaptive Grids. *IEEE Communications Magazine*, 48(7):94–100, Jul 2010.

[3] D. M. Batista, N. L. S. da Fonseca, F. Granelli, and D. Kliazovich. Self-adjusting grid networks. In *Proc of IEEE International Conference on Communications 2007 (ICC 2007)*, pages 344–349, 2007.

[4] D. M. Batista, N. L. S. da Fonseca, F. K. Miyazawa, and F. Granelli. Self-adjustment of Resource Allocation for Grid Applications. *Computer Networks*, 52(9):1762–1781, Jun 2008.

[5] D. M. Batista, A. C. Drummond, and N. L. S. da Fonseca. Robust scheduler for grid networks. In *Proc of the 24th ACM Symposium on Applied Computing*, pages 35–39, 2009.

[6] Daniel M. Batista and Nelson L. S. da Fonseca. Robust scheduler for grid networks under uncertainties of both application demands and resource availability. *Computer Networks (1999)*, 55:3–19, 2011.

[7] C. Fayad, J.M. Garibaldi, and D. Ouelhadj. Fuzzy grid scheduling using tabu search. In *Fuzzy Systems Conference*, pages 1–6, july 2007.

[8] FICO. FICO Xpress Optimization Suite 7, 2010.

[9] I. Gonzalez-Rodriguez, C.R. Vela, and J. Puente. A memetic approach to fuzzy job shop based on expectation model. In *Fuzzy Systems Conference, 2007. FUZZ-IEEE 2007. IEEE International*, pages 1 –6, july 2007.

[10] M. Imran, I. A. Niaz, S. Haider, N. Hussain, and M. A. Ansari. Towards Optimal Fault Tolerant Scheduling in Computational Grid. In *Proceedings of the ICET 2007*, pages 154–159, Nov 2007.

[11] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. Brite: An approach to universal topology generation. In *MASCOTS '01: Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, page 346, Washington, DC, USA, 2001. IEEE Computer Society.

[12] P. Blaha and K. Schwarz and G. Madsen and D. Kvasnicka and J. Luitz. WIEN2K, 2011.

[13] Jennifer M. Schopf. Ten Actions when Grid Scheduling. In Jarek Nabrzyski and Jennifer M. Schopf and Jan Weglarz, editor, *Grid Resource Management: State of the Art and Future Trends*, pages 15–23. Springer, 2003.

[14] Sathish S. Vadhiyar and Jack J. Dongarra. A Performance Oriented Migration Framework for the Grid. In *Proceedings of CCGRID'03*, pages 130–137, May 2003.

[15] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott. Proactive Process-Level Live Migration in HPC Environments. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 43:1–43:12, 2008.

[16] Marek Wieczorek, Radu Prodan, and Thomas Fahringer. Scheduling of scientific workflows in the askalon grid environment. *SIGMOD Rec.*, 34:56–62, September 2005.

[17] H.-J. Zimmermann. *Fuzzy set theory — and its applications*, pages 11–13;336–347. Kluwer Academic Publishers, Norwell, MA, USA, 4 edition, 1996.