Multi-scale analysis of large distributed computing systems

Lucas Mello Schnorr, Arnaud Legrand, Jean-Marc Vincent

Multi-scale Analysis of Large Distributed Computing Systems

Lucas Mello Schnorr
INRIA MESCAL, CNRS LIG
Grenoble, France
Lucas.Schnorr@imag.fr

Arnaud Legrand
INRIA MESCAL, CNRS LIG
Grenoble, France
Arnaud.Legrand@imag.fr

Jean-Marc Vincent
INRIA MESCAL, CNRS LIG
Grenoble, France
Jean-Marc.Vincent@imag.fr

ABSTRACT

Large scale distributed systems are composed of many thousands of computing units. Today's examples of such systems are grid, volunteer and cloud computing platforms. Generally, their analyses are done through monitoring tools that gather resource information like processor or network utilization, providing high-level statistics and basic resource usage traces. Such approaches are recognized as rather scalable but are unfortunately often insufficient to detect or fully understand unexpected behavior. In this paper, we investigate the use of more detailed tracing techniques –commonly used in parallel computing– in distributed systems. Finely analyzing the behavior of such systems comprising thousands of resources over several months may seem infeasible. Yet, we show that the resulting trace can be analyzed using tools that enable to easily zoom in and out on selected area of space and time. We use the BOINC volunteer computing system as a basis of this study. Since detailed activity traces of the BOINC clients are not available yet, we rely instead on traces obtained through a BOINC simulator developed with the SimGrid toolkit and which uses as input real availability trace files from the Seti@Home BOINC project. We show that the analysis of such detailed resource utilization traces provides several non-trivial insights about the whole system and enables the discovery of unexpected behavior.

Categories and Subject Descriptors

I.6.7 [**Simulation and Modeling**]: Simulation Support Systems; C.4 [**Performance of Systems**]: Performance attributes

General Terms

Experimentation, Measurement, Performance

Keywords

Cloud computing, Grid computing, Large-scale distributed systems, Performance visualization analysis, Resource usage anomalies, Volunteer computing, Triva, Simgrid, BOINC

LSAP'11, June 8, 2011, San Jose, California, USA.

1. INTRODUCTION

Today's large scale distributed systems, such as grid, volunteer and cloud computing platforms, are composed of many thousands of computing units. Such resources, interconnected by large-scale hierarchical networks, are very heterogeneous and are shared by applications and users. Generally, checking the good use of such platforms is done through monitoring tools like the Ganglia monitoring system [15], the Network Weather Service [23], Monalisa [17] and others [25]. Most of these collection systems gather resource information such as the processor or network utilization and provide high-level statistics or basic resource usage traces. Such approaches are recognized as rather scalable but are unfortunately often insufficient to detect or fully understand unexpected behavior.

On the other hand, the parallel computing community generally relies on more precise and fine grain data collection and analysis. Data collection is generally initiated from the application level: tools mostly focus on registering local and global states of the program, the amount of application-data transferred in messages, and counters for specific functions. Such "application-level" observation allows the detection of complex patterns like late communications, costly synchronization or convoy effects. Examples of such tools include TAU [21], Scalasca [7], VampirTrace [16], and the MPI standard profiling interface [8, 11]. Such tools can be used either for profiling, i.e., to get statistics about the application, or for tracing, i.e., to log time-stamped information during the execution without summarizing them. Such traces enables much more in-depth analysis with custom visualization tools like JumpShot [24] VaMPIr [16], and Paje [5]. Yet this approach suffers major scalability issues at both tracing and analysis level.

The tracing technique is generally much more intrusive than simple monitoring or profiling, since it registers detailed information in the form of events, and thus does not scale well. The rather involved analysis and visualization techniques used in parallel computing also face scalability issues because of the large amount of monitored entities and the detailed information registered on the traces. This last issue is often circumvented by using clustering techniques to spot outliers or identify general behavior [14] and then reuse the same classical visualization methodology with a smaller set of entities and a shorter timescale. The main drawback of such approaches is that they require to specify before the beginning of the analysis which kind of pattern or behavior to look for, decreasing the possibility of detection for unknown or unexpected patterns. Another interesting

approach is based on multi-level aggregation techniques. It enables to seamlessly zoom in and out at both space and time scale to let the analyst find interesting configurations by himself [18] in an exploratory manner.

The scalability issues faced in the parallel computing techniques seem to discourage the use of such detailed techniques to large-scale distributed systems. Yet, we establish in this article that they can be exploited in distributed systems by using multi-scale aggregation based techniques, effectively identifying and understanding non-trivial behavior of large-scale distributed systems such as BOINC. We also briefly discuss the feasibility of tracing such a system at this level of detail. To the best of our knowledge, it is the first time that such elaborated analisys and visualization techniques are applied to large-scale distributed systems.

The paper is organized as follows. Section 2 introduces the multi-scale aggregation techniques applied to analyze large-scale distributed systems. Section 3 presents our experimental framework and investigates two scenarios arising from the BOINC resource sharing mechanism: the first one is related to fairness, whereas the second one is related to global response time. The results and analysis are described in Section 4. We end with a conclusion and future work.

2. BEHAVIOR VISUALIZATION OF LARGE SYSTEMS

When observing a given resource over a very long time frame, the corresponding amount of information is very large and cannot easily be comprehended. Such traces often show very different behavior at small scale and at larger scale. Hence, their understanding requires the ability to easily zoom in and out on the trace. We illustrate this issue by looking at a typical availability trace obtained from the Failure Trace Archive (FTA) [13] and which represents whenever a given client is working for the SETI@home project over a 8 months period. Such a trace is thus a time-stamped sequence of zeros and ones and is represented in gray on Figure 1.(a). Depending on the anti-aliasing method used by the document viewer or on the resolution of the printer, it may appear either as a very fine-grained barcode or as a series of a few wide gray rectangles. In both cases, it is rather difficult to quantitatively analyze this information.

When zooming on a twelve-day period (Figure 1.(b)), we obviously see details that are hidden when considering eight months. But more interestingly, this time zoom in allows to realize that the full time frame view summarizes very poorly the twelve-day period. One could easily believe, from Figure 1.(a) that the client was available all the time during the twelve-day period even if it is only available 60% of the time. This is mainly because the full time frame representation is a *graphical* zoom out of the whole trace.

Figure 1.(b) and 1.(c) also depicts a black curve which represents the average availability over a one-day period. Unlike the gray barcode, such representation is much more faithful to the trace and gives a better feeling of the behavior of the client, even over an eight-month period (see Figure 1.(c)). This simple example illustrates that one should not rely on graphical zoom (i.e., zoom out on graphical representations of traces) but rather use aggregation techniques directly on the traces (i.e., use graphical representations of summarized traces). Indeed, graphical zooming techniques often result in extremely misleading interpretation and cannot correctly

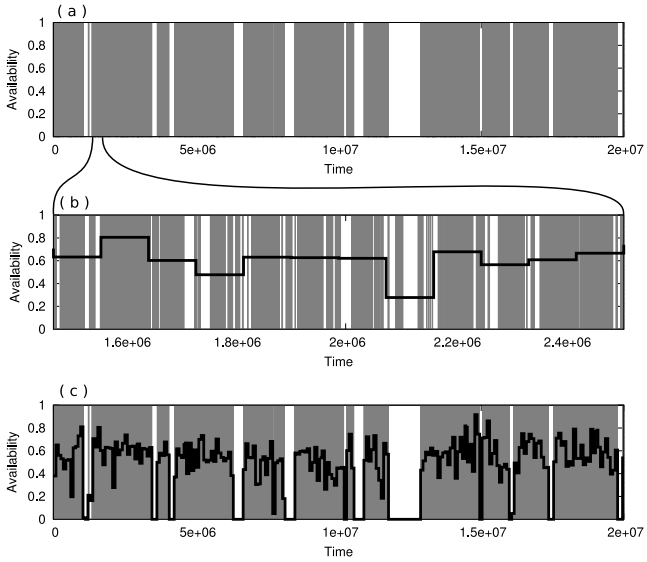

Figure 1: The gray areas represent resource availability, the black line represent a one-day integration: (a) eight months of raw traces; (b) twelve-day zoom with raw and integrated traces and (c) same as the first plot, plus one-day integration.

account for the multi-scale complexity of such traces. This same issue arises at any scale since even the second one-day period of Figure 1.(b) may appear either as fully available or half available depending on the printer resolution, whereas the one-day average availability is around 80%.

In a distributed system with a very large number of resources, this issue is even more problematic. We have to deal with a huge amount of information in both time and space. Faithfully representing on a single screen the behavior of thousands of resources at various scales of time is extremely challenging but is yet necessary to analyze such systems.

We briefly detail how data aggregation is formally defined in our context. Let us denote by $\mathcal{R}$ the set of resources and by $\mathcal{T}$ the observation period. Assume we have measured a given quantity ρ on each resource:

$$\rho : \begin{cases} \mathcal{R} \times \mathcal{T} & \to \mathbb{R} \\ (r,t) & \mapsto \rho(r,t) \end{cases}$$

In our context, $\rho(r,t)$ may represent the CPU availability of resource r at time t; or the (instantaneous) amount of CPU power allocated to a given project on resource r at time t. In most situations, we have to depict several such functions at once to investigate their correlation.

As we have have just illustrated in Figure 1, ρ is generally complex and difficult to represent. Studying it through multiple evaluations of $\rho(r,t)$ for many values of r and t is very tedious and one often miss important features of ρ doing so. Assume we have a way to define a neighborhood $N_{\Gamma,\Delta}(r,t)$ of (r,t), where Γ represents the size of the spatial neighborhood and Δ represents the size of the temporal neighborhood. In practice, we could for example choose $N_{\Gamma,\Delta}(r,t) = [r-\Gamma/2, r+\Gamma/2] \times [t-\Delta/2, t+\Delta/2]$, assuming our resources have been ordered. Then, we can define an

approximation $F_{\Gamma,\Delta}$ of ρ at the scale Γ and Δ as:

$$F_{\Gamma,\Delta} : \begin{cases} \mathcal{R} \times \mathcal{T} & \to \mathbb{R} \\ (r,t) & \mapsto \iint_{N_{\Gamma,\Delta}(r,t)} \rho(r',t').dr'.dt' \end{cases} \quad (1)$$

Intuitively, this function averages the behavior of ρ over a given neighborhood of size Γ and Δ. For example a crude view of the system is given by considering the whole system as the spatial neighborhood and the whole timeline as the temporal neighborhood. Since Δ can be continuously adjusted, we can temporally zoom in and consider the behavior of the system at any time scale. Once this new time scale has been decided, we can observe the whole timeline by shifting time and considering different time intervals.

3. EXPERIMENTAL FRAMEWORK

We propose the use of different case studies in the scheduling of bag-of-tasks in BOINC volunteer computing platforms to illustrate the benefits of using detailed resource utilization tracing in large-scale platforms. Since most of the traces obtained in real volunteer computing platforms are already reduced with different techniques, we decided to use simulation of such systems to gather raw monitoring traces. During the simulations, we register the contribution given to the projects in computing power from each volunteer client. The result of each simulation is a trace file that contains the computing power given by any client to each project. This section details the distributed application scenario, how the traces are collected by simulating BOINC with SimGrid, and how they are analyzed using different data reduction techniques with Triva.

3.1 Distributed Application Scenario

The scenario used in this paper is the scheduling of bag-of-tasks in volunteer computing (VC) [2] distributed platforms. We use the BOINC (Berkeley Open Infrastructure for Network Computing) [1] architecture as an example of a volunteer computing platform. BOINC is the most popular VC infrastructure today with over 580,000 hosts that deliver over 2,300 TeraFLOP per day. Such VC architectures are composed by volunteer clients that choose to which projects their unused CPU cycles will be given. Each project (such as SETI@home, Climateprediction.net, Einstein@home, or World Community Grid) is hosted on a BOINC server that provides the volunteer clients with work units. Once a host has fetched at least one work unit, it disconnects from the server and computes work unit results. Several mechanisms and policies determine when the host may do these computations, accounting for volunteer-defined rules (e.g., caps on CPU usage), for the volunteer's activity (e.g., no computation during keyboard activity), and for inopportune shutdowns. We explore thus the following scenarios:

- **Fair sharing** Volunteers define preferences and project shares. They expect these preferences to be respected whatever the project work units characteristics. In this first scenario, we propose to check that the local scheduling algorithm keeps all volunteer machines busy and fairly share these resource between the different projects.

- **Response time** As such, the BOINC architecture is not perfectly suited to VC projects that consist of small numbers of short work units. Such projects typically have burst of tasks arriving and are not able to keep all volunteers busy all the time. However, they need the volunteers to crunch their work units as soon as they are available and to return the results as soon as possible to minimize the response time of the whole bag of tasks.

A third scenario involving data-intensive projects was also investigated (see [20] for more details), but it has been omitted for space reasons.

3.2 SimGrid Simulation

Real large-scale platforms are hard and time consuming to study. Although traces of real volunteer computing platforms can be found, most of the information stored is already reduced in time, and sometimes also in space. Yet, we need the raw traces to evaluate the data reduction techniques previously described. This reason lead us to use simulation to generate traces with the right level of details. We used a BOINC Simulator [6] that executes the most important features of the real scheduler (deadline scheduling, long term debt, fair sharing, exponential back-off) and has already been validated [6] against the official BOINC client simulator developed by the BOINC team.

The simulator used in our experiments is developed using the SimGrid framework [4]. With this framework, the simulations can consider real availability traces as defined by the Failure Trace Archive (FTA) [13] and can be configured to generate resource utilization traces very easily. The tracing can be customized to register the resource utilization in a per-project fashion, instead of simply registering the utilization of a resource.

For sake of clarity during the analysis of the data reduction techniques, we only use two projects with different task creation policies. Every client is configured to evenly share its resources between the two projects. Depending on the case study, one of the projects might create tasks from time to time (a *burst* project), while the other keeps a steady flow of task creation to be computed by the volunteer clients (a *continuous* project). This two-projects setting for running simulations, mixed with the FTA availability traces, allows the analysis of the global and local fairness behavior considering situations where the failure of multiple clients might happen at the same time. Every simulation run generates a single trace file that contains detailed traces of categorized resource utilization, and which are then analyzed with Triva, described in the next subsection.

3.3 Data Reduction and Visualization

Triva [19] is a visualization tool focused on the analysis of parallel and distributed application traces. It implements different visualization techniques and also serves as a sandbox for the creation of new techniques to do a visual analysis of data. The tool is equipped with algorithms to do temporal and spatial aggregation, allowing the analysis in configurable time frames and level of details. We used Triva in this work to analyze the traces obtained with the simulation execution described in the previous section.

The temporal aggregation feature is based on integration of the variables within a time frame configured by the user. It is the user responsibility to define a significant time frame for the analysis, either for a full trace observation or on a smaller time interval. The tool is also capable to move dy-

namically the configured time frame, so the user can observe the evolution of the variables along time. The spatial aggregation works, on the other hand, by using simple operators to group detailed information from hosts and links in higher level representations, like a cluster for instance.

As of today, Triva implements the Squarified Treemap [3] visualization technique and a configurable topology-based visualization. On the treemap view, the user is capable to select the categories used in the representation. Several categories might be used at the same time, allowing a visual comparison of their resource utilization. The topology-based visualization technique implemented in Triva uses customizations defined by the user to set which trace components are taken as nodes and edges of the topology. Since the instrumented version of SimGrid registers all the hosts participating on the simulation, and also the links that interconnect them, we can used such information to create the topological interconnection of the resources for the visualization within Triva. The mapping of variables from the categorized resource utilization can also be applied on the graph so the user can compare their values taking into account the topology itself and the dynamic evolution over time.

The traces from the simulations were used as input for Triva to generate different representations that helped us to analyze the behavior behind the three different case studies. For each of them, we present the expected and observed behavior, and a discussion about how a bad spatial or temporal data reduction technique would have hidden the observed behavior and mislead the analysis.

4. RESULTS AND ANALYSIS

This section presents two case studies for the BOINC scheduling scenarios: a fairness analysis and projects interested in response time. The case studies are separated in three parts: setting, for detailing how the experiment was configured; expected, listing what we hope to get from the experiment; and observed, showing the results we obtained through the visualization analysis.

4.1 Fair Sharing

Setting. For this case study, we create two projects that generate continuous tasks, and two categories for the tracing: *Continuous-0* and *Continuous-1*. The only difference between the two projects is the size of the jobs and their corresponding deadline. The tasks from the *Continuous-0* project are 30 times longer than tasks from the *Continuous-1* project. The deadlines are set to 300 hours and 15 hours for project *Continuous-0* and *Continuous-1*. There were a total of 65 clients[1] whose power and availability traces are taken from the Failure Trace Archive [13]. Last, every client was configured to evenly share its resource between the two projects. We configured the BOINC simulator to run the projects for the period of ten weeks, tracing the resource utilization by category over the different clients.

Expected. As we previously explained, volunteers want to keep control on the resource they provide. In particular, they expect that the BOINC client respect the resource shares they define. In our setting, this means that the amount of CPU cycle given to each project should be roughly the same, regardless of the differences between the two projects (task size and deadline). The global and local share for each project should thus remain around 50%.

Observed. The Figure 2 shows the visualization analysis for the trace file obtained with the simulation. On the top of the figure, the treemap with aggregated data from all the clients shows the global resource utilization division between the two categories. It considers the whole simulation time for all clients. We can observe that clients are reasonable fair, executing tasks from the project *Continuous-0* in 52.30% of the simulated time.

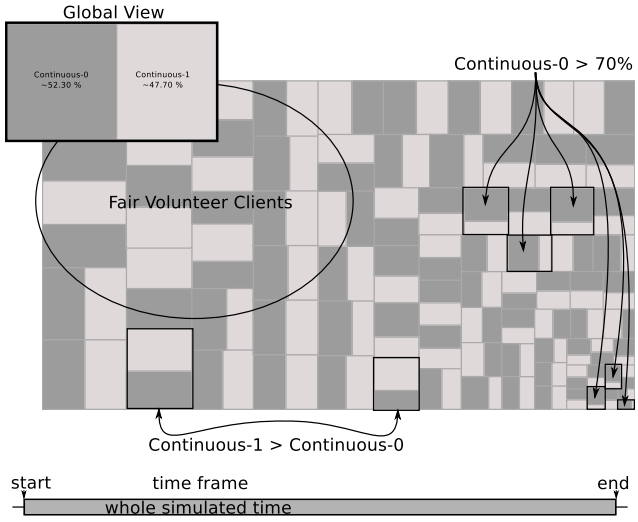

Figure 2: Fairness anomaly detected in some BOINC clients. Many clients with a small area (i.e., a small power or a low availability) favor application *Continuous-0*.

The bigger treemap on Figure 2 was calculated in the same way, but detailing the share for every client. In this level of detail, we can notice that some clients have an unexpected behavior, working severely more for one project than for another. On the right part of the bigger treemap, we can observe that some of the clients worked more than 70% of time for the *Continuous-0* project, while some other clients worked more for the *Continuous-1* project. This behavior is unexpected since the volunteer clients should be fair no matter the size of the tasks defined by the different projects and the deadline for completion. By analyzing the treemap view, we can notice that smaller clients (occupying a smaller area of the drawing), which reflects less contribution to the work, present such strange behavior. On the other hand, bigger clients, which contributed more to the work, are mostly fair. The size difference between the clients is related to both the power of the hosts and their availability. Since the power heterogeneity is not that large, it can easily be deduced that the smaller clients have very long unavailability periods. Therefore unfairness seemed to be related to unavailability.

This observation was done during the early stages of the development of the BOINC simulator so we informed the developers about this unexpected phenomenon. The origin of the problem, which was identified later on, was related to the algorithm that counts the time worked for each project on the clients. After the problem resolution, we ex-

[1]We illustrate our observations with only 65 clients for sake of readability but we performed similar experiments with thousands of hosts

ecuted again the simulation, with exactly the same parameters and obtained a trace file which was again analyzed with Triva. Figure 3 shows the analysis of this trace file, with the global view of fairness among all clients depicted on its top left corner. Now, the division between the two projects reaches 50.20% for the *Continuous-0* tasks, and 49.80% for the *Continuous-1* tasks, considering the simulation time of ten weeks. The bigger treemap on the figure shows the division by project for each client, allowing the observation that even clients that contribute less are also fair.

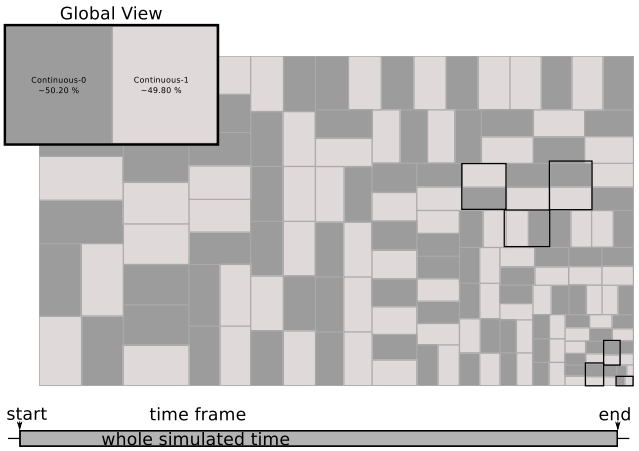

Figure 3: Fairness visualization after the correction on the implementation of the scheduling algorithm

In this first example, the anomaly came from a bug in the simulator in its earliest development stages. This bug would probably never have been identified without the visualization-based approach. Indeed, even after having selected unfair clients, correlating unfairness and unavailability would have been hard to do with standard statistical techniques whereas it appeared clearly on the treemap. The identification of this phenomenon enabled to narrow where the problem could come from and thus to correct it very quickly. Even though this bug was found in a simulation, the same kind of issue could have happened in a real large scale distributed system. Last, it was hardly noticeable at large scale and was made possible by tracking the activity of every resource according to the two projects.

Triva's treemap visualization, with the global and detailed view, gives the possibility to developers to spot outliers rapidly.

4.2 Response Time

Setting. As previously discussed on Section 3, BOINC targets projects with CPU-bound tasks interested in throughput computing. One of the main mechanism that give project servers some control over task distribution is the completion deadline specification. This parameter allows projects to specify how much time will be given to volunteer clients to return the completed task. This is used as a soft deadline specification upon task submission, but above all as a way of tracking task execution. On the client side, this deadline is employed to decide which project to work for. Thus, a client never starts working on an overdue task but always finish a started task. When a deadline is likely to be missed, the client switches to *earliest deadline first* mode. By setting tight deadlines, a project may thus temporary gain priority over other projects (this phenomenon is balanced by other long-term sharing mechanisms).

Some research has been done to design mechanisms enabling to improve the average response time of batch of tasks in such unreliable environments [12]. Resource selection (discard unreliable hosts) and prioritization (send tasks to fast hosts in priority) and replication toward the end of the batch seem to be the key elements of such a strategy. GridBot [22] already implement many of these features.

For this case study, we configured the BOINC simulator to execute two projects: *Continuous* and *Burst*. The *Continuous* project continuously generates tasks 30 times larger than the other project, and with a loose completion deadline given to volunteer clients of 300 hours. The *Burst* project creates smaller tasks every 10 days with a tighter completion deadline of six hours and allows up to 5 replicas of the same task. The client configuration was the same as in Section 4.1. Our analysis is based upon a simulation that details the behavior of BOINC during ten weeks.

Expected. The BOINC architecture is designed using a pull style architecture. This means that the volunteer clients only contact from time to time the project servers to which they wish to donate their CPU cycles. Hence, when a given project generates a burst of tasks, it has to wait for clients to contact him before being able to start the task distribution. There may thus be a rather long time before all volunteer clients realize that the server has a bunch of new tasks to be executed. Yet, once a volunteer client starts working for a burst, it is expected to try to work for it as much as possible. Indeed, the client scheduling algorithm tries to comply to a long-term sharing policy. Hence projects that have not sent tasks since a long time should get a temporary higher priority than projects whose tasks are available all the time.

This mechanism lead us to expect a slow-start execution of tasks from the project that generates bursts of tasks, from time to time. Such behavior was already anticipated and optimized in other works [9]. What we would like to observe here is the shape of this slow-start.

Observed. To observe the slow-start effect, we decided to observe the system with a treemap view configured to show only aggregated states from all the simulated machines. Figure 4 shows a series of treemaps that classify the aggregated work executed by all the volunteer clients for the *Continuous* (light gray) or the *Burst* (darker gray) projects. The generated treemaps were aligned horizontally according to the beginning of the time interval considered for their rendering. Since the average completion of a burst of task was around 26 hours, we decided to start the analysis with time intervals of two hours, represented by the treemaps on the top. In the middle, the intervals considered are of one hour and, on the bottom, time intervals of half an hour. The figure also depicts the beginning and the end of the burst period, denoted by the rectangle that encapsulates all the screenshots taken inside the period.

The expected slow-start behavior of volunteer clients during the simulation is visible on Figure 4. It takes from 2 to 3 hours for a client to realize the activity of the *Burst* project server and start the execution of its tasks. According to the view of two hours, it takes 18 hours to the burst tasks consume more than half of the power of the platform, despite its tighter deadlines that indicate that it should be given a higher priority. Considering that the activity pe-

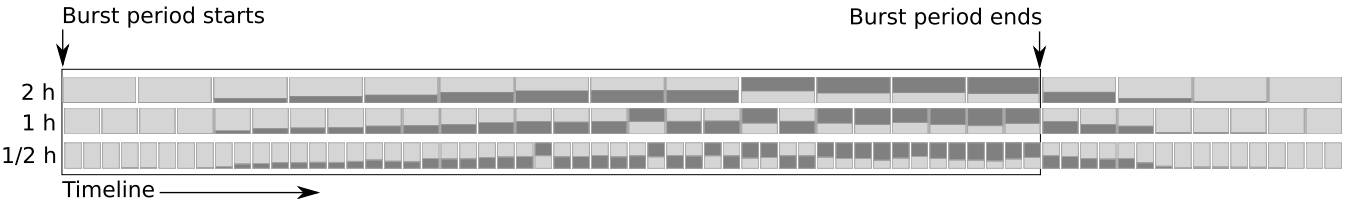

Figure 4: Observing the slow-start behavior of volunteer clients giving resources to the burst tasks (shown as dark gray) when the burst server becomes active. Using a 2-hour time frame, we analyze that about 18 hours are required for the burst project to get more than half of the platform computing power and that it hardly gets more. This last observation is surprising as bursts are infrequent and should thus have a temporary high priority compared to the continuous project (clients try to comply to a long-term fair sharing policy). Using a smaller time frame, we observe oscillations: the dominant project is alternatively the burst or the continuous project. Furthermore, active burst tasks remain in the system after the termination of the burst. This waste can be explained by "loose" deadlines and replication.

riod of the burst project is about 26 hours, this slow-start occupies about 70% of that time. Using the 1/2 hour view which shows small variations, the time taken could even be estimated to 77% of the time.

Besides the expected behavior caused by the connection mechanism of BOINC clients, we can notice also two different anomalies on the analysis. The first one is the execution of *Burst* tasks after the end of the burst. The second one is the apparent difficulty of the *Burst* project to overwhelm (at least for a short time period) the *Continuous* project during the burst period, even with a larger priority.

- **Wasted Computations.** Figure 4 shows the first anomaly observed in the analysis. We can notice the end of the activity period of the *Burst* project. This information, registered on the trace file, indicates that the server received at least one answer for all the tasks it submitted to the volunteers. Remember that tasks may be submitted many times at the end. This replication enables to deal with straggling tasks. Such behavior can thus be considered as normal since clients are not always connected to the server. But this illustrates that an aggressive replication algorithm, mixed with a bad deadline configuration on the server side, leaves some tasks in the system. Such tasks waste computing resources and may make volunteer unhappy since they may not be rewarded credit for their work. Furthermore, even if the project is not going to use the results, clients count the work donated to each project. This means that they will be less likely to work for this project later on.

- **Surprisingly Low Priority of the *Burst* Project.** The second anomaly is related to the execution of *Continuous* tasks during the burst period. We noticed this unexpected behavior by visualizing, in Figure 4, the aggregated amount of work executed by the clients for each of the projects. We can observe, in the timeframe of half hour, that the project which receives more computational power fluctuates between the two projects: sometimes the darker gray tonality occupies more space than the other; sometimes no.

To understand why the *Burst* Project struggles to get computing resources compared to the *Continuous* project, we decided to look for more details to each volunteer share evolution. At this level of detail, treemaps are not be the best solution because the location of volunteers in the representation may change along time depending on the aggregate computing it delivered (in this kind of representation, the area of the screen is occupied following a space-filling algorithm that considers the nodes value; if the values change, the nodes position on the screen might change).

Therefore, we used a graph-based view, where each machine is positioned on the screen and its visual parameters (size, filling, etc) are associated to a trace variable. The leftmost image of Figure 5 illustrates such representation. It details the behavior of the *Continuous* and *Burst* server projects and some volunteer clients. For the two servers, represented by the squares located in the middle of the screenshot, the black color indicates that for the time frame in question, the server is active. If it is white, the server is not generating tasks, and it received all the results from previously submitted tasks. The other squares represent the volunteer clients, and their gray tonalities indicate for which project they are working for the time frame in question, and how much. If the square is full of light gray, for instance, it means that the client worked only for the *Continuous* project at full computational power. If a square has two gray tonalities, it means that the client worked on that time frame for both projects. The space occupied for each tonality indicates on this case the amount of power given for each project. Still on the same image of Figure 5, the size of the volunteer clients square is directly related to its computational power on the time-frame in question. Hence, the client representation is not drawn if it is inactive on the period.

The screenshots, from left to right, show the evolution of volunteer clients behavior using time intervals of one hour. On the leftmost image, rendered with the time frame 11 to 12 hours after the burst started, shows two volunteer clients (inside the two circles) fully executing *Burst* tasks. The subsequent images show that these clients starts to work for the continuous project and then come back to execute *Burst* tasks. Such situation can happen if the deadline given for one of the *Continuous* tasks fall exactly during the burst period. In that case, the scheduling algorithm implemented on the client-side decides to work for the continuous project. Such situations should be rather rare though and happen at most once on each volunteer. However, such unexpected and strange behavior appears re-

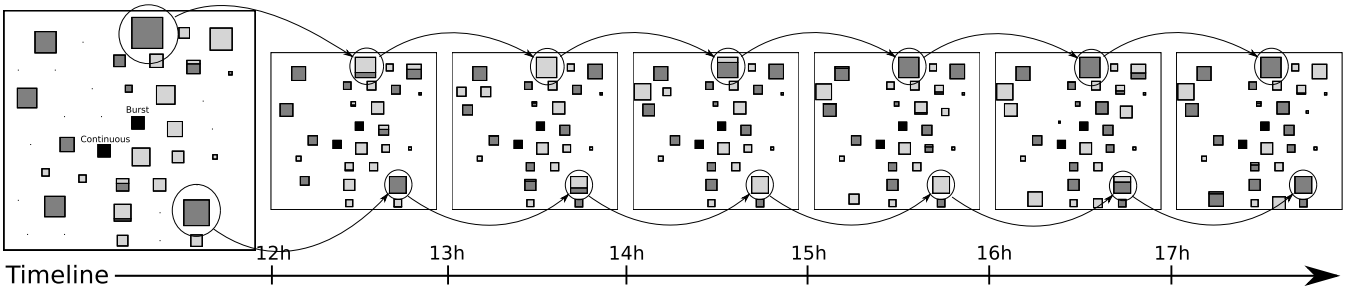

Figure 5: Observation of an anomaly consisting in the cyclic execution of continuous (light gray) and burst (dark gray) tasks on volunteer clients, about 11 hours after the beginning of the burst period. The arrows between the circled hosts help following their evolution. Executing continuous (low priority) tasks while burst (high priority) tasks are available results in a poor overall resource usage. This anomaly was originally discovered using animations (available at `http://triva.gforge.inria.fr/2011-lsap.html`) that reveal "blinking" hosts (periodic switch between dark and light gray) that caught our eyes.

peatedly on all volunteer clients before and after the time frames shown on this figure. We observed a cyclic behavior on some volunteer clients, where clients work for the *Burst* project, then for the *Continuous* project, and back to the *Burst* project, and so on. This phenomenon was particularly striking in the animated version (available at `http://triva.gforge.inria.fr/2011-lsap.html`) since it resulted in a blinking of the volunteers between light and dark gray.

We informed the BOINC simulator developers of such phenomenon. Further investigation revealed that this anomaly comes from the *short time* fairness requirements of the BOINC scheduling algorithm and shows how inadequate it might be in this context. Like in the first scenario, this issue would probably never have been identified without both the spatial and time aggregation and zooming capabilities of Triva and its ability to seamlessly move from one representation to another.

4.3 Trace Size Limitations

The scalability of detailed tracing in large-scale distributed systems is probably one of the main reasons against its adoption. One may even consider that such traces are too large for conducting an analysis like those of the previous. This is not the case. Assuming resource utilization of a single volunteer client is recorded every minute, we would have 1440 trace events per day for that client. If we consider a binary resource allocation: all or none computing power to execute the tasks, we need only 180 bytes to record that information per day. If the large-scale computing system is composed of one million volunteer clients that are continuously donating computing power, we need only 180 Megabytes of data for a full year.

This detailed data can be even more compressed if necessary, since long periods of continuous donation or not can be grouped. Similar orders of magnitude are reported in real large scale observations. For example, the availability traces of SETI@home from the FTA [13], which comprise 102,416,434 clients [10] with very different lifetimes over 20 months, are available in a single 2.5GB compressed file. Such amount of data can thus easily be handled on a standard workstation such as the one we used to conduct our analysis and create all graphics and animations referenced in this article.

5. CONCLUSION

Large-scale distributed systems usually rely on monitoring tools that gather high-level statistics about the resource utilization. The main reason for using such techniques is related to scalability, as a system gets larger, the amount of trace data becomes an important issue. In this paper, we have investigated what kind of analysis would be possible if detailed traces were available about the resource utilization of large-scale platforms. The analysis we present relies on data aggregation in space and time scales, coupled with elaborated visualization techniques. This combination of techniques, from tracing to analysis, provides interesting insights about the behavior of the whole distributed system.

The results we present are based on two scenarios related to scheduling of bag-of-tasks in the BOINC volunteer computing platform. The analysis of the first scenario enabled us to detect a problem in the fairness of the scheduling algorithm of the simulated BOINC clients. Such problem appeared mainly on clients with low availability. The treemap visualization of Triva, combined with temporal integration on the whole simulated time, allowed the problem to be immediately spotted.

The second scenario analysis, related to the use of replication algorithms and tight deadlines to improve the response time of some projects, allowed to observe three different phenomenons. The first one, which was expected, was related to the slow-start of such projects. The second one was that the abuse of fault-tolerant features may leave many tasks in the system after the completion of batches, increasing resource waste. The third phenomenon was much surprising and was related to the difficulty of such projects to get more computing resources than the more classical ones despite these deadline and replication mechanisms which should have favored them. Thanks to the ability of Triva to represent the state of the system at different spatial and temporal scales, we were able to easily spot this phenomenon and then to identify its origin as the short-term fairness feature of the client scheduling algorithm.

The analysis of these two scenarios, and the different anomalies observed, is only possible thanks to detailed traces of resource utilization. Without access to raw traces, most of the behavior and possibly the problems related to the platform would probably never have been noticed. Although most large-scale distributed systems avoid the use of such detailed

tracing, we have shown how beneficial they are to the understanding of their behavior. As future work, we plan to apply such techniques to more classical HPC and grid scenarios so as to keep on improving the spatial aggregation capabilities of Triva. Indeed, in the case studies we presented in this article, we used spatial aggregation techniques mainly for the whole system at a time and not much for a smaller set of resources. When a hierarchy can be defined on resources (e.g., in a grid or in a cloud), this technique is likely to reveal very precious.

6. ACKNOWLEDGMENT

This work is partially supported by ANR (*Agence Nationale de la Recherche*), project reference ANR 08 SEGI 022 (USS SimGrid). We thank Derrick Kondo for the insightful discussions and who provided us with many references and information on BOINC. We also thank Bruno Donassolo for providing the implementation of the BOINC simulator for our experiments.

7. REFERENCES

[1] D. P. Anderson. Boinc: A system for public-resource computing and storage. In *The 5th IEEE/ACM International Workshop on Grid Computing (Grid)*, pages 4–10. IEEE Computer Society, 2004.

[2] D. P. Anderson and G. Fedak. The computational and storage potential of volunteer computing. In *The Sixth IEEE Int. Symp. on Cluster Computing and the Grid (CCGrid)*, pages 73–80. IEEE Computer Society, 2006.

[3] M. Bruls, K. Huizing, and J. van Wijk. Squarified treemaps. In *Proceedings of Joint Eurographics and IEEE TCVG Symposium on Visualization*, pages 33–42. IEEE Press, 2000.

[4] H. Casanova, A. Legrand, and M. Quinson. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *10th IEEE International Conference on Computer Modeling and Simulation*, 2008.

[5] J. C. de Kergommeaux and B. de Oliveira Stein. Pajé: An extensible environment for visualizing multi-threaded programs executions. In *The 6th International Euro-Par Conference on Parallel Processing*, pages 133–140, 2000.

[6] 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.

[7] M. Geimer, F. Wolf, B. J. N. Wylie, E. Abraham, D. Becker, and B. Mohr. The scalasca performance toolset architecture. *Concurrency and Computation: Practice and Experience*, 22(6):702–719, 2010.

[8] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: portable parallel programming with the message-passing interface*. MIT Press, Cambridge, MA, USA, 1994.

[9] E. Heien, D. Anderson, and K. Hagihara. Computing low latency batches with unreliable workers in volunteer computing environments. *Journal of Grid Computing*, 7(4):501–518, 2009.

[10] B. Javadi, D. Kondo, J.-M. Vincent, and D. Anderson. Discovering Statistical Models of Availability in Large Distributed Systems: An Empirical Study of SETI@home. *IEEE Transactions on Parallel and Distributed Systems*, 2010.

[11] R. Keller, G. Bosilca, G. Fagg, M. Resch, and J. Dongarra. Implementation and Usage of the PERUSE-Interface in Open MPI. In *The 13th European PVM/MPI Users' Group Meeting*, 2006.

[12] D. Kondo, A. Chien, and H. Casanova. Resource management for rapid application turnaround on enterprise desktop grids. In *The ACM/IEEE conference on Supercomputing*, page 17, 2004.

[13] D. Kondo, B. Javadi, A. Iosup, and D. Epema. The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. In *The 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2010.

[14] C. W. Lee, C. Mendes, and L. Kale. Towards scalable performance analysis and visualization through data reduction. In *IEEE International Symposium on Parallel and Distributed Processing*, pages 1 –8, 2008.

[15] M. L. Massie, B. N. Chun, and D. E. Culler. The ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[16] M. S. Muller, A. Knupfer, M. Jurenz, M. Lieber, H. Brunst, H. Mix, and W. E. Nagel. Developing scalable applications with vampir, vampirserver and vampirtrace. *Parallel Computing: Architectures, Algorithms and Applications*, 38:637–644, 2007.

[17] H. Newman, I. Legrand, P. Galvez, R. Voicu, and C. Cirstoiu. Monalisa : A distributed monitoring service architecture. *CoRR*, cs.DC/0306096, 2003.

[18] L. M. Schnorr, G. Huard, and P. O. A. Navaux. Towards visualization scalability through time intervals and hierarchical organization of monitoring data. In *The 9th IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, 2009.

[19] L. M. Schnorr, G. Huard, and P. O. A. Navaux. Triva: Interactive 3d visualization for performance analysis of parallel applications. *Future Generation Computer Systems*, 26(3):348 – 358, 2010.

[20] L. M. Schnorr, A. Legrand, and J.-M. Vincent. Visualization and Detection of Resource Usage Anomalies in Large Scale Distributed Systems. Research Report RR-7438, INRIA, 10 2010.

[21] S. Shende and A. Malony. The TAU parallel performance system. *International Journal of High Performance Computing Applications*, 20(2):287, 2006.

[22] 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*, pages 1–12. ACM, 2009.

[23] R. Wolski, N. T. Spring, and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5–6):757–768, 1999.

[24] O. Zaki, E. Lusk, W. Gropp, and D. Swider. Toward scalable performance visualization with jumpshot. *International Journal of High Performance Computing Applications*, 13(3):277–288, 1999.

[25] S. Zanikolas and R. Sakellariou. A taxonomy of grid monitoring systems. *Future Generation Computing Systems*, 21(1):163–188, 2005.