Non-Cooperative Scheduling Considered Harmful in Collaborative Volunteer Computing Environments

Bruno Donassolo
blmdonassolo@inf.ufrgs.br
Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves, 9500 - Porto Alegre, RS, Brazil

Arnaud Legrand
arnaud.legrand@imag.fr
CNRS – University of Grenoble
INRIA MESCAL Project
LIG Laboratory - 51 Av. Jean Kuntzmann - 38330 MontBonnot Saint-Martin - France

Cláudio Geyer
geyer@inf.ufrgs.br
Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves, 9500 - Porto Alegre, RS, Brazil

***Abstract*—Advances in inter-networking technology and computing components have enabled Volunteer Computing (VC) systems that allows *volunteers* to donate their computers' idle CPU cycles to a given *project*. BOINC is the most popular VC infrastructure today with over 580,000 hosts that deliver over 2,300 TeraFLOP per day. BOINC projects usually have hundreds of thousands of independent tasks and are interested in overall throughput. Each project has its own server which is responsible for distributing work units to clients, recovering results and validating them. The BOINC scheduling algorithms are complex and have been used for many years now. Their efficiency and fairness have been assessed in the context of throughput oriented projects.**

Yet, recently, burst projects, with fewer tasks and interested in response time, have emerged. Many works have proposed new scheduling algorithms to optimize individual response time but their use may be problematic in presence of other projects. In this article we show that the commonly used BOINC scheduling algorithms are unable to enforce fairness and project isolation. Burst projects may dramatically impact the performance of all other projects (burst or non-burst). To study such interactions, we perform a detailed, multi-player and multi-objective game theoretic study. Our analysis and experiments provide a good understanding on the impact of the different scheduling parameters and show that the non-cooperative optimization may result in inefficient and unfair share of the resources.

I. Introduction

Recent evolutions in inter-networking technology and the decreasing cost-performance ratio of computing components have enabled Volunteer Computing (VC) systems. Among these platforms, BOINC (Berkley Open Infrastructure for Network Computing) [1] is the most representative, having hundreds of thousands clients and many projects deployed. In such platforms, volunteers usually donate their machines' CPU idle time to scientific projects and indicate some preferences on how to share their processing power among them. In a typical situation, projects have thousands of independent CPU-bound tasks and they are interested in optimizing their throughput, i.e., maximize the total number of tasks completed. To handle this complexity, BOINC implements a simple and robust distributed protocol which ensures that local shares and volunteer priorities are respected. This protocol has been used for many years now and its efficiency and fairness have been assessed in the context of throughput oriented projects [2], [3]. However, the popularization of BOINC calls for new usages of such platforms. New projects with different characteristics have emerged, in particular, projects whose workload is made of batches of a small number of work units (Bag of Tasks) [4] arriving periodically, in contrast to regular and continuous throughput projects. These projects are usually interested in response time. As the workload of such burst projects is different from classical throughput-oriented projects, it calls for alternative server project configuration.

In BOINC systems, each project deploys its own server and configures it so as to fit to its workload characteristics. The interaction between projects may lead to unexpected consequences that require tools like game theory to be carefully studied. In this article, we make the following contributions:

- We perform a game theoretic modeling and experimental analysis of such system, in order to verify the potential performance issues raised by such complex configurations.
- Then, we study the influence of the main server parameters in a multi-player context. This study illustrates that the current scheduling mechanism is unable to enforce fairness and project isolation. Burst projects may dramatically impact the performance of other projects.
- Last, we show that when such burst projects share volunteer machines with throughput projects, the non-cooperative optimization of their project configuration may result in inefficient and unfair sharing of resources.

This article is organized as follows. Section II discusses related work and presents the main characteristics of the BOINC architecture and scheduling algorithms. Section III presents a game theoretic modeling and the notations used throughout this article. Section IV describes the experimental framework we used and presents experimental results. Section V concludes the paper and highlights directions for future work.

II. Background and Related Work

A. Volunteer Computing

Volunteer Computing is a kind of distributed computing platform on which clients donate their idle resources to projects. VC became famous thanks to SETI@home project [5], which started in 1999, searching for extraterrestrial intelligence. Later, SETI@home evolved and became the open-source BOINC (Berkeley Open Infrastructure for Network Computing) project. Nowadays, BOINC harnesses more than 580,000 hosts that deliver over 2,300 TeraFLOP per day. Several projects have been deployed, such as ClimatePrediction.net, Einstein@home or the World Community Grid. Each project has its own server which is responsible for distributing work units to clients, recovering results and validating them. Work units run on clients' machines according to specific rules defined by the clients.

Salient characteristics of such systems are: scalability, heterogeneity, volatility, unpredictability and unreliability. Therefore, typical VC workloads are made of a large (i.e., orders of magnitude larger than the number of available hosts) sets of independent CPU-bound tasks. Thus, supporting new kinds of applications on VC platform is very challenging. A new promising class seems to be the case where applications have a relatively small number of work units in infrequent time intervals. As these projects do not have large amount of tasks, they are interested in getting tasks back as soon as possible. More precisely, when receiving a batch of tasks, such projects try to minimize the completion time of the last finishing task of the batch. New algorithms and techniques have been proposed to address this problem. Kondo *et al.* [6] claim that rapid application turnaround can be achieved through resource selection, resource prioritization, and replication. More recently, Heien *et al.* [7] proposed to tune the connection interval parameter of BOINC projects to optimize the response time of batches. Last, the GridBot project [4] recently made use of hybrid computing platform composed of grid, clusters and VC to execute workload resulting from mix of throughput and response time oriented applications. Yet, all projects rely on the same BOINC protocol and scheduling algorithms which we expose in this article.

B. BOINC Protocol

BOINC relies on a classical client/server architecture. Each project has a specific server from which clients request work units to execute. Clients, during the install process for example, decide the projects that they want to crunch for. The behavior of clients and servers are described below.

1) Project Server: The main activity of a BOINC server is to distribute jobs to clients. Upon client request it selects from a job list which tasks can run on the client. The server must take care of the system's constraints which could preclude the client from running these tasks. Due to the high resource volatility and unpredictability, the server is also responsible for keeping track of jobs and uses to this end a simple deadline mechanism. When work units are distributed to a client, they are associated a deadline before which the client should send the task back. Whenever a work unit is received on time, servers reward the client with credits. If a client takes a long time to execute a task and misses its deadline, no credit is granted to him.

In this article, we regard two kinds of projects:

- **Continuous projects**: Such projects have an extremely large number of tasks and are thus interested in throughput, i.e., the average number of tasks done per day. Most existing BOINC projects actually fall in this category.
- **Burst projects**: Unlike the previous ones, these projects receive batches of tasks (or Bags of Tasks) and are interested in the average response time of batches.

In order to achieve a better performance, the server may use some strategies, such as replication, deadline or scheduling algorithms. The replication can be used to improve average response time, avoiding the last-finishing task issue [6]. Replication has some variants, such as **homogeneous redundancy**, which replicates tasks only to hosts with the same characteristics (OS and CPU), and **adaptive replication**, which only replicates tasks if the host is not trustful [8]. The deadline, in the other hand, can be configured to keep a track of the tasks running on clients. Tighter deadlines implies in more interaction between client and server. Also, it can be used to give urgency to some tasks (last tasks in a batch) and so, get the results earlier. Also, servers may use some special strategy to select which tasks they will send to clients. In short, they are described below.

- **Fixed**: Server does not do any kind of test before sending tasks to clients.
- **Saturation**: Server receives the saturation date of client, i.e., the date when client finishes running all urgent tasks (running in EDF mode). Then, it verifies whether a task, starting at saturation date, will finish before its deadline. This is the most commonly used scheduling algorithm.
- **Earliest Deadline First (EDF)**: The most restrictive test does a detailed simulation of the scheduling of all tasks running on the client (the name comes from the fact that the client uses an EDF scheduling algorithms when it is getting late). Then, it checks whether, when sending a new task, all already existing tasks (even from other projects) would not miss their deadline by more than they did previously.

BOINC's default configuration utilizes the saturation test. However, each project decides of activating or not these features according to its workload, objectives and clients' characteristics, such that it yields to the best possible behavior (e.g., throughput or response time improvement).

2) Clients: According to [2], client scheduling policy has been designed with 3 main goals in mind:

- Maximize the amount of credit given to user.
- Enforce long-term fairness: client must work the same for each project if project shares are equal.
- Maximize variety: client should avoid long periods working to same project.

So, clients try to fairly share (instantaneously) the resource between projects with respect to their priorities. However, this instantaneously fair share incurs overhead which slightly reduces client throughput and delays task completion. Therefore, BOINC clients implement a complex mix of short-term/long-term fairness scheduling algorithm. The exception is when a task is near to miss its deadline. In order to avoid deadline misses, the scheduling algorithm switches to EDF (Earliest Deadline First) mode and executes such tasks with higher priority. Consequently, continuous projects generally have loose deadlines, whereas burst projects should prefer tighter deadlines so that their tasks are executed in priority. It is important to notice that the long-term debt sharing mechanism prevents projects with tight deadline to always bypass other projects. When a client overworks for a project, it simply momentarily stops downloading new tasks from this project.

III. Game Theoretic Modeling and Notations

A Volunteer Computing system such as BOINC is made of volunteers that offer resources to be shared among a set of projects. The underlying scheduling mechanisms are supposed to ensure a fair and efficient sharing of resources but several parameters may affect this sharing. In the following, we explain how this situation can be modeled through the use of game theory notions. We start by defining the main actors of such situations: volunteers and projects.

Definition 1 (Volunteer V_j)**.** A BOINC volunteer is characterized by the following parameters:

- a **peak performance** (in $MFLOP.s^{-1}$) indicating the amount of $MFLOP$ it can process per second when it is available.
- an **availability** trace, i.e., an ordered sequence of disjoint time intervals indicating when the volunteer machine is available. This trace is unknown from the volunteer scheduler, which also does not try to use past historical information about this trace to obtain a better schedule.
- the **project shares**, i.e., the list of projects the volunteer is ready to work for and which priorities he/she assigned to each project.

A collection of volunteers is denoted by V.

In our modeling, we consider the volunteers to be passive and to only provide resources. We will see later how their welfare can be accounted for but in a first approximation, they are considered to be completely passive.

We now define the main characteristics of BOINC projects.

Definition 2 (Project P_i)**.** A BOINC project is characterized by the following parameters:

- w_i $[MFLOP.task^{-1}]$ denotes the **size of a task**, i.e., the number of $MFLOP$ s required to perform a task. We assume that the size of the tasks of P_i is uniform. This is an approximation, but it generally holds true for many projects at the time scale of a few weeks.
- b_i $[task.batch^{-1}]$ denotes the **number of tasks within each batch**. Again, we assume that all batches of a given project have the same size. This is a rather strong assumption for projects like GridBOT [4] but we consider this as reasonable in a preliminary study such as the one we propose. Furthermore, since we are not interested yet in how a given project should prioritize its bursts, this assumption should not really affect our conclusions.
- r_i $[batch.day^{-1}]$ denotes the **input rate**, i.e., the number of batches per day. Again, we assume this is fixed and we neglect the potential bursts and off-peak periods that may arise at the scale of the week or of the month. Also, we also assume that r_i, b_i and w_i are such that they do not fully saturate the system, i.e., such that a batch always ends before the submission of a new one. Indeed, as we previously explained, we are not interested yet in how a given project should prioritize its bursts. We only focus on how the different projects interfere with each others so the previous assumptions should not be harmful to this respect.
- Obj_i is the **objective function** of the project. Depending on the nature of the project, it could be either the **throughput** ϱ_i, i.e., the average number of $task$ processed per day, or the **average completion time of a** $batch$ α_i.
- q_i is the **quorum**, i.e., the number of successfully processed results that have to be returned before the task can be considered a valid. In our experiments, we assume that q_i is always equal to 1.

A project P_i is thus a tuple (w_i, b_i, r_i, Obj_i) and a project instance is thus a collection of projects $P = (P_1, \dots, P_K)$. The set of all such possible project instances is denoted by $\mathcal{P}$.

Using the game theory terminology, projects will be referred to as **players**. Along the same lines, the term **strategy** is used to account for the set of options that players have and that may influence the sharing of the available resources.

Definition 3 (Strategy S_i)**.** The server of each project can be configured with specific values that directly influence the performance of the project. In our study, we focused on the following parameters:

- π_i is the task **work send policy** [3] used by the server upon reception of a work request. When a client connects to the server, it asks for enough work to keep him busy for a period of time (e.g., one day). Then, it is the server that determines how many tasks to send. The simplest strategy $\pi_{cste=c}$ sends a fixed amount (c) of tasks, whatever the state of the client and of the server. In this simple strategy, c could be determined from the average task duration and from the requested amount of work. More elaborate work send policies like saturation (π_{sat}) and EDF (π_{EDF}) have been introduced in Section II-B1.
- σ_i is the **slack** [3]. This value is used to determine the deadlines assigned to tasks upon submission to clients. For example, if the average computation time of a task on a dedicated standard reference machine is one hour, then a fixed slack of 2 would result in a deadline of two hours. The simplest strategy $\sigma_{cste=s}$ means that a fixed slack of s is used. In practice, more elaborate strategies, like adapting the slack to volunteers speed/availability/reliability or to the progress of the batch, could be used but our preliminary study does not explore such possibilities.
- τ_i is the **connection interval** [7] and indicates clients how often they should reconnect to the server. This parameter influences on how fast volunteers realize that new tasks need to be processed for a burst project. In our experiments, this parameter ranges from 12 minutes to 30 hours.
- γ_i is the **replication strategy** [6]. This replication strategy is completely different from the replication used to reach a given quorum. γ_i is used to avoid straggler volunteers to delay the completion of a batch. Therefore, the strategy

$\gamma_{cste=r}$ allows to submit at most r replicas of the same task and whose deadlines have not expired. Again, smart strategies adapting to the reliability of volunteers and to the progress of the batch could be used but our preliminary study does not explore such possibilities.

The strategy S_i of a project P_i is thus a tuple $(\pi_i, \sigma_i, \tau_i, \gamma_i)$ and the set of all possible strategies for all projects is denoted by $\mathcal{S}$.

Definition 4 (Outcome $\boldsymbol{O}$)**.** Once a set of project P has decided a given strategy S, the resources of a given set of volunteers V are shared through the BOINC scheduling mechanisms.

The outcome $O(V, P, S)$ is the set of all information about completion of tasks and batches from different projects. In the following, when needed, we will denote by O_i the restriction of O to information related to P_i.

From this outcome, we can compute the following values:

- **Throughput** of continuous projects. If P_i is a continuous project, then we can compute the total number of tasks from P_i that have been processed over a given time period.
- **Average batch completion time**. If P_i is a burst project, then we can sum the time needed to complete each batch (from the arrival of the batch in the system to the completion of the last finishing task of the batch) and average it over the total number of batches that have been submitted over a given time period.
- **Waste**. Sometimes, tasks fail to be completed before their deadlines. This can happen because the volunteer's machine was too slow, or because it went unavailable for a long time, or maybe even because the slack of the project was too tight. In such cases, the task is often resubmitted and the client may not get reward for it. The time spent working on missed tasks is thus wasted both from the volunteer and project perspectives.
  For a given project P_i and a given volunteer V_j, we denote the waste by $W_{i,j}$, the ratio of tasks from P_i missed by V_j over the total number of tasks he/she received from P_i. Similarly the waste W_i of P_i denotes the ratio between total number of tasks from P_i missed by volunteers over the total number of tasks.

From a given outcome, we need to define the satisfaction (or **utility** using the game theory terminology) of each player. In our context, depending on the nature of the project, the satisfaction is based either on the effective throughput or on the average batch completion time. Yet, these two metrics do not express in the same units at all. One of them is to be maximized (the throughput) whereas the other one is to be minimized (the average batch completion time). Therefore, we propose to translate these two metrics into a common one: the **cluster equivalence** metric. The cluster equivalence metric was proposed in [9] in the context of throughput optimization and represents the number of dedicated standard reference machines that would be needed to achieve the same performance. This metric can thus be computed for both types of projects, only with a different formula.

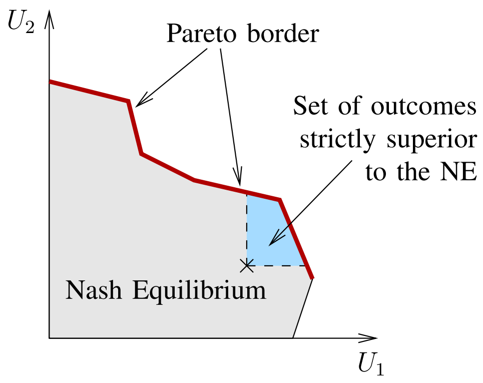

Fig. 1. Utility set for two projects. Any point in the gray area correspond to a possible outcome. Different project configurations (or strategies) may lead to the same outcome. The Pareto border (i.e., the set of points that are not dominated by any other strategy) is depicted with a bold line. As illustrated, the Nash equilibrium may be dominated by many other solutions.

Definition 5 (Cluster equivalence $\boldsymbol{CE}$)**.** We denote by W the performance (in $MFLOP.s^{-1}$) of the reference machine chosen to estimate the CE. Depending on the objective of project i, its cluster equivalence can thus be defined either as

$$CE_{continuous} = \frac{TasksDone \cdot w_i}{W \cdot TotalTime}$$

or as

$$CE_{burst} = \frac{b_i \cdot w_i}{W \cdot \alpha_i},$$

where α_i is the average batch response time of project P_i.

We can now define the utility of projects easily:

Definition 6 (Utility $\boldsymbol{U_i}$)**.** The utility of P_i is equal to its cluster equivalence:

$$U_i(V, P, S) = CE_{Obj_i}\big(Obj_i(O_i(V, P, S))\big)$$

It is thus very important to understand here that the utility of P_i depends on both its own strategy S_i, but also on the strategy choices made by the other projects.

In the following, $U(V, P, S)$ denotes the utility vector of all projects.

The utility of a project is thus what it should aim at maximizing. Yet, as it has been observed in the context of GridBOT [4], projects should also try to ensure that the waste they cause to other projects is not too large. Otherwise volunteers may consider this as selfish and decide to sign off from P_i. Therefore, even though we do not model such situations and consider the volunteers to be passive and do not put them in the decision loop, the waste (not only W_i but rather W as a whole) should be considered as a second objective to be minimized.

In the fifties, John Nash introduced the notion of Nash equilibrium [10], [11], where each player optimizes its own utility and cannot improve it by unilaterally changing its strategy. Let us reformulate this notion using the previous notations.

Definition 7 (Nash Equilibrium)**.** S is a **Nash equilibrium** for (V, P) iff

$$\text{for all } i \text{ and for any } S'_i,\ U_i(V, P, S_{|S_i=S'_i}) \leqslant U_i(V, P, S_i),$$

where $S_{|S_i=S'_i}$ denote the strategy set where P_i uses strategy S'_i and every other player keeps the same strategy as in S.

Nash equilibria do not necessarily exist and it is even undecidable to determine their existence in the general case. Similarly, when they exist, they may not be unique and computing them may be computationally intractable. Yet, they are commonly considered as an interesting way of modeling non-cooperative or non-coordinated situations. In particular, when we consider a system where each player constantly optimizes its utility and tries to learn its "optimal" selfish strategy, if such a system ever reaches a stable state, then it would be a Nash equilibrium.

Even though the BOINC project administrators do not necessarily keep tuning the parameters of their project, they certainly monitor the outcome and we think that such equilibria can be considered as a good approximation of what would happen in reality.

Nash equilibria are somehow the result of non-cooperative optimization and are **stable** (hence the name equilibrium) in the sense that no player has interest in unilaterally deviating from its strategy. Yet, Nash equilibria are not resilient to coalitions. A subset of players could have interest in simultaneously changing their strategy to all obtain a better utility. Such coalition notions are more advanced game theory notions [12] and will not be discussed in this document.

More important, Nash equilibria are not particularly fair nor efficient. In particular, they are generally Pareto-inefficient, i.e., there may exist another strategy S' such that:

$$\forall P_i : U_i(V, P, S^{(NE)}) < U_i(V, P, S').$$

The following issue can be understood by the use of utility set.

Definition 8 (Utility set)**.** The utility set $\mathcal{U}$ of a given scenario (V, P) is the image of all possible strategies $\mathcal{S}$ by the U function.

Figure 1 depicts the utility set for an imaginary scenario with $K = 2$ projects, along with the position of the corresponding Nash equilibrium. The Pareto border is the set of points that are not dominated by any other strategy (i.e., it is impossible to improve the utility of a player without decreasing the one of another player). Note that the utility set has no reason for having a shape as simple as the one used on this illustrative example.

Yet, being able to estimate how bad is a Nash equilibrium would be very helpful since it would enable to know whether it is worth regulating the system or whether we can let it live as such. Notions like the price of anarchy [13] or the selfishness degradation factor [14] have been proposed to address this issue. Yet, computing the utility set or estimating its shape is extremely hard in the general case and getting inefficiency estimation or bounds is extremely difficult. Instead, we sample it in the experimental section so as to provide insights on how bad such equilibria may be in practice.

Last, Pareto-inefficient Nash equilibria can lead to dramatic situations like Braess paradoxes where increase of resources leads to a decrease of utility for every player. Indeed, since the Nash equilibrium is not tied to the Pareto border, the increase of resource indeed leads to an improvement of a Pareto border but may lead to a Nash equilibrium worse than earlier. Such phenomena were initially observed by Braess [15] in 1968 and still receives a lot of attention. Indeed, if Pareto-inefficiency of the Nash equilibrium is a necessary condition, it is not sufficient (see [16] for example) and one still does not really understand when such phenomena occur. This is another motivation for studying the potential inefficiency of Nash equilibrium in the context of BOINC.

IV. Experimental Study

The outcome of such complex scheduling algorithms is extremely hard to put into equations and the outcome of the dynamic of thousands of volunteers running it is even more difficult to model. Therefore, we conducted a series of experiments to analyze the performance of many independent players sharing BOINC resources. The performance evaluation methodology is described in Section IV-A. Since our study is multi-parametric (deadline, connection interval, ...), multi-player (burst and continuous projects) and multi-objective (throughput and response time), we start by performing a careful study of parameter influence on global system performance (Section IV-B). Then, we illustrate the impact of burst projects on continuous projects (Section IV-C). Last, we explain in Section IV-D how we restrict our parameter space and how we sample the utility set and search for Nash equilibria.

A. Configuration Setup

We used the generic simulation toolkit SimGrid [17] to perform the experiments described in this section. SimGrid provides a powerful toolkit to simulate large-scale distributed systems. We implemented a simulator of the BOINC architecture, including both client and server components [18]. Based on reading of both articles describing BOINC and its freely available source code, we designed our simulator so as to take the main features of BOINC into account and thus perform simulations as realistic as possible.

The set of results we present in this article was obtained using a set of 1,000 clients fed with real availability traces. Clients' traces were taken from SETI@home project, available at Failure Trace Archive (FTA) [19]. We also performed similar experiments with larger set of clients but as we did not observe significant differences, we chose to report complete results with only 1,000 clients.

We restrict our study to the following project configurations:

- **Burst projects** receive one batch per day, comprising only short-live jobs which take about 1 hour running on a standard machine.
- **Continuous projects** have hundreds of thousands of CPU-bound jobs. We used a typical configuration from SETI@home project, with tasks of 30 hours[1]. The deadline of

[1]See http://www.boinc-wiki.info/Catalog_of_BOINC_Powered_Projects

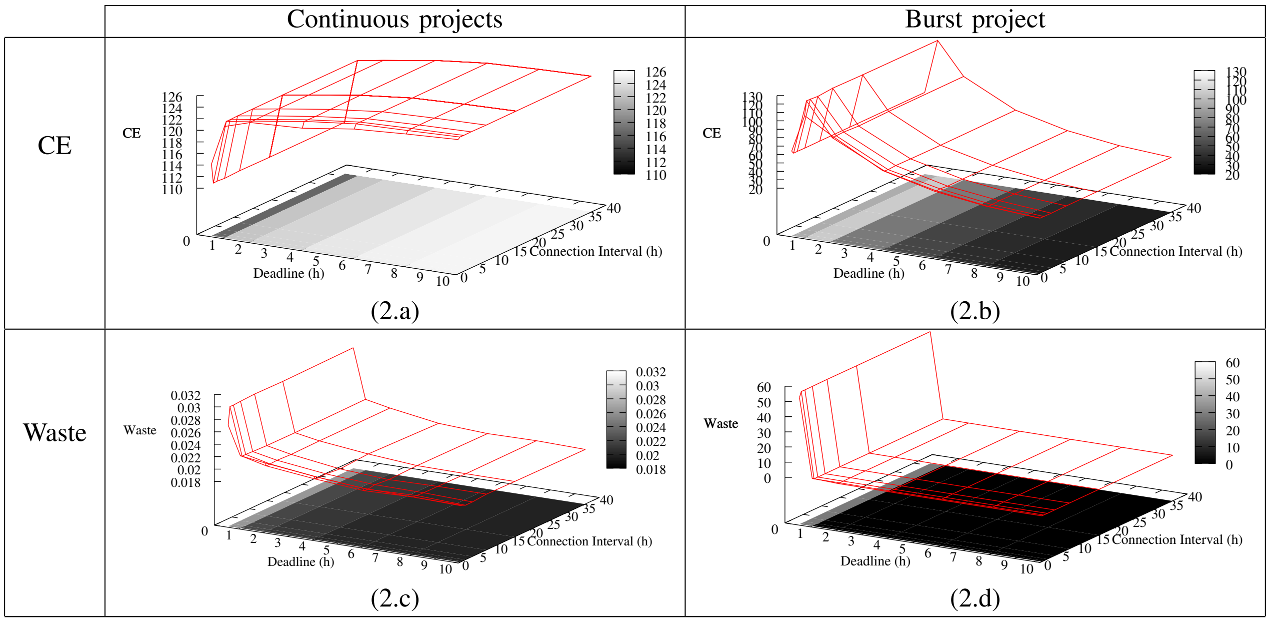

Fig. 2. Analyzing the influence of deadline and connection interval on CE and waste using a fixed scheduling strategy.

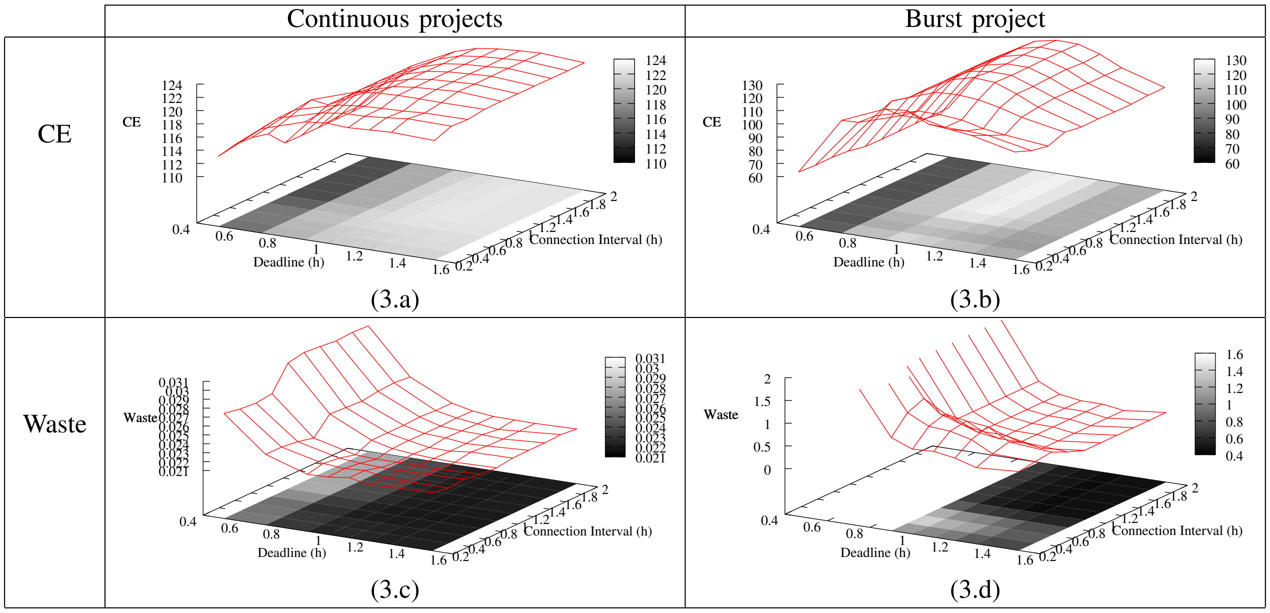

Fig. 3. Analyzing the influence of deadline and connection interval on CE and waste using a fixed scheduling strategy.

such tasks is set up to 300 hours as it is commonly accepted to provide good results [3].

The duration of each test is equivalent to about 139 days. It is important to have a very long period to ensure the effectiveness of the long-term share of BOINC system.

B. Project Parameter Influence and Optimization

This section aims to analyze the influence of each parameter configuration in burst performance. In this first series of experiments, the workload is made of 4 continuous projects (with a very large number of tasks), and 1 burst project with 1,000 tasks per day. We assume that the configuration of continuous projects does not change as we variate the deadline, connection interval and scheduling strategy of the burst project.

1) Influence of the deadline and of the connection interval: We start our analysis with the most simple scheduling strategy $\pi_{cste=1}$: the server satisfies every request sent by clients, regardless of their potential ability to process or not the task in due date. We also assume that servers do not replicate ($\gamma_{cste=0}$) to avoid straggler computers. Figure 2 depicts both the CE and the waste for the two type of projects when varying the deadline and the connection interval of the burst projects. The most obvious observation (all subfigures) is that connection interval has a very limited impact whereas deadline has a dramatic influence. In particular, it appears that optimal deadline setting for the burst project is very tight (around 1) and that suboptimal configuration leads to extremely poor performance (Figure (2.b)) for the burst project. A too tight

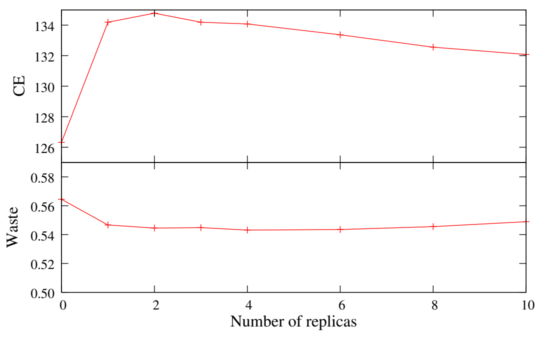

Fig. 4. Influence of replication on CE and waste when using a fixed scheduling strategy for burst projects.

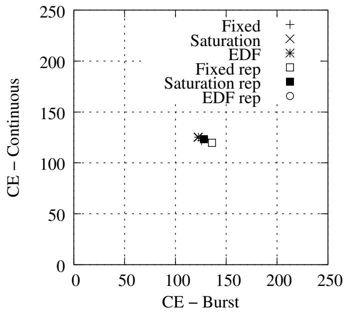

Fig. 5. Comparing cluster equivalence of scheduling strategies.

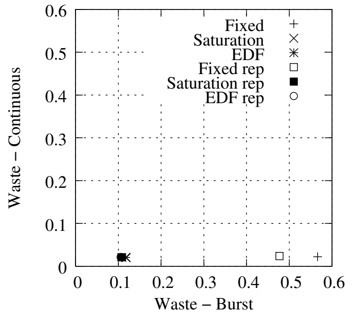

Fig. 6. Comparing waste of scheduling strategies.

deadline (less than an hour) creates an extremely large waste for the burst project (Figure (2.d)) and may almost double the waste of continuous projects (Figure (2.c)) even though it remains very low. It may even lead to a poor CE for both burst (Figure (2.b)) and continuous projects ((Figure (2.a))). Loose deadlines do not affect too much continuous projects (Figure (2.a)) but lead to rather poor performance for burst projects (Figure (2.b)).

A finer sampling of the values of these parameters around interesting values (Figure 3) enables to see that setting a deadline around 1.1h and a connection interval around 2h leads to the best CE for the burst project (Figure (3.b)). Interestingly, this configuration does not degrade significantly the CE of continuous projects (Figure (3.a)), which augurs well for coexistence of such projects. Last, although the waste for continuous projects remains low when burst projects select their optimal parameter set (2.5% compared to an absolute minimum of 1.8% as can be read on Figure (2.c)), the waste of burst projects is extremely high (around 56% on Figure (3.d)).

2) Replicas: The previous subsection presents the deadline and connection interval parameters that burst projects should choose to optimize their performance. This section briefly presents the influence of replication on burst performance. To this end, we get the best configuration using the fixed scheduling strategy and we vary the number of extra replicas that the burst project uses for each task. As we can see in Figure 4, burst can improve its performance about 7% using 2 replicas and further replicas lead to a CE decrease. Surprisingly such replication also decreases waste from 56% to 48% but this remains extremely high.

3) Influence of the scheduling strategies: The performance surface of the saturation and EDF strategies shows very similar shapes to those of fixed (presented in Section IV-B1 and IV-B2), except that waste is much smaller. Indeed, waste is significantly smaller because of saturation test, which is conservative and avoids sending tasks when clients cannot finish them. Therefore, due to space requirements, we omit the corresponding graphs and only provide the outcome of the optimal configurations for each scheduling strategy (Figures 5 and 6).

We can notice that in these experiments, there is no significant difference between the saturation π_{sat} and the EDF π_{EDF} scheduling strategy. Yet, we think that π_{EDF} may reveal superior to π_{sat} in more complex situations. Both scheduling strategies give every project roughly same CE and a rather low waste (except for burst projects, which is about 12%).

This is the result of the combination of two strategies in client's local scheduling policy: EDF and long-term fairness. The burst project manages to access resources when needed, running its tasks with high priority and getting results quickly. In other hand, the long-term fairness ensures that continuous projects get at least a reasonable share of the resources.

Last, it is interesting to note that the fixed strategy has a slightly better CE, due to the high number of tasks sent, which incurs a very important waste (around 48% in the optimal configuration). In most of our experiments, we observed that this strategy often thrashes the system even though it is the most efficient way for burst projects to steal computing resources from continuous projects. Servers should thus disregard this scheduling strategy as it wastes resources and could disappoint clients, since they may not get credits for missed tasks. The important waste of this strategy was already identified by Kondo *et al.* [3] but it had only be compared with the one of the EDF strategy which is considered to be to costly and is thus often not activated. Similar issue about unsatisfied users was also reported in [4].

C. Influence of Burst Projects on Continuous Projects

1) Burst size influence: Beside the high waste of the burst project, the previous configuration seems to indicate that both kind of projects can seamlessly share resources. Yet, this nice behavior is largely due to the fact that the burst project does not exhaust available resources. This section shows the influence of burst size on system performance, when resources become scarce. The workload is the same as in previous section: 4 continuous projects and 1 burst project, every one using the saturation scheduling strategy. The burst project uses a slack of 1.1 and 2 hours of connection interval since it was the optimal parameter configuration determined from our experiments.

As it is possible to see in Figure 7, the CE of burst project reaches a maximum when burst has near 1,000 tasks per day. After that, the system starts being saturated and, consequently, CE decreases. However, the burst project does not steal resources from continuous projects when it has a lot of tasks. The system fairness guarantees the CE of continuous projects. Analyzing the waste, as seen in Figure 8 (note the log

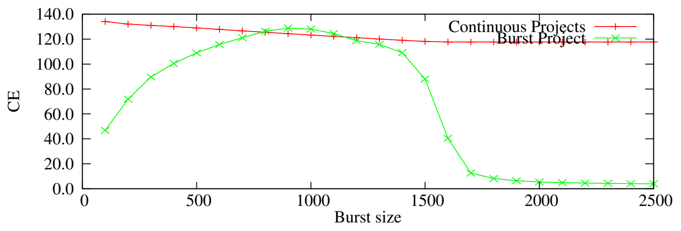

Fig. 7. Analyzing burst size influence on CE using the saturation scheduling strategy.

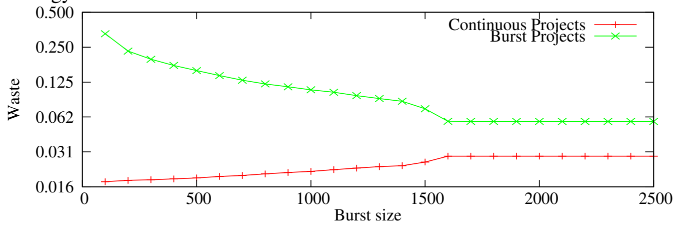

Fig. 8. Analyzing burst size influence on Waste using the saturation scheduling strategy.

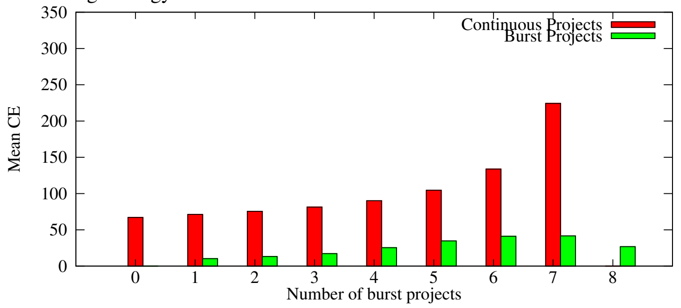

Fig. 9. Analyzing burst impact on CE using the saturation scheduling strategy.

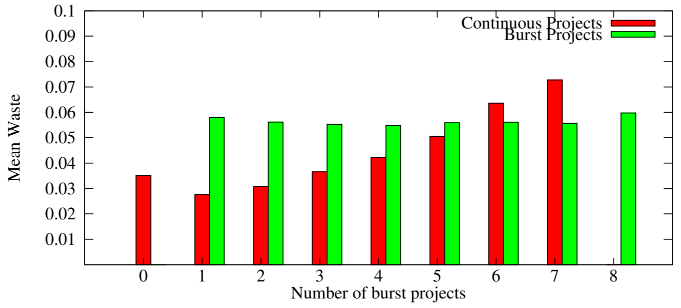

Fig. 10. Analyzing burst impact on Waste using the saturation scheduling strategy.

scale in y-axis), we see an important influence on the waste of continuous projects. It almost doubles with the increase in burst size. Another interesting fact is the stabilization in waste after a threshold. This is due to the saturation test that avoids burst tasks to be sent to clients and to thrash the system.

2) Number of burst projects influence: The number of burst projects may also affect the waste of continuous projects. In this section, we fix the total number of 8 projects and we vary the number of burst projects. Each burst project receives a batch of 1,000 tasks per day and uses a slack of 1.1.

Considering the CE, the Figure 9 shows that continuous CE increases because burst projects do not exhaust resources (unlike continuous projects). Despite the use of the saturation

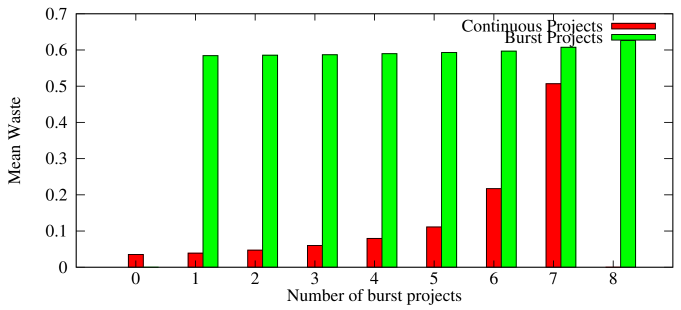

Fig. 11. Analyzing impact of burst projects on waste when using the fixed scheduling strategy.

scheduling strategy, we can see on Figure 10 that the continuous' waste increases from about 3% with 1 burst project to 7% with 7 burst projects. Note that although it was already known [3] that the fixed strategy lead to an unacceptable waste, it was not known (to the best of our knowledge) that it could also increase the waste of other projects. We illustrate this issue on Figure 11, which represents the waste when replacing continuous projects by burst projects, using the fixed scheduling strategy and a workload of one batch of 1,700 tasks per day. Although the waste of burst project remains stable, using more burst projects also considerably increases the waste of continuous projects (from 3% to over 50%). This shows that the current protocol is unable to enforce fairness and project isolation when using burst projects.

Examining the results of our tests, we conclude that burst projects may have an important influence on waste of others projects and thus, they can cause users dissatisfaction since they will miss more tasks and loose credits.

D. Utility Set Sampling and Nash Equilibrium

In this section we explain how we proceed to search for Nash equilibrium and locate these Nash equilibrium in a sample of the corresponding utility set.

1) Best Response Strategy: In Section IV-B, we showed that deadline is the most influential parameter and that all others have an optimal value almost independent from the deadline. Therefore, we consider a situation where all projects use the saturation strategy with a fixed replication of 2 and a connection interval of 2 hours.

The previous sections also show that the negative effect of burst projects may become more visible when resource become scarce. Hence, we first present a configuration with 6 projects among which 4 of them are burst projects receiving a batch of 900 1 hour tasks per day. In this configuration, continuous projects use 24h and 30h tasks with a deadline of 336h and 300h (these values are representative of the Einstein@home and SETI@home projects, respectively).

Although best-response strategy does not necessarily converge, it is known that upon convergence, its limit state is a Nash equilibrium. Furthermore, since in our particular setting, all our burst projects are identical, they should all have the same configuration at equilibrium. Therefore, we apply the following methodology. Assuming burst projects agree on a given value of slack σ_{cons}, we compute the CE_{cons} of burst

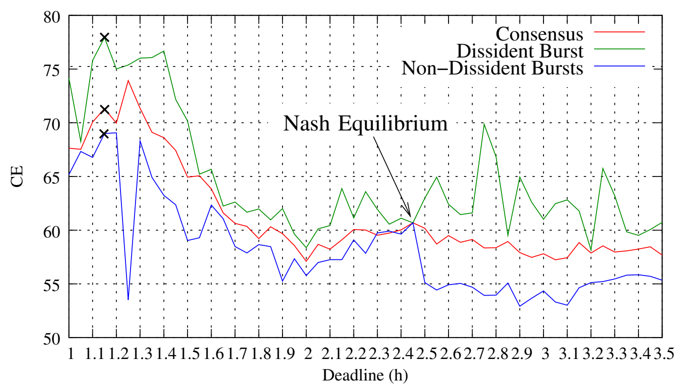

Fig. 12. Illustrating the willingness to depart from a consensus. The consensus line represents the CE CE_{cons} of each burst project if they agree on a given deadline σ_{cons}. The dissident line represents how much a dissident can improve its CE_{dis} by selecting a different deadline σ_{dis}. The impact of such a strategy modification on the CE of other burst projects is depicted by the line $CE_{non-dis}$.

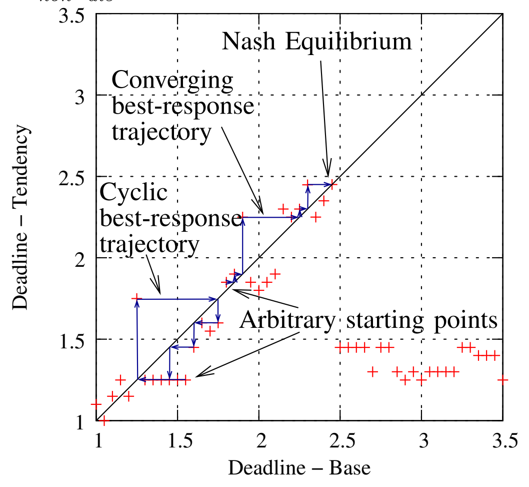

Fig. 13. This plot depicts for a given σ_{cons}, the corresponding σ_{dis} to which burst projects tend to use. With such information, it is possible to depict the outcome of a best-response strategy.

projects (consensus value in Figure 12). For a given value of slack σ_{cons}, a dissident burst project may increase its CE by unilaterally changing its slack to a different value σ_{dis}. We compute this value so that it maximizes the CE_{dis} of the dissident burst project. Yet, this strategy modification generally results in a decrease of the $CE_{non-dis}$ of the other burst projects. Both CE_{dis} and $CE_{non-dis}$ are represented on Figure 12. For example when $\sigma_{cons} = 1.15$ (i.e., a value close to the optimal one found in the previous section), we see that a burst project can change its CE from 71.3 to 77.5 by changing its slack, thus causing a CE decrease of 2 for every other burst projects. As a consequence, even though this deadline leads to the largest CE_{cons}, the other burst projects should disregard $\sigma_{cons} = 1.15$ and switch to a better position as well. Interestingly, the three curves join for $\sigma_i = 2.45$, which is thus a Nash equilibrium.

Figure 13 depicts for a given σ_{cons}, the corresponding σ_{dis} that leads to the best CE_{dis}. Therefore, if all projects used a simplistic best response strategy, we can follow the dynamic of the system and see that it converges (when initial $\sigma_{init} \in [1.8, 2.45]$) to the Nash equilibrium $\sigma_{NE} = 2.45$, that leads to a CE_{NE} of 60.7 for burst projects. Yet, we can read on Figure 12 that a common choice of $\sigma_{cons} = 1.15$ (reference point) would have lead to a $CE_{cons} = 71.3 > CE_{NE}$. Still, it could be the case that this loss of efficiency did benefit to the continuous projects. This is unfortunately not true as can easily be checked by representing the outcome of all such situations in the utility space (Figure 14).

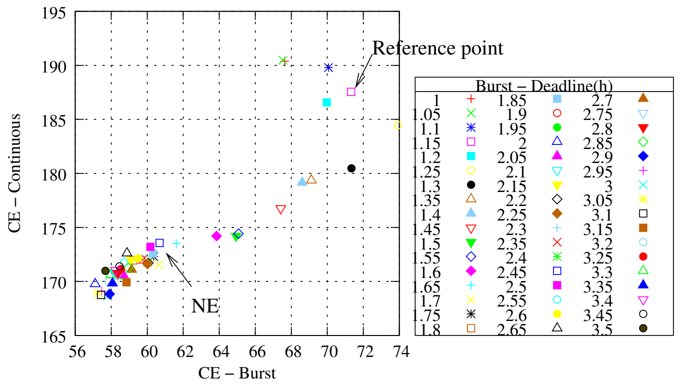

Fig. 14. Sampling utility set and illustrating the inefficiency of the Nash equilibrium. 2 continuous projects and 4 burst project with 900 1 hour tasks per day each. CE could be *improved by more than 10% for all projects* by collaboratively choosing $\sigma_{cons} = 1.15$ instead of the NE ($\sigma_{NE} = 2.45$).

With such a representation, we can check that the Nash equilibrium is indeed Pareto inefficient and that the CE *could be improved by more than 10% for all projects* by collaboratively choosing $\sigma_{cons} = 1.15$.

Such inefficiencies can be observed for many different configurations and the utility set samples seem to always have more or less the same structure. For example, when using a configuration with 8 projects among which 7 of them are burst projects receiving a batch of 600 tasks per day, we also obtain an inefficient Nash equilibrium (see Figure 15). Note that in this example, the continuous project uses 30 hours tasks with a deadline of 300 hours, which is a typical configuration from the SETI@home project.

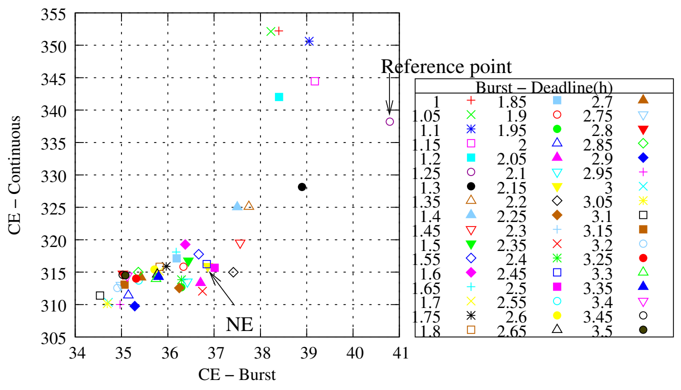

Fig. 15. Sampling utility set and illustrating the inefficiency of the Nash equilibrium. 1 continuous project and 7 burst project with 600 1 hour tasks per day each. CE could be improved by more than 10% for burst projects and by 7% for the continuous project by collaboratively choosing $\sigma_{cons} = 1.25$ instead of the NE ($\sigma_{NE} = 2.45$).

Again, with such a representation, we can check that the

Nash equilibrium is indeed Pareto inefficient and that the CE could be improved by 10% for burst projects and by 7% for continuous project by collaboratively choosing $\sigma_{cons} = 1.25$.

The main conclusion to draw from this set of experiments is that burst projects not only step on each others toes but also impact rather negatively the efficiency of continuous projects (not even speaking about their waste).

V. Conclusion

In this article, we have studied the situation where burst projects interested in response time share resources with classical throughput-oriented projects. We perform a game theoretic modeling and experimental analysis of such system, in order to verify the potential performance issues raised by such complex configurations. Such game theory concepts are rarely applied in such "complex" systems. Indeed, our modeling considers a very large strategy space, a rather large number of players, and a multi-objective optimization problem since project should not only optimize their throughput or their response time but also their waste (i.e.,, the fraction of tasks missing their deadline) and the waste they incur to others.

Our experimental analysis relies on thorough and realistic simulations and enables us to study the influence of the main server parameters (slack, replication, work sending strategy, ...) in a multi-player context. This study illustrates that the current scheduling mechanism is unable to enforce fairness and project isolation (burst projects may dramatically impact the performance of other projects).

In particular, we show that when such burst projects share volunteer machines with continuous projects, the non-cooperative optimization of their project configuration may result in rather inefficient sharing of resources. We exhibit situations where every project could use resources 10% more efficiently if burst projects agreed on some of their scheduling parameters. Even though this result should not surprise people acquainted with game theory, it is, to the best of our knowledge, the first time inefficient Nash equilibrium is exhibited in the context of Volunteer Computing systems. It is also interesting to note that these inefficiencies occur even though the whole system relies on a protocol where each volunteer produces a locally fair and efficient sharing of its computing resources. Such results can be linked with those presented in [16] except that we do not assume an over-simplistic resource model but use a very realistic simulation instead.

As future work, we intend to continue the analyze such situations to verify whether even worse situations can be found. Then, we would like to try to characterize the workload on which these equilibria occur. This would allow to decide on sound basis whether the BOINC architecture should be modified to prevent such situations to happen. Furthermore, the existence of inefficient Nash equilibria may lead the system to even more undesirable behavior, like Braess's paradox where the addition of resources may degrade the performance of every project. Note that game theory provides many tools (correlated equilibria, pricing mechanisms, Shapley value ...) to cope with and improve such situations and that some of them may be used to implement a form of cooperation (possibly distributed) between servers.

Acknowledgment

This work is partially supported by ANR (Agence National de Recherche), project reference ANR 07 JCJC 049 (DOCCA) and the Coordination of Improvement of Higher Education (CAPES). Special thanks to INRIA for partially supporting Bruno Donassolo's internship.

Many thanks to Derrick Kondo who has provided us with many references and information on BOINC.

References

[1] D. P. Anderson, "BOINC: a system for public-resource computing and storage," in *Proc. of the 5th IEEE/ACM Intl. Workshop on Grid Computing*, 2004.

[2] D. P. Anderson and J. McLeod VII, "Local Scheduling for Volunteer Computing," in *Proc. of the Intl. Parallel and Distributed Processing Symp. (IPDPS 2007)*. IEEE Intl., 2007.

[3] D. Kondo, D. P. Anderson, and J. V. McLeod, "Performance evaluation of scheduling policies for volunteer computing," in *Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing e-Science'07*, Bangalore, India, dec 2007.

[4] M. Silberstein, A. Sharov, D. Geiger, and A. Schuster, "Gridbot: execution of bags of tasks in multiple grids," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09. New York, NY, USA: ACM, 2009.

[5] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer, "SETI@home: an experiment in public-resource computing," *Commun. ACM*, vol. 45, no. 11, 2002.

[6] D. Kondo, A. A. Chien, and H. Casanova, "Scheduling task parallel applications for rapid application turnaround on enterprise desktop grids," *Journal of Grid Computing*, vol. 5, no. 4, 2007.

[7] E. Heien, D. Anderson, and K. Hagihara, "Computing low latency batches with unreliable workers in volunteer computing environments," *Journal of Grid Computing*, vol. 7, pp. 501–518, 2009.

[8] D. P. Anderson and K. Reed, "Celebrating Diversity in Volunteer Computing," in *System Sciences, 2009. HICSS '09. 42nd Hawaii*, January 2009.

[9] D. Kondo, M. Taufer, C. Brooks, H. Casanova, and A. A. Chien, "Characterizing and Evaluating Desktop Grids: An Empirical Study," in *Proc. of the Intl. Parallel and Distributed Processing Symp. (IPDPS)*, 2004.

[10] J. F. Nash, Jr., "Equilibrium points in N-person games," *Proceedings of the National Academy of Sciences*, vol. 36, no. 1, pp. 48–49, 1950.

[11] ——, "Non-cooperative games," *The Annals of Mathematics*, vol. 54, no. 2, pp. 286–295, 1951.

[12] R. B. Myerson, *Game theory: Analysis of conflict*. Harvard University Press, 1997.

[13] T. Roughgarden, *Selfish Routing and the Price of Anarchy*. The MIT Press, May 2005.

[14] A. Legrand and C. Touati, "How to measure efficiency?" in *Proceedings of the 1st International Workshop on Game theory for Communication networks (Game-Comm'07)*, 2007.

[15] D. Braess, "Über ein Paradoxen aus der Verkehrsplanung," *Unternehmensforschung*, vol. 12, pp. 258–268, 1968.

[16] A. Legrand and C. Touati, "Non-cooperative scheduling of multiple bag-of-task appplications," in *Proceedings of the 25th Conference on Computer Communications (INFOCOM'07)*, Alaska, USA, may 2007.

[17] H. Casanova, A. Legrand, and M. Quinson, "SimGrid: a Generic Framework for Large-Scale Distributed Experiments," in *Proc. of the 10th IEEE Intl. Conf. on Computer Modeling and Simulation*, 2008.

[18] B. Donassolo, H. Casanova, A. Legrand, and P. Velho, "Fast and scalable simulation of volunteer computing systems using simgrid," in *Workshop on Large-Scale System and Application Performance (LSAP)*, 2010.

[19] D. Kondo, B. Javadi, A. Iosup, and D. Epema, "The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems," in *Proc. of the IEEE Int. Symp. on Cluster Computing and the Grid (CCGrid)*, 2010.