Promoting Cooperation in Selfish Computational Grids

Krzysztof Rzadca[(1)(2)] Denis Trystram[(1)]
rzadca@imag.fr

(1) LIG, Grenoble University
51, avenue Jean Kuntzmann
38330 Montbonnot Saint Martin
France

(2) Polish-Japanese Institute of Information Technology
Koszykowa 86
02-008 Warsaw, Poland

Abstract

In distributed computing the recent paradigm shift from centrally-owned clusters to organizationally distributed computational grids introduces a number of new challenges in resource management and scheduling. In this work, we study the problem of *Selfish Load Balancing* which extends the well-known Load Balancing (LB) problem to scenarios in which each processor is concerned only with the performance of its local jobs. We propose a simple mathematical model for such systems and a novel function for computing the cost of the execution of foreign jobs. Then, we use the game-theoretic framework to analyze the model in order to compute the expected result of LB performed in a grid formed by two clusters. We show that, firstly, LB is a socially-optimal strategy, and secondly, for similarly loaded clusters, it is sufficient to collaborate during longer time periods in order to make LB the dominant strategy for each cluster. However, we show that if we allow clusters to make decisions depending on their current queue length, LB will never be performed. Then, we propose a LB algorithm which balances the load more equitably, even in the presence of overloaded clusters. Our algorithms do not use any external forms of compensation (such as money). The load is balanced

only by considering the parameters of execution of jobs. This analysis is assessed experimentally by simulation, involving scenarios with multiple clusters and heterogeneous load.

1 Introduction

Computational grids [15], i.e. large-scale, distributed computing systems, are expected to provide computing infrastructure required by many areas of modern science and technology (see e.g. [8] for example applications). However, the truly decentralized nature of such systems introduces a number of new problems in fields as diverse as programming models, scheduling and resource management or security. In resource management, conflicts may arise in grids formed by resources owned by different organizations [14], which have different, often contradictory, goals. Consequently, there must be some kind of motivation for resources' owners (e.g. a university computational center) to allow the users coming from the outside (e.g. from a different university) to access the local resources.

In this paper, we study the problem of resource management in computational grids by analyzing the problem of *selfish* Load Balancing (LB). The computational grid is a collection of independent, but interconnected clusters. LB techniques are classic algorithms that move jobs from overloaded processors to underloaded ones, in order to spread the load on the system as evenly as possible. While it is well known that a system with LB performs better both in terms of sum of jobs' completion times and maximum completion time (makespan), LB implicitly requires a centralized control over the whole system. Processors must share a common goal, that is the optimization of the set of jobs submitted to the system. *Selfish LB* extends the problem of LB to distributed systems owned by multiple organizations. Jobs of an organization are submitted to a local cluster. Each organization is concerned only with the performance of its jobs. We study two resulting problems. Firstly, whether the clusters owned by different organizations should balance their load. Secondly, how to organize LB in order to make the process profitable for all organizations.

An example computational grid is the CiGri [2] project, connecting clusters owned by 6 academic institutions, among others, by the Department of Physics and ID-IMAG laboratory. CiGri allows users' jobs to run on the whole infrastructure. This functionality completely changes the context of the load-balancing, as jobs coming from e.g. the Department of Physics can delay local jobs executed on e.g. ID-IMAG's cluster, and therefore reduce the performance experienced by ID-IMAG's local users. Consequently, there

must be some kind of motivation for resources' owners (ID-IMAG) to allow users coming from the outside (from the Department of Physics) to access to local resources and to execute their jobs (considered as *foreign* or *grid* jobs by the host).

This paper has several contributions. Firstly, we propose a simple mathematical model that describes distributed computing systems owned by multiple organizations. Along with the model, we propose a cost function for resource utilization that takes into account the expected delay in the computation of local jobs. The cost function is not linear with resource utilization time and depends also on the local load. Then, we show that a system performing LB is more efficient than the one without, thus LB is socially profitable. Furthermore, we show that LB becomes the dominating strategy for each site if clusters are similarly loaded and the period of cooperation is significantly longer than the average size of jobs. However, we prove that if individual participants are able to decide on balancing the load based on their current queue length, cooperation will never appear and thus the grid will work inefficiently. Finally, we propose a LB algorithm that achieves acceptable results even if the load is highly heterogeneous. We achieve to balance the system without introducing any form of external compensation (such as money) for using foreign resources, only by measuring delays in the execution of jobs.

The rest of the paper is organized as follows. Section 2 presents related work in scheduling, grid resource management and game theory. Section 3 presents mathematical models for the grid and the cost function. Section 4 analyzes the proposed model from the game theoretical point of view. Section 5 shows how the strategies dependent on the current queue length fail to promote collaboration in the grid. Section 6 details a LB algorithm that addresses the problem of heterogeneity in the load of the clusters. Section 7 contains a discussion of the results obtained. Finally, Section 8 provides some concluding remarks.

2 Related Work

2.1 Scheduling

The mainstream of the current research on scheduling and resource management [13] concerns systems in which the performance of all jobs is optimized. Usually, a common metric (such as the maximum completion time called makespan and denoted C_{max}, or the sum of completion times ΣC_i of all jobs) is optimized and thus all the jobs are treated in a more or less equal man-

ner (sometimes adding weights to express their relative importance). This approach is justified by the fact that the system has one owner (an organization) which is able to enforce policies on local users. Load-balancing can be achieved in such systems by *Work Stealing* (WS). Each time a processor becomes free, it connects to its (loaded) neighbors in order to take off some of their load. It can be proved that WS delivers good performance (under some hypotheses) when optimizing C_{max} in an on-line system [3].

Nevertheless, the main difference between a grid and a large-scale supercomputer is that a grid is, by its definition [14], formed by resources controlled by different administrative entities. In the context of Grid computing, *multi-criteria* approaches may be used. Different criteria either express performance of different jobs [27] (*centralized scheduling*), or different factors, such as completion time and cost, of one job [22] (*broker-based scheduling* [6]). A scheduling algorithm is expected to deliver Pareto-optimal solutions. The main problem with the first approach is that it requires a centralized control over all the resources and all the jobs. A user does not have guarantee that his/her solution is optimal, given the action enforced by the scheduler on the others. Brokers schedule each job independently of other jobs. In grids with many users, trying to execute their jobs in parallel, actions of each user will influence the state of the environment and consequently the parameters of functions optimized by other users' brokers. Therefore users might be tempered to influence the environment in order to create most favorable conditions for their job. Optimization alone is not able to model such feedback loops.

2.2 Grid Economy

Grid economic approaches [10, 36] analyze the problem of grid resource management by means of market economy. Each resource has a (monetary) cost for its usage. Each user has a budget to spend on the execution of his/her jobs.

The simplest approach to match users and resources is to organize auctions [10] each time a user is looking for resources for the execution of his/her job. In such an auction resources of a requested type compete for the job. However, most studies proposing auctions as a way of trading resources leave too many important questions unanswered. It is not clear how should the bids be formulated and how should they depend on previous auctions. Additionally, the resulting prices are hard to predict. For an user, it is hard to balance his/her budget between a number of different jobs (although there is a mechanism of combinatorial auction, in its general form it is computationally expensive to implement [35]). For an owner, who considers investing

in the costly infrastructure, it is hard to calculate the expected return of investment.

Commodity markets for trading the access to the resources [21], similar to markets for e.g. soybean, seem to be more stable [37], as a large number of buyers and sellers of homogeneous good is centrally matched. However, computational power is not a trivial commodity, because both resources and applications are heterogeneous. Usually, an application needs a certain operating system, libraries, system architecture, size of RAM locally available, moreover, its performance depends heavily on such parameters like e.g. speed of interconnection network [20]. Heterogeneity divides the computational market into many sub-markets, in which one player is able to manipulate the price. Consequently, market is far from the *perfect competition* principle, the fundamental assumption of free-market economy, and its stability cannot be guaranteed. The other problem is that commodity markets (with derivative tools, such as futures or options, players' strategies etc.) are rather complex. Considering that grids are already one of the most complicated computer systems ever created, the introduction of another complex system on the top of them seems to be both risky and, as we will show in this paper, sometimes not necessarily.

The concept of "money" itself also bears problems. Firstly, it requires a centralized entity, a form of a grid bank, to keep track of it. Secondly, money makes people self-oriented and not willing to cooperate [33].

Some semi-market approaches were also proposed. The *Network of Favors*, implemented in the OurGrid project [4], implements a kind of barter trade. A grid job is executed if there are no local jobs, but it is canceled if a local job appears. This approach thus does not solve the problem whether to execute a grid job on a machine, only which grid job should be executed. There are also no guarantees on the finish time of a grid job.

2.3 Game-theoretic approaches

Approaches presented in this section analyze the problem of grid resource management by means of game theory. Game theory [29] studies the problems in which players maximize their returns which depend also on actions of other players.

A number of papers focus on creating rules to avoid the problem of users who gain from the system without contributing [30, 9, 18, 17]. There were some theoretical works on this subject, where game theory models were formed to model users' behavior. Unfortunately, the models proposed captured mainly steady-state behavior of the participants, as early P2P systems were focused on file-sharing. In file-sharing (and similarly, in selfish caching

problem [12, 24]), it is possible to formulate a static (not time-dependent) utility function, which can express the gain each user gets from the system in a closed formula. In the context of distributed resource management, such a solution is proposed in [34] and focuses on maintaining good relationships of a node with its neighbors by accepting neighbors' jobs to be executed on the node and therefore increasing the probability that the node's jobs will be accepted by its neighbors in the future. However, the need for computational power is usually highly time-dependent with peaks of activity followed by periods of considerably lower needs. While the ability to use foreign computational power is very valuable when the local demand is high (and, at the same time, the execution of foreign jobs becomes then very costly), it becomes almost useless when the local demand drops below some threshold.

[23] proposes a model where individual clusters (placed in e.g. different departments of a university) are visible as one site in the grid. The model assumes that a job has been already accepted for the execution by the site. [23] studies which cluster from that site should eventually execute the job. Our model concerns that assumption which [23] made a priori, as we are studying the problem if a site should accept a job coming from another site.

[5, 26] study systems in which selfish jobs compete for community-owned processors. We consider that our model is better for academic grids, in which a job is viewed through the organization that has submitted it.

Another related field of research is mechanisms design for the load balancing problem [19]. Generally, a mechanism can balance the agent's cost of invoking an action by a payoff paid to the agent [16]. This enables the mechanism to control the global behavior of the system. In [19], central scheduler assigns a fraction of a global job queue to each participating site. By a mechanism, the sites are encouraged to report their true processing power, which allows the scheduler to take globally-optimal decisions. The major disadvantage is that a common "currency" for both the cost and the payoff is required.

In this paper, we propose a collaboration framework that takes into account the institutional heterogeneity of the grid. We achieve to balance the system without introducing external forms of compensations, only by measuring delays in the execution of jobs. Furthermore, our system provides better Quality-of-Service then the best-effort execution as it is able to guarantee the latest finish time of every submitted job in the moment of the job's submission.

3 Grid Model

3.1 Description

We assume that the computational grid is formed by a set of independent, yet interconnected, clusters $\{M_1, \ldots, M_n\}$. Each cluster M_i is administratively owned by an organization O_i, whose members submit their individual jobs $J_{i,j}$ locally. $J_{i,j}$ is the jth job submitted by the members of the ith organization, and J_i is the set of all jobs submitted by O_i. As each cluster is expected to prioritize the needs of its organization, the cluster must optimize the performance of its local jobs J_i. Each cluster measures the sum of flow times F_i (the difference between the job's completion time $C_{i,j}$ and its release date, or arrival time $r_{i,j}$) of cluster's local jobs: $F_i = \sum_j (C_{i,j} - r_{i,j})$. We consider on-line [32], clairvoyant scheduling [7]. A job $J_{i,j}$ is unknown to the scheduler before its arrival time $r_{i,j}$. The size $p_{i,j}$ of job $J_{i,j}$ is known immediately after the job has arrived. At a given moment, let the *local load* L_i stand for the time moment when the computation of the last currently known local job ends.

For the sake of the theoretical analysis, unless otherwise stated, we assume that the jobs J_i are produced by a Poisson process with a known mean time between arrivals λ_i. The sizes $p_{i,j}$ of the jobs follow exponential distribution with known μ_i. Note that those parameters are not needed by the algorithms presented in the paper.

Jobs follow the divisible load model. Each job can be divided into a large, but finite, number of fragments, that can be computed in parallel. Every job can be computed on every cluster. The processing rates of the clusters are the same. It takes one unit of time to process a unit load.

The clusters are managed by time-sharing resource management systems. At a given moment, every node of a cluster computes the same job. Once a job (or its fragment) is finished, another job (or its fragment) can be computed. Local resource management systems queue incoming jobs on First Come First Served (FCFS) basis.

Although we will discuss several LB algorithms, they all share a common property of being able to guarantee the latest finish time of every job submitted to the system, which is announced to the user in the moment of job submission. We consider that such guarantee, as opposed to best-effort execution, provides better usage experience for the users (better Quality of Service) and prevents job starvation.

We can formulate the problem of selfish LB both from the Grid's (infrastructure's) and from the cluster's point of view. The grid should provide a LB algorithm which, firstly, optimizes the performance of the whole system

and, secondly, satisfies as many clusters as possible. Each cluster, given the algorithm and the observed behavior of other clusters, must decide whether to participate in the LB process, or not.

3.2 Discussion

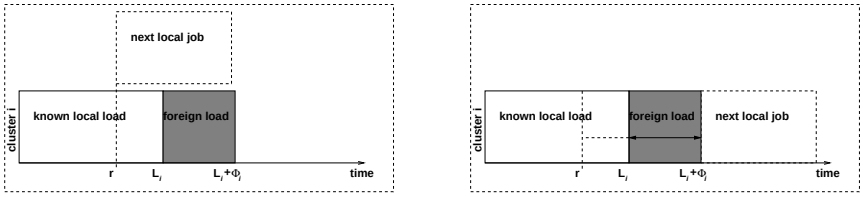
Time-sharing resource management systems, used in our model, emphasize the cost of accepting foreign jobs. In space-sharing, an alternative approach, parts of clusters, defined by a certain number of nodes, execute in parallel different jobs. In space-sharing, there is a fairly common situation when a small number of free nodes cannot be utilized by any waiting local job. Accepting small foreign jobs which fit in the schedule by filling such gaps does not slow down the execution of local jobs. We decided not to follow that model because we want to stress out the negative effects of non-local jobs on the local jobs' performance. The results of our model are an upper approximation (the worst case) for the space-sharing resources.

3.3 Cost Model

A number of approaches to grid resource management assume that the cost of running a job on a resource is set externally (it is an input to the model). But what is the real cost of running a job during a certain amount of time? One possible approach might consider that a job should participate in cluster's running cost, such as the cost of the electricity consumed, or the cost of the staff. However, if no jobs are executed, those costs remain constant. The clusters are not switched off, nor the staff is dismissed. On the other hand, in space-sharing systems, a small (and therefore cheap) job can block resources required by a large job, which would pay more.

We can alternatively suppose that the cost of accepting foreign load Φ_i is determined by the delay in the computation of the "next" local job $J_{i,n}$, unknown at the moment of decision. Such a measure expresses the "irritation" of a local user who experiences his/her job being delayed because of a foreign load being executed (see see Fig. 1). Note that if Φ_i is accepted, the arrival of $J_{i,n}$ cannot interrupt Φ_i's scheduled or ongoing execution, because such interruption could break the announced finish times for jobs which fragments belong to Φ_i. It would essentially turn our system into best-effort one.

More formally, we can formulate a function $g(r, L_i, \Phi_i)$ which expresses the delay in the computation of $J_{i,n}$ in function of known local load L_i, size of the foreign load Φ_i, and $J_{i,n}$'s arrival time $r = r_{i,n} > 0$. We assume that the decision whether to accept a foreign load Φ_i is made at time $t = 0$. We also do not consider the impact of the following local jobs nor LB which might



a b

Figure 1: A dilemma (a) faced by a cluster M_i accepting a foreign load. Accepting it (b) can delay the execution of the next local job by Φ_i, if the local job arrives before $L_i + \Phi_i$.

occur in the future. Using this notation,

$$g(r, L_i, \Phi_i) = \begin{cases} \Phi_i & \text{if } r \in (0, L_i), \\ -r + L_i + \Phi_i & \text{if } r \in (L_i, L_i + \Phi_i), \\ 0 & \text{if } r \in (L_i + \Phi_i, +\infty). \end{cases}$$

If $J_{i,n}$ arrives before the foreign load is started, it is delayed by Φ_i, i.e. the size of the foreign load (first row). If $J_{i,n}$ arrives after the foreign load is finished, it is not delayed (last row). Finally, during computation of Φ_i, the delay linearly decreases with r (second row).

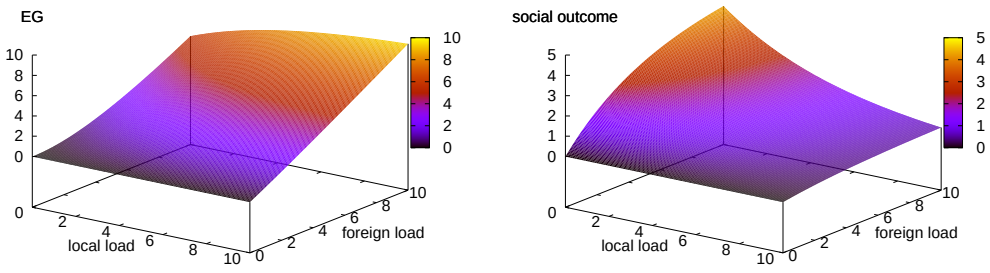
Knowing that r is a random variable with distribution $f(r)$, we can compute the expected value of $g(r, L_i, \Phi_i)$ as

$$\begin{aligned} EG(L_i, \Phi_i) &= \int_{-\infty}^{+\infty} g(r, L_i, \Phi_i) f(r) dr = \\ &= \int_0^{L_i} \Phi_i f(r) dr + \int_{L_i}^{L_i+\Phi_i} (-r + L_i + \Phi_i) f(r) dr. \end{aligned} \tag{1}$$

If the local jobs arrive according to a Poisson process with known intensity λ_i, then the delays between the jobs follow the exponential distribution, thus $f(r) = \lambda_i \cdot \exp(-\lambda_i r)$ for $r > 0$. After computing the integrals, we get the following formula:

$$EG(L_i, \Phi_i) = \Phi_i + \frac{e^{-\lambda_i L_i}}{\lambda_i} \Big(e^{-\lambda_i \Phi_i} - 1 \Big).$$

This function is plotted on Fig. 2.a. It should be noted that, firstly, the expected delay $EG(L_i, \Phi_i)$ is not linear with foreign load Φ_i. This suggests that the basic approach to calculate the cost of execution of a job on a cluster, *cost* = *coefficient*$\cdot$ *jobsize* [1], may miss some important phenomena. Secondly, $\Phi_i - EG\,(L_i, \Phi_i) > 0$ for all possible (positive) values of Φ_i and L_i.



a b

Figure 2: The expected delay in the execution of the next local job $EG(L_i, \Phi_i)$ plotted against local load L_i and the accepted foreign load Φ_i (a) and the social outcome $\Phi_i - EG(L_i, \Phi_i)$, i.e. the difference between the sender's gain Φ_i and the expected delay (b). All variables are measured in time units. $\lambda = 0.2$.

4 Game Theoretical Analysis

4.1 A Basic Two-Player Game

We present now how the foregoing cost function $EG(L_i, \Phi_i)$ can be used to analyze the expected result of LB performed in a grid formed by two clusters.

Let us define a load balancing game with two players (clusters). The heavier loaded cluster M_H, called the sender, has two strategies: *send* part of its load Φ_L to the other cluster or *compute* everything *locally*. The less loaded cluster M_L, called the receiver, can either *accept* the foreign load coming, or *reject* it. Table 1.a shows the structure of payoffs in such a game.

The pair of strategies corresponding to LB (*send* and *accept*) is the global optimum regarding the sum of players' payoffs. If the sender *sends* and the receiver *accepts* the load, the receiver expects that its local jobs will be delayed by $EG(L_L, \Phi_L)$. As it will affect M_L's criterion F_L, M_L's payoff will be always negative. However, sender's jobs are finished faster by at least Φ_L (assuming that the sender's load after sending ends at time $L_L + \Phi_L$ or later, otherwise Φ_L should be reduced). Therefore, the sender's payoff is at least Φ_L. As $\Phi_i - EG\,(L_i, \Phi_i) > 0$, the sum of payoffs of both players, or the *social outcome* (plotted on Fig. 2.b) is always positive. LB is globally-

Table 1: The structure of payoffs in the basic game (a) and when clusters cooperate during longer period (b). The row player's payoff is in the top left corner, the column player's in the bottom right corner.

a

	receive	reject
send	Φ_i, $-EG(L_i, \Phi_i)$	0, 0
compute locally	0, 0	0, 0

b

	cooperate	compute locally
cooperate	$\frac{1}{2}(\Phi_i - EG(L_i, \Phi_i))$, $\frac{1}{2}(\Phi_i - EG(L_i, \Phi_i))$	0, 0
compute locally	0, 0	0, 0

optimal, thus it the best solution from the infrastructure's point of view. The social outcome reaches the maximum for small local load L_i and large foreign load Φ_i as $\lim_{\Phi_i \to \infty} \Phi_i - EG(0, \Phi_i) = \frac{1}{\lambda_i}$. Social outcome diminishes with the increase of L_i, because the risk of delaying the next local job becomes greater, yet the gain of the sender is the same.

The main issue is that, for the receiver, the action *accept* is dominated by *reject*, as it always leads to a lower payoff ($-EG(L_L, \Phi_L)$ if the heavier loaded cluster *sends*, 0 otherwise). Thus, even if LB leads to the social optimum, it is always inefficient for the receiver, and the receiver will never choose the action *accept*.

However, the probability of being the receiver is similar to that of being the sender, if the clusters are similarly loaded and the system forces the clusters to commit to their decisions for some period, instead of letting them cooperate only instantly. Consequently, both clusters should gain from cooperation. Table 1.b informally shows the idea. In such a game, as $\Phi_i - EG(L_i, \Phi_i) > 0$, the action *cooperate* dominates the local computation. The transition from the basic game to the cooperation game can be viewed as a transition from an one-shot game to a repeated game with players committing to their strategies. Actions which are dominated in a simple game became feasible when the game is repeated. There is, however, an important difference between repeating the simple game and the cooperation game. The simple game is asymmetric, so in order to obtain payoffs from Table 1.b, each player must alternate between being the sender and the receiver.

4.2 Cooperation Game

Since it is hard to extend the theoretical results to cases with more than two players and more than one next job, we decided to validate the proposed model empirically by constructing a grid simulator with learning participants. In the simulator each cluster measures the performance of its local jobs and

accordingly adjust its willingness to join the LB coalition.

A *coalition* C is formed by a subset of clusters. Clusters belonging to the coalition balance their load by a centralized, averaging LB algorithm. Periodically, each cluster decides whether to join the coalition for the next time period $T \gg p_{i,j}$. A cluster cannot leave, nor enter, the coalition until the next decision moment. This decision is made locally, and it depends only on the performance of local jobs J_i. There are no outside, administrative rules that would enforce clusters to join the coalition. However, once a cluster is in the coalition, it must obey the results of the LB algorithm. Clusters outside the coalition compute all the locally produced jobs and no foreign jobs.

If a cluster M_i participating in C has been assigned some foreign load Φ_i, M_i must execute Φ_i immediately after time L_i, i.e. immediately after finishing the last local job known at the moment of the execution of the algorithm. As both local and foreign jobs are never postponed, we can guarantee that a job $J_{i,j}$ will start no later than $\max(L_i, L'_i + \Phi_i)$ (where L'_i is the local load known at the moment of executing the last LB procedure before $r_{i,j}$). The job can be finished faster if the next LB is performed before the job's scheduled finish time.

The decision taken by each cluster whether to join the coalition is modelled by two pure strategies (corresponding to the ones from Table 1.b): s_1 – join the coalition; and s_2 – do not join the coalition. In this game, each cluster uses mixed strategy σ_i, which specifies probability p_i that the cluster M_i will join the coalition (i.e. the probability of using strategy s_1).

4.2.1 Description of Algorithms

Here, we will describe the LB algorithm used to balance the load in the coalition and the algorithm that the clusters execute to adjust the probability of participation in the coalition.

The LB algorithm is executed periodically, and it balances the load by moving jobs or their fragments between clusters. The algorithm starts by computing the mean load $L_{mean} = \frac{1}{|C|} \sum L_i$ from the loads of clusters in the coalition. Then, the clusters with load greater than the mean send fractions of their load to the ones with the load lower than the mean. We assume no priorities in the load. Every job which parts were load balanced is completed in exactly the same time moment L_{mean}. Consequently, after LB, the load of all clusters in the coalition finishes exactly at L_{mean}. The next iteration of LB is performed at L_{mean}.

Cluster M_i that participated in the coalition adjusts its strategy σ_i at the moment of deciding whether to join the coalition for the next time period. σ_i is based on the observed sum of flow times F_i of local jobs submitted

during the last time period. M_i computes what would have been the sum of flow times $\widetilde{F_i}$ of local jobs if the cluster had not joined the coalition. By comparing F_i and $\widetilde{F_i}$, the cluster can measure if LB was profitable, and adjust p_i accordingly. More specifically, each cluster M_i that was participating in the coalition computes the ratio $R_i = \frac{F_i}{\widetilde{F_i}}$. In order to limit the maximum possible changes, we bound the ratio (by $\frac{1}{2}$ and 2). If $R_i > 1$, LB worsened the performance of local jobs, so $\delta = -1$. Otherwise, $\delta = +1$ and the ratio is inverted ($R_i \leftarrow \frac{1}{R_i}$), in order that the impact of $R_i = \frac{a}{b}$ and $R_i = \frac{b}{a}$ is the same. Then, the cluster computes the one-round desire to join the coalition q_i, which depends linearly on R_i: $q_i = \frac{1}{2} + \delta\left(\frac{R_i}{2} - \frac{1}{2}\right)$. The value of p_i is then adjusted by the value of q_i: $p_i = (1-\sigma)p_i + \sigma q_i$, where σ is a small number expressing the learning coefficient.

A cluster that remained outside of the coalition cannot compute the relative performance. In this case p_i does not change. As initially clusters are indifferent to the coalition, the starting value of p_i is $\frac{1}{2}$.

4.2.2 Experimental Analysis

We simulated the grid environment with $m = 10$ clusters participating in the previously described game using a custom-built discrete event simulator.

Unless otherwise stated, we assume that the clusters are similarly loaded. The expected delay between two consecutive jobs is the same $\frac{1}{\lambda}$, $\lambda = 0.2$. Jobs' sizes follow the gamma distribution with parameters (α, β), $\alpha = 2$, $\beta = 2$. The period of cooperation in coalition is $T = 1000$. We repeated each experiment 50 times. The results obtained in each repetition were very similar to each other and the same phenomena could be observed.

In algorithm adjusting p_i, when computing $\widetilde{F_i}$ and F_i, we took into account only jobs submitted between $t_{prev}+0.2T$ and $t_{prev}+0.8T$ (where t_{prev} is the time at which the previous decision was taken), in order to avoid transitory effects and to capture the performance of the system in the steady-state.

Fig. 3.a depicts a typical course of a simple cooperation game with identically-loaded clusters. After a short period of adjustment, the probability of joining the coalition oscillates between 0.8 and 0.9 with a few bigger peaks. We see that joining the coalition is, in general, more profitable than computing the whole load locally. Otherwise, p_i values would drop below 0.5. The probabilities do not, however, converge to 1.0. This suggests that it was quite common that LB delayed the execution of local jobs. We have also varied the number of clusters m from 2 to 30. We observed similar results, although the more clusters, the higher final value of p_i was observed.

The speed-up experienced by a cluster M_i can be defined as M_i's *score*

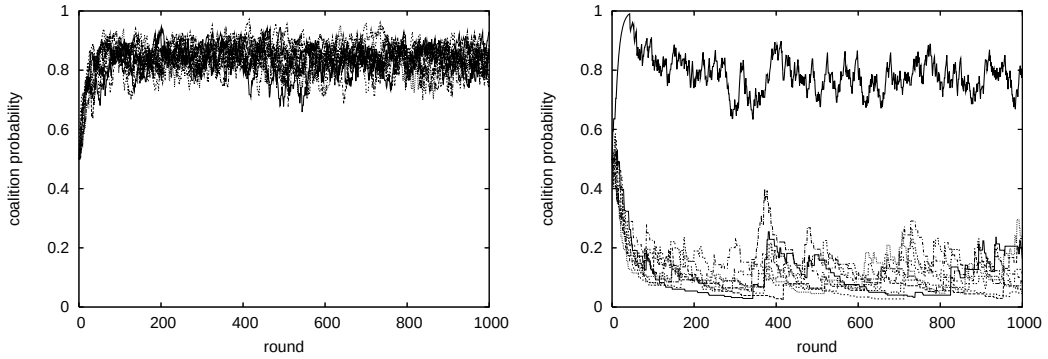


Figure 3: The probability of joining the load-balancing coalition p_i plotted against the round number in the homogeneous case (a) and when one of the clusters (its p_i is represented by a thick black line) is overloaded (b). p_i values for individual clusters are depicted by different styles of lines.

$S_i = \frac{\widetilde{F_i}}{F_i} - 1$, if $F_I < \widetilde{F_i}$ (LB was better for M_i than local computation) and $S_i = -(\frac{F_i}{\widetilde{F_i}} - 1)$ otherwise. If $S_i = 1$, M_i's jobs' flow time when LB was used was reduced by a half comparing to local computation, if $S_i = -1$, M_i's jobs' flow time was twice as much. S_i expresses how well the LB algorithm performed.

In order to measure the score, we fixed $p_i = 1.0$ for all the clusters, and performed $N = 1000$ rounds of the game without using the p_i's adjustment algorithm described in the previous section. Table 2 presents the aggregated results. As scores for individual clusters were similar, for brevity we present only the average score (computed as an average from individual clusters' scores, which, in turn, are averages over 1000 runs), the actual range in which individual clusters' scores belonged, and the average standard deviation, computed as the average from standard deviations of score for each cluster, again with the actual range.

We can see that, on the average, if the clusters were similarly loaded and their load was high ($\alpha = \beta = 2.0$ with $\lambda = 0.2$ gives average load of 80%), jobs were speeded up from 50% to 237%. The more clusters in the grid, the better the results. This is because if there are more clusters, more clusters can actually send their load and profit from the LB. In the underloaded case

Table 2: Mean score and standard deviation achieved by individual clusters in 1000 rounds of the simple cooperation game. $\lambda = 0.2$

no. of clusters	parameters	score / standard deviation typical	overloaded
homogeneous configuration			
2	$\alpha = \beta = 2.0$	0.50 ± 0.00 / 0.65 ± 0.00	
5	$\alpha = \beta = 2.0$	1.25 ± 0.06 / 1.29 ± 0.10	
10	$\alpha = \beta = 2.0$	1.76 ± 0.12 / 1.64 ± 0.11	
30	$\alpha = \beta = 2.0$	2.37 ± 0.11 / 2.06 ± 0.20	
2	$\alpha = \beta = 1.0$	0.13 ± 0.00 / 0.06 ± 0.005	
5	$\alpha = \beta = 1.0$	0.23 ± 0.00 / 0.10 ± 0.00	
10	$\alpha = \beta = 1.0$	0.28 ± 0.01 / 0.11 ± 0.01	
30	$\alpha = \beta = 1.0$	0.30 ± 0.01 / 0.12 ± 0.01	
heterogeneous configuration, $\alpha_1 = \beta_1 = 3$			
2	$\alpha = \beta = 2.0$	-7.13 ± 0.00 / 3.66 ± 0.00	0.93 / 0.24
5	$\alpha = \beta = 2.0$	-1.98 ± 0.06 / 1.53 ± 0.04	5.11 / 2.05
10	$\alpha = \beta = 2.0$	-0.14 ± 0.03 / 0.93 ± 0.04	14.23 / 5.53
30	$\alpha = \beta = 2.0$	1.55 ± 0.11 / 1.57 ± 1.17	36.99 / 9.29
2	$\alpha = \beta = 1.0$	-24.21 ± 0.00 /12.29 ± 0.00	4.45 / 2.14
5	$\alpha = \beta = 1.0$	-0.55 ± 0.01 / 0.27 ± 0.01	62.42 /11.92
10	$\alpha = \beta = 1.0$	0.01 ± 0.00 / 0.10 ± 0.01	98.67 /18.96
30	$\alpha = \beta = 1.0$	0.23 ± 0.01 / 0.11 ± 0.01	131.99 /25.42

($\alpha = \beta = 1.0$, which gives average load of 20%), the algorithm was not as efficient, because there were not as many jobs to be load-balanced. However, performance varied heavily, as standard deviations are high.

However, one overloaded cluster, for instance M_1 in Fig. 3.b (with $\alpha_1 = 3$, $\beta_1 = 3$), is able to hinder others from cooperating. p_1 of such a cluster quickly reaches a value slightly higher than in the previous experiment. The rest of the clusters does not want to collaborate with M_1, so their probabilities oscillate around 0.1. There are sharp increases, when LB gets profitable for an underloaded cluster. After such an increase we usually observe a gradual downfall, meaning that the cluster constantly looses performance in comparison to individual computing. Such an overloaded cluster has also enormous impact on the mean score achieved by other clusters. While the score achieved by M_1 is high, other clusters are constantly loosing by cooperating with this cluster. Only in largest grids (30 participants in the normally loaded case, 10 and 30 participants in underloaded case) others clusters gain, although this gain is much smaller comparing to the homogeneous case.

5 Strategies Dependent on the Current Queue Length

Both effects observed in the previous section suggest that, although LB is indeed profitable, clusters should perhaps make their decisions on a finer level. However, it will be shown in this section that if each cluster is able to decide about participating in the load-balancing process based on its current queue length, cooperation will never appear, and therefore the grid will work inefficiently.

Let us assume that a grid is composed of two clusters, M_1 and M_2, that participate in an averaging LB algorithm. The cluster with larger load sends a half of the difference in load to the less loaded cluster. The amount of load s_i sent by cluster M_i is:

$$s_i(L_i, L_j) = \frac{L_i - L_j}{2}.$$

If $s_i > 0$, cluster M_i sends a fraction of its load to cluster M_j, if $s_i < 0$ cluster M_i receives a fraction of cluster M_j's load.

Given cluster's M_i current load L_i and its expectations about the distribution of load in the other cluster, M_i can compute the expected result of LB process. Let us assume that clusters are identically loaded and they know their average arrival rate λ and the average service rate μ. Let's denote $\gamma = \mu - \lambda$ and $\rho = \frac{\lambda}{\mu}$. Then, for each cluster, its queue length L_i has a mixed distribution having an impulse at $L_i = 0$ and being exponential for $L_i > 0$ [25]. For our analysis, this distribution can be approximated by the following continuous distribution:

$$f(L_i) = \begin{cases} 0 & \text{for } L_i < 0 \\ (1-\rho)\delta(L_i) + \rho\gamma e^{-\gamma L_i} & \text{for } L_i \geq 0 \end{cases},$$

where $\delta(L_i)$ is Dirac delta function. Knowing this, M_i can compute the expected value of $s_i(L_i, L_j)$ as:

$$\begin{aligned} ES_i(L_i) &= E\Big(\frac{L_i - L_j}{2}\Big) = \frac{1}{2}\Big(L_i - EL_j\Big) = \\ &= \frac{1}{2}\Big\{L_i - \Big(0 \cdot (1-\rho) + \\ &+ \int_0^\infty \rho\gamma L_i e^{-\gamma L_i} dL_i\Big)\Big\} = \frac{1}{2}\Big(L_i - \frac{\rho}{\gamma}\Big). \end{aligned}$$

Consequently, cluster M_i expects that it will send its jobs only if its current queue is longer than the expected queue length of M_j, $\frac{\rho}{\gamma}$. Only in

this case LB is profitable for M_i. Cluster M_j, however, is capable of the same analysis. If M_j's current queue length L_j is less than the average $\frac{\rho}{\gamma}$, M_j will not participate in LB process. In order to carry out the LB, we need two clusters willing to participate. Consequently, if M_i wants to participate, it must know that if M_j participates (and the LB occurs), M_j's queue length will be longer than $\frac{\rho}{\gamma}$. This modifies the probability distribution function (pdf) of M_j's queue length. Following the general formula for conditional probability $Pr(A|B) = \frac{Pr(A \text{ and } B)}{Pr(B)}$ the pdf of M_j's queue length, given that $L_j > \frac{\rho}{\gamma}$ can be expressed as:

$$f^*(L_j|L_j > \frac{\rho}{\gamma}) = \begin{cases} \frac{f(L_j)}{1-F(\frac{\rho}{\gamma})} & \text{if } L_j \geq \frac{\rho}{\gamma} \\ 0 & \text{if } L_j < \frac{\rho}{\gamma} \end{cases},$$

where $f(L_j)$ is the original pdf and $F(L_j) = 1 - \rho e^{-\gamma L_j}$ is its cumulative distribution function. Then, the expected value of s becomes:

$$\begin{aligned} ES_i(L_i|L_j > \frac{1}{\gamma}) = & \frac{1}{2}\Big\{L_i - \\ & - \Big(\int_{\frac{\rho}{\gamma}}^{\infty} \frac{\rho\gamma L_i e^{-\gamma L_i}}{1-(1-\rho e^{-\gamma\frac{\rho}{\gamma}})} dL_i\Big)\Big\} \\ & = \frac{1}{2}\Big(L_i - \frac{\rho+1}{\gamma}\Big). \end{aligned}$$

Consequently, LB becomes profitable for cluster M_i only if its queue length is longer than $\frac{\rho+1}{\gamma}$. We can, however, iterate this reasoning further. Generally, the following formula holds:

$$ES_i(L_i|L_j > \frac{\rho+k}{\gamma}) = \frac{1}{2}\Big(L_i - \frac{\rho+k+1}{\gamma}\Big).$$

It makes the one-step LB analogous to the *beauty contest* game [11]. The more iterations of the above reasoning the players make, the longer the minimum queue length which makes LB profitable. As the players are completely rational, the minimum profitable queue length increases indefinitely. The LB will thus never occur.

Obviously, in the longer time period, M_i's queue length will also follow the same distribution as M_j's. The expected result of LB (computed as $ES_i = \frac{1}{2}(EL_i - EL_j)$) becomes then zero. Each cluster can expect that it will send, on average, as much load as it will receive. In the analysis in Section 3.3 we have shown that the cost EG of accepting a fraction of load of size Φ is always less than Φ, the profit the sender gets from sending it. Therefore the LB process is indeed profitable.

6 Bounded, Iterative Load Balancing

In Section 4 we have shown that, although cooperation is profitable, one overloaded cluster is able to make the LB unprofitable for the rest of the clusters. In Section 5 we have demonstrated that it is impossible to formulate globally-optimal strategies based on the current queue length. Consequently, the only part of the system that can be improved is the LB algorithm. In this section we will propose Bounded, Iterative Load Balancing algorithm (BILB) that balances the load in a more equitable way. The algorithm uses two mechanisms: iterative LB, in which firstly the least-loaded clusters are balanced, then the following clusters are iteratively added in the order of their local load; and bounded LB, in which no cluster is forced to accept foreign load which would increase its local queue too much. The former technique favors underloaded clusters, the latter directly uses conclusions from the cost model presented in Section 3.3, where we have shown that accepting foreign load is much cheaper when local queue is short.

6.1 Algorithm Description

BILB allows each cluster to control the maximum foreign load it will receive. Each cluster, once every T time units, instead of declaring whether it participates to the coalition or not, announces its declared participation level l_i. l_i value is expressed in the same units as jobs' length. The algorithm guarantees that, after BILB finishes, if a cluster has been assigned some foreign load Φ_i, the cluster's queue length will not extend l_i. On the other hand, in order to prevent that clusters declare $l_i = 0$, an overloaded cluster will not be able to send more of his load than l_i. Let us denote as Ψ_i the total load that cluster M_i sends as a result of BILB. Using this notation, $\Psi_i < l_i$. By l_i, a cluster can control the risk of taking part in LB. Lower values mean lower risk, but also smaller load to be sent if the cluster is overloaded. Higher values enable the cluster to send much load, if it is overloaded, but also could force it to accept much foreign load.

BILB balances the load of a cluster with clusters that have smaller loads. The algorithm starts by collecting the loads L_i, and the declarations of maximum participation l_i from the clusters. Firstly, the clusters are sorted according to the increasing queue lengths. Let us assume that $L_1 \leq L_2 \leq \ldots \leq L_n$. Then, for each cluster M_k, its load is balanced with $\{M_1, \ldots, M_{k-1}\}$. Those clusters have their load already balanced, with mean queue length $\overline{L_{k-1}}$. The

algorithm computes the current mean queue length:

$$\overline{L_k} = \frac{(k-1)\overline{L_{k-1}} + L_k}{k} = \frac{1}{k}\sum_{i=1}^{k} L_i$$

The maximum load M_k will send, Ψ_k^*, is bounded by the average $L_k - \overline{L_k}$ and M_k's announced participation l_k, $\Psi_k^* = \max(L_k - \overline{L_k}, l_k)$. Ψ_k^* is then divided onto $\{M_1, \ldots, M_{k-1}\}$. On each receiver M_i, the space available for Ψ_k^* is bounded by the delay to the average $(L_i + \Phi_i - \overline{L_k})$ and by the declared participation $l_i - L_i - \Phi_i + \Psi_i$ (Φ_i is the total load already received by M_i, Ψ_i is the total load sent by M_i). Receivers are sorted by increasing space available for M_k load. Then, for each receiver M_i, the load $\Phi_{i,k}$ actually received is bounded by the space available and the number of remaining receivers $\Phi_{i,k} = \frac{\Psi_k^*}{k-i}$. In other words, if M_i receives anything, $L_i + \Phi_i - \Psi_i \leq l_i$. Note that after such a bounding, the actual load $\Psi_k = \sum_{i=1}^{k-1} \Phi_{i,k}$ that the cluster M_k sends can be lower than Ψ_k^*.

The computational complexity of BILB is in $O(n^2)$.

6.2 Algorithm Analysis

The two mechanisms used by the algorithm are difficult to analyze theoretically together, however we will show that they both contribute to the desired effects.

Firstly, iterative LB achieves more equitable solutions than averaging LB. Let us assume that $L_1 \leq \ldots \leq L_k \leq \ldots \leq L_n$. As the load s_k sent in the averaging LB is a function of the average load $s_k = L_k - \overline{L}$, cluster M_k will profit from averaging LB only if its load is greater than the average load $\overline{L}$. Otherwise it will be just receiving foreign load. However, in iterative LB, M_k sends a fraction $\Psi_k = L_k - \overline{L_k}$. Since the cluster balances its load with less loaded clusters, $L_1 \leq \ldots \leq L_k$, the average is smaller than the cluster's load $\overline{L_k} \leq L_k$, so the load send $\Psi_k \geq 0$. Only the least loaded cluster will not send its jobs. Iterative LB is also profitable for the heavier loaded clusters. In fact, for each cluster M_k, $\overline{L_k} \leq \overline{L}$, so $\Psi_k \geq s_k$ (and this becomes an equality only for the heaviest loaded cluster). Each cluster sends at least the same amount of load as in averaging LB.

Additionally, iterative LB makes jobs finish earlier. Cluster M_k finishes all its jobs that are known in the moment of LB until the time $\overline{L_k}$, whereas in averaging LB jobs are finished later ($\overline{L} \geq L_k$). Only the jobs of the most loaded cluster M_n are not accelerated in comparison with averaging LB.

Bounding the load that a receiver may accept has two goals. Firstly, through the parameter l_i a cluster can control its participation in the algorithm on a finer level, as it is able to balance the risk of getting a load of

size l_i (which would induce the cost $EG(L_i, l_i)$), and the gain from sending its job of size l_i. Secondly, this mechanism uses the observation that the longer is the cluster's queue, the more expensive is to execute a foreign job. In fact, the longer the queue, the more probable that the next job will arrive before the queue's end, and therefore it will be delayed by the foreign load. When a cluster is heavily loaded, it should not receive any foreign load, as it is almost certain that another local job would arrive before the foreign load would have been finished.

6.3 Experimental Analysis

We simulated the BILB algorithm in a system similar to the one described in Section 4.2. The parameters of the system were the same as in the previous experiments. For the plots of optimum participation level, we have repeated each experiment 10 times and observed no significant differences between individual runs. Each cluster modified its value of declared participation l_i according to the following formula:

$$l_i = l_i + \sigma l_i \delta(R_i - 1),$$

where $\sigma \in (0, 1]$ is the learning coefficient, R_i and δ have the same meanings as in Section 4.2.2. Similarly to the previous adjustment algorithm, this function also guarantees that the change of l_i is proportional to the performance achieved.

When clusters are similarly loaded, LB is highly profitable as the declared levels of cooperation l_i quickly reach the upper bound (Fig. 4). As such a bound is strongly greater than observed queue lengths, a cluster, by declaring $l_i = 1000$, essentially turns off the bounding mechanism and declares that it will accept everything.

When clusters' loads differ, both mechanism used in BILB are needed depending on the number of overloaded clusters in the system. Firstly, in smaller grids composed of 2 clusters, the underloaded cluster M_2 declares its participation level l_2 less than 1.5 (Fig. 5.a), which is rounded to 1.0 by the algorithm. This is a consequence of the discrete nature of our simulator, as the first job after LB may come at earliest in the next time unit. Clearly, if LB was performed before job submission, l_2 would be equal to 0. This result shows, however, that the algorithm used for l_i modification works correctly: M_2 cannot gain by load-balancing its jobs (as M_1's queue is almost always longer), but it can accept foreign load of size 1, since it will not delay its local jobs.

Secondly, in larger grids, the declared participation level of all clusters eventually reaches the upper bound (Fig. 5.b). This result is justified by

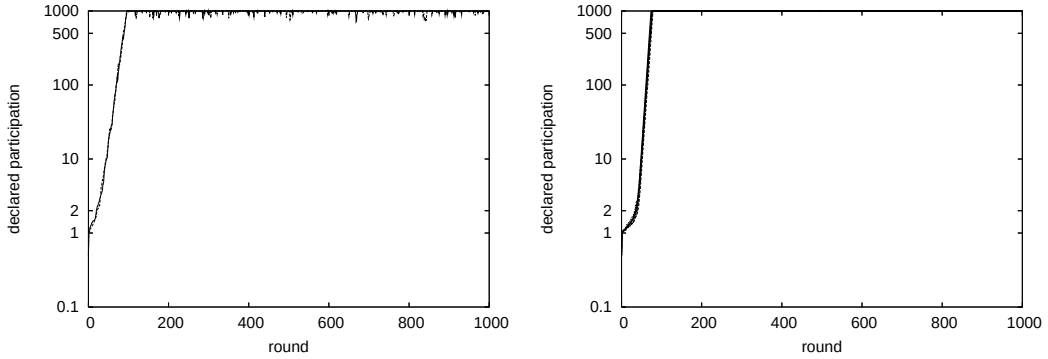


Figure 4: The declarations of maximum participation of each cluster l_i plotted against the round number in the homogeneous case of two clusters (a) and of ten clusters (b). $\alpha = \beta = 2$, $\lambda = 0.2$, l_i values for individual clusters are depicted by different styles of lines.

the "iterativeness" of the LB algorithm. As clusters with smaller loads are balanced firstly, they are able to gain from the process, without being flooded by overloaded cluster's jobs.

In order to measure the score (computed as in Section 4.2.2), we fixed the declared level of participation l_i and performed $N = 1000$ rounds of the game without using the l_i's adjustment algorithm described in the previous section.

Firstly, we tested different levels of participation l_i in order to assess the quality of the adjustment algorithm. Tables 3 and 4 present the example results for grids composed of 2 and 10 clusters aggregated in the same way as in Section 4.2.2 (each row is an average over 1000 runs with the same settings). In homogeneous cases, all clusters fixed their values of l_i to the same value. In heterogeneous cases, the overloaded cluster M_1 declared $l_i = 1000$, because it almost never receives any load and it is profitable that it sends as much as possible. The rest of the clusters declared the same l_i, shown in the different rows of the table.

In the homogeneous case (Tables 3), we can see that, indeed the maximum participation leads to best performance, although the score achieved is similar as for $l_i = 4$. What is, however, important, is that for lower participation

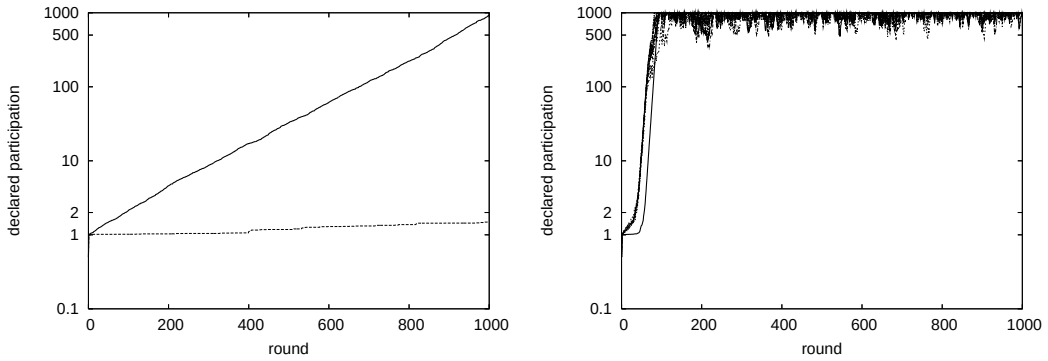


Figure 5: The declarations of maximum participation of each cluster l_i plotted against the round number in the heterogeneous case of two clusters (a) and of ten clusters (b). For M_1, plotted in thick, black line, $\alpha_1 = \beta_1 = 3$, whereas for the other clusters $\alpha = \beta = 2$. l_i values for individual clusters are depicted by different styles of lines. $\lambda = 0.2$

Table 3: Mean score and standard deviation achieved by individual clusters in 1000 rounds of the BILB in function of different levels of declared participation l_i. 2 and 10 clusters with $\lambda = 0.2$, $\alpha = \beta = 2$.

	two clusters				**ten clusters**			
l_i	score	$\pm$	std dev	$\pm$	score	$\pm$	std dev	$\pm$
1	0.125	0.006	0.123	0.014	0.076	0.006	0.055	0.006
2	0.258	0.003	0.251	0.012	0.261	0.016	0.190	0.016
3	0.387	0.01	0.364	0.001	0.710	0.043	0.515	0.039
4	0.48	0.006	0.467	0.004	1.429	0.072	1.059	0.131
5	0.574	0.006	0.568	0.008	2.148	0.092	1.541	0.153
6	0.587	0.018	0.587	0.021	2.334	0.085	1.701	0.054
7	0.609	0.013	0.642	0.033	2.518	0.124	1.878	0.150
8	0.606	0.017	0.639	0.019	2.661	0.095	1.999	0.174
9	0.608	0.011	0.617	0.007	2.793	0.114	2.095	0.184
10	0.592	0.017	0.648	0.026	2.912	0.083	2.197	0.155
20	0.613	0.007	0.713	0.019	3.116	0.140	2.413	0.195
50	0.623	0.004	0.737	0.031	3.134	0.230	2.372	0.253
100	0.636	0.039	0.774	0.035	3.135	0.153	2.427	0.169
1000	0.628	0.011	0.729	0.006	3.123	0.139	2.393	0.212

Table 4: Mean score and standard deviation achieved by individual clusters in 1000 rounds of the BILB in function of different levels of declared participation l_i. Heterogeneous grid, one overloaded cluster M_1 has $\alpha_1 = \beta_1 = 3$, the rest $\alpha = \beta = 2$. $\lambda = 0.2$. The overloaded cluster's declared participation level is fixed to $l_1 = 1000$.

	two clusters				**ten clusters**					
	typical		overloaded		typical				overloaded	
l_i	score	std dev	score	std dev	score	±	std dev	±	score	std dev
1	0.003	0.017	0.034	0.022	0.079	0.006	0.057	0.006	0.003	0.004
2	-0.005	0.047	0.074	0.046	0.267	0.014	0.196	0.021	0.019	0.017
3	-0.03	0.058	0.12	0.078	0.687	0.033	0.510	0.050	0.097	0.069
4	-0.062	0.062	0.152	0.106	1.290	0.055	0.946	0.084	0.382	0.251
5	-0.098	0.081	0.185	0.146	1.807	0.058	1.343	0.098	1.116	0.855
6	-0.138	0.101	0.194	0.144	1.748	0.069	1.370	0.152	1.921	1.737
7	-0.167	0.116	0.192	0.141	1.696	0.066	1.375	0.098	3.086	2.823
8	-0.202	0.128	0.195	0.133	1.605	0.049	1.363	0.088	4.201	3.845
9	-0.233	0.152	0.202	0.159	1.545	0.086	1.315	0.128	5.664	5.116
10	-0.278	0.167	0.203	0.149	1.448	0.071	1.290	0.108	6.651	5.938
20	-0.661	0.353	0.232	0.166	0.887	0.059	1.181	0.081	13.635	10.712
50	-1.822	0.903	0.316	0.189	0.607	0.026	1.226	0.126	20.841	10.551
100	-3.753	2.005	0.501	0.272	0.591	0.057	1.226	0.089	21.845	9.602
1000	-7.054	3.641	0.949	0.253	0.609	0.104	1.229	0.161	22.018	9.423

levels, the standard deviation is also lower, which means a smaller risk for the individual clusters.

In the heterogeneous case (Table 4) with two clusters, the minimum participation of the least loaded cluster, which was proposed by the adjustment algorithm, indeed leads to the best performance of this cluster. Nevertheless, in heterogeneous grid composed of 10 clusters, less loaded clusters achieve better performance for $l_i = 5$, than for $l_i = 1000$, proposed by the adjustment algorithm. It should be noted that, with $\lambda = 0.2$, 5 is the average time between two jobs are released. If the cluster accepts a foreign load which extends its queue length to 5, in average half of the local jobs will be delayed. We conclude that the adjustment algorithm is too straightforward, thus the clusters should use more sophisticated techniques for learning their optimal participation level, similar to ones used in other games with incomplete information, such as [31] or genetic algorithms [28].

In order to compare scores achieved by clusters under BILB with the simple algorithm (Table 2), we set l_i values which maximized the average score in Table 4. The results are shown in Table 5. We see that, firstly, BILB provides better results in the homogeneous setting, and secondly, it allows positive gains for all the clusters even when one of the clusters is

Table 5: Mean score and standard deviation achieved by individual clusters in 1000 rounds of the BILB. $\lambda = 0.2$. In heterogeneous configurations, the overloaded cluster declared $l_1 = 1000$.

no. of clusters	parameters	score / standard deviation typical	overloaded
homogeneous configuration			
2	$\alpha = \beta = 2.0$, $l = 1000$	0.65 ± 0.00 / 0.73 ± 0.010	
5	$\alpha = \beta = 2.0$, $l = 1000$	2.00 ± 0.10 / 1.72 ± 0.14	
10	$\alpha = \beta = 2.0$, $l = 1000$	3.15 ± 0.17 / 2.45 ± 0.22	
30	$\alpha = \beta = 2.0$, $l = 1000$	4.56 ± 0.30 / 3.36 ± 0.24	
2	$\alpha = \beta = 1.0$, $l = 1000$	0.64 ± 0.02 / 0.79 ± 0.01	
5	$\alpha = \beta = 1.0$, $l = 1000$	1.65 ± 0.07 / 1.64 ± 0.12	
10	$\alpha = \beta = 1.0$, $l = 1000$	3.15 ± 0.01 / 2.41 ± 0.14	
30	$\alpha = \beta = 1.0$, $l = 1000$	4.55 ± 0.40 / 3.32 ± 0.60	
heterogeneous configuration, $\alpha_1 = \beta_1 = 3$			
2	$\alpha = \beta = 2.0$, $l = 1$	0.03 ± 0.02 / 0.00 ± 0.00	0.02 / 0.00
5	$\alpha = \beta = 2.0$, $l = 3$	0.11 ± 0.07 / 0.56 ± 0.02	0.44 / 0.03
10	$\alpha = \beta = 2.0$, $l = 5$	1.12 ± 0.82 / 1.80 ± 0.10	1.33 / 0.15
30	$\alpha = \beta = 2.0$, $l = 10$	29.77 ± 8.73 / 3.40 ± 0.15	2.52 / 0.23
2	$\alpha = \beta = 1.0$, $l = 1$	0.18 ± 0.05 / 0.00 ± 0.00	0.00 / 0.00
5	$\alpha = \beta = 1.0$, $l = 3$	1.90 ± 1.14 / 0.14 ± 0.00	0.07 / 0.00
10	$\alpha = \beta = 1.0$, $l = 5$	8.75 ± 4.55 / 0.21 ± 0.01	0.10 / 0.00
30	$\alpha = \beta = 1.0$, $l = 10$	40.36 ± 10.87 / 0.31 ± 0.02	0.12 / 0.03

overloaded. BILB is especially profitable in heterogeneous configurations with many clusters.

7 Discussion

The proposed solutions are simple, yet distributed systems applying them manage to avoid some common pitfalls. Firstly, every job submitted to the system has guaranteed completion time. This is a clear advantage over the best-effort mode used commonly for grid jobs.

Typical free-riding is impossible. By participating in the coalition, clusters will share their resources, as long as they are underloaded. If the cluster is heavily loaded, it does not impede others from LB, it just uses the resources which are free after the less loaded clusters have balanced their load.

There are several ways in which clusters can attempt to cheat the algorithm in order to obtain better gain. Firstly, cluster can delay the execution of foreign load. However, other clusters can immediately detect such behavior, because of the guarantees on the completion time. As we have shown, generally, LB is profitable for each cluster. Therefore, by declaring that such

delayers will be penalized by temporal or permanent exclusion, community is able to impose the desired behavior on every cluster. This is a consequence of folk theorem in repeated games.

Secondly, clusters may be tempted to overstate their queue length in order to reduce, or even suppress the load assigned to them by the algorithm. However, such behavior may reduce cluster's performance when the BILB algorithm is used. As the algorithm sorts clusters by increasing queue lengths, a cluster which declares larger local load can be overtaken by another cluster, which, in turn, can fill the queues of less loaded clusters with its jobs, leaving no space for the cheater's jobs. Only the least-loaded cluster will profit from overstating its actual load. In the general case, this problem is, unfortunately, similar to the n-player repeated Prisoner's Dilemma [29]. In two-player, one-shot Prisoner's Dilemma (PD in short), each player has two strategies: *Cooperate* or *Defect*. For a player, the payoff when both players *Cooperate* is lower than the payoff when the player *Defect*, leaving the other one *Cooperating*. However, the payoff when both players *Defect* is the lowest. Although mutual *Cooperation* is socially-optimal, the Nash equilibrium (the pair of strategies in which no unilateral deviation is profitable for any player) is mutual *Defection*. In our game, a cluster M_i *Defects* when it declares its queue length $\widetilde{L_i}$ equal to the declared participation level l_i ($\widetilde{L_i} = l_i$) (higher declarations are dominated by this action), although the true queue L_i is shorter $L_i < \widetilde{L_i}$. Additionally, other clusters cannot observe the true L_i, and therefore can guess the action undertaken by M_i only by the observed result of the LB algorithm. We think that, similarly to other real-world PD occurrences, the only way to prevent such situations is an out-of-game verification of clusters. Such control can be performed e.g. by sporadic test of a cluster's true queue length. Each organization must provide an account, indistinguishable from other accounts of that organization. Using this account, an auditor is able to observe the current queue length by submitting his/her job of a known length.

One of the drawbacks of the proposed solutions is the centralization of the load balancing algorithm. This, however, should not be a problem in typical grid systems, composed of tens (rather than thousands) clusters. Recall that the computational complexity of the algorithm is small ($O(n^2)$). In larger systems, the load could be balanced in a more distributed fashion, with a number of instances of BILB running on overlapping fragments of the system.

8 Concluding Remarks

In this paper, we have proposed a novel method for resource management in decentralized systems owned by multiple parties. Game-theoretic approach allowed us to model organizational heterogeneity of grids. We presented a new cost function, in which the cost of execution depends both on the size of a job and on the local load. Then, using game-theoretical approach, we have demonstrated that cooperation in the LB process is profitable, although it may be hard to achieve when players are myopic. We have overcome this issue by forcing players to commit to their decisions for longer periods of time. When clusters were similarly loaded, even a simple queue-averaging load balancing algorithm was acceptable for all the players. Then we have proved that if clusters choose their actions according to their current queue length, cooperation will never occur. Finally, we have addressed the issue of load heterogeneity by designing Bounded Iterative Load Balancing algorithm (BILB). BILB delivers more equitable solutions by employing two techniques: bounding the foreign load assigned for execution on a cluster by limiting the maximum length of the queue; and load balancing the least-loaded clusters first, so that they could also gain from the process. The algorithm delivers better results than the previous algorithm in all configurations considered. Moreover, it is able to improve performance of also the less-loaded clusters. Additionally, our system provides basic Quality of Service parameters for every submitted job, as it is capable of determining the job's worst-case finish time in the moment of the submission of the job.

Our future work is twofold. Firstly, we want to further enhance our algorithm in order to reduce the dispersion of the results observed in the experiments. Secondly, we would like to port this solution to real-world grids, by adding the implementation of our algorithm to CiGri software.

References

[1] Sun grid compute utility. `http://www.sun.com/service/sungrid/`, 2005.

[2] Cigri: Ligthweight grid solution. `http://cigri.imag.fr/`, 2006.

[3] U. A. Acar, G. E. Blelloch, and R. D. Blumofe. The data locality of work stealing. *Theory Comput. Syst.*, 35(3):321–347, 2002.

[4] N. Andrade, W. Cirne, F. Brasileiro, and P. Roisenberg. Ourgrid: An approach to easily assemble grids with equitable resource sharing. In *9th*

Workshop on Job Scheduling Strategies for Parallel Processing, volume 2862 of *LNCS*, pages 61–86. Springer, 2003.

[5] E. Angel, E. Bampis, and F. Pascual. The price of approximate stability for a scheduling game problem. In *Proceedings of Euro-Par (to appear)*, LNCS. Springer, 2006.

[6] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, S. Spring, A. Su, and D. Zagorodnov. Adaptive computing on the grid using apples. *IEEE Trans. on Parallel and Distributed Systems*, 14(4):369–382, 2003.

[7] J. Blazewicz, K. Ecker, B. Plateau, and D. Trystram, editors. *Handbook on Parallel and Distributed Processing*. Springer, 2000.

[8] K. H. Buetow. Cyberinfrastructure: Empowering a third way in biomedical research. *Science*, 308:821–824, 2005.

[9] C. Buragohain, D. Agrawal, and S. Suri. A game theoretic framework for incentives in p2p systems. In *P2P '03: Proceedings of the 3rd International Conference on Peer-to-Peer Computing*, page 48, Washington, DC, USA, 2003. IEEE Computer Society.

[10] R. Buyya, D. Abramson, and S. Venugopal. The grid economy. In *Special Issue on Grid Computing*, volume 93, pages 698–714. IEEE Press, 2005.

[11] C. F. Camerer. *Behavioral Game Theory: Experiments in Strategic Interaction*. Princeton University Press, 2003.

[12] B.-G. Chun, K. Chaudhuri, H. Wee, M. Barreno, C. H. Papadimitriou, and J. Kubiatowicz. Selfish caching in distributed systems: a game-theoretic analysis. In S. Chaudhuri and S. Kutten, editors, *PODC*, pages 21–30. ACM, 2004.

[13] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn. Parallel job scheduling a status report. In *Job Scheduling Strategies for Parallel Processing: 10th International Workshop, JSSPP 2004*, volume 3277 of *Lecture Notes on Computer Science*, pages 1–16. Springer, 2005.

[14] I. Foster. What is the grid. `http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf`.

[15] I. Foster and C. Kesselman, editors. *The Grid 2. Blueprint for a New Computing Infrastructure*. Elsevier, 2004.

[16] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, 1991.

[17] D. Ghosal, B. K. Poon, and K. Kong. P2p contracts: a framework for resource and service exchange. *Future Generation Comp. Syst.*, 21(3):333–347, 2005.

[18] P. Golle, K. Leyton-Brown, and I. Mironov. Incentives for sharing in peer-to-peer networks. In *EC '01: Proceedings of the 3rd ACM conference on Electronic Commerce*, pages 264–267, New York, NY, USA, 2001. ACM Press.

[19] D. Grosu and T. Chronopoulos. Algorithmic mechanism design for load balancing in distributed systems. *IEEE Trans. on Systems, Man and Cybernetics–Part B: Cybernetics*, 34(1):77–84, 2004.

[20] R. Gruber, V. Keller, P. Kuonen, M.-C. Sawley, B. Schaeli, A. Tolou, M. Torruella, and T.-M. Tran. Towards an intelligent grid scheduling system. In *Proceedings of PPAM 2005: Sixth International Conference on Parallel Processing and Applied Mathematics*, volume 3911 of *Lecture Notes on Computer Science*, pages 751–757, 2006.

[21] Chris Kenyon and Giorgos Cheliotis. Grid resource commercialization: economic engineering and delivery scenarios. In J. Nabrzyski, J. M. Schopf, and J. Weglarz, editors, *Grid resource management: state of the art and future trends*, pages 465–478. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[22] K. Kurowski, J. Nabrzyski, A. Oleksiak, and J. Weglarz. Multicriteria aspects of grid resource management. In J. Nabrzyski, J. M. Schopf, and J. Weglarz, editors, *Grid resource management: state of the art and future trends*, pages 271–293. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[23] Y.-K. Kwok, S. Song, and K. Hwang. Selfish grid computing: Game-theoretic modeling and nas performance results. In *Proceedings of the International Symposium on Cluster Computing and the Grid (CCGrid-2005)*, 2005.

[24] N. Laoutaris, O. Telelis, V. Zissimopoulos, and I. Stavrakakis. Distributed selfish replication. *IEEE Trans. on Parallel and Distributed Systems*, 2005.

[25] R. C. Larson and A. R. Odoni. *Urban Operations Research*. Pretince Hall, 1981.

[26] J. Liu, X. Jin, and Y. Wang. Agent-based load balancing on homogeneous minigrids: Macroscopic modeling and characterization. *IEEE Transactions on Parallel and Distributed Systems*, 16(7):586–598, 2005.

[27] L. Marchal, Y. Yang, H. Casanova, and Y. Robert. A realistic network/application model for scheduling divisible loads on large-scale platforms. *ipdps*, 01:48b, 2005.

[28] R.E. Marks. Playing games with genetic algorithms. In S.-H. Chen, editor, *Evolutionary Computation in Economics and Finance*. Springer-Verlag, 2001.

[29] M. J. Osborne. *An Introduction to Game Theory*. Oxford, 2004.

[30] K. Ranganathan, M. Ripeanu, A. Sarin, and I. Foster. Incentive mechanisms for large collaborative resource sharing. In *Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE International Symposium on*, pages 1–8, 2004.

[31] P.S. Sastry, V.V. Phansalkar, and M.A.L. Thathachar. Decentralized learning of nash equilibria in multi-person stochastic games with incomplete information. *IEEE Transactions on Systems, Man, and Cybernetics*, 24, 1994.

[32] J.l Sgal. On-line scheduling. In *Developments from a June 1996 seminar on Online algorithms*, pages 196–231, London, UK, 1998. Springer-Verlag.

[33] K.D. Vohs, N.L. Mead, and M.R. Goode. The psychological consequences of money. *Science*, 314(5802):1154 – 1156, 2006.

[34] D. E. Volper, J. C. Oh, and M. Jung. Game theoretical middleware for cpu sharing in untrusted p2p environment. In *Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems (PDCS-2004)*, 2004.

[35] W. E. Walsh, M. P. Wellman, P. R. Wurman, and J. K. MacKie-Mason. Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 35:271–303, 2001.

[36] R. Wolski, J. Brevik, J. S. Plank, and T. Bryan. Grid resource allocation and control using computational economies. In F. Berman, G. Fox, and A. Hey, editors, *Grid Computing: Making The Global Infrastructure a Reality*. John Wiley & Sons, 2003.

[37] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan. G-commerce: Market formulations controlling resource allocation on the computational grid. In *IPDPS*, page 46. IEEE Computer Society, 2001.