Influence of Grid Economic Factors on Scheduling and Migration

Rafael Moreno-Vozmediano[1] and Ana B. Alonso-Conde[2]

[1] Dept. de Arquitectura de Computadores y Automática,
Universidad Complutense. 28040 - Madrid, Spain
Tel.: (+34) 913947615 - Fax: (+34) 913947527
rmoreno@dacya.ucm.es

[2] Dept. Economía Financiera, Contabilidad y Comercialización,
Universidad Rey Juan Carlos. 28032 - Madrid, Spain
Tel./Fax: (+34) 914887788
abac@fcjs.urjc.es

Abstract. Grid resource brokers need to provide adaptive scheduling and migration mechanisms to handle different user requirements and changing grid conditions, in terms of resource availability, performance degradation, and resource cost. However, most of the resource brokers dealing with job migration do not allow for economic information about the cost of the grid resources. In this work, we have adapted the scheduling and migration policies of our resource broker to deal with different user optimization criteria (time or cost), and different user constraints (deadline and budget). The application benchmark used in this work has been taken from the finance field, in particular a Monte Carlo simulation for computing the value-at-risk of a financial portfolio.

1 Introduction

A grid is inherently a dynamic system where environmental conditions are subjected to unpredictable changes: system or network failures, addition of new hosts, system performance degradation [1], variations in the cost of resources [2], etc. In such a context, resource broker becomes one of the most important and complex pieces of the grid middleware. Efficient policies for job scheduling and migration are essential to guarantee that the submitted jobs are completed and the user restrictions are met.

Most of the resource brokers dealing with job migration face up to the problem from the point of view of performance [3] [4] [5]. The main migration policies considered in these systems include, among others, performance slowdown, target system failure, job cancellation, detection of a better resource, etc.

However, there are hardly a few works that manage job migration under economic conditions [6] [7]. In this context, new job migration policies must be contemplated, like the discovery of a new cheaper resource, or variations in the resource prices during the job execution. There are a broad variety of reasons that could lead resource providers to dynamically modify the price of their resources, for example:

– Prices can change according to the time or the day. For example, the use of a given resource can be cheaper during the night or during the weekend.
– Prices can change according to the demand. Resources with high demand can increase their rates, and vice versa.
– A provider and a consumer can negotiate a given price for a maximum usage time or a maximum amount of resources consumed (CPU, memory, disk, I/O, etc.). If the user jobs violate the contract by exceeding the time or the resource quota, the provider can increase the price.

In this work we have extended the GridWay resource broker capabilities [1] [3] to deal with economic information and operate under different optimization criteria: time or cost, and different user constraints: deadline and/or budget.

This paper is organized as follows. Section 2 describes the GridWay framework. In Section 3 we analyze the extensions to the GridWay resource broker to support scheduling and migration under different user optimization criteria (time optimization and cost optimization), and different user constraints (cost limit and time limit). Section 4 describes the experimental environment, including the grid testbed and the application benchmark, taken from the finance field [12]. In Section 5 we show the experimental results. Finally, conclusions and future work are summarized in Section 6.

2 The GridWay Resource Broker

The GridWay (GW) framework [1] [3] is a Globus compatible environment, which simplifies the user interfacing with the grid, and provides resource brokering mechanisms for the efficient execution of jobs on the grid, with dynamic adaptation to changing conditions.

GW incorporates a command-line user interface, which simplifies significantly the user operation on the Grid by providing several user-friendly commands for submitting jobs to the grid ("gwsubmit") along with their respective configuration files (job templates), stopping/resuming, killing or re-scheduling jobs ("gwkill"), and monitoring the state and the history of the jobs ("gwps" and "gwhistory"). For a given job, which can entail several subtasks (i.e., an array job), the template file includes all the necessary information for submitting the job to the grid:

– The name of the executable file along with the call arguments.
– The name of the input and output files of the program.
– The name of the checkpoint files (in case of job migration).
– The optimization criterion for the job. In this implementation, we have incorporated two different optimization criteria: *time* or *cost*.
– The user constraints. The user can specify a time limit for its job (*deadline*), as well as a cost limit (*budget*).

The main components of the GW resource broker are the following:

2.1 Dispatch Manager and Resource Selector

The Dispatch Manager (DM) is responsible for job scheduling. It invokes the execution of the Resource Selector (RS), which returns a prioritized list of candidates to execute the job or job subtasks. This list of resources is ordered according to the optimization criterion specified by the user.

The DM is also responsible for allocating a new resource for the job in case of migration (re-scheduling). The migration of a job or job subtask can be initiated for the following reasons:

– A forced migration requested by the user.
– A failure of the target host.
– The discovery of a new better resource, which maximizes the optimization criterion selected for that job.

2.2 Submission Manager

Once the job has been submitted to the selected resource on the grid, it is controlled by the Submission Manager (SM). The SM is responsible for the job execution during its lifetime. It performs the following tasks:

– **Prolog**. The SM transfers the executable file and the input files from the client to the target resource
– **Submission**. The SM monitors the correct execution of the job. It waits for possible migration, stop/resume or kill events.
– **Epilog**. When the job execution finishes, the SM transfers back the output files from the target resource to the client.

3 Scheduling and Migration Under Different User Specifications

We have adapted the GW resource broker to support scheduling and migration under different user optimization criteria (time optimization and cost optimization), and different user constraints (budget limit and deadline limit). Next, we analyze in detail the implementation of these scheduling alternatives.

3.1 Time Optimization Scheduling and Migration

The goal of the time optimization criterion is to minimize the completion time for the job. In the case of an array job, this criterion tries to minimize the overall completion time for all the subtasks involved in the job.

To meet this optimization criterion, the DM must select those computing resources being able to complete the job – or job subtasks – as faster as possible, considering both the execution time, and the file transfer times (prolog and epilog). Thus, the RS returns a prioritized list of resources, ordered by a rank function that must comprise both the performance of every computing resource, and the file transfer delay. The time optimization rank function used in our RS implementation, is the following:

$$TR(r) = PF(r)(1 - DF(r)) \tag{1}$$

Where

$TR(r)$	is the Time-optimization Rank function for resource r
$PF(r)$	is a Performance Factor for resource r
$DF(r)$	is a file transfer Delay Factor for resource r

The Performance Factor, $PF(r)$, is computed as the product of the peak performance (MHz) of the target machine and the average load of the CPU in the last 15 minutes. The Delay Factor, $DF(r)$, is a weighted value in the range [0-1], which is computed as a function of the time elapsed in submitting a simple job to the candidate resource – for example, a simple Unix command, like "date" – and retrieving its output.

The resource selector is invoked by the dispatch manager whenever there are pending jobs or job subtasks to be scheduled, and also at each resource discovery interval (configurable parameter). In this case, if a new better resource is discovered, which maximizes the rank function, the job can be migrated to this new resource. To avoid worthless migrations, the rank function of the new discovered resource must be at least 20% higher than the rank function of the current resource. Otherwise the migration is rejected. This condition prevents from reallocating the job to a new resource that is not significantly better than the current one, because in this case the migration penalty time (transferring the executable file, the input files, and the checkpoint files) could be higher than the execution time gain.

3.2 Cost Optimization Scheduling and Migration

The goal of the cost optimization criterion is to minimize the CPU cost consumed by the job. In case of an array job, this criterion tries to minimize the overall CPU consumption for all the job subtasks. In this model we have only considered the computation expense (i.e. the CPU cost). However, other resources like memory, disk or bandwidth consumption could be also incorporated to the model.

To minimize the CPU cost, the resource selector must know the rate of every available resource, which is usually given in the form of price (Grid $) per second of CPU consumed. These rates can be negotiated with some kind of trade server [8] [9], and different economic models can be used in the negotiation [10]. Once the resource selector has got the price of all the resources, it returns an ordered list using the following rank function:

$$CR(r) = \frac{PF(r)}{Price(r)} \tag{2}$$

Where

$CR(r)$	is the Cost-optimization Rank function for resource r
$PF(r)$	is a Performance Factor for resource r (similar to the time-optimization scheduling)
$Price(r)$	is the CPU Price of resource r, expressed in *Grid \$ per (CPU) second*

It is important to point out that the CPU Price of the resource, $Price(r)$, can not be considered by itself as an appropriate rank function, since the total CPU cost, which is the factor to be minimized, is given by the product of the CPU price and the execution time. In this way, a low-priced but very slow resource could lead to a higher CPU consumption than another more expensive but much faster resource. So the most suitable resource is that one that exhibits the best ratio between performance and price.

As in the previous case, the resource selector is invoked by the dispatch manager whenever there are pending jobs to be scheduled, an also at each resource discovery interval. If a new better resource is discovered, whose rank function exceeds more than 20% the current resource rank value, then the job is migrated to that new resource.

3.3 Scheduling and Migration Under User Constraints (Budget/Deadline Limits)

Independently of the optimization criterion selected (time optimization or cost optimization), the user can also impose a budget and/or a deadline limit. In this case, the resource broker must be able to minimize the specific optimization user criterion (time or cost), but without exceeding the budget or deadline limits. To implement this behavior, the RS uses the same rank functions - (1) or (2), depending on the optimization criterion - to get an ordered list of candidates, but in addition, it must be able to estimate if each candidate resource meets the user budget and/or deadline limits. Otherwise, the specific resource can not be eligible for executing the given task.

These estimations have been implemented exclusively in array jobs, since we make use of the known history of the first executed subtasks to estimate the cost or deadline of the pending subtasks for every available resource.

First we analyze the model developed for job scheduling and migration under budget limit. Let:

B_L	Budget Limit imposed by the user
$B_C(r,s,t)$	Budget Consumed by resource r to execute task s, at time t
$N_C(r,t)$	Number of tasks completed by resource r at time t
$N_P(t)$	Total Number of Pending tasks (not started) at time t
$N_S(t)$	Total Number of Started tasks, but not completed, at time t

The Budget Available at time t, $B_A(t)$, is

$$B_A(t) = B_L - \sum_{\forall s, \forall r} B_C(r,s,t) \tag{3}$$

Assuming that, in average, the started tasks have been half executed, the average Budget Available per pending task at time t, $\overline{B}_A(t)$, is estimated as

$$\overline{B}_A(t) = \frac{B_A(t)}{N_P(t) + \frac{N_S(t)}{2}} \tag{4}$$

On the other hand, the average Budget per task Consumed by resource r at time t, $\overline{B}_C(r,t)$, is

$$\overline{B}_C(r,t) = \frac{\sum_{\forall s} B_C(r,s,t)}{N_C(r,t)} \tag{5}$$

If the average budget per task consumed by resource r is higher than the average budget available per pending task, i.e.,

$$\overline{B}_C(r,t) > \overline{B}_A(t) \tag{6}$$

then it is assumed that resource r is not eligible for executing a new task at the current time t.

It is obvious that this budget estimation model can not be applied when $N_C(r,t) = 0$, so it is only useful for array jobs. In fact, when the first subtasks of the array are scheduled, there is no information about the average budget consumed by each resource, and hence every available resource is considered an eligible candidate. For the subsequent subtasks, the historical information about the average budget consumed by a given resource is used to estimate whether it is likely to violate the user budget limit. If so, the resource is excluded from the candidate list.

Similarly, we have developed a model for scheduling and migration under deadline limit. Let:

D_L	Deadline Limit imposed by the user (expressed as maximum elapsed time, in seconds)
$T_C(r,s)$	Time consumed by resource r for completing task s
$N_C(r,t)$	Number of tasks completed by resource r at time t

The Remaining Time at time t, $T_R(t)$, is

$$T_R(t) = D_L - t \tag{7}$$

The average time per task consumed by resource r, $\overline{T}_C(r)$, is

$$\overline{T}_C(r) = \frac{\sum_{\forall s} T_C(r,s)}{N_C(r,t)} \tag{8}$$

If the average time per task consumed by resource r is higher than the remaining time, i.e.,

$$\overline{T}_C(r) > T_R(t) \tag{9}$$

then resource r is not eligible for executing a new task at the current time t.

As in the previous case, this time estimation model is only applicable for array jobs, when there is available historical information about the time consumed by a given resource, and hence $N_C(r,t) > 0$.

4 Experimental Environment

In this section we analyze the grid testbed and the application benchmark used in our experiments.

4.1 Grid Testbed

The main features of the computational resources employed in our grid testbed are summarized in Table 1.

Table 1. Characteristics of the machines in the experimental testbed

Hostname	Architecture / OS	Perf. Factor (peak MHz)	Delay Factor (x 100)	CPU Price (Gris $)	Slots
hydrus.dacya.ucm.es	i686 / Linux	2539	2	20	1
cygnus.dacya.ucm.es	i686 / Linux	2539	2	20	1
cepheus.dacya.ucm.es	i686 / Linux	650	3	5	1
aquila.dacya.ucm.es	i686 / Linux	662	3	5	1
belle.cs.mu.oz.au	i686 / Linux	2794	15	5	2

Resources hydrus, cygnus, cepheus, and aquila, which are located in Spain (Computer Architecture Dept., Univ. Complutense of Madrid), are uniprocessor systems, and it is assumed that they have only one available slot, i.e., only one task can be issued simultaneously to each computer. Resource belle, which is located in Australia (Computer Science Dept., Univ. of Melbourne), is a 4-processor system, and it is assumed that it has two available slots, i.e., up to two tasks can be issued simultaneously to this computer.

The client machine is located in Spain, so the delay factor for the belle system, located in Australia, is much higher than the delay factors for the systems located in Spain (hydrus, cygnus, cepheus, and aquila).

With regard to the CPU prices, it is assumed that the systems are used at European peak time, and Australian off-peak time, so that belle system exhibits a CPU price significantly lower than hydrus and cygnus systems. On the other hand, cepheus and aquila are low-performance systems, and hence their rates are also lower.

4.2 Application Benchmark

The experimental benchmark used in this work is based on a financial application, specifically, a Monte Carlo (MC) simulation algorithm for computing the Value-at-Risk (VaR) of a portfolio [12] [13]. We briefly describe this application.

The VaR of a portfolio can be defined as the maximum expected loss over a holding period, Δt, and at a given level of confidence c, i.e.,

$$Prob\{|\Delta P(\Delta t)| < VaR\} = 1 - c \tag{10}$$

where $\Delta P(\Delta t) = P(t + \Delta t) - P(t)$ is the change in the value of the portfolio over the time period Δt.

The Monte Carlo (MC) approach for estimating VaR consists in simulating the changes in the values of the portfolio assets, and re-evaluating the entire portfolio for each simulation experiment. The main advantage of this method is its theoretical flexibility, because it is not restricted to a given risk term distribution and the grade of exactness can be improved by increasing the number of simulations.

For simulation purposes, the evolution of a single financial asset, S(t), can be modelled as a random walk following a Geometric Brownian Motion [11]:

$$dS(t) = \mu S(t)dt + \sigma S(t)dW(t) \tag{11}$$

where $dW(t)$ is a Wiener process, μ the instantaneous drift, and σ the volatility of the asset.

Assuming a log-normal distribution, using the Itô's Lemma, and integrating the previous expression over a finite time interval, δt, we can reach an approximated solution for estimating the price evolution of $S(t)$:

$$S(t + \delta t) = S(t)e^{(\mu - \sigma^2/2))\delta t + \sigma\eta\sqrt{\delta t}} \tag{12}$$

where η is a standard normal random variable.

For a portfolio composed by k assets, $S_1(t)$, $S_2(t)$,..., $S_k(t)$, the portfolio value evolution can be modelled as k coupled price paths:

$$\begin{aligned} &S_1(t + \delta t) = S_1(t)e^{(\mu_1 - \sigma_1^2/2))\delta t + \sigma_1 Z_1\sqrt{\delta t}} \\ &S_2(t + \delta t) = S_2(t)e^{(\mu_2 - \sigma_2^2/2))\delta t + \sigma_2 Z_2\sqrt{\delta t}} \\ &.... \\ &S_k(t + \delta t) = S_k(t)e^{(\mu_k - \sigma_k^2/2))\delta t + \sigma_k Z_k\sqrt{\delta t}} \end{aligned} \tag{13}$$

where $Z_1, Z_2, ..., Z_k$ are k correlated random variables with covariance $cov(Z_i, Z_j) = cov(S_i, S_j) = \rho_{ij}$

To simulate an individual portfolio price path for a given holding period Δt, using a m-step simulation path, it is necessary to evaluate the price path of all the k assets in the portfolio at each time interval:

$$S_i(t + \delta t), S_i(t + 2\delta t), ..., S_i(t + \Delta t) = S_i(t + m\delta t), \quad \forall i = 1, 2, ..., k \tag{14}$$

where δt is the basic simulation time-step ($\delta t = \Delta t/m$).

For each simulation experiment, j, the portfolio value at target horizon is

$$P_j(t + \Delta t) = \sum_{i=1}^{k} w_i S_{i,j}(t + \Delta t) \quad \forall j = 1, ..., N \tag{15}$$

where w_i is the relative weight of the asset S_i in the portfolio, and N is the overall number of simulations.

Then, the changes in the value of the portfolio are

$$\Delta P_j(\Delta t) = P_j(t + \Delta t) - P(t) \quad \forall j = 1, ..., N \tag{16}$$

Finally, the VaR of the portfolio can be estimated from the distribution of the N changes in the portfolio value at the target horizon, taking the $(1-c)$ percentile of this distribution, where c is the level of confidence.

The Monte Carlo solution for VaR estimation is inherently parallel, since different simulation experiments can be distributed among different computers on a grid, using a master-worker paradigm. In order to generate parallel random numbers we use the Scalable Parallel Random Number Generators Library [14].

To adapt the application to a grid environment and allow migration, the application must save periodically some kind of checkpoint information. This adaptation is very straightforward, since saving regularly the output file, which contains the portfolio values generated by the Monte Carlo simulation up to this moment, it is sufficient to migrate and restart the application on a different host from the last point saved. The checkpoint file is saved every 1,000 simulations. To save properly the checkpoint file, the application should never be interrupted while the file is being written to disk. To avoid this problem we have protected the checkpoint file update code section against interruption signals, by masking all the system signals before opening the file, and unmasking them after closing the file. The C code is the following:

```
/*************************************************/
/*   C code fragment for saving checkpoint file    */
/*************************************************/
/* Initialize signal group including all the signals */
sigfillset(&blk_group);
/* Mask all the system signals */
sigprocmask(SIG_BLOCK, &blk_group, NULL);
/* Open checkpoint file in append mode */
chkp = fopen("output_file","a");
/* Save checkpoint information */
..........................
/* Write the last 1,000 computed values to the file */
..........................
/* Close checkpoint file */
fclose(chkp);
/* Unmask system signals */
sigprocmask(SIG_UNBLOCK, &blk_group, NULL);
/*************************************************/
```

5 Experimental Results

In this section we analyze how this implementation of the GridWay resource broker behaves under different optimization criteria and user constraints, using the grid testbed and the application benchmark described in the previous section.

5.1 Cost Optimization and Time Optimization Scheduling

First we show the behavior of the resource broker under different optimization criteria (cost and time) without considering user constraints. We assume that the user submits a 4-subtask array job, each subtask performing 500,000 simulations.

Table 2 shows the ordered resource list returned by the resource selector when the user selects the CPU cost as optimization criterion.

Fig. 1.a shows the resulting cost-optimization scheduling for the four subtasks submitted, assuming that no job migration is allowed (remember that belle system has two execution slots). As we can observe, this scheduling completes all the four tasks in a period of 22:35 mm:ss, with a CPU expense of 20,090 Grid $.

If job migration is allowed, as shown in Fig. 1.b, the resource broker gets a much better scheduling, since the CPU cost is reduced to 16,770 Grid $, and elapsed time is also reduced to 20:57 mm:ss. So, these results highlight the relevance of job migration in grid scheduling.

Notice that, for a given task, the CPU cost is computed as the product of the execution time and the CPU Price of the resource. Prolog, epilog, and migration

Table 2. Resource ranking for cost-optimization scheduling

Ranking	Host	Cost-optimization rank function value
1	belle.cs.mu.oz.au	559
2	aquila.dacya.ucm.es	132
3	cepheus.dacya.ucm.es	130
4	cygnus.dacya.ucm.es	127
5	hydrus.dacya.ucm.es	127

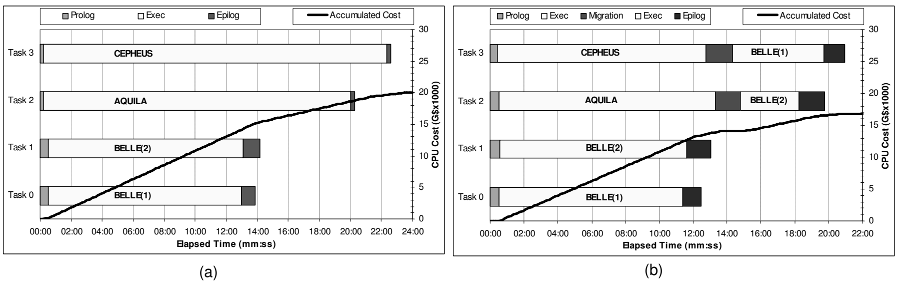

Fig. 1. Cost-optimization scheduling (no user constraints). a) Without migration. b) With migration

Table 3. Resource ranking for time-optimization scheduling

Ranking	Host	Time-optimization rank function value
1	cygnus.dacya.ucm.es	2488
2	hydrus.dacya.ucm.es	2488
3	belle.cs.mu.oz.au	2375
4	aquila.dacya.ucm.es	642
5	cepheus.dacya.ucm.es	630

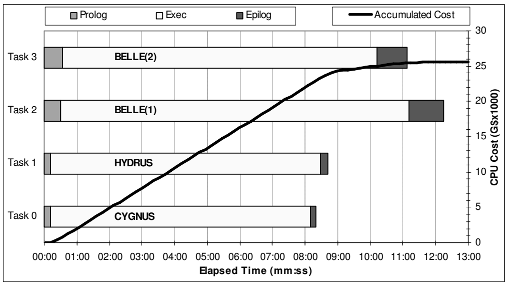

Fig. 2. Time-optimization scheduling (no user constraints)

times are not contemplated for computing the CPU cost, since these periods are used just for file transmission, and no CPU expense is considered.

Table 3 shows the ordered resource list returned by the resource selector when the user selects the time as optimization criterion, and Fig. 2 shows the resulting scheduling. As we can observe, the time-optimization criterion gets an important time reduction, since the four tasks are completed in 12:14 mm:ss, but at the expense of increasing the CPU cost up to 25,610 Grid $. In this case, scheduling with or without migration leads to similar results, since the rank functions of both cygnus and hydrus systems do not exceed more than 20% the rank function of belle system, so job migration is discarded.

5.2 Scheduling and Migration Under User Constraints

In this section we examine the effects of user constraints over scheduling. We first analyze how a budget limit can modify the resulting scheduling.

Fig. 3.a shows the time-optimization scheduling of a 8-subtask array job (each subtask performing 500,000 simulations), without user constraints. As we can observe, initially tasks 0-5 are issued using the six available resources, and tasks 6-7 stay pending until some resource is available. When cygnus and hydrus systems complete their respective tasks, the two pending tasks 6-7 are assigned to these resources. Furthermore, when belle system completes its two tasks and becomes available, tasks 4 and 5 in aquila and cepheus systems are migrated to belle, since it exhibits a higher rank function. This scheduling can be considered optimal in time, and it takes 17:55 mm:ss, consuming a CPU cost of 53,500 Grid $.

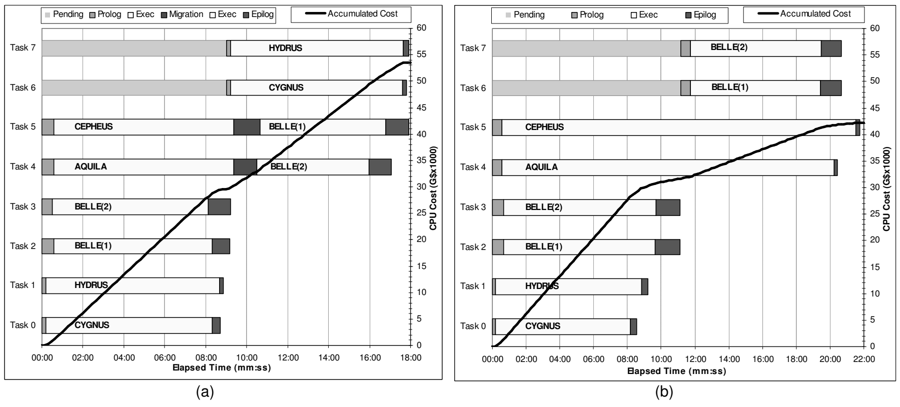

Fig. 3. Time-optimization scheduling. a) No user constraints. b) Scheduling under budget limit

Now, we are going to consider a budget limit $B_L = 45{,}000$ Grid \$. With this user constraint, the resulting scheduling is shown in Fig. 3.b. When cygnus and hydrus systems complete their respective tasks, they are not eligible to execute the two pending tasks 6 and 7, since the average budget consumed by these two systems exceeds the average available budget per task. Instead of selecting these two expensive systems, the resource broker waits until belle system is available, and then the two pending tasks are assigned to it. This scheduling keeps the budget limit imposed by the user – the overall CPU consumption is 42,190 Grid \$ –, but it is not optimal in time, since it takes 21:45 mm:ss to complete all the tasks.

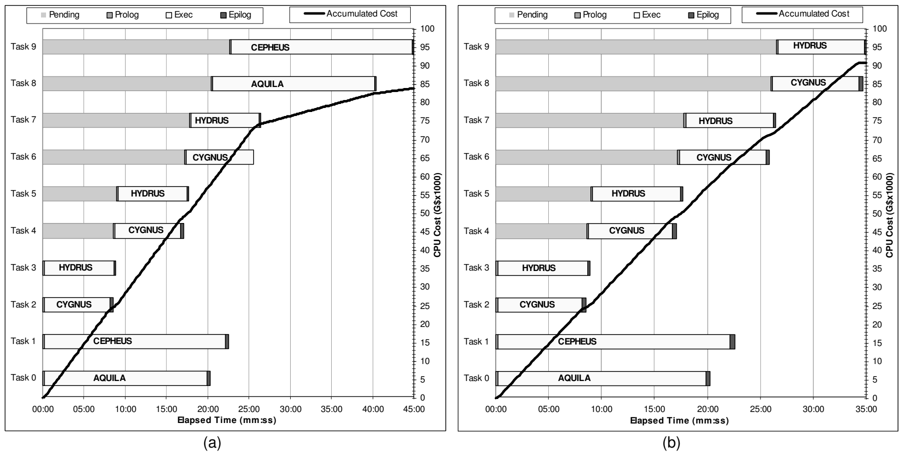

Fig. 4. Cost-optimization scheduling. a) No user constraints. b) Scheduling under deadline limit

Finally, we analyze how a deadline limit can also alter the resulting the scheduling. To observe clearly this effect, we have considered a 10-subtask array job, with cost-optimization scheduling. In this experiment, we use only four systems from our testbed: hydrus, cygnus, aquila and cepheus (belle is not used in this case). Fig. 4.a shows the resulting scheduling without user constraints. In this scheduling, tasks are allocated to resources as soon as they become available. This scheduling takes almost 45 min. and the CPU expense is 83,925 Grid $.

Fig. 4.b. shows the cost-optimization scheduling of the 10 subtasks with a deadline limit $D_L = 35$ min. imposed by the user. In this scheduling, tasks 8 and 9 are not assigned to the systems aquila and cepheus when these resources become available, because the average time consumed by these resources in the previous tasks exceeds the remaining time until the user deadline. In this case, these two pending tasks are delayed until the hydrus and cygnus are available. The resulting scheduling keeps the deadline limit imposed by the user, since it takes 35 min to complete all the subtasks, but the CPU cost goes up to 91,015 Grid $.

6 Conclusions and Future Work

In this paper we have adapted the scheduling and migration strategies of our resource broker to deal with economic information and support different user optimization criteria (time or cost), and different user constraints (deadline and budget). This implementation of the resource broker uses different rank functions to get an ordered list of resources according to the optimization criteria specified by the user, and performs different time and cost estimations based on historical information to discard those resources that are likely to violate the user constraints. The results also show that migration is essential to adapt resource mapping to changing conditions on the grid, and guarantee that optimization criteria and user constraints are met.

In a future work, we plan to incorporate to the brokering model the cost of other physical resources, like the cost of the memory and disk space used by the application, the cost of the network bandwidth consumed, etc. The incorporation of these new elements will lead to the development of new scheduling and migration policies based on alternative optimization criteria, as well as new cost estimations to meet user constraints.

References

1. Huedo, E., Montero, R.S., Llorente, I.M.: An Experimental Framework For Executing Applications in Dynamic Grid Environments. NASA-ICASE T.R. 2002-43 (2002)
2. Abramson, D., Buyya, R., Giddy, J.: A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker. Future Generation Computer Systems Journal, Volume 18, Issue 8, Elsevier Science (2002) 1061-1074
3. Huedo, E., Montero, R.S., Llorente, I.M.: An Framework For Adaptive Execution on Grids. Intl. Journal of Software - Practice and Experience. In press (2004)

4. Allen, G., Angulo, D., Foster, I., and others: The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment. Journal of High-Performance Computing Applications, Volume 15, no. 4 (2001)
5. Vadhiyar, S.S., Dongarra, J.J.: A Performance Oriented Migration Framework For The Grid. http://www.netlib.org/utk/people/JackDongarra/papers.htm (2002)
6. Moreno-Vozmediano, R., Alonso-Conde, A.B.: Job Scheduling and Resource Management Techniques in Economic Grid Environments. Lecture Notes in Computer Science (LNCS 2970) - Grid Computing (2004) 25-32
7. Sample, N., Keyani, P., Wiederhold, G.: Scheduling Under Uncertainty: Planning for the Ubiquitous Grid. Int. Conf. on Coordination Models and Languages (2002)
8. Buyya, R., Abramson, D., Giddy, J.: An Economy Driven Resource Management Architecture for Global Computational Power Grids. Int. Conf. on Parallel and Distributed Processing Techniques and Applications (2000)
9. Barmouta, A. and Buyya, R., GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration. 17th Annual Int. Parallel and Distributed Processing Symposium (2003)
10. Buyya, R., Abramson, D., Giddy, J., and Stockinger, H.: Economic Models for Resource Management and Scheduling in Grid Computing The Journal of Concurrency and Computation, Volume 14, Issue 13-15 (2002) 1507-1542
11. Jorion, P.: Value at Risk: The Benchmark for Controlling Market Risk McGraw-Hill Education (2000)
12. Alonso-Conde, A.B., Moreno-Vozmediano, R.: A High Throughput Solution for Portfolio VaR Simulation. WSEAS Trans. on Business and Economics, Vol. 1, Issue 1, (2004) 1-6
13. Branson, K., Buyya, R., Moreno-Vozmediano, R., and others: Global Data-Intensive Grid Collaboration. Supercomputing Conference, HPC Challenge Awards (2003)
14. Mascagni, M., Srinivasan, A.: Algorithm 806: SPRNG: a scalable library for pseudorandom number generation. ACM Trans. on Mathematical Software (TOMS), Volume 26, Issue 3, September (2000) 436-461