Economic Model for Grid Resource Sharing

Bernard Miegemolle
LAAS - CNRS
Toulouse, France

Rémi Sharrock
LAAS - CNRS
Toulouse, France

Thierry Monteil
LAAS - CNRS
Toulouse, France

Abstract ***The main objective of grid computing is to aggregate an important number of geographically distributed heterogeneous resources, owned by different organizations, in order to enable their utilization in a transparent way by the grid users.***

In such an environment, resource owners and users have different objectives and strategies that have to be satisfied. Because they often need a complete state information about the grid, traditional approaches can not be used in this context. An approach based on economic models used in human economy can be a solution to deal with decentralization and heterogeneity.

In this paper, an economic model is proposed. First, its characteristics are given, and a mathematical model is defined. Then, the performances of an algorithm implementing this model are studied in order to evaluate its usability in a grid environment.

Keywords: Grid computing, Resource management, Economic model, Scheduling

1 Introduction to Economic Models

1.1 Objectives

Grid computing enables to solve large-scale advanced scientific and engineering problems by aggregating an important number of geographically distributed resources owned by different organizations [1, 2]. The heterogeneity, the dynamic nature and the structure of grid environments introduce a significant number of challenging issues. Systems like *Globus* [3] or *Legion* [4] address many of these issues, but resource managing and trading is still an important investigation field concerning grid computing.

In grid environments, resource owners must be distinguished from resource users [5, 6]. The grid clients are in quest for resources that best fit their requirements in terms of power, load, availability, etc. Resources are available on a grid only because their owners accept to share them. This can be possible in the long term only if the resource owners are offered an incentive to share them.

The objective of economic models is to permit both *producers* (resource owners) and *consumers* (resource users) to meet their requirements. They encourage producers to share their resources, and also consumers to think about a compromise solution between computational time and cost, depending on the desired quality of service.

Economic-based resource managers help regulating the supply and demand for resources. No central agent is needed during negotiations, and resource owners and users may make their own choices at every moment in order to maximize their benefits.

1.2 General Functioning

Resource managers based on economic models provide all features, such as negotiation protocols and scheduling systems, that are necessary for consumers and providers to get in contact and lead a transaction for a resource.

Major economic models taken from human economy can be considered to manage resources that are present on a computational grid [6]. The choice of one of these models, such as *commodity market model*, *bargaining model* or *auction model*, determines the resource allocation policy.

Different resources may be charged to the user: the consumed CPU time, the quantity of memory used, the amount of required storage, the network bandwidth consumption, the received signals and context switches, and also the softwares and libraries used.

According to the chosen economic model, a resource price can be constant or can vary with time. In this case, many parameters have to be considered to establish a resource price: the usage duration, the usage timing (peak, off-peak), supply and demand, the loyalty of consumers, the resource power, the physical cost of the resource and the resource overload.

1.3 Related Work and Contributions

An early example of pricing computational resources was reported by Sutherland in 1968 [7]. At Harvard University, the PDP-1 computer was shared by all students and faculty members for experiments, and an auction system was set up to reserve time on it. Users could reserve a period of dedicated access by giving the price they were willing to pay.

The appearance of computer networks introduced a new approach in applying economics to resource allocation schemes [8]. Indeed, the multiplication of resource providers leaded to replace the simple pricing models used until then by real markets, with producers and consumers.

Thus, Ferguson et al. demonstrate in [5] that load-balancing algorithms based on concepts drawn from microeconomics are efficient for managing the allocation and sharing of computational resources in a distributed system. Such competitive algorithms enable to deal with the system complexity by converting a complex global allocation system into a set of smaller independant ones.

Spawn [9] is a market-based computational system that utilizes idle resources in a distributed network of heterogeneous high-performance workstations in order to enable unused resources to be accessed by users. CPU time slices are sold using an auction process to end-users.

Another example of system that relies on auction processes is in POPCORN [10]. The POPCORN project provides a market-based mechanism for trade in CPU time. Its purpose is to motivate processors to provide their idle CPU cycles for other peoples' computations. Thus selling their CPU cycles allows them to consume CPU time of other people.

In [6], Buyya describes the Nimrod-G grid resource broker. This broker uses a computational economy driven architecture for managing resources and scheduling task farming on large-scale distributed systems. Some different deadline and budget constrained algorithms are incorporated in order to demonstrate the effectiveness of an economic paradigm for managing resources.

Economic paradigms are not only used to schedule applications over execution hosts. Thus, Mariposa [11] is a distributed data base system that uses a distributed microeconomic approach to deal with query execution and storage management. Queries submitted by user applications use a tender/contract-net model to select the server sites that will process the result of those queries.

This paper focuses on grids used in the Application Service Provider (ASP) context, in which resources are dedicated to the execution of submitted applications. In [12], the authors state that commodities markets are a better choice in terms of price stability, market equilibrium and overall efficiency, for controlling grid resources than auctions. Thus this paper proposes a commodity-market based model, and introduces a pricing model for resources that includes several parameters, such as the power of the processors used, the usage duration and the usage timing (peak and off-peak periods). The economic model proposed in this paper allows to schedule concurrent applications composed of several parallel and homogeneous tasks. Users that submit an application can choose to give greater importance to the cost of the application or to its execution time. The model not only selects resources that best fit users requirements, it also chooses the best moment to execute submitted applications. Indeed, according to the compromise between application execution time and cost, and also considering peak and off-peak periods (in terms of load and prices with special offers for example), executing an application at the time it is submitted is not necessarily the best solution. In some cases, it is better, for the user needs, to postpone the execution of the application he submits.

2 Definition of an Economic Model

In this section, an economic model is defined. This model aims to manage resources available on a grid. First, the model characteristics are given. Then the definition of variables and the mathematical model are proposed.

2.1 Model Characteristics

This model is designed to be a grid resource exchange model: resource providers are also resource consumers. The resources they share enable them to access all resources that are available on the computational grid. We assume that this model targets grids used in an ASP context, for which all resources are dedicated. This allows to have a complete control over their use, and to be able to predict their load at every time.

The economic model defined in this paper is based on the *commodity market model*. The prices of the different resources are set in advance, but can vary with time. The evolution of the prices is known in advance, so that it can be considered when the resources are chosen by the user. Note that the user is charged proportionally to the amount of resources he consumes.

The money used in this model can be assimilated to tokens. Consumers pay an amount of tokens to execute their applications on a set of resources. The owners of these resources get back these tokens, and will use them whenever they need to execute an application on the grid. Note that producer prices need to be competitive in order to enable them to earn tokens for their own resource consumption.

The resources considered in the model are the ones related to the processors used. An application is divided into several tasks, which are identical (i.e. they consume the same amount of resources). Each task will be mapped to one and only one processor, and will occupy it during a given time. Each available processor can execute several tasks. All the tasks of an application are synchronous, and start simultaneously. Note that even if tasks are identical to each other, the CPU time they consume can vary from one task to another depending on the power of the processors that execute them.

All tasks of an application can communicate with each other through a network. These communications are synchronous. Nevertheless coarse-grain applications are considered. For these kinds of applications, time spent in communication and synchronization is neglected compared to computation time.

The economic model defined in this section proposes to charge the user with the following resources : the consumed CPU time to execute each task, the number of processors used, and the power of the processors used.

The goal of a scheduler using this economic model to manage resources present on a grid is to choose the best processors to execute the tasks of an application. The scheduler also has to choose the starting date of the application execution. These choices must lead to a good compromise between the application execution time and its cost.

2.2 Definition of Variables

The variables used to correctly deal with this problem can be separated into two categories: the problem data and the unknowns.

2.2.1 The Problem Data

These variables are known at the beginning of the schedule. They can be given by the system, or given by the user.

Some of theses data are constant:

- P^R = power of the reference processor. When the user performs a request, he gives an estimation of application duration time corresponding to a reference processor. This value can be estimated with *SPECint*, *SPECfp* or *Mflops* values.
- T^R = application duration time requested by the user on the reference processor (sum of all tasks computations).
- T^C = the average time spent in communications for a task, depending on the number of exchanged messages, on their size, and on the network bandwidth.
- A = number of tasks of the application.
- N_a^P = number of available processors.
- N_{min}^P and N_{max}^P = bounds requested by the user on the number of processors to use.
- $P_{j;1 \le j \le N_a^P}$ = power of each available processor.
- C^T, C^N and C^P = weighting coefficient of the cost corresponding respectively to the consumed CPU time, the number of processors used and the power of the processors used.

The resources price and the load of the different processors may vary with time. By consequence, it can be better for a task to defer its execution, and thus to take advantage of a drop of processors price or load (off-peak periods).

The time is divided into N^K intervals $[D_k, D_k + \Delta T[_{1 \le k \le N^K}$, each one beginning at the date D_k and lasting ΔT time units, with $D_1 = 0$. The total time period can cover a day, a week or more.

The load and the price of the processors vary with time, but they are supposed to be constant during each time interval. These two variables are noted:

- $L_{j,k;1 \le j \le N_a^P, 1 \le k \le N^K}$ = load on the processor j during the interval $[D_k, D_k + \Delta T[$. This load is entirely predictable as the grid resources are dedicated. It depends only on the applications that are already scheduled at the date $D_1 = 0$, and is updated whenever a new application is scheduled.
- $C^P_{j,k;1 \le j \le N_a^P, 1 \le k \le N^K}$ = coefficient applied to the cost corresponding to the power of the processor j during the interval $[D_k, D_k + \Delta T[$. It enables the resource owners to locally adjust the cost of a processor, in order to offer them a way to control supply and demand on their processors.

The load of the different processors is known at the beginning of the schedule, but it must be updated during the schedule process, since launching new tasks on the available processors has an influence on their load.

2.2.2 The Unknowns

These variables are calculated from the problem data in order to minimize the target function given below. They give some information about the schedule of the application tasks over the available processors. Their expressions are given in section 2.3.

- $u_{j;1\leq j\leq N_a^P}$ = number of tasks of the application to run on the processor j.
- N_u^P = number of processors used. This variable is deduced from the u vector, and must be bounded by N_{min}^P and N_{max}^P.
- k^s = the time interval during which the application execution will start. Thus, the starting date of the application will be: $t^s = (k^s-1)*\Delta T$ (the tasks are considered to start and end at the beginning of time intervals).
- C_{App} = the application cost.
- T_{App} = the end date of the application, i.e. the date when the last task of the application finishes.

2.2.3 Target Function

The target function is a compromise between the application cost and its termination date. These two parameters are weighted by the W^C and W^T coefficients which values are chosen according to what is wished to favor.

Note that the application execution time itself is not necessarily minimized. The model only considers what the users see, i.e. the termination date of their applications, corresponding to their execution times added to their starting dates.

Therefore, target function is:

$$Target = W^C \cdot C_{App} + W^T \cdot T_{App} \qquad (1)$$

The u vector and the application starting date t^s will be calculated in order to minimize this target function.

2.3 Mathematical Model

In this section, the mathematical model is explained.

2.3.1 Calculation of the Application Execution Time

The first step is to estimate the time needed by a task to be executed. All the application tasks are supposed to start simultaneously, at the date t^S, i.e. at the beginning of the interval $[D_{k^S}, D_{k^S}+\Delta T[$. The number of time intervals needed to execute a task, from the $[D_{k^S}, D_{k^S}+\Delta T[$ interval, on a given processor, must be evaluated.

Let Q be the quantity of computations that must be done by each task. This variable represents something like the number of operations that each task has to accomplish. It is given by the user, with the T^R and P^R parameters.

$$Q = \frac{T^R \cdot P^R}{A} \qquad (2)$$

The maximum quantity of computations that can be done during an interval $[D_k, D_k+\Delta T[$ on a processor j depends on the power of the processor (P_j), and on the load of this processor during the considered time interval ($L_{j,k}$). Its expression is:

$$q_{j,k} = \Delta T \cdot P_j \cdot (1 - L_{j,k}) \qquad (3)$$

Let K_{j,k^S} be the minimal number of time intervals needed to execute a task, from the $[D_{k^S}, D_{k^S}+\Delta T[$ interval, on the processor j. Its expression is:

$$K_{j,k^S} = min\{K / \sum_{k=k^S}^{k^S+K-1} q_{j,k} \geq Q\} \qquad (4)$$

The complete expression of K_{j,k^S} is deduced from equations (4), (2) and (3):

$$K_{j,k^S} = min\{K / \sum_{k=k^S}^{k^S+K-1} (1-L_{j,k}) \geq \frac{T^R \cdot P^R}{P_j \cdot A \cdot \Delta T}\} \qquad (5)$$

By this way, the value of K_{j,k^S} can be explicitly calculated, by successive iterations. An approximation is made to calculate the time needed by a processor to execute several tasks of a same application: these tasks are supposed to be executed sequentially, in order to use the result given by equation (5). This is correct if all the tasks of the application have no communication. The hypothesis of coarse-grain applications justifies this approximation.

Let $k_{i,j}^S$ be the starting interval of the task i on the processor j. For each processor j used (i.e. $u_j \neq 0$), the tasks are numbered from 1 to u_j. The first task ($i = 1$) starts during the k^S interval (beginning of the application execution), each other task starts when the previous one ends.

$$\begin{cases} k_{1,j}^S = k^S \\ \forall i \in [2, u_j], k_{i,j}^S = k_{i-1,j}^S + K_{j,k_{i-1,j}^S} \end{cases} \qquad (6)$$

The interval during which the application execution will end corresponds to the last interval during which a task ends. Let k^E be this interval. Its expression is:

$$k^E = \max_j\{k^S_{u_j,j} + K_{j,k^S_{u_j,j}}\} \tag{7}$$

Actually, the application spends some time in communications between its tasks. Only coarse-grain applications are considered in this paper, which allows to suppose that this communication time does not have a significant impact on the k^E calculation.

Without considering the tasks communication time, the application running time is thus:

$$t^E = (k^E - k^S) \cdot \Delta T \tag{8}$$

If tasks communication time is taken in account, the application running time, noted t^{EC} becomes:

$$t^{EC} = t^E + T^C = (k^E - k^S) \cdot \Delta T + T^C \tag{9}$$

This time is associated with a number of time intervals noted k^{EC} :

$$k^{EC} = \frac{t^{EC}}{\Delta T} + 1 \tag{10}$$

Finally, the application will finish at T_{App}:

$$T_{App} = t^S + t^{EC} \tag{11}$$

This time can be calculated by successive iterations, and corresponds to the first criteria of the target function (1).

2.3.2 Application Cost Calculation

In this section, the calculation of the application cost is explained. This cost depends on three resources :

- consumed CPU time: the CPU time needed to execute each task,
- number of processors used,
- power of the processors used.

Cost Due to the Consumed CPU Time

The CPU time consumed by a task depends on the quantity of computations Q that has to be done. The total CPU time consumed by the application on a given processor j corresponds to the sum of the CPU time consumed by each task launched on this processor (2):

$$T_j = Q \cdot \frac{u_j}{P_j} = \frac{T^R \cdot P^R}{A} \cdot \frac{u_j}{P_j} \tag{12}$$

In this case, the cost due to the consumed CPU time is:

$$C^T_{App} = \frac{T^R \cdot P^R}{A} \cdot \sum_{j=1}^{N^P_a} \frac{u_j}{P_j} \tag{13}$$

Cost Due to the Number of Processors Used

This cost is given by the following expression:

$$C^N_{App} = N^P_u \tag{14}$$

N^P_u represents the number of processors used, i.e. the number of processors for which $u_j \neq 0$.

Cost Due to the Power of the Processors Used

The cost due to the power of the processors used takes in account the coefficients enabling to adjust locally the cost of each processor, in order to set up off-peak periods for some specified processors. An average of these coefficients is taken during the intervals of time implied in the execution of the application. The expression of this cost is:

$$C^P_{App} = \frac{t^E}{t^{EC}} \cdot \sum_{j=1}^{N^P_a} \Big[u_j \cdot P_j \cdot (\frac{1}{t^{EC}} \cdot \sum_{k=k^S}^{k^{EC}} C^P_{j,k} \cdot \Delta T) \Big] \tag{15}$$

The average of the cost coefficients $C^P_{j,k}$ is calculated on a time interval that includes the tasks computation and communication times. Indeed, actually, the tasks communicate during their computation, so that all time intervals, not only from k^S to k^E but from k^S to k^{EC}, are concerned.

The result obtained is multiplied by $\frac{t^E}{t^{EC}}$ in order to charge the user only with the computation time, not with the communication time.

Global Cost of the Application

The global cost of the application is a weighted sum of the different resources cost:

$$C_{App} = C^T \cdot C^T_{App} + C^N \cdot C^N_{App} + C^P \cdot C^P_{App} \tag{16}$$

3 Implementation and Performances

3.1 Implementation of the Economic Model

The problem is an optimization problem, or more exactly a non-linear problem, with discrete variables. A genetic algorithm has been designed and developed in order to solve it.

By minimizing the objective function of the economic model, a schedule is obtained. This schedule maps each task of an application with the processor used to run it, and determines the beginning date of the application.

3.2 Simulation Context

This part focuses on the usability of the algorithm in grid environments, in terms of convergence and completion time. These simulations use a grid that is composed of 1000 processors. Some applications are already running on the grid, or are scheduled for a future starting date which is known.

The load of each processor and its variation in the future are supposed to be known, as stated in section 2.1.

The objective of these simulations is to measure the algorithm performances for the computing of a schedule (processors and starting date) of a new application to be launched.

The application considered here is composed of 40 tasks to schedule over the 1000 processors of the grid, in function of their power and availability, in order to find a solution that best fits the user requirements in terms of cost and achievement speed.

3.3 Algorithm Performances

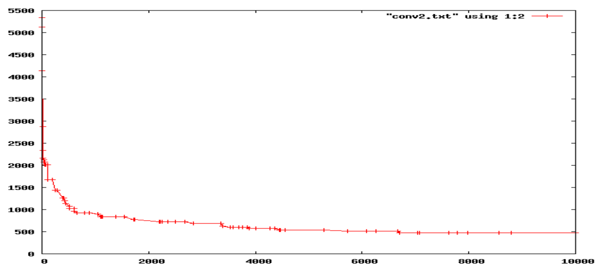
The genetic algorithm has been developed using the *C* programming language. Several optimizations have been made in its source code (improvement of the memory usage, multi-threading, etc). The algorithm was run on a host powered by an Intel bi-processor (2 * 3.2 GHz).

Figure 1 shows the convergence of the algorithm in three different situations: if the user only wants to minimize the application termination date whatever its cost may be (fig. 1(a)), if he wants to minimize only the application cost whatever the termination date may be (fig. 1(b)), or if he wants to minimize a compromise between these two values (fig 1(c)).

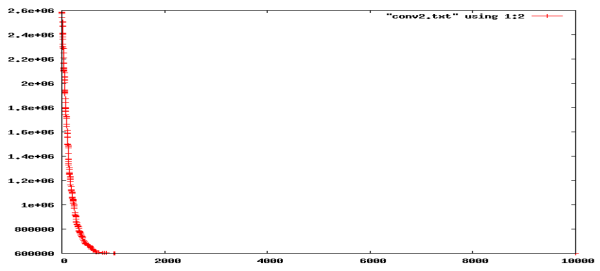
The algorithm converges faster in the cases that only one of the two variables has to be minimized. If a compromise between the application termination date and its cost has to be considered, the algorithm takes more time, and 5000 iterations are needed to obtain a correct result.

Under these conditions, the algorithm completion time for the schedule of 40 tasks of an application over 1000 processors is less than 5 seconds.

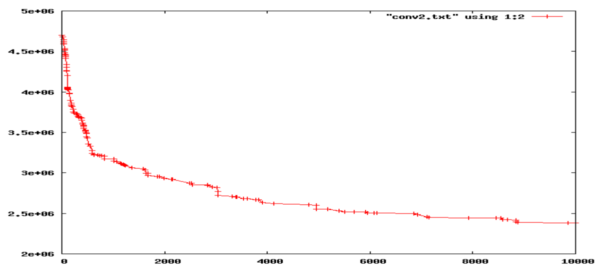
Actually, the 1000 processors will be divided into different domains, which will be in competition to run the applications. In this case, the algorithm can be run in parallel for each domain, and then the best domain will be chosen. This solution decreases the complexity of the algorithm and so its completion time.



(a) Termination date minimization



(b) Cost minimization



(c) Minimization of a compromise between cost and termination date

Figure 1: Algorithm convergence (target value in function of the algorithm iterations number)

4 Conclusion and Future Directions

Scheduling is one of the main problems of grid environments. The dynamicity of the grid in terms of availability of its resources and load on them, and also the problems of heterogeneity and scalability, can not be easily managed by traditional approaches.

This paper proposes to deal with the scheduling problem by using an economical approach, similar with the one used in human economy.

In this paper, an economic model is proposed. It corresponds to an optimization problem, which enables to find a schedule for the tasks of an application over different processors. In order to solve this problem, a genetic algorithm has been designed and tested. Its completion time is quite acceptable, which means that it can be used in grid environments in order to manage their resources.

This type of algorithm can be used in grid environments with reservation mechanisms. The cost of a reservation can be calculated with the target function of the economic model described in this paper. This cost depends on the duration, the number of processors and the type of machines. Each user can have an amount of a fictive money (tokens) that he will spend at each reservation with a value calculated by the target function. It will help regulate conflicts of utilization. An experiment will be done between three academic laboratories and two industrial partners.

Some improvements can be made in the economic model. Other interesting resources can be added to the model, like the memory usage, or the network bandwidth consumption. In the proposed model, all the tasks of a same application have to start simultaneously and need the same computation time. A complete model would take in account non-synchronous applications, so that starting each task at different moments would optimize the execution time or the cost of the application. The spirit of the model would be the same. It would only create some new constraints to satisfy, and more cases to study in the implementation. It would also be interesting to group together processors by domains, in order to optimize the communications between the different tasks of the applications. Finally, the concept of classes of service could be introduced in the model. This concept, presented in [13], introduces several levels of quality of service (QoS), and thus enables to guarantee that a QoS level chosen by the user will be respected. In the economic model, the chosen class could be charged to the user proportionally to the required level of QoS.

References

[1] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure.* San Francisco, Morgan Kauffman, 1999.

[2] I. Foster, C. Kesselman and S. Tueke. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations.* Intl. J. Supercomputer Appl., 2001.

[3] I. Foster and C. Kesselman. *Globus: A Metacomputing Infrastructure Toolkit.* International Journal of Supercomputer Applications and High Performance Computing, 11(2): 115-128, 1997.

[4] A. Grimshaw, W. Wulf, J. French, A. Weaver and P. Reynolds. *Legion: The Next Logical Step Toward a Nationwide Virtual Computer.* Computer Science Technical Report, University of Virginia, CS 94-21, June, 1994.

[5] D. Ferguson, Y. Yemini and C. Nikolaou. *Microeconomic Algorithms for Load Balancing in Distributed Computer Systems.* Proceedings of the 8th International Conference on Distributed Computing Systems, June 13-17 1988.

[6] R. Buyya. *Economic-based Distributed Resource Management and Scheduling for Grid Computing.* PhD Thesis, Monash University, 2002.

[7] I. E. Sutherland. *A futures market in computer time.* Communications of the ACM, June 1968.

[8] J. Nakai. *Pricing Computing Resources: Reading Between the Lines and Beyond.* Technical Report, NASA Ames Research Center, January 2002.

[9] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart and W. S. Stornetta. *Spawn: A Distributed Computational Economy.* IEEE Transactions on Software Engineering, February 1992.

[10] O. Regev and N. Nisan. *The POPCORN Market - an Online Market for Computational Resources.* Proceedings of the First International Conference on Information and Computation Economies, October 25-28 1998.

[11] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah and C. Staelin. *An Economic Paradigm for Query Processing and Data Migration in Mariposa.* Proceedings of 3rd International Conference on Parallel and Distributed Information Systems, September 28-30 1994.

[12] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan. *G-commerce: Market Formulations Controlling Resource Allocation on the Computational Grid.* Proceedings of 15th International Parallel and Distributed Processing Symposium, April 23-27 2001.

[13] P. Pascal, S. Richard, B. Miegemolle, and T. Monteil. *Tasks Mapping with Quality of Service for Coarse Grain Applications.* 11th International Euro-Par Conference (Euro-Par 2005), Lisbon, Portugal, Aug. 30 - Sept. 2 2005, Lecture Notes in Computer Science 3648 Parallel Processing, Springer, 2005.