Scheduling for Volunteer Computing on BOINC server infrastructures

Internship report
June 8, 2011

Duco van Amstel

LIG - Grenoble, France

Supervisor : Arnaud Legrand
April 1, 2011 - June 8th, 2011

Acknowledgements

I would like to express my great gratitude towards Arnaud Legrand, my advisor, who despite his own full schedule managed to accompany me throughout my short but instructive internship. He offered me a subject that not only allowed me to get results within the short allowed time-frame but also to learn much more about scheduling issues. I would also offer some special thanks to Panayotis Mertikopoulos for his great insights into the mathematics behind fairness criteria as well as for his unlimited amount of positive thinking. Last but not least I am grateful to the entire team of colleagues that I learned to know during my stay at the LIG office in Montbonnot, for the vivid discussions, the many cups of coffee shared and games of Go that were played.

Introduction

One of the first and main usages for which computers were developed is the computation and solving of numerical problems that are too vast or too long to be done by hand using the human brain. This has been in a straight line with the historical trend for scientific research to come up with more and more effective ways to manage computationally intensive problems: from hand writing to abacuses, to calculators, to mainframes, to super-calculators and more recently large scale distributed computing systems. The rationale for such a development is simple: the more computing resources are available, the more computational work can be processed. Hence the success of Volunteer Computing that allows *volunteers* to donate their computers' idle cycles to scientific or humanitarian projects. The Berkeley Open Infrastructure for Network Computing (BOINC) is the most popular Volunteer Computing infrastructure today with over 580,000 hosts that deliver over 2,300 TeraFLOP per day [1].

It should be noted that using a computational tool such as BOINC for a scientific project requires a considerable effort; a complex hardware infrastructure has to be maintained and a large amount of publicity is necessary to attract volunteers to the new project. These requirements are prohibitive for small-scale projects and as a consequence a new way of using BOINC has been observed over the last few years: umbrella projects. An example of such a project is the World Community Grid operated by IBM. This project proposes to scientific teams to host their computational tasks thereby eliminating the need for both infrastructure maintenance and publicity. As a consequence much smaller research teams can access the computational resources that Volunteer Computing offers.

However this evolution of Volunteer Computing calls for more elaborated scheduling strategies. The existing BOINC software, as an evolution of the initial SETI@Home project [21, 2] was designed to optimize the computational throughput of the server. That is to say that BOINC optimizes the number of calculations done per time-slot but not, for example, the time necessary to run a particular job or sub-set of the whole computation. Furthermore the scheduling mechanism currently used in BOINC has been developed for a single project running on the server whereas umbrella projects host multiple projects. Last but not least, a single project may work on several sets of data at the same time.

The first research goal of this report is to study the structure and abstractions necessary for the fair scheduling of multiple projects capable of each running multiple campaigns of computations over time. A second goal involves the analysis of existing solutions provided by related work. The third and last goal is to propose a new scheduling mechanism that adapts well to Volunteer Computing umbrella projects. This report will start with a short presentation of the history and principles underlying Volunteer Computing. We will also present the mechanisms that are currently used in such systems. Secondly we will show the hierarchical structure that is required for a fair scheduling which leads us to the study of the individual layers of this hierarchy. From there on we propose a new scheduling heuristic and end this report with some preliminary results from our simulations.

Contents

Acknowledgements **i**

Introduction **iii**

Table of contents **v**

1 Scheduling in the Volunteer Computing Context **1**
1.1 Volunteer Computing in a Nutshell 1
1.1.1 SETI@Home and the Development of BOINC 1
1.1.2 Toward Umbrella Projects and Online Submission 3
1.2 Optimization of Campaign Completion 3
1.3 A Theoretical Perspective on Campaign Scheduling 5
1.4 Monolithic Organization 6

2 A hierachical approach to the analysis of scheduling **7**
2.1 Independent Optimization Levels 7
2.1.1 Campaign-level Scheduling 7
2.1.2 Application-level Scheduling 8
2.1.3 Server-level Scheduling 9
2.2 A Novel Heuristic: Pressurized Scheduling 9
2.2.1 Rationale 9
2.2.2 Defining and Enforcing a Deadline 10
2.2.3 Scheduling Campaigns and Applications 10

3 Experimental result with SimGrid **11**
3.1 Simulating BOINC with SimGrid 11
3.2 Preliminary Results 11
3.2.1 Experimental Configuration and Results 12
3.2.2 Analysis of the Results 12

Conclusion **15**

Bibliography **16**

Appendices **18**
A. Vocabulary 18

Chapter 1

Scheduling in the Volunteer Computing Context

1.1 Volunteer Computing in a Nutshell

In this Section, we briefly present key concepts of Volunteer Computing as well as the usage for which it has been successful. Then, we describe a recent evolution of its usage and why such evolution requires major modifications to the current server architecture.

1.1.1 SETI@Home and the Development of BOINC

Confronted to an enormous amount of data to process and analyze, the SETI[1] project developed a server infrastructure [21] in which volunteers could install software on their personal computers and donate all their unused computational power to the project. The client software would contact the main server with a request for a computational job which it would then run whenever the computer was idle. Results would be sent back to the server whenever the job is completed. The project was launched in the first half of 1999. Such an infrastructure was similar to that of the Distributed.net project launched two years earlier in 1997 that focuses on cryptographic challenges and is still running today [7]. The SETI@Home initiative, as it was called, became very popular and quickly grew to become the largest distributed computing infrastructure in the world. This success allowed the SETI project to process all the data it was acquiring over time. To extend this approach to other scientific domains, the SETI@Home code was further developed and open-sourced to become the BOINC software [1].

On a high-level view, the scheduling mechanism of such a system is very simple: [2]:

1. The computational work that has to be done is divided into *tasks* on the server;
2. Each task is allocated to a volunteer;
3. Upon completion, volunteers send the result back to the server and receive computational credits in return;

In practice, there are several issues that need to be addressed to obtain an efficient system:

[1] Search for Extra-Terrestrial Intelligence

[2] A precise definition of all the vocabulary pertaining to BOINC is given in Appendix A.

Pull mode Most volunteer resources are behind firewalls or private networks and many only have a limited connectivity. It is thus impossible for the server to contact clients directly, which imposes a pull-mechanism: clients have to issue a *request* to the server to obtain some tasks to process. Some projects may have large periods of time with no task to perform and it is thus impossible for the server to warn clients whenever new tasks are available. This issue is simply solved by adding a re-connection interval to work requests that cannot be fulfilled. Thus, clients check for task arrival with a server-defined frequency.

Network traffic Some clients have a low connectivity and thus need to minimize their connection to the server. Hence, a client typically requests for several days of computation to ensure the resource will not run out of work, even if it is disconnected from the Internet for a long period of time. Likewise, a client generally does not send results as soon as they are obtained but aggregates them and tries to send them at once if possible. Such a mechanism allows also to decrease the network traffic on the server side.

Invalid results The results that are returned by the clients are not always correct. Multiple issues can influence the quality of the obtained results. Some clients for example have enabled the overclocking of their CPU in order to increase their computational power. In other cases the performed computation itself is unstable and results may thus change between operating system types and versions. There is also the possibility of encountering malicious clients that try to acquire more credits by faking results. As a consequence the server runs the same task on multiple hosts, a process that is called replication. To avoid the instability inherent to an algorithm the replication is done on similar systems, a process that is called homogeneous redundancy. For a task to be considered finished the server has to obtain a minimum number of valid results. This number is called the quorum.

Churn and unreliable clients As the clients are made up of hosts that belong to volunteers they do not have a continuous availability and up-time. The owner of the client may leave the project without notice or not turn on his computer for several days or weeks. In this case the server has to wait for a long period before receiving the results of the tasks that were allocated to this client. Another consequence is that the server needs to maintain the task in its database for a longer period, a situation that should be avoided in the case of projects handling over a million results each day[3]. The solution here is the presence of a deadline associated with each task. Whenever a client fails to meet the deadline of one of its tasks the server may sent a new replica to another client. The deadline itself is defined based on a parameter called slack. The actual deadline is defined by the start date of the task to which is added the average processing time multiplied by the slack.

Volunteer community A Volunteer Computing initiative relies by definition on the participation of a large number of volunteers. These volunteers donate their computing power and expect in return that their donation is spent in a useful way. If the scheduling policy of the project is prone to waste resources, for example by using to much replication, volunteers may stop supporting the project. This means that every change in the functioning of the project should be evaluated carefully before it is brought out on the real-world system.

Fair sharing between projects A client may participate in multiple projects. The resource-share between these projects is manually defined by the owner of the client. The client software then uses a *Fair-Sharing* policy to schedule tasks belonging to different projects. A second layer in

[3]The World Community Grid handles such a quantity

the client tries to respect the deadlines associated to tasks and may thus resort to an Earliest-Deadline-First policy. Again another layer tries to enforce a long-term fairness between the projects to correct any unfairness induced by the *Fair-Sharing* policy.

These various issues form the basis of the research in the Volunteer Computing context. The solutions that are applied in the case of BOINC result in an almost optimal scheduling relatively to the throughput of the server, *i.e* the number of tasks completed each day. However there has been a growing demand for the possibility to optimize other criteria in order to adapt to new use cases for Volunteer Computing.

1.1.2 Toward Umbrella Projects and Online Submission

Over time a new type of BOINC project has appeared. These servers do not differ in the software that is used but they do in the usage that is made of the service. Whereas projects such as SETI@Home run their own dedicated server, many smaller teams do not have the infrastructure, funds or size required to support a Volunteer Computing initiative. Therefore so-called *umbrella projects* have appeared that host multiple computational experiments, called *applications* in this report, simultaneously.

An example of this is the World Community Grid. The developers of BOINC have further predicted the evolution of such projects towards hosting even smaller computational jobs in the spirit of what is done more classically in the grid context. This would be in a much more interactive way: users could submit a single or multiple *campaigns* to be run on the resources of the project, a campaign being an independent set of data associated with the algorithm to be performed on it. The users then wait for the response from the server and it is up to the server to schedule the execution of the requests. The GridBot project [19] is a typical such BOINC project where users submit requests through a portal but only one application is supported. There is thus a real need for a support of multi-application/multi-user *response-time* oriented scheduling.

Such an evolution however requires a major modification of the BOINC software. As there are two sides to the software: the client-side and the server-side one could try to optimize both. However the number of clients involved and the fact that there are numerous implementations of the client software prevents any practical modification of this side. We will therefore now only focus on the server-side and the elements pertaining to the scheduling of tasks from the point-of-view of the project.

1.2 Optimization of Campaign Completion

The Straggler Problem

The change in optimization criterion from throughput to response-time that corresponds to the evolution of the BOINC use-cases as explained in Section 1.1.2 raises another issue that existed only partially until now. As the current version of BOINC has been designed as to optimize the throughput value [3, 11] no attention is paid to the response-time of individual tasks. Previous studies have observed a specific behavior in the execution of campaigns on current Volunteer Computing networks that relates directly to the response-time of the tasks that compose the campaigns. At the start of a campaign we observe a near-linear progression in the number of tasks that are completed over time. However the last tasks of the campaign can take a significantly longer period before completion: these tasks are called *stragglers* [14]. This problem especially arises when the last replicas of a campaign are sent to frequently inactive and/or slow clients and the completion of

corresponding tasks takes a long time. The significant way in which this delays the completion of a whole campaign can be seen in Figure 1.1.

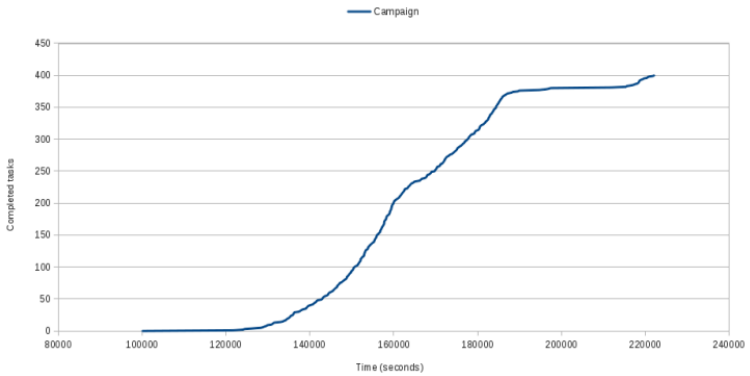

Figure 1.1: A campaign with a straggler issue

Besides impacting the response-time of campaigns this issue has also a more practical impact on the server in the same way the absence of deadlines on tasks would. Because the tasks that have not yet been completed need to be maintained in the server-database together with all the information related to the almost finished campaign the straggler problem can have considerable effect on the server infrastructure. This should again be considered while keeping in mind the orders of magnitude that are involved in Volunteer Computing initiatives where the number of tasks managed each day is counted in hundreds of thousands.

Aggressive Replication and Client Selection

The GridBot initiative has proposed two solutions in order to solve the straggler problem [19]:

Aggressive replication As the straggler problem originates with clients that are inactive and/or slow an intuitive idea is to allow campaigns that are almost finished to replicate tasks more than the standard quorum number limit. This is called aggressive replication. As a consequence, with more replicas being processed there is a higher chance of having a quick return of results and to attain the quorum. However a significant drawback is the computational time and power lost by the clients who's result arrives after the end of the campaign. This lost quantity is also called *waste* and has, as described earlier, a significant psychological effect on the volunteers participating in the project [18]. In order to set a bound to the generated amount of waste a maximum amount of replicas is set.

Client Selection Using inactive or slow clients to end a campaign is ineffective. It has thus been proposed to select only adequate clients for finishing the last tasks. These clients should ideally be available continuously and have high computational power. This solution has been tested within the context of GridBot which is different from the standard Volunteer Computing concept as it relies on the hosts of a dedicated grid to perform the last tasks of a campaign.

Detecting the End of Campaign

Optimizing the end of a campaign relies directly on the definition of what the end represents and several possibilities arise. It could for example be done by relying on a fixed completion percentage

or a fixed number of remaining tasks. However the campaigns may not all be of the same size and the straggler issue itself is not related to the size of the campaign but to the type and number of clients that compose the resources of the server. Another possibility would be to define a threshold of remaining tasks based on the number of clients that participate in the project. The number of clients has directly been related to the appearance of the straggler problem [13]. We will thus specify the end of a campaign when the number of tasks that remain drops below the number of clients that work for the project. However this definition and the optimization of task completion do not suffice for a BOINC server to optimize campaign response-times and the fair sharing of resources among projects. The scheduling structure itself has to be adapted in a more fundamental way.

1.3 A Theoretical Perspective on Campaign Scheduling

At high level, an umbrella project has several campaigns, each belonging to a particular application. As we have explained earlier, the scheduling of jobs within a campaign to optimize the campaign completion time is a problem in itself that can be addressed in various ways. Another novelty of this context is the fact that several campaigns can now be ready for execution and that it is the responsibility of the umbrella server to fairly share resources between the campaigns.

A natural strategy would be to allocate an instantaneous fair share of resources to all campaigns. This is for example what is currently implemented in the World Community Grid project. Although this is a reasonable approach when optimizing throughput, it is known to be a bad strategy when optimizing individual response-time. Indeed, assume two campaigns are submitted at the same time in the system and would take one day each if they were allocated all volunteer resources. If the server allocates half of the resources, they will both complete within two days. However, if the server decides to allocate all resources to the first campaign then all resources to the second campaign, the campaigns will respectively complete within one day and two days. By processing campaigns in sequence, one of the campaign has a reduced processing time whereas the processing time of the second is unchanged. Although this may seem unfair at first sight, by randomizing the choice of which campaign is processed first, both campaigns could get an expected running time of one day and a half, which is strictly better than what would be obtained through instantaneous fair sharing (FS).

At the campaign level, the server is thus solving an online scheduling problem where campaigns are released one after the other and are to be fairly distributed amongst resources. A similar problem was already studied and we briefly review the main concepts [15, 16]. If we consider the set of volunteer resources as a single processing resource, the campaign scheduling reduces to online scheduling of preemptive jobs on a single processor. Using the Graham notation [9, 17], this problem is denoted as $\langle 1|pmtn, r_j|.\rangle$. A key question to fairly share resources is thus to come up with a "fair" objective function involving response time.

The flow F_j of a job is the difference of its completion time C_j and of its release date r_j. It corresponds to the response-time as perceived by a user that would submit the campaign. It is easy to show that the well-known *First-Come First-Served* (FCFS) strategy is optimal for $\langle 1|pmtn, r_j|\max_j F_j\rangle$. Such a criteria is however arguably fair and does not really correspond to the idea one may have about reactivity. Instead of focusing on the worst case, optimizing the average flow could seem more representative of the overall system performance. Again, it is not difficult to show that the *Shortest Remaining Processing Time* (SRPT) strategy that schedules at any time the campaign with the smallest remaining amount of work is optimal for $\langle 1|pmtn, r_j|\sum_j F_j\rangle$. Such a strategy requires to have a good estimation of the amount of work to be processed for each campaign but this is a reasonable assumption in our context.

It has been proved in [15, 16] that $\max_j F_j$ and $\sum_j F_j$ cannot be simultaneously approximated. Hence, any guaranteed heuristic for $\sum_j F_j$ (with a competitive ratio strictly better than the one of FCFS) may lead to starvation of one of the campaigns, which may be problematic in this context. Other more elaborated strategies (based on linear programing and recursive optimization) have been proposed [15, 16] but we will restrict our study to the simple classical FS (the one currently used in the World Community Grid), FCFS and SRPT. We will also evaluate both maximum response time (flow) and average response time to make sure that the overall performance is correct while no particular job is particularly slowed down compared to the others.

1.4 Monolithic Organization

The existing server and client architecture of BOINC could be described as a coherent block. On the server-side the scheduling method does not allow for the specification of separate applications or campaigns. This means that using the current implementation for an umbrella project such as the World Community Grid would result in a far-from-optimal scheduling with respect to the response-time of individual campaigns. Another problem would be the control of the resource-share between separate applications as these are not recognized either by the server. This brought the World Community Grid to implement a work-around under the form of *Fair-Sharing* scheduling.

Fair-Sharing implies that the scheduling instance keeps track of the computational power allocated to each application. The application with the lowest resource-share is systematically prioritized to obtain an equilibrium. The distinction between applications still remains minimal and the *Fair-Sharing* does not allow at all to maintain seperate campaigns: the server structure remains monolithic. For the predicted evolution in the usage of BOINC as posed in section 1.1.2 to take place this monolithic organization should be reconsidered. Instead of having a single block the structure itself suggests a hierarchical vision. Our next step will thus be the presentation of this new organization.

Chapter 2

A hierachical approach to the analysis of scheduling

In the new BOINC server usage and structure four seperate entities can be distinguished:

1. computational tasks
2. campaigns made up of computational tasks
3. applications that contain one or more campaigns
4. the server that runs a set of applications

This also thus depicts a hierarchy as the four entities are enclosed one inside the next one up to the unique server that encompasses all of them. As we will show in this chapter these hierarchical levels are independent one of another and corresponds each to a different optimization issue. In order to do this we will use a bottom-up approach starting with the individual tasks and ending with the applications that run on the server.

2.1 Independent Optimization Levels

2.1.1 Campaign-level Scheduling

Task Units

The smallest entity that can be found on a BOINC server is the computational task. A task may have multiple instances: the replicas, however these are not considered as full entities as they are mere seperate representations of the same object. As stated previously, in order for a task to be considered completed it the server should receive a minimal quorum of valid results corresponding to the given task. This is combined with the fact that the server initially releases the exact number of replicas for each task requested by the quorum and the fact that further replicas are only being added when invalid results are received or time-outs occur. Earlier on we described the resulting straggler issue that is associated with these time-outs and invalid results, especially when slow clients are involved. We also presented two solutions that have been presented in papers related to the GridBot project.

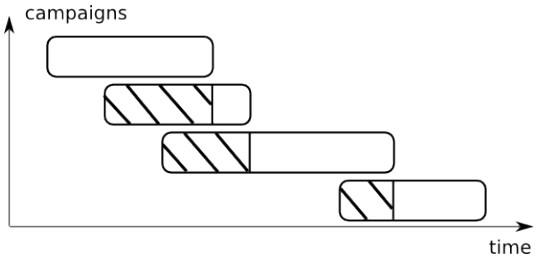

Figure 2.1: Excessive response-time for short campaigns in FCFS-scheduling

Dynamic Slack

Aside from these two existing methods we here present a third one that deals with the slack parameter of the server. As explained this parameter influences the deadline of replicas that are sent to clients. In their turn the clients adapt their local scheduling to keep up with this deadline. The modification we propose is the dynamic definition of the slack parameter for each replica. We should keep in mind that using a slack value that is too low may result in clients failing to meet the imposed deadline and thus the accumulation of waste. It has however been shown that for values down to 4 there is no impact on the amount of waste that is generated [12]. By reducing the value of the slack for close-to-end campaigns we force clients to prioritize the execution of these tasks.

Parameter Specification

In our simulations that will be presented in Chapter 3 we implement the three presented optimizations. We allow for aggressive replication up to two times the quorum value and perform client selection by only assigning tasks to fast clients. These are defined as having a speed above the average of currently active clients. Concerning the dynamic slack definition our implementation predicts the finish-date of a campaign by the current rate at which results are received. It then adjusts the slack parameter for the campaign to have replica deadlines corresponding to this date with a minimum slack value of 5.

2.1.2 Application-level Scheduling

Campaigns and Their Completion Time

Above the task-level the next hierachical level of scheduling is that of the multiple campaigns that belong to the same application. For this particular level the optimization issue is the completion date of each campaign. In cases where there is a dependency chain with think-time, *i.e* a given campaign has to be completed in order to analyze the results and start a new campaign, the completion date is even more important.

Optimization

A *First-Come First-Served* (FCFS) scheduling strategy would allow for all campaigns to be certain that they will receive the highest priority at a given moment. A drawback would be that during the processing of a large campaign, a small campaign will have to wait before being processed. This means that the second campaign could be waiting more time than it's actual processing would take as is shown in Figure 2.1.

As an alternative strategy one could take the *Shortest Remaining Processing Time* (SRPT) scheduling. This would prioritize the small campaign of the previous FCFS scheduling issue. When being flooded with small campaigns this strategy would suffer from the inverse drawback of delaying the larger campaigns of the application, for an infinite period of time in a worst-case scenario. Knowing in advance when applications release new campaigns could enable us to choose between the two scheduling policies but such a model does not exists as stated earlier.

Last but not least remains the already presented *Fair-Sharing* policy. It has the advantage of allocating resources to all projects, the disadvantage being that it does not allow for any type of prioritization.

2.1.3 Server-level Scheduling

User-specific Applications

The highest level of the scheduling hierarchy comes in with the contention for resources between applications. Each application is associated with a specific user. A user can represent an individual, a research team or any other entity that wants to use the BOINC project. If our hierarchy had stopped at the application-level then the scheduling could have been easily exploited by a malicious user: this user would fork his campaign into multiple smaller ones that each contend for resources on an equal basis, an issue that has already been observed in networks [4]. For this reason the server-level scheduling issue is that of giving a fair share of the available resources to each user.

A Fair Resource Distribution

The policy that can be used here is in accordance with the objective: *Fair-Sharing* scheduling. This strategy is the best algorithm to ensure that each user receives an equal amount of processing power. It can also easily be adapted to allow for the server administrator to give preference to an application. This would be through the specification of a multiplicative coefficient $k < 1$ on the allocated resources of the application.

2.2 A Novel Heuristic: Pressurized Scheduling

In the previous section we showed the different scheduling possibilities for the application-level. We also noted that the three proposed policies are all flawed in their worst-case scenarios. In order to propose a scheduling that is both attentive to response-time and to the fair-sharing of resources we developed a new measure to be optimized by a specific algorithm. Both will be specified in this section.

2.2.1 Rationale

When looking to the existing FCFS and SRPT policies one can observe that they are greedy algorithms performing on the current situation of the server. They do not rely on knowledge of either past or future to function. A consequence of this is that they do not account for any possible unfairness that happened in the past. This can directly be related to their worst-case scenarios: in our example for the SRPT measure it does not account for the fact that the multiple small campaigns have been delaying the large campaign for a long time. On the opposite side the *Fair-Sharing* policy does account for past unfairness by relying on the resource allocation over the last time-period. The drawback of this strategy is its monotonous nature as all campaigns are automatically allocated the

same resource share, or in other words the fact that *Fair-Sharing* is not "conscious" of any form of deadline for a given campaign.

2.2.2 Defining and Enforcing a Deadline

In order to optimize a completion date we should ask the question of how to seperate a good from a bad result. This means we have to define an expectation of what the completion date should be. We thus define a deadline that reflects this expectation and a baseline to establish this is to pose that resources should be shared in a fair way. This brings us to the deadline τ_c of a campaign c depending on the available resources R_A for the corresponding application A that runs n_A campaigns, on the remaining amount of work to be done W_c as well as on the current time stamp t. The final formula is then

$$\tau_c = t + \frac{W_c \cdot n_A}{R_A}$$

As the formula depends on environment variables such as n_A and R_A deadlines should be refreshed every time their values change in a significant way, *i.e* when a campaign is started or finishes.

Between the moment a campaign starts and the moment at which it finishes the ideal situation would be a near-linear progression in the number of completed tasks. As a consequence there is need for a measure that indicates the manner in which the progress of a campaign is on schedule or not. We call this measure the pressure of a campaign : being late on the schedule to meet the deadline induces a high pressure and being early inversely provokes a low pressure value. We define the following additional variables: μ the advancement of the campaign (between 0 and 1) and s_c the starting date of the campaign. The pressure ρ_c of the campaign then is

$$\rho_c = \frac{\tau_c - t}{\mu \cdot (\tau_c - s_0)}$$

2.2.3 Scheduling Campaigns and Applications

Once the pressure measure is defined the priority on campaigns is straightforward: the greater the pressure, the higher the priority. This is defined for the campaigns of an application. Furthermore the pressure measure can also be extended to the server-level in order to schedule applications. To achieve this we can add the pressures of all campaigns and so define the general pressure of the application to which they belong. The variable number of campaigns per application is already been taken into account in the definition of the deadlines and there is thus no compensation necessary to compare between different applications. In the last part of this report we will show some results of our simulations. These include all described scheduling policies in order to give a preliminary analysis of the effectiveness of the proposed optimizations and the new hierarchical model.

Chapter 3

Experimental result with SimGrid

3.1 Simulating BOINC with SimGrid

As stated in section 1.1.1 the client part of the software has a very complex behavior that prevents any mathematical modeling. Modifying the BOINC software is possible since BOINC is open source. Yet, although we have contacted the developers of the World Community Grid during this internship to obtain information about their system, experimenting on a real production system is extremely difficult and would not have been possible in such a short time-frame as it could take several months before being able to asses the impact of a server modification. Hence, we resort to realistic simulation that account for the complexity of such systems.

To this end, we choose to use the open-source simulation toolkit SimGrid [5, 20]. This toolkit allows for fast simulation development with an efficient execution core capable of reproducing the behavior of multi-threaded applications distributed over many nodes that are linked together through a realistic user-specified network. SimGrid has grounded more than a hundred of articles in the fields of Distributed Computing and Network Applications and is still under active development. The rationale for the choice of SimGrid also included the proximity of the development team as well as an already existing BOINC simulator coded using SimGrid [8] and which comprises a very detailed simulation of both client and server components. This simulator was directly derived from the real-world client code, the same being true for the server code. The simulator was then tested in comparison with data from real BOINC projects in order to experimentally validate the simulated behavior [8]. This simulator has been used to study the consequences of non-cooperative scheduling in a system such as BOINC [6].

This simulator was originally 2 000 source lines of code. No modification of the client was required but the server part was totally rewritten to reflect the previously described hierarchical structure of a multi-application, multi-campaign BOINC project. This rewriting and the development of all different scheduling strategies and statistics output results in a 3 000 line simulator.

3.2 Preliminary Results

Because of the short time that has been available to run simulations during the internship the results presented below are a mere outline of an in-depth study that still has to be performed. They should be considered as a strong encouragement for the continued study of the scheduling policies that were presented within this report.

Application	First release	Release interval	Tasks per campaign	Quorum	Average task duration
1	0	100000	1500	3	$\tilde{2}200$
2	60000	60000	1300	2	$\tilde{1}500$

Table 3.1: Application characteristics and release schedules. Time is given in seconds.

3.2.1 Experimental Configuration and Results

The set of results we present in this report was obtained using a set of 100 clients fed with real availability traces. Clients' traces were taken from the SETI@Home project, available at Failure Trace Archive (FTA) [10]. 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 100 clients.

For our simulations we used a BOINC server with two applications. Both applications run multiple campaigns over time that are released at regular intervals. The campaigns of the same applications share the same characteristics which are given in table 3.1 together with their release schedule.

Statistics were gathered on all scheduling levels and the simulations that were ran include all possible combinations of application and campaign scheduling policies: *Fair-Sharing* (FS), *Pressurized* (PR), /emphFirst-Come First-Served (FCFS) and *Shortest Remaining Processing Time* (SRPT). For each combination three runs were performed:

- Without dynamic slack, aggressive replication or client selection
- With dynamic slack and client selection activated
- With dynamic slack, aggressive replication and client selection activated

A first result that we can interpret is the evolution of the completion rate of the campaigns over time. This will enable us to observe the effect of task optimizations on the straggler problem. We have traced the completions rates for two specific cases: FS/FCFS scheduling without any optimization relating to the straggler problem and with variable slack and client selection. The results are presented in Figure 3.1.

As stated in Section 1.3 a second characteristic in which we are interested is the response-time of individual campaigns. Ideally we are looking for a combination of scheduling policies at the application and server-levels that results in a low average response-time while trying to minimize the maximum response-time. Figures 3.2 and 3.3 represent the average (resp. the maximum) response-time of campaigns for each application. From left to right are represented the simulation with no task optimizations (left), only dynamic slack and client selection (middle) and all optimizations activated (right). Figure 3.4 represents the waste rate for the same three cases.

3.2.2 Analysis of the Results

Eventhough these experimental results are insufficient by far to draw any conclusions about the effectiveness of each scheduling policy they do allow us to construct a standard experimental protocol for further experimentations.

On one hand we can deduce from Figure 3.1 that the usage of client selection and a dynamic slack definition has a positive effect on the straggler issue. On the other hand, the relatively bad performances observed in the graphs on the right side in Figures 3.2, 3.3 and 3.4 seem to indicate

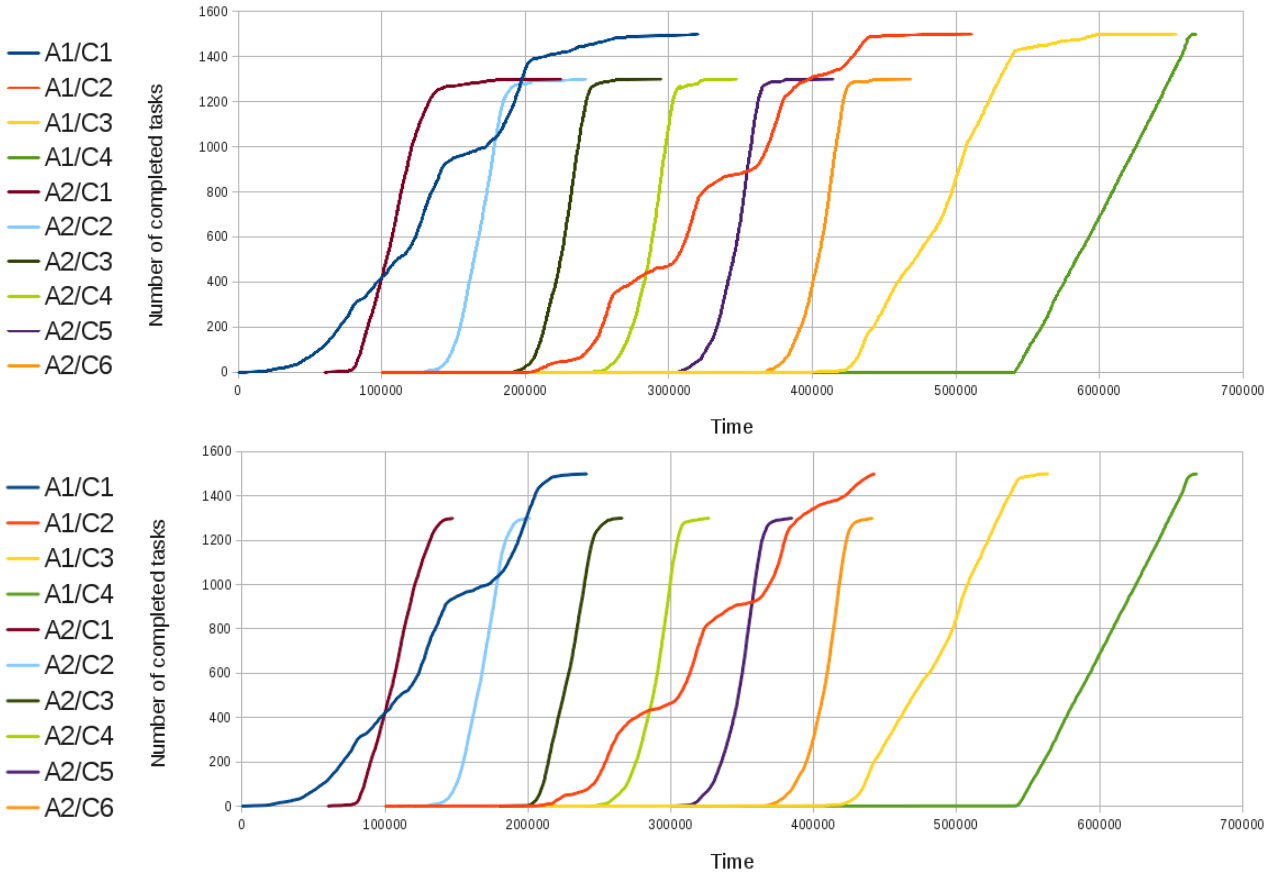

Figure 3.1: Campaign completion over time under FS/FCFS scheduling with no task optimizations (top) / dynamic slack and client selection (bottom)

that the aggressive replication method as we have implemented decreases performances. The most noticeable effect of the aggressive replication is the increase in the waste-rate by two orders of magnitude. This however does not imply directly that aggressive replication can not be usefull, it merely indicates the need for further study and simulations.

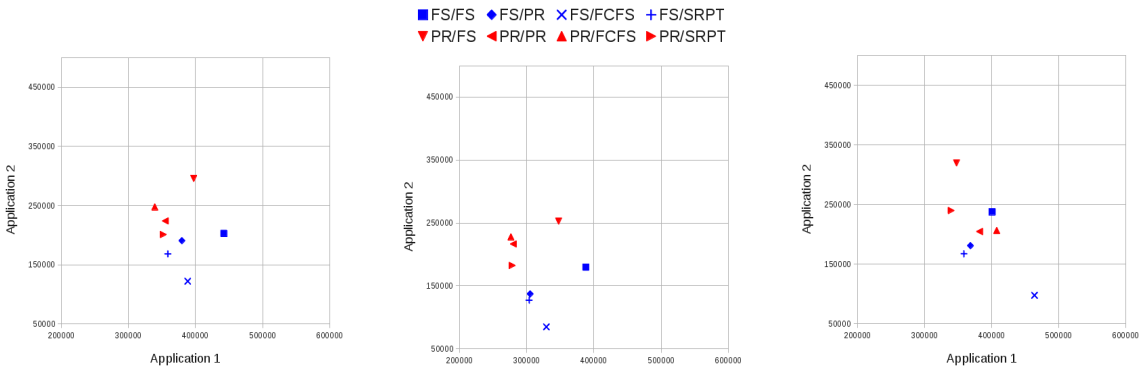

Figure 3.2: Average response-time for each application/campaign scheduling combination

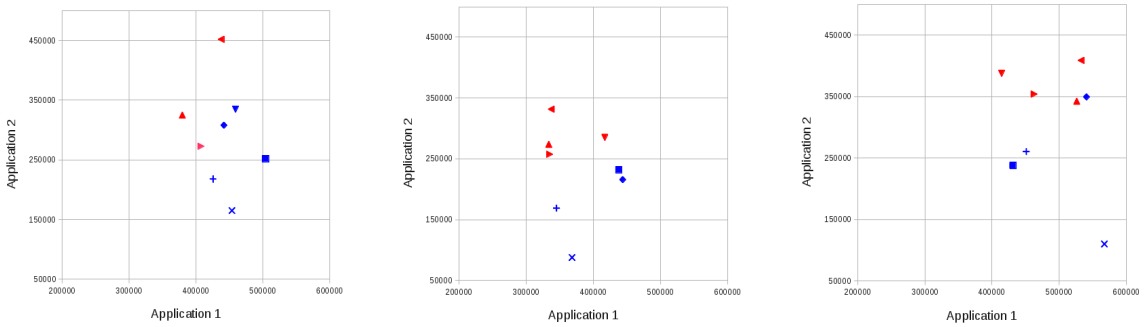

Figure 3.3: Maximum response-time for each application/campaign scheduling combination

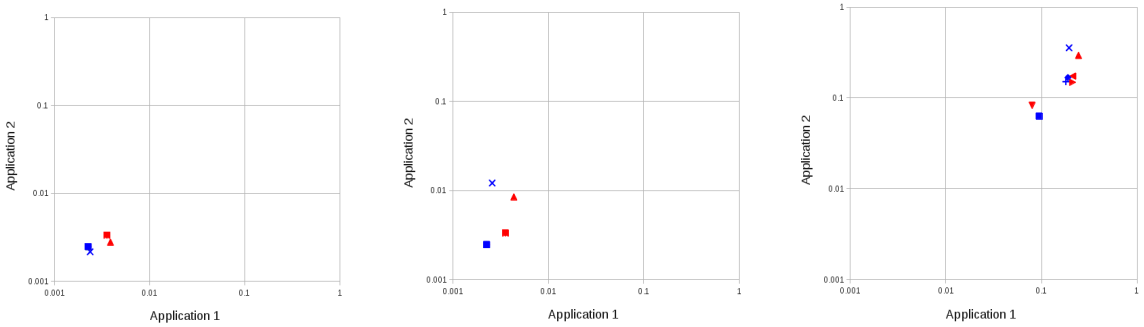

Figure 3.4: Waste rate for each application/campaign scheduling combination

Conclusion

At the start of this internship the goal was to obtain a better understanding of the scheduling mechanisms that are involved in the current version of the BOINC Volunteer Computing software. This had to be done while keeping in mind the current and future evolutions of the system. In the first part of this report we presented the origin of BOINC and the global context of Volunteer Computing, its issues and the solutions that BOINC has developed. This enabled us to introduce some complementary issues relating to new use-cases for Volunteer Computing as well as some related-work on the topic of scheduling.

The second part build on our analysis of the current version of BOINC and presented an alternative point-of-view on the scheduling structure. This new hierarchical structure is better adapted to the multi-application/multi-campaign outlay of recent Volunteer Computing initiatives. Aside from the hierarchy we have shown the scheduling policies and optimizations specific to each level. This resulted in the developement of a new strategy, named *Pressurized Scheduling* (PS). The pressure measure favors long-term over short-term fairness by including past and current events in order to anticipate the future.

Finally, in the last and third part, we took up the challange of evaluating the various optimization and scheduling proposals that were made earlier in the report. Eventhough the simulations that were made are far from being extensive they gave us some indications on what type of further research and simulations are needed. Our current results indicate that the combination of two campaign-level optimization, namely dynamic slack and client selection, can have a positive influence on the straggler problem.

Future work in line with this report would need to extend the diversity of experimental use-cases on which our simulations are being performed. This would allow to conform or infirm our initial conclusions based on the preliminary experiments. If the success of the campaign-level optimizations and the *Pressurized scheduling* are confirmed than this would allow for experimentations on a larger scale, for example within the World Community Grid. This would also be the first step towards an eventual integration of these modifications into the BOINC open-source software itself and the birth of a new way of using Volunteer Computing initiatives as a resource for distributed computing.

Bibliography

[1] David P. Anderson. BOINC: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID '04, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[2] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@home: an experiment in public-resource computing. *Commun. ACM*, 45(11), 2002.

[3] David P. Anderson and John McLeod. Local scheduling for volunteer computing. *Parallel and Distributed Processing Symposium, International*, 0:477, 2007.

[4] Bob Briscoe. Flow rate fairness: Dismantling a religion. *ACM SIGCOMM Computer Communication Review*, 37(2):63–74, April 2007.

[5] Henri Casanova, Arnaud Legrand, and Martin Quinson. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *Proc. of the 10th IEEE Intl. Conf. on Computer Modeling and Simulation*, 2008.

[6] Bruno De Moura Donassolo, Arnaud Legrand, and Claudio Geyer. Non-Cooperative Scheduling Considered Harmful in Collaborative Volunteer Computing Environments. In *Proceedings of the 11th IEEE International Symposium on Cluster Computing and the Grid (CCGrid'11)*. IEEE Computer Society Press, may 2011.

[7] Distributed.net. http://www.distributed.net.

[8] Bruno Donassolo, Henri Casanova, Arnaud Legrand, and Pedro Velho. Fast and scalable simulation of volunteer computing systems using simgrid. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 605–612, New York, NY, USA, 2010. ACM.

[9] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[10] 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.

[11] Derrick Kondo, David P. Anderson, and John VII 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.

[12] Derrick Kondo, David P. Anderson, and John VII McLeod. Performance evaluation of scheduling policies for volunteer computing. Rapport technique, GRAND-LARGE - INRIA Futurs , Berkeley University of California - UC BERKELEY , Sybase Inc., 2007.

[13] Derrick Kondo, Andrew A. Chien, and Henri Casanova. Resource management for rapid application turnaround on enterprise desktop grids. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, SC '04, pages 17–, Washington, DC, USA, 2004. IEEE Computer Society.

[14] Derrrick Kondo, Andrew A. Chien, and Henri Casanova. Scheduling task parallel applications for rapid application turnaround on enterprise desktop grids. *Journal of Grid Computing*, 5(4), 2007.

[15] Arnaud Legrand, Alan Su, and Frédéric Vivien. Off-Line Scheduling of Divisible Requests on an Heterogeneous Collection of Databanks. In *Proceedings of the 14th Heterogeneous Computing Workshop*, Denver, Colorado, USA, apr 2005. IEEE Computer Society Press.

[16] Arnaud Legrand, Alan Su, and Frédéric Vivien. Minimizing the Stretch When Scheduling Flows of Divisible Requests. *Journal of Scheduling*, 2008.

[17] Joseph Y.-T. Leung, editor. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, 2004. ISBN: 1584883979.

[18] Lucas Mello Schnorr, Arnaud Legrand, and Jean-Marc Vincent. Multi-scale analysis of large distributed computing systems. In *Proceedings of the third international workshop on Large-scale system and application performance*, LSAP '11, pages 27–34, New York, NY, USA, 2011. ACM.

[19] Mark Silberstein, Artyom Sharov, Dan Geiger, and Assaf Schuster. Gridbot: execution of bags of tasks in multiple grids. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 11:1–11:12, New York, NY, USA, 2009. ACM.

[20] Simgrid. http://simgrid.gforge.inria.fr.

[21] W.T. Sullivan, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, and D. Anderson. A new major SETI project based on Project Serendip data and 100,000 personal computers. In C.B. Cosmovici, S. Bowyer, and D. Werthimer, editors, *Proceedings of the Fifth International Conference on Bioastronomy*, Bologna, Italy, 1997. Editrice Compositori.

Appendices

A. Vocabulary

Application A *user* account on the *server*. An *application* is associated with an algorithm that is used to process the data of the *campaigns* of the *user*.

Campaign A set of data submitted by a *user* that is to be processed within the *application*. A *campaign* is made-up of *tasks* and is considered finished when all it's *tasks* have been completed. This is also the moment at which the *project* will return the set of results to the *user*.

Client A computer that participates in a Volunteer Computing *project*. An individual can possess multiple computers that all participate in a *project* but they are all to be considered as seperate *clients*.

Deadline (campaign) The date at which a *campaign* should be finished. This parameter is not used in all scheduling strategies and it's mathematical definition may vary. As there are no service guarantees in a Volunteer Computing *project* a *deadline* is only an indication and failing to meet one is not considered as a failure of whole the *campaign*, this is called a soft-deadline.

Deadline (replica) The date at which the result of the processing of a *replica* is due to be sent to the *server* by the *client* running the *replica*. Once the *deadline* has been missed, the result of the *replica* is no longer considered relevant to the *server* and the client is allowed to discard the *replica*. When a *replica* passes it's *deadline* without a result being returned a new *replica* of the associated *task* is instantiated.

Project A Volunteer Computing *project* is made-up by a unique *server* and the *clients* that connect to it.

Replica A non-unique instance of a given *task*, a *replica* can only be sent once to a single *client*.

Response-time (campaign) Amount of time passed between the moment of submission of a *campaign* to the server and the moment at which the *campaign* is finished and the *results* are send back to the *application*.

Result Once the computation of a *replica* is finished by a *client* the obtained result is returned to the *server* responsable for the *campaign* and *application* to which it belonged. The *result* of a *replica* that is returned to the *server* is considered either valid or invalid.

Task A portion of data that has to be processed by *clients*. On creation of the *task* by the *server* a given number of *replicas* of the *task* are instantiated. A *task* is considered complete once a given number of valid results have been received for it's *replicas*.

Server A computer or host that is reponsable for sending work and receiving results. The *server* also manages the databases of information that should be processed by the *clients*. There is a unique *server* for each Volunteer Computing project.

Slack Ratio between the *deadline* given for a *task* and the average required processing time for the *task* on a *client*.

Waste Percentage of the total computational work done by clients and returning valid results that is not considered usefull by the server because of time-out or aggressive *task* replication.