CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
*Concurrency Computat.: Pract. Exper.* 2011; **00**:1–26
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1885

# Detection and Analysis of Resource Usage Anomalies in Large Distributed Systems Through Multi-scale Visualization

Lucas Mello Schnorr*, Arnaud Legrand, Jean-Marc Vincent

*INRIA MESCAL research team, CNRS LIG Laboratory, Grenoble, France*

SUMMARY

Understanding the behavior of large scale distributed systems is generally extremely difficult as it requires to observe a very large number of components over very large periods of time.
Most analysis tools for distributed systems gather basic information such as individual processor or network utilization. Although scalable because of the data reduction techniques applied before the analysis, these tools are often insufficient to detect or fully understand anomalies in the dynamic behavior of resource utilization and their influence on the applications performance.
In this paper, we propose a methodology for detecting resource usage anomalies in large scale distributed systems. The methodology relies on four functionalities: characterized trace collection, multi-scale data aggregation, specifically tailored user interaction techniques, and visualization techniques. We show the efficiency of this approach through the analysis of simulations of the volunteer computing BOINC architecture. Three scenarios are analyzed in this paper: analysis of the resource sharing mechanism, resource usage considering response time instead of throughput, and the evaluation of input file size on BOINC architecture. The results show that our methodology enables to easily identify resource usage anomalies, such as unfair resource sharing, contention, moving network bottlenecks, and harmful short-term resource sharing. Copyright © 2011 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: performance visualization analysis; large-scale distributed systems; volunteer computing; grid computing; cloud computing; resource usage anomalies

## 1. INTRODUCTION

Today's large scale distributed computing systems such as grids, clouds, and volunteer computing systems typically comprise thousands to millions of computing units collaborating over complex large-scale hierarchical networks. Such resources are typically very heterogeneous, volatile, and shared by applications and users, making the understanding of the behavior of these systems extremely challenging. The analysis of such systems has to address many observed entities over very large periods of time.

Generally, checking the good use of such platforms is done through monitoring tools like the Ganglia monitoring system [29], the Network Weather Service [46], Monalisa [34] and others [48]. Most of these collection systems gather resource information such as the processor or network utilization and provide overall statistics created from basic resource usage traces. Such approaches are recognized as rather scalable since the information from the large amount of observed resources is reduced at source to be analyzed. Although these statistics are useful to get an overview about the distributed system, they are often insufficient to detect or fully understand anomalies in the dynamic behavior of resource utilization and its influence on the applications performance.

*Correspondence to: Lucas.Schnorr@imag.fr. 51, avenue Jean Kuntzmann, 38330 Montbonnot Saint Martin, France

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" observations allow the detection of complex patterns like late communications, costly synchronization or convoy effects. Examples of tools relying on this approach include TAU [42], Scalasca [12], VampirTrace [32], and the MPI standard profiling interface [14, 20]. 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 [47] VaMPIr [32], and Paje [8]. Yet this approach suffers major scalability issues at both tracing and analysis level. The scalability and intrusiveness issues faced in the parallel computing community seem to discourage the use of such detailed techniques to large-scale distributed systems. However, We believe that abnormal resource usage in large scale distributed systems can only be detected if the information collected about the resources is detailed enough in time and space.

In this paper, we propose a methodological approach for detecting resource usage anomalies in distributed systems. It relies on four functionalities: (a) characterized trace collection, where the resource usage is classified based on application-level categories, such as tasks of a given type, processes of a certain nature, and so on; (b) multi-scale data aggregation, used to control the amount of data to be analyzed at the same time; (c) a set of specifically tailored user interaction techniques, to let the analyst navigate through the different levels of detail in the space/time dimensions; and (d) visualization techniques, effective in identifying non-trivial and unexpected behavior of large applications in distributed systems.

We validate our anomaly detection approach through the analysis of a simulated BOINC architecture [2, 9] using the SimGrid framework [7] with various scenarios and workload. The results show that our approach allows an easy identification of various types of anomalies such as unfair resource sharing, contention, moving network bottlenecks, and harmful short-term resource sharing.

The paper is structured as follows. Next section presents the related work, classified in monitoring and tracing tools, visualization techniques, and analysis methodologies. Section 3 presents our methodology for the anomaly detection in distributed systems. Section 4 presents the framework we used to validate the approach. Section 5 presents three different scenarios. The first one investigates the fairness of the BOINC client scheduling algorithm when subscribed to multiple BOINC projects and reveals a *fairness sharing anomaly*. Our second case study examines the resource usage of projects having bursts of batches of tasks and that are thus rather more interested in optimizing response-time than throughput. This case study illustrates a *slow-start* effect and a surprising *resource usage oscillation* revealed by a *blinking* during the visualization. Last, our third scenario investigates the effect of large input files on the overall platform usage. This last case study illustrates *moving bottlenecks*. Different visualization techniques are used to analyze the characterized resource utilization traces and show the effectiveness of our approach. We finish the paper with a conclusion and perspectives, in Section 6.

## 2. RELATED WORK

We classify related research in three sections: monitoring and tracing systems, analysis methodologies, and visualization techniques. The first section details tools that collect resource usage data and if they characterize that information. The second section presents the data analysis methodologies employed to analyze distributed and parallel applications. Finally, the third section presents visualization techniques are used to analyze application traces together with resource usage. The section ends with a discussion about how our approach differs in these three areas from other existing solutions.



DOI: 10.1002/cpe.1885

### 2.1. Monitoring and Tracing Systems

Most tools for monitoring distributed systems are focused on observing only the resource usage. Examples of these tools include Ganglia [29], the Network Weather Service [46], MRNet [37], Monalisa [34], and many others [48]. They use different distributed collection configurations to gather periodic resource usage from a possibly large number of resources. These tools consider in their implementation several constraints related to sensor distribution, intrusiveness and historical data retention policies. Ganglia, for example, registers the computer load and other metrics in different time frames (minute, hour, week). Monalisa, for instance, registers also network traffic, flows, connectivity and the topology configuration. The information collected by these tools is generally composed by how much a given resource is used during a time period. The applicative categorization of resource usage is generally not available on these tools.

Application-centric tracing systems register much more information about the application behavior during the execution. These tools can be event- or profiling-based, registering the behavior in time or immediately aggregating the time spent in critical functions of the application. Examples of tracing tools include TAU [42], Scalasca [12], VampirTrace [32], and the widely used MPI standard profiling interface [14, 20]. Such traces enables much more in-depth analysis with custom visualization tools like JumpShot [47] Vampir [32], and Paje [8].

A different approach to provide a better identification of what caused a given resource usage is brought by multi-level tools. These tools try to bridge the gap between resource-based monitoring tools and application-based tracing. Most of them work by merging different types of traces and analyzing them together [36, 35, 41]. The common characteristic found in these tools is that sensors independently collect information from the different abstraction levels of the system. The information is merged before a centralized analysis considering the whole data available takes place.

### 2.2. Data Analysis Methodologies

Several methodologies exist to do performance analysis. Perhaps the easiest way to register the general behavior of an application is to apply profiling techniques [17, 11] to measure the time spent in parts of the program. Event-based analysis, such as the ones provided by tracing tools presented in previous subsection, is used when more details are necessary to understand performance behavior. Events from the different processes of a parallel application are especially useful if the analyst needs to correlate performance issues with causal constraints. These techniques are generally coupled with visualization techniques that helps to identify reasons behind bad performance.

For profiling or event-based techniques, the analysis and behavior comprehension itself is done by the analyst. More recently, automatic trace analysis [30, 13] emerged as a solution where previously known performance problems are harvested by a program. The development of automatic pattern detection appeared to avoid the change of the source code to try to reduce trace size. While there is development to provide intra-process pattern detection [10], the combination of this with inter-process pattern detection is also already explored [21]. The automatic approach for trace analysis is also used to correlate the application-level communication topology with possible performance issues [44]. The automatic approach is especially useful for large-scale scenarios, where much tracing information must be analyzed. Another methodology in performance analysis is known as clustering techniques [27, 19], where the idea is to group processes behavior by similarity.

Obviously, the data analysis methodology used for a given large-scale application highly depends on the nature of performance issues that is under investigation. Questions such as the characteristics of the application, whether it is CPU or network-bounded, among others, must be taken into account and be used to select the best analysis methodology for each case.

### 2.3. Visualization Techniques

Visualization techniques have been employed in performance analysis of distributed and parallel programs since the earliest times of parallel and distributed application analysis. Besides the traditional scatter plot used to analyze profiling metrics of a program, several interactive views have been proposed to analyze the behavior evolution through time.



DOI: 10.1002/cpe.1885

Most of tools today have the well-known and intuitive timeline view, derived from Gantt-charts [45], where the horizontal axis represents the detailed behavior of each observed entity. Examples of tools providing this type of visualization are Vampir [33], ParaProf [4], Jumpshot [47] among many others [28, 36, 8]. The main advantage of timeline views is that they emphasize time and causality in events, thus enabling to finely debug and track down the roots of anomalies through the inspection of all details about every observed entity of the system. The increase in number of processes of today's applications has lead further developments in visualization techniques used to explore execution behavior. The space-time view, for instance, is naturally limited by the screen size, more particularly by the vertical resolution. Only a subset of entities can then be observed at the same time. Some work [31, 1] have explored clustering techniques, discussed in previous section, to group similar processes according to their behavior. Some tools, such as Vampir, already incorporate these techniques to reduce the number of entities listed in the vertical dimension of its space-time view. Even if timeline views are widely used and generally useful, they lack topological information about the network or program structure that might be crucial for program understanding.

ParaGraph [15], a precursor of performance visualization, uses several representation techniques for performance analysis. Among them, the topology-based technique is capable of showing details about network interconnection and the state of the parallel program execution in a given timestamp. More recently, this type of visualization has been explored again [39] to correlate application behavior with network infrastructure of large-scale grid environments.

*Discussion*

Our work builds on different approaches from the three areas of related research presented in this section and comes up with different trade-offs. First, our methodology, presented in Section 3, differs from existing resource usage monitoring systems by registering categorized traces of resource usage. The generated traces also contain information about the parallel application or system software that caused the resource usage in a given timestamp. This detailed tracing can give the analyst important information about the correlation of resource usage and application execution. Current tools only register raw resource usage, without any kind of categorization. Second, considering data analysis methodologies, our approach makes use of a different multi-scale data analysis methodology that transforms event-based traces into higher-level spatial/temporal aggregated information that is appropriately visualized. Our approach differs from existing methods since we allow the analyst to interactively select time intervals and entities subset to fit the analysis needs. The data related to these subsets of time and space are aggregated to obtain a scalable and representable snapshot of the application behavior on the distributed platform. And third, regarding visualization techniques, our approach uses different methods to visualize data. Although expected by most of users of high performance computing, the traditional timeline view has some limitations when applied to the analysis of large-scale distributed systems. Observing several thousands of entities with such type of visualization can sometimes be too much complex and impractical. Besides, as previously discussed, timeline views are depicted without topological information, which is sometimes crucial for the comprehension of application behavior in distributed systems. In our approach, we explore alternative visualization techniques that can prove extremely useful for understanding the behavior in large scale scenarios. The unique combination of our approach on these three areas make the novelty of our method, presented in next section.

## 3. ANALYSIS METHODOLOGY

This section presents our analysis approach for the anomaly detection in large-scale distributed system from resource usage traces. It is divided in four parts: characterized trace collection, multi-scale analysis with spatial/temporal aggregation, user interaction, and visualization techniques. The next four subsections detail each of them.



DOI: 10.1002/cpe.1885

### 3.1. Characterized Trace Collection

The first part of our methodology consists in collect resource usage traces categorized according to application components. The basic idea is to match platform information with application data. We consider that platforms are made of a set of resources $R = \{r_1, r_2, ..., r_p\}$. Typical resources are processing units or network links. The capacity of these resources generally changes over time, depending on platform configurations, resource usage, or external load. Therefore, we assume that the capacity of each resource $r \in R$ at time $t$ is denoted by $\rho_r(t)$. The *instantaneous* value $\rho_r(t)$ is typically a rate expressed in Mflops/s or in Mb/s, depending on the resource type.

Before associating application-level information to the resources, we need to explain our assumptions on the parallel or distributed applications we consider. We assume that the developers use a set of categories $C = \{c_1, c_2, ..., c_n\}$ that can be used to classify the components of the application, such as a set of tasks of a given type, request messages, specific functions of an algorithm and so on. The categories can also reflect code regions of and application (e.g., initialization or configuration), the communication and calculation steps, or the gathering of results, for example. Categories can also be used to distinguish between different applications running on the same platform.

Figure 1 depicts different possibilities of categorization of the components of a parallel or distributed application, depending on its structure. If the application is rather organized by tasks (e.g., a workflow), the developer can use the categories to differentiate the tasks. For example, an application can have tasks for requesting data or for sending back results, different type of computation tasks, or synchronization tasks. If the application is rather structured in term of process, then similar categorization can be applied. The process can then be categorized according to which function they provide to the application. The Figure depicts three categories for processes: sender, receiver and master.

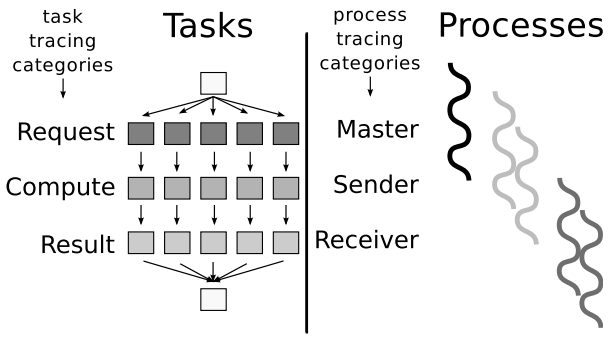


Figure 1. Two examples of tracing categories for application tasks and processes.

After the categorization of the application components, our approach works by tracing the resource utilization classified by the different categories used in the application-level. Therefore, for a given resource $r$ and a given category $c$, we assume we can trace $\rho_{r,c}(t)$, the *instantaneous resource usage* of resource $r$ by category $c$ at time $t$. Therefore, we have

$$\forall r \in R, \forall t, \quad \sum_{c \in C} \rho_{r,c}(t) \leqslant \rho_r(t) \tag{1}$$

Figure 2 shows an example of the utilization tracing for a given resource $r$, with two categories from the application-level: $cat1$ and $cat2$. The same type of tracing is performed for all the resources, considering all the categories used by the developer. Sometimes, the values of $\rho_r$ and $\rho_{r,c}$ might not be directly available from a monitoring system, but most of the time, they can be inferred from samples collected in the monitored environment.

This process of resource monitoring based on categories is different from the traditional approach, where traces register for a given period of time how much of the resource capacity was used. In our approach, instead, we register how much each resource was used by each of the categories in



DOI: 10.1002/cpe.1885

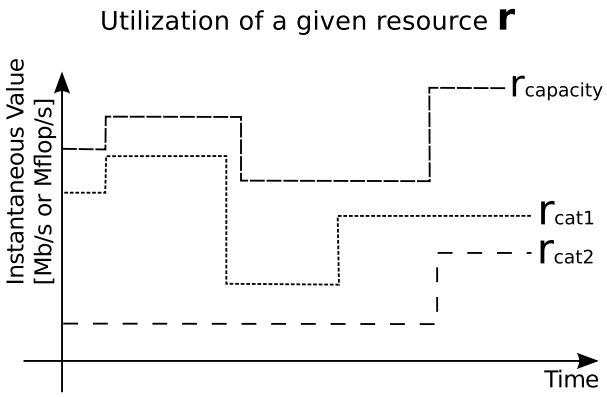


Figure 2. Tracing resource utilization by category.

the application-level. This classification of the resource utilization allows a finer-grain information on the application impact over the resources, and, combined with a good visualization technique, enables a better understanding of the overall application behavior.

Technically speaking, tracing resource utilization by categories might be less intrusive than a raw profiling (where possibly all the functions are traced), since it reduces the amount of data to be registered in trace files. At the same time, categorized tracing is more complete than a simple load trace for a processor. In practical situations, it is already possible to obtain trace data that corresponds to what we need in our approach. The computing power of a machine can be classified in a per-process base, and the traffic through the network cards of a computer can be separated in flows that can be easily associated to an application or process. Such mechanisms, associated to special tracing libraries, can isolate a part of the application and use that part as a category to classify resource utilization.

The only issue that poses some difficulty is the classification of resource utilization for inner network links, especially in the case of backbones and interconnections that the application (or a library associated with) does not have direct access. A possible solution is either to use only the current network utilization for those links, or to infer the utilization by category based on the information available in the endpoints of the network link.

### *3.2. Multi-Scale Analysis*

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) [23] 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 3.(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 3.(b)), we obviously see details that are completely hidden when considering eight months. But more interestingly, this zoom in time allows to realize that the full time frame view summarizes very poorly such twelve-day period. One could easily believe, from Figure 3.(a) that the client was available all the time during the twelve-day period whereas it was 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 3.(b) and 3.(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



DOI: 10.1002/cpe.1885

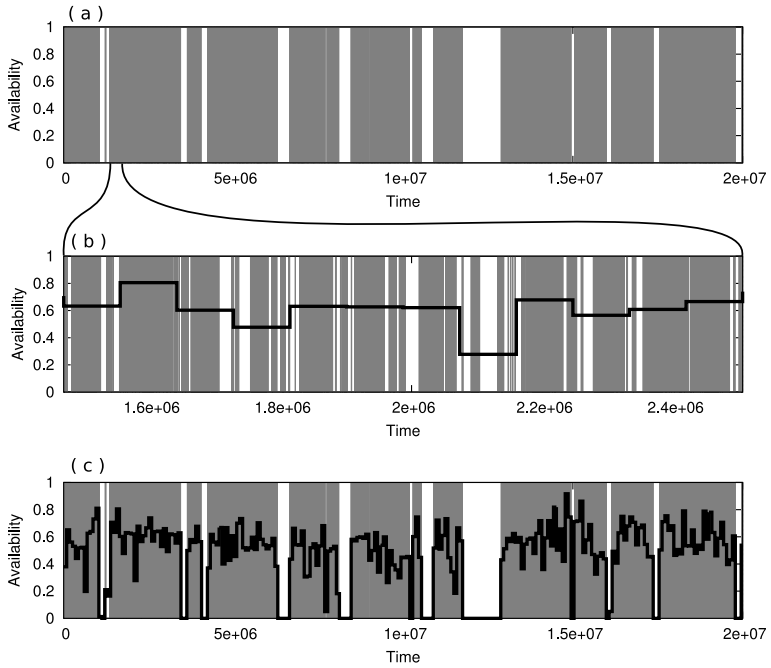


Figure 3. The gray areas represent resource availability, the black line represent a one-day integration: (a) eight months of raw traces; (b) twelve-day zoom with raw and integrated traces and (c) same as the first plot, plus one-day integration.

Figure 3.(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 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 3.(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 large-scale distributed system with a very large number of resources, this issue is even more problematic as one has 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. Since such visualization artifacts are even more likely to happen, designing specific approaches allowing to expect and control such bias is thus essential to sound analysis.

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 $\rho$ on each resource:

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

In our context, $\rho(r,t)$ could for example represent the CPU availability of resource $r$ at time $t$. It could also represent 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 3, $\rho$ 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 $\rho$ doing so (this is one of the reasons explaining the previous visualization artifact). Assume we have a way to define a neighborhood $N_{\Gamma,\Delta}(r,t)$ of $(r,t)$, where $\Gamma$ represents the size of the spatial neighborhood and $\Delta$ represents the size of the temporal neighborhood. In practice, we could for example choose $N_{\Gamma,\Delta}(r,t) = [r - \Gamma/2, r + \Gamma/2] \times [t -$



DOI: 10.1002/cpe.1885

$\Delta/2, t + \Delta/2]$, assuming our resources have been ordered. Then, we can define an approximation $F_{\Gamma,\Delta}$ of $\rho$ at the scale $\Gamma$ and $\Delta$ as:

$$F_{\Gamma,\Delta} : \begin{cases} \mathcal{R} \times \mathcal{T} & \rightarrow \mathbb{R} \\ (r,t) & \mapsto \iint_{N_{\Gamma,\Delta}(r,t)} \rho(r',t').dr'.dt' \end{cases} \quad (2)$$

Intuitively, this function averages the behavior of $\rho$ over a given neighborhood of size $\Gamma$ and $\Delta$. 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 $\Delta$ 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.

The analyst have to be careful about the conclusions that are taken during an analysis based on aggregated data. The nature of the data aggregation technique as presented here leads to the attenuation of behaviors registered in scales smaller than the one used to aggregate the data. For example, if a temporal aggregation is configured to integrate data using a two seconds interval, all the details smaller than two seconds are attenuated by the integration. At the same time, the analyst needs to be aware that, although some information is lost, such aggregation generally lead to better visualization which can allow for the detection of anomalies that could pass undetected without data aggregation. Our approach deals with such questions by letting the analyst choose freely which scale is used to aggregate trace data. Next subsection details these techniques.

### *3.3. User Interaction Techniques*

Our analysis methodology also consists in different interaction techniques applied by the analyst to navigate through the traces. These techniques consider the multi-scale treatment made on the traces, as described in previous subsection, to improve the customization of the visualization views, described in the next subsection.

Because of the possibly large number of observed components (space) and the observation periods (time), such analysis requires visualization techniques combined with the ability to efficiently navigate through space and time. This approach enables the identification of behavior at different space and time scales and thus helps the analysis process.

We detail here the time and space navigation techniques we adopt in our approach. The next subsection details two visualization techniques that can be combined with these navigation methods and that allow the observation of different aspects of the traces.

*Time Navigation* The idea of time navigation is to observe the behavior evolution of the observed components through time. Generally, the number of time stamps $t$ such that $\rho_{r,c}(t)$ changes is huge. Looking at all these events would thus be extremely tedious. Furthermore, visualizing the $\rho_{r,c}(t)$ values for a given $t$ often does not make any sense since the traces may have clock drifts or sampling issues. Last, the system global behavior is often defined at a rather large time scale.

Therefore, navigation through time requires to *select* a specific time period, to analyze it at a given *level of details*, and finally to *interact* using techniques such as animation to get conclusions about the monitoring data. We briefly review these three steps:

- **Selection:** We allow the user to define a time interval within the period of time of the trace file. It is based on this time slice that the analysis takes place.
- **Adjusting details:** To this end, we rely on temporal aggregation. It requires to define a time slice $[t_1, t_2]$, which is part of the period from the observation phase. Based on this definition, the *temporal aggregation* to compute the *average usage* is derived from equation (2):

$$\rho_{r,c}^{[t1,t2]} = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \rho_{r,c}(t).dt \quad (3)$$




DOI: 10.1002/cpe.1885

Thus, the previous inequality (1) still holds true for any $t_1 < t_2$, which enables to keep the visualization coherent, using the same kind of representation technique as if we were directly using $\rho_{r,c}(t)$.

- **Interaction:** The representation can be animated by moving the considered time slice dynamically, using information that is temporally aggregated. This animation is defined with two parameters: *frequency*, which defines how frequent the time slice will be moved forward; and *step*, defining of how much the considered time slice will be advanced. During the animation, the frequency parameter is also used to define the redraws of the representation technique used on the analysis.

In this article, it was not possible to display animations. Therefore, we chose to put online the animations that enabled us to identify phenomenons and to include in this article series of snapshots along a timeline. This latter representation allows to explain the phenomenon so it is enough for the need of this article but it is generally not enough to detect unexpected phenomenons.

*Space Navigation* Analysis of large platforms suffers from the same difficulties as large observation periods. The behavior observation of a single resource among many rarely brings conclusions. To analyze large quantities of components at the same time, we need the same kind of abilities as we previously reviewed for time navigation, i.e., the ability to easily *select* the set of observed resources, to observe them at different levels of *details*, and easy way to *interact* through this set at this level of detail if there are still too much information to display. A description of each of these steps follows:

- **Selection:** We allow the user to filter the observed components that will be analyzed. On our context, this filtering can be applied to analyze only a sub-set of monitored resources, or only a given category of the observation phase.
- **Adjusting details:** The level of detail that is observed can be defined using spatial aggregation. Its objective is to reduce the amount of data that is analyzed at the same time, reducing the complexity of the information and allowing the analyst to group components with similar behavior. Assume for example that resources or categories have been organized in a hierarchical way (e.g., grid/cluster/node/CPU/core or application/macro-step/micro-step). Then a cut at any level of this hierarchy defines a partition of $R$ into $R_1, \ldots, R_q$ such that for any $R_i$, all resources $r$ in $R_i$ have the same type. Such a partition enables us to define a *spatial aggregation* of $\rho$ by:

$$\rho_{R_i,c}(t) = \sum_{r \in R_i} \rho_{r,c}(t) \tag{4}$$

  A *category aggregation* is easily defined in a similar way. The usefulness and ease of use of such aggregation is then very dependent from the type of representation used in the visualization analysis.
- **Interaction:** We use two interaction techniques to navigate through the selected and detailed data: *shifting* and *zooming*. The shifting method works by moving the representation in a way to put in evidence a smaller part of the platform, easing the observation of the components that appear on that part. This method works together with zooming to get close to a given part of the platform, obtaining more details about it.

Here we mean *graphical* zooming in and out here, which should not be confused with spatial disaggregation and aggregation. Graphical zooming technique is related to techniques used in image manipulation programs whereas aggregation is related to techniques used for example in cartography, where one needs to reduce the complexity of the information to depict and eliminate information that are not relevant to the purpose of the map.

### *3.4. Visualization Techniques*

We use the techniques for *time* and *space* navigation in combination with two visual representations: treemap and topological views. Their common characteristic is the lack of a timeline, as the one used



DOI: 10.1002/cpe.1885

on Gantt-charts, for example. This lack of timeline allows us to use both screen dimensions to draw the components and thus display more information.

*Treemap Representation* The treemap technique [18] represents an annotated hierarchical structure on the screen using a space-filling approach. The recursive technique starts on the root of the tree, dividing the screen space among its children depending on their values. Nodes of the hierarchical structure with bigger values occupy a bigger space on the screen (more precisely, the *surface* they are allotted in the treemap is proportional to their value). This space-filling mechanism allows an easy comparison of the characteristic of the different nodes of the structure. We use treemaps on our approach to represent the resource utilization by the categories defined on the observation phase. When there is no natural hierarchical structure in the considered system or when such information is not available, we hierarchically organize the traces collected in the first phase, using each resource as a child of the root node, and then using each category defined as their children, with their respective categorized values of resource utilization. Such a hierarchy of depth one does not fully illustrate the power of spatial aggregation but already provides good insight in many settings. We plan to investigate this additional capability in future studies.

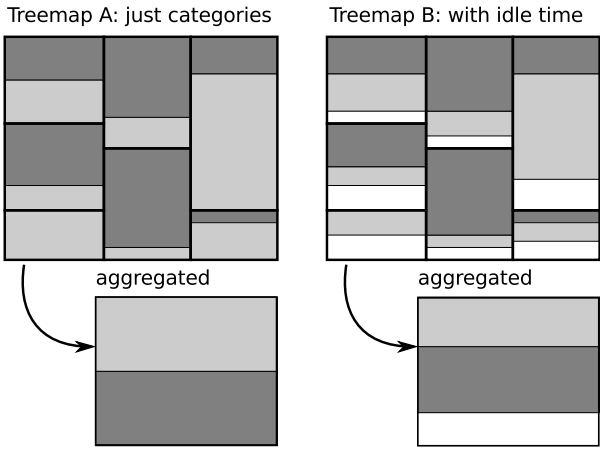


Figure 4. Treemap visualization scheme used to analyze characterized resource utilization, (A) without and (B) with idle time.

Figure 4 shows two examples of treemaps adapted to visualize our approach. Both treemaps show the resource utilization for 7 machines, limited by the thicker black lines, with their respective categorized utilization, denoted by the two different gray tonalities. The treemap on the left shows only the categorized resource utilization, whereas the treemap on the right shows also the idle time (white areas) for each resource. As previously discussed, the main advantage of this representation is that it enables to glance at the whole system and possibly to rapidly spot anomalies or unexpected behavior, especially when using aggregation. By looking at the right treemap of this figure, we instantaneously see that the can the category represented by the light gray tonalities requires less resource consumption than the other category, even though this is not true for every machine at all. Such kind of perception is still very effective when a very large number of machines is represented on the screen at the same time. We refer the interested reader to [38] for more advanced examples illustrating how scalable this representation can be.

As previously described, we explore spatial aggregation on treemaps since the monitoring data in this case can easily be hierarchically organized. The hierarchy is used to create aggregated values for the inner nodes. In the context of our approach, we can summarize the utilization counts for each resource in the root node of the hierarchy, giving a broad view of the system behavior. Figure 4 shows on the bottom part this spatial aggregation for the left and right treemaps. The aggregated view of treemap A, for instance, allows the perception that the global resource utilization is almost equal between the two types of utilization, depicted by the two gray tonalities. The same analysis is



DOI: 10.1002/cpe.1885

possible when considering the idle or non-categorized resource utilization, at the aggregated view of treemap B.

The treemap representation also uses information that was temporally aggregated. This means that what is observed in one screenshot, such as the examples of Figure 4, represents the resource behavior in a given time slice. The animation methodology previously described is also used here by dynamically updating the treemap when the time-slice moves forward. Yet, such animations do not necessarily go along well with flat non-aggregated views. Indeed, treemap use space-filling approaches that generally sort the $\rho_{r,c}$ values. Hence, when $\rho_{r,c}(t)$ evolves through time, the rectangle corresponding to a given $r, c$ may jump from one place to another, which makes such animations hard to follow.

*Topological Representation* Although there are many visualization techniques used to analyze distributed and parallel application traces, very few use topology-aware representations to understand resource utilization. Such representation are however essential to the study of spatiotemporal correlations that often exist between various measured values. For this type of representation, our approach uses a graph resource representation and customize it associating different shapes or colors to the trace variables. Such approach eases the perception of correlations and patterns on resource utilization.

A simple scheme for the topological representation is depicted on Figure 5. In this example, there are four machines, from $A$ to $D$, and three links interconnecting them ($AB$, $BD$ and $BC$). Two categories of the application are represented by the two gray tonalities. The size of the machines is related to the current power capacity (i.e., $\rho_A(t)$, $\rho_B(t)$ and so on). For the link, the width of the line is associated to the current bandwidth (i.e., $\rho_{AB}(t)$). The length of the line has now no meaning and thus, unlike machine representations, the area the rectangle has no meaning. The level inside the link is used to represent how much of the resource is used by the different categories. The remaining white area of the resources (i.e., machine $D$ and link $BD$) can be considered as idle resource if all the application components were categorized, or non-categorized resource utilization otherwise. We can observe that, in machine $A$, both categories use the computing resource. In machine $B$ and $C$, only the category represented by the darker gray tonality is consuming power, which is the same case for the link $BC$ interconnecting them.

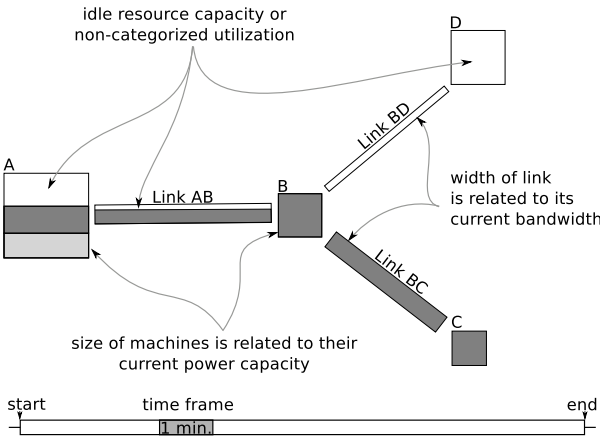


Figure 5. Visualization scheme to analyze the characterized resource utilization traces.

The topological-aware representation described here also benefits from the methodology of time and space navigation. The data that is represented is calculated based on a time slice, reflecting the behavior of all the resources and their utilization on that period of time. The Figure 5, for instance, represents the behavior of resources during a time interval of one minute. When changing the time frame (either by reducing/increasing or shifting it), the location of resources does not change. The size of resources (their inner filling) evolves according to their average power evolution (the average consumption of each category). Regarding space navigation, we use only the shifting and zooming




DOI: 10.1002/cpe.1885

methodologies, as previously described. The spatial aggregation, which can be used to summarize information about the resources, is not applied here since it would require a hierarchical organization of resources which was not available in our case study and would thus not have given more insight than the treemap representation. Yet, it would not suffer from the same problems as the treemap when using animations. Therefore, we are investigating this capability for hierarchical platforms such as grids, large clusters or clouds.

Next section details the framework used to validate our resource usage analysis approach. We present the distributed application scenario, how we obtain characterized resource utilization traces and a brief description of the visualization analysis using a tool named Triva [40], which was adapted to create topological and customized graph representations of resources. In Section 5, we present the detection of anomalies in three scenarios.

## 4. FRAMEWORK

In the section, we describe the context in which we propose to validate our resource usage anomaly detection approach. The application scenarios we choose are centered around the exploitation of volunteer computing platforms. Such large-scale distributed systems rely on many *ad hoc* mechanisms to address heterogeneity, volatility, user preferences. Furthermore, the global behavior of such systems results from the interactions of many different participants and is thus very hard to analyze. We detail these scenarios in Section 4.1. Then, we explain how we implemented the observation of the interactions and how we obtained the traces from resource utilization in Section 4.2. Last, we give a brief description of the visualization tool used to analyze the traces in Section 4.3.

### *4.1. Distributed Application Scenario*

The scenario used to evaluate the proposed anomaly detection approach is the scheduling of bag-of-tasks in volunteer computing (VC) [3] distributed platforms. We use the BOINC (Berkeley Open Infrastructure for Network Computing) [2] architecture as an example of a volunteer computing platform, depicted on Figure 6. 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 (e.g., SETI@home, Climateprediction.net, Einstein@home, 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 units, 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.

Salient characteristics of VC systems are thus their scale, their heterogeneity, and their volatility and unpredictability [24]. One way in which to cope with these characteristics is to run applications that consist of *large numbers* of *independent CPU-bound* work units (i.e., orders of magnitude larger than the number of available hosts). The BOINC architecture now implements many mechanisms perfectly suited to this kind of workload. For example, a deadline is assigned to every work unit submitted to the clients. Clients are expected to report the results before this deadline. Otherwise, the work unit will be considered lost by the server. This enables to keep track of submitted work units while ensuring that communications are always initiated by the clients. On the volunteer-side, the client tries to complete all tasks before their deadlines while respecting the project priority shares defined by the volunteer. This is implemented through a mix of *earliest deadline first* scheduling and *fair sharing* scheduling based on short term and long term debt [25]. The fair sharing issue is thus of uttermost importance since volunteers monitor that their resource is indeed used according to their preferences.

Supporting new application classes mandates that open research questions be addressed so that the effects of heterogeneity and volatility can be mitigated intelligently. For instance, scheduling



DOI: 10.1002/cpe.1885

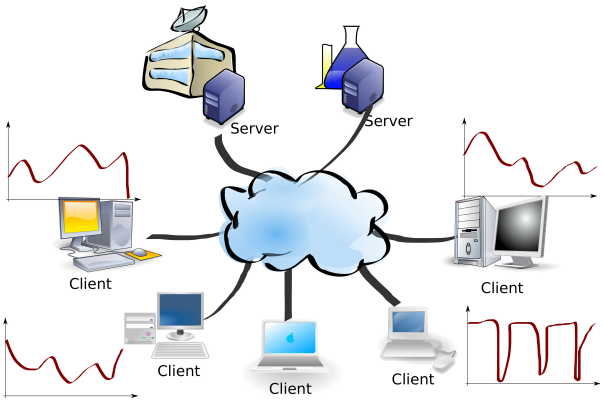


Figure 6. The BOINC architecture with two project servers and clients with varying availability.

techniques have been proposed for VC applications that consist of small numbers of work units [26, 16, 43]. Another possible extension is to allow the execution of applications that are not only CPU-bound but also incur large data transfers [5].

We explore thus the following three 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.
- **Large data transfers** When large files are associated to work units, network saturation is likely to happen next to the server. It may thus be interesting to identify these saturation to provision enough bandwidth to efficiently use all volunteers.

### *4.2. Obtaining Traces through Simulation*

The observation in real platforms of the application scenarios previously described is hard and time consuming for several reasons. In the case of BOINC, we would have to change the code for every client to get traces of resource consumption to each task. Similar information is already collected on the most recent versions of the BOINC client but we do not have access to such traces yet. Furthermore, we would like to study the interactions between BOINC projects in situations that have not been experimented too much in the real world.

These reasons lead us to use simulation to validate our approach for detecting anomalies in resource utilization. In the experiments, we used a BOINC Simulator [9] that implements most important features of the real scheduler (deadline scheduling, long term debt, fair sharing, exponential back-off). The behavior of this simulator was validated [9] against the BOINC client simulator designed by the BOINC developer team [†].

In our experiments, we only used two projects with different task creation policies. Every client was 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, while the other keeps a steady flow of task creation to be computed by the volunteer clients. One of the most interesting parts of

[†]This simulator has been used to evaluate and improve the BOINC client scheduler [25]. Therefore it only enables to simulate the behavior of a single client, whereas the BOINC simulator we use allows to simulate the whole BOINC infrastructure



DOI: 10.1002/cpe.1885

this simulator is that it considers real availabilities traces from BOINC, as defined by the Failure Trace Archive [23]. This configuration allows the analysis of the global and local fairness behavior considering situations where the failure of multiple clients might happen at the same time.

The simulator used in our experiments is developed using the SimGrid framework [7]. This framework allows the construction of different types of simulators following a well-defined interface, simulating CPU and network consumption with *ad hoc* validated mathematical models. To be able to trace the resource utilization, we instrumented the SimGrid framework allowing the association of a category with every task created and simulated on a given platform. The instrumentation guarantees that during the simulation all the CPU or network resources used by tasks of a given category will be monitored and traced. The instrumentation generates as result a trace file where the resources are listed along with their utilization by the categories used during the simulation.

We defined two categories for the BOINC simulator: *burst* and *continuous*. The burst category helps to tag all the tasks that are created by the server that generate bursts of tasks; the continuous category marks the tasks that are continuously created by the other BOINC server. By doing this, we differentiate the resource utilization according to the server that created the tasks. This classification allows a clear identification of which project is using the resources during a time period, and if this utilization is being fair considering the simulated execution time.

To complement the tracing, the instrumented SimGrid library also registers, for hosts, the maximum processing capacity (or *power*). For the interconnection, the library registers the *bandwidth* available. SimGrid is capable of dealing with resource fluctuation with availability trace files. When the *power* of a host or the *bandwidth* of a link changes, an event registering to what value was changed is registered in the trace file. With such information, we can analyze the resource utilization taking into account the maximum capacity of resources.

### *4.3. Visualization Analysis with Triva*

Triva [40] 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 works by integrating the variables inside a time frame configured by the user. It is the user responsibility to set a significant time frame for the analysis, either for a full trace observation or a small time interval. The tool is also capable to move dynamically 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. The spatial aggregation can also explore the natural hierarchical organization of the traces [38].

As of today, Triva implements the Squarified Treemap [6] visualization technique and a configurable topology-based visualization. On the treemap view, the user is capable to filter which category used during the observation phase is considered 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 detect the anomalies and understand unexpected behaviors. These visual



DOI: 10.1002/cpe.1885

representations are used in next section to analyze three different case studies based on the BOINC simulator.

## 5. RESOURCE USAGE ANOMALIES

Our approach is validated against three case studies for the BOINC scheduling scenarios: fairness analysis, projects interested in response time, and the effect of large input files. As we will see, our approach allows an easy identification of various types of anomalies such as unfair resource sharing, contention, moving network bottlenecks, and sub-optimal resource usage.

We separate these cases in three parts: setting, for detailing how the experiment was configured; expected, listing the expected results from the experiment; and observed, showing the results we obtained using our resource usage analysis approach.

### *5.1. 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[‡] whose power and availability traces are taken from the Failure Trace Archive [23]. 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 7 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. The data used to define the treemap was aggregated on 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.

The bigger treemap on Figure 7 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.

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




DOI: 10.1002/cpe.1885

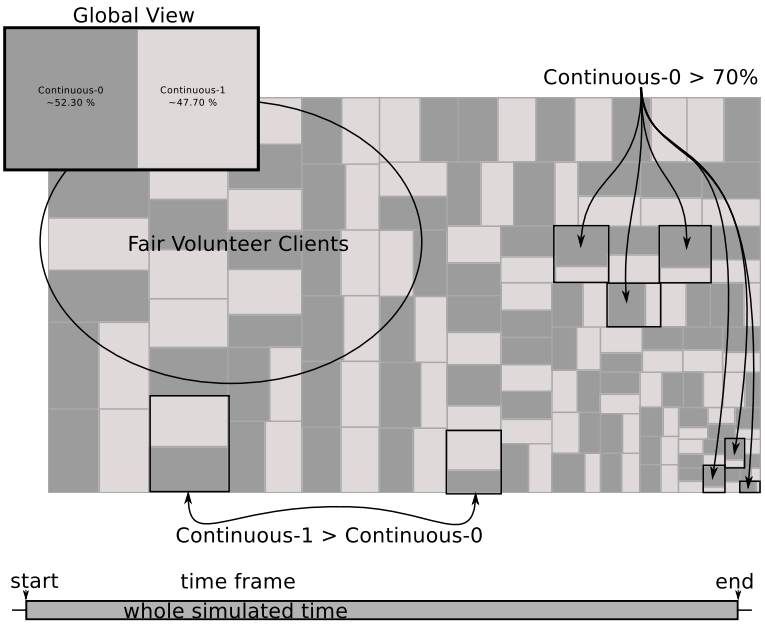


Figure 7. **Fairness anomaly** detected in some BOINC clients. Many clients with a small area (i.e., a small power or a low availability) favor application *Continuous-0*.

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 executed again the simulation, with exactly the same parameters and obtained a trace file which was again analyzed with Triva. Figure 8 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 background of this figure shows the division by project for each client. Most of the clients are now fair or more balanced than before. The lack of fairness in smaller clients (lower right corner) appears because they have a small contribution. This means that these clients have been mostly unavailable during the ten weeks and that their availability periods have been so short that being fair between the two projects while avoiding context switching every two minutes is simply not possible. The scheduling algorithm could just not balance the work between the two projects.

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 rapidly outliers.

### *5.2. Projects Interested in Response Time*

*Setting* As previously discussed on Section 4, 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



DOI: 10.1002/cpe.1885

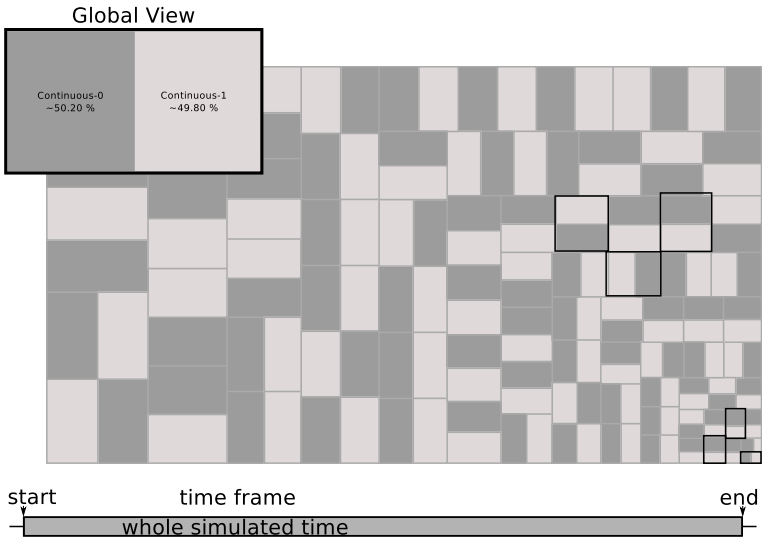


Figure 8. Fairness visualization after the correction on the implementation of the scheduling algorithm

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 [22]. 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 [43] 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 5.1. 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 [16]. What we would like to observe here is the shape of this slow-start.



DOI: 10.1002/cpe.1885

*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 9 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 9. 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.

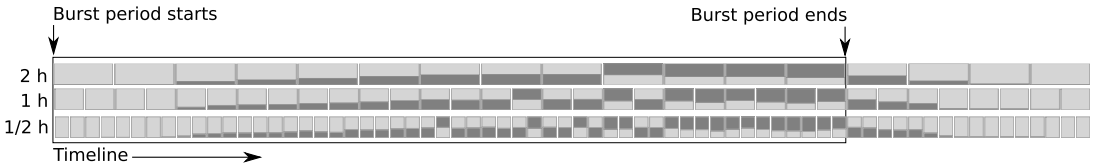


Figure 9. Observing the **slow-start** behavior of volunteer clients giving resources to the burst tasks (shown as dark gray) when the burst server becomes active. Using a 2-hour time frame, about 18 hours are required for **the burst project** to get more than half of the platform computing power and 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.

Considering that the activity period 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 execution of *Burst* tasks after the end of the burst; and the apparent difficulty of the *Burst* project to overwhelm (at least for a short time period) the *Continuous* project during the burst period.

– Wasted Computations Figure 9 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 9, the aggregated amount of work executed by the clients for each of the projects. We can observe, in the time-frame of half hour, that the project which receives more computational power



DOI: 10.1002/cpe.1885

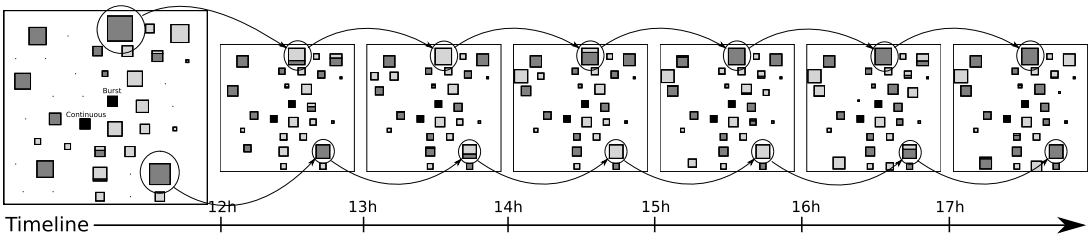


Figure 10. Observation of an anomaly consisting in the cyclic execution of continuous (light gray) and burst (dark gray) tasks on volunteer clients, about 11 hours after the beginning of the burst period. The arrows between the circled hosts help following their evolution. Executing continuous (low priority) 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-detection.html) that reveal "**blinking**" hosts (periodic switch between dark and light gray) that caught our eyes.

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 (as explained in Section 3.4, 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 10 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 10, 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 repeatedly 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 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. As in the first scenario,



DOI: 10.1002/cpe.1885

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.

### 5.3. *Projects with Large Input Files*

*Setting* In the previous experiments, the considered BOINC projects used only very small input files. Thus, we had configured SimGrid to use a very simple constant time model for network communications. Such a model enables extremely high scalability and speeds up the simulation. To study the situation where projects require larger input files, we need to take into account the possible network contention as well as the latency heterogeneity. Thus, to observe the task distribution across the platform, we configured SimGrid and the BOINC simulator to use a complex interconnection platform, and a network model capable to simulate the TCP behavior. Our goal here is to identify network bottlenecks caused by the use of large input files needed to execute the tasks by the volunteer clients. Figure 11 presents the graph view of Triva during the visualization of the platform with links to interconnect hosts. The bandwidth defines the thickness of each link, while the power configures the size of each host representation. Internally to each link or host, the gray tonalities represent the amount of resource capacity used for the configured time frame. These amounts are always proportional to the maximum capacity of a resource. The time frame, for this figure, consists of the whole simulation time. It is also worth noting the position of the two BOINC project servers, represented by the black squares. The black color here is also related to the user variable that indicates if a project is active or not. For the simulation we executed, both projects are always active.

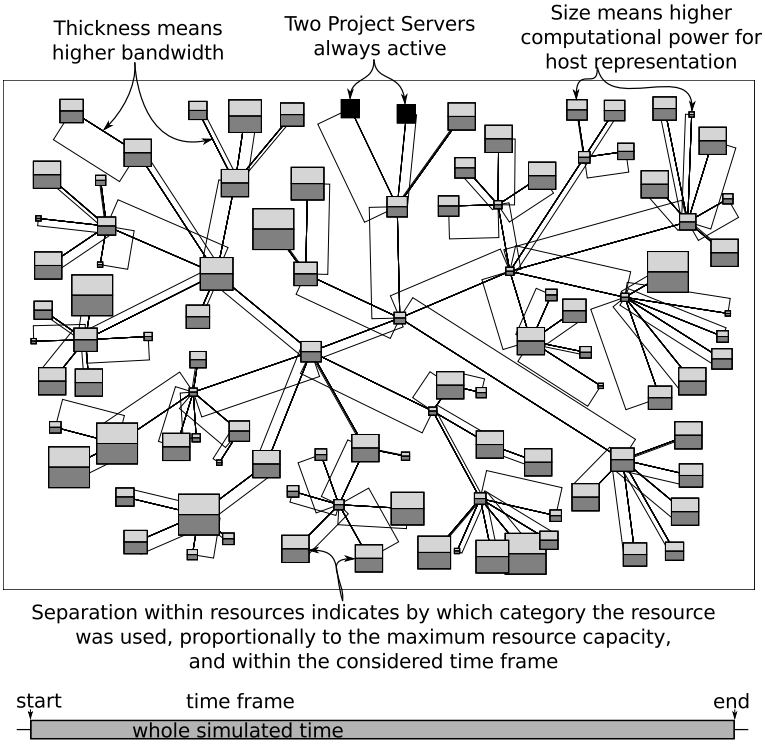


Figure 11. Full resource utilization when BOINC task distribution uses small files as input for volunteer clients. No client shows idle time whereas network links are mostly empty.

*Expected* In the first place, we simulated the BOINC task distribution using small files as input for the volunteer clients, using two project servers that are always active. This experiment will be used as a reference. The two projects compete for the communication links depending on the availability




DOI: 10.1002/cpe.1885

and network position of the volunteer clients but since the input files are very small, they should not interfere with each others on the network part.

Figure 11 depicts the overall behavior of this simulation, considering the whole simulated time of one week. As expected, all the clients, no matter their computation power, were able to execute tasks from both projects (represented by the gray tonalities inside each host) in a fair way. Each link interconnecting hosts on the figure depicts also their utilization from each project. We can observe that these links were mostly unused because of the small input files. At the same time all the hosts were fully used because tasks could arrive quickly to the volunteer clients. In the next section, we increase the size of the input files, while using exactly the same platform. We expect, by doing this, that a network bottleneck will appear somewhere around the project servers, because of the smaller bandwidth of the links, particularly for the rightmost project which has a smaller bandwidth compared to the leftmost one.

*Observed* Figure 12 shows the resulting visualization created with Triva considering the whole simulated time. The first thing to be noticed on this visualization is that the hosts are no longer fully utilized. The white space inside each node represents the amount of capacity that was not used by the two projects tasks. Such unused capacity is observed in all the hosts present in the platform. This lack of full use of hosts could be expected because of network limitation, since the tasks to be executed by clients take a longer time to be transferred. However, considering the whole simulated time, the links were mostly unused, despite the large files used as input, and the bottleneck we were expecting to see around the project servers links does not appear.

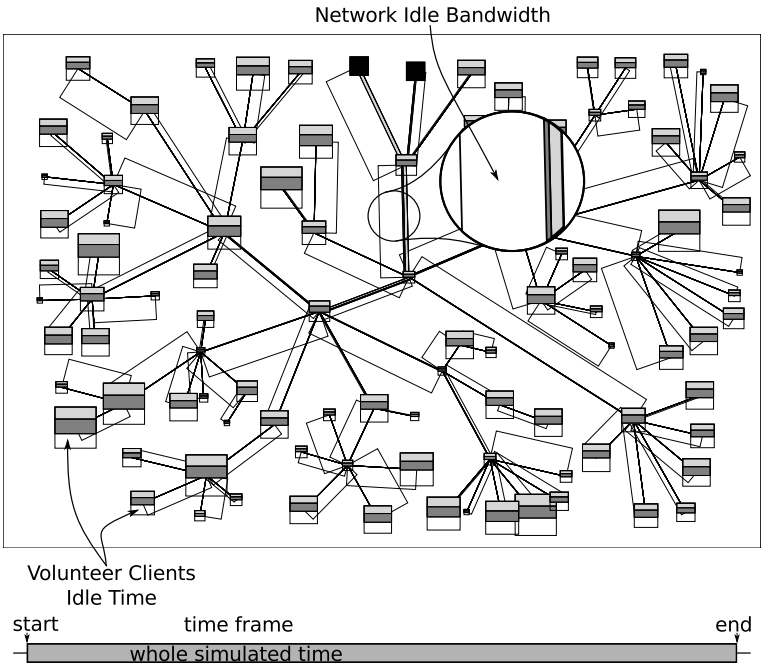


Figure 12. **Resource waste** when BOINC task distribution uses large files as input for volunteer clients. All clients show a large idle time. Surprisingly, network interconnection links around project servers do not appear as bottlenecks and have a rather low usage.

To identify which links are the origin of the bottleneck, we measured how much time each project takes to send the input files to the volunteer tasks. Then, based on this value, we configure the time frame used to calculate a given visualization screenshot, so each individual communication flow can be spotted. The Triva screenshots of Figure 13 depicts two situations (A and B) were the network link causing the bottleneck can be easily identified through the visualization. The arrows identified by the markers **a1** and **a2** point to the network links that are causing contention because they are the smaller bandwidth links among all links used for the two active communication flows (light and




DOI: 10.1002/cpe.1885

dark gray colors). The arrow **b1** on situation B, on the contrary, shows the network link causing contention because it is being shared by two communicating flows at the same time.

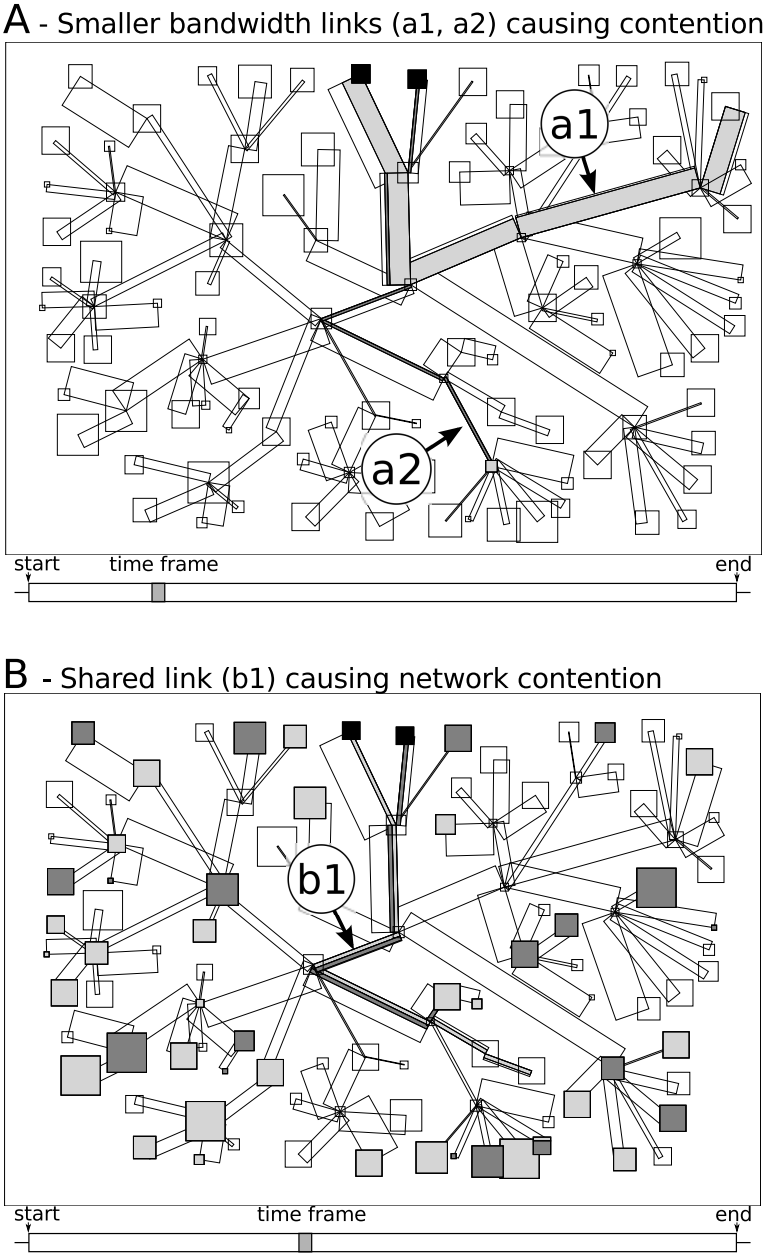


Figure 13. Identification of two types of network contention: screenshot A, caused by smaller bandwidth links (arrows a1 and a2) for each communication flow (light and dark gray colors); and B, caused by a shared network link (arrow identified by the marker b1). Therefore, every communication has indeed a bottleneck (often close to poorly connected clients) but these **bottlenecks keep moving** over time, which explains why no bottleneck appears at a large time scale.

In summary, considering the platform used to simulate the BOINC behavior, the bottlenecks appears either on the link with the smaller bandwidth of the path, either on the links that are mostly shared by the communication of the two servers with their volunteer clients. Differently from what we expected, such contentions are not specific to the links around the servers, but move over time and are distributed in different parts of the platform. Such type of network bottlenecks are commonly




DOI: 10.1002/cpe.1885

observed in real platforms, but are generally hard to spot without the aid of visualization techniques capable of showing a global and configurable view of the application and resource behavior.

## 6. CONCLUSION

In this paper, we presented an analysis methodology for traces originating from large-scale distributed systems that relies on four functionalities: the collection of categorized resource usage traces based on the application; a multi-scale aggregation technique to refine the information to be analyzed; specifically tailored user interaction techniques, to let the analyst navigate through the different levels of detail in the space/time dimensions; and two alternative visualization techniques to understand the resource usage traces. Understanding the behavior of such system requires to observe a very large number of components over a very large period of time.

One of the strengths or our approach lies in the user-defined and application-level characterization of resource usage. Our approach was exemplified through the analysis of different scheduling aspects in volunteer computing platforms, using as example a faithful simulation of BOINC. The scheduling aspects used to validate our approach are the analysis of fairness between projects inside the BOINC volunteer clients; the effect of replication and tight deadlines for projects interested in better response time; and the analysis of network contention in presence of large input files. These three scenarios were simulated, using our approach to collect resource utilization, and were analyzed using the visualization tool Triva.

These visualizations enabled to rapidly detect anomalies or unexpected behaviors. 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 a deadline and long-term fairness mechanism 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 requirements of the client scheduling algorithm.

The third scenario analyzed the effect of large input files in a volunteer computing platform using a realistic TCP network model. We have been able to identify through the visualization that overall low resource usage was due to contention constantly moving on the different links across the distributed platform, which was thus not visible at a large time-scale. Although multi-threading was not necessary in classical situations, it becomes essential to address data-intensive scenarios. Yet, implementing such anachronisms could raise other issues and would deserve further investigations.

Our contribution is certainly not related to scheduling issues that may arise in volunteer computing scenarios. Our contribution is that the use of characterized resource utilization, combined with a multi-scale aggregation technique, specifically tailored user interaction techniques and a visual interpretation of the data leads to a very effective blend enabling to identify problems that would have been otherwise extremely difficult to detect. These problems could also be analyzed in more realistic environments. Today, the BOINC client is capable of tracing local resource utilization without any kind of classification. With minor modifications to the way the BOINC client works, the local resources could be traced in a per-project manner. Almost all the proposed techniques in this paper could then be used for the analysis of these real traces, except the graph visualization (BOINC uses the Internet to transfer tasks). In more controlled platforms, such as high-performance clusters equipped with a dedicated interconnection, real network traces are available by monitoring the communication flows, giving the analyst a full graph-based analysis.



DOI: 10.1002/cpe.1885

We envision at least two important developments that should be pursued after this work. First, as we already mention, treemap visualization enable very intuitive and efficient aggregation, which is essential to the visualization of very large systems. Unfortunately, treemap do not mix well with time evolutions since location of entities is not fixed in treemap views, which make spatiotemporal correlations study rather cumbersome. Graph-based visualization faithfully depicts topological information but do not allow for spatial aggregation, which reduces their efficiency for very large configurations. We are thus investigating how the best of these two techniques could be combined.

Second, the aggregation technique we used so far is mainly based on integration, which implies that the variables we are interested in should be additive and that the analyst is somehow interested in the average behavior. There are many situations where such hypothesis do not apply and where alternative aggregation techniques should thus be defined. As we already mentioned, such issues have been at the heart of cartography since centuries. Hence, we believe techniques from the cartography community could thus reveal very useful in large-scale distributed systems study.

## 7. ACKNOWLEDGMENT

This work is partially supported by the french National Agency for Research (ANR – Agence Nationale de la Recherche) project USS SimGrid (08-ANR-SEGI-022).

## REFERENCES

1. G. Aguilera, P.J. Teller, M. Taufer, and F. Wolf. A systematic multi-step methodology for performance analysis of communication traces of distributed applications based on hierarchical clustering. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 8 pp., April 2006.
2. 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.
3. 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.
4. R. Bell, A.D. Malony, and S. Shende. Paraprof: A portable, extensible, and scalable tool for parallel performance profile analysis. *Lecture Notes in Computer Science*, pages 17–26, 2003.
5. I. Bird, L. Robertson, and J. Shiers. Deploying the lhc computing grid - the lcg service challenges. In *LGDI '05: Proceedings of the 2005 IEEE International Symposium on Mass Storage Systems and Technology*, pages 160–165, Washington, DC, USA, 2005. IEEE Computer Society.
6. 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.
7. 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.
8. Jacques Chassin de Kergommeaux and Benhur 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.
9. 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.
10. F. Freitag, J. Caubet, and J. Labarta. A trace-scaling agent for parallel application tracing. In *Tools with Artificial Intelligence, 2002. (ICTAI 2002). Proceedings. 14th IEEE International Conference on*, pages 494 – 499, 2002.
11. K. F ”uerlinger, N.J. Wright, and D. Skinner. Effective Performance Measurement at Petascale Using IPM. In *16th International Conference on Parallel and Distributed Systems*, pages 373–380. IEEE, 2010.
12. 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.
13. Juan Gonzalez, Judit Gimenez, and Jesus Labarta. Automatic detection of parallel applications computation phases. *Parallel and Distributed Processing Symposium, International*, 0:1–11, 2009.
14. William Gropp, Ewing L. Lusk, and Rajeev Thakur. *Using MPI-2: Advanced Features of the Message Passing Interface*. The MIT Press, 1999.
15. MT Heath and JA Etheridge. Visualizing the performance of parallel programs. *IEEE software*, 8(5):29–39, 1991.
16. E.M. Heien, D.P. Anderson, and K. Hagihara. Computing low latency batches with unreliable workers in volunteer computing environments. *Journal of Grid Computing*, 7(4):501–518, 2009.
17. D. Jeon, S. Garcia, C. Louie, S.K. Venkata, and M. Taylor. Kremlin: Like gprof, but for parallelization. In *Principles and Practice of Parallel Programming*, 2011.
18. B. Johnson and B. Shneiderman. *Tree-Maps: a space-filling approach to the visualization of hierarchical information structures*. IEEE Computer Society Press Los Alamitos, CA, USA, 1991.
19. Ajay Joshi, Aashish Phansalkar, Lieven Eeckhout, and Lizy Kurian John. Measuring benchmark similarity using inherent program characteristics. *IEEE Transactions on Computers*, 55:769–782, 2006.



DOI: 10.1002/cpe.1885

20. 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.
21. Andreas Knüpfer, Bernhard Voigt, Wolfgang E. Nagel, and Hartmut Mix. Visualization of repetitive patterns in event traces. In *Proceedings of the 8th international conference on Applied parallel computing: state of the art in scientific computing*, PARA'06, pages 430–439, Berlin, Heidelberg, 2007. Springer-Verlag.
22. D. Kondo, A.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.
23. 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.
24. Derrick Kondo, Michela Taufer, Charles L. Brooks III, Henri Casanova, and Andrew A. Chien. Characterizing and evaluating desktop grids: An empirical study. *International Parallel and Distributed Processing Symposium*, 1:26b, 2004.
25. Derrrick Kondo, David Anderson, and John VII McLeod. Performance Evaluation of Scheduling Policies for Volunteer Computing. In *Proc. of the 3rd IEEE Intl. Conf. on e-Science and Grid Computing (e-Science)*, Bangalore, India, 2007.
26. Derrrick Kondo, Andrew A. Chien, and Henri Casanova. Rapid application turnaround on enterprise desktop grids. In *Proc. of the ACM Conf. on High Performance Computing and Networking (SC)*, 2004.
27. C. W. Lee, C. Mendes, and L.V. Kale. Towards scalable performance analysis and visualization through data reduction. In *IEEE International Symposium on Parallel and Distributed Processing*, pages 1 –8, 2008.
28. C.W. Lee, C. Mendes, and L.V. Kalé. Towards scalable performance analysis and visualization through data reduction. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, pages 1–8. IEEE, 2008.
29. 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.
30. B. Mohr and F. Wolf. Kojak-a tool set for automatic performance analysis of parallel programs. *Lecture notes in computer science*, pages 1301–1304, 2003.
31. Kathryn Mohror and Karen L. Karavanic. Evaluating similarity-based trace reduction techniques for scalable performance analysis. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 55:1–55:12, New York, NY, USA, 2009. ACM.
32. 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.
33. W.E. Nagel, A. Arnold, M. Weber, H.C. Hoppe, and K. Solchenbach. Vampir: Visualization and analysis of mpi resources. *Supercomputer*, 12(1):69–80, 1996.
34. H.B. Newman, I. Legrand, P. Galvez, R. Voicu, and C. Cirstoiu. Monalisa : A distributed monitoring service architecture. *CoRR*, cs.DC/0306096, 2003.
35. F.G. Ottogalli, C. Labbé, V. Olive, B. de Oliveira Stein, J.C. de Kergommeaux, and J.M. Vincent. Visualisation of distributed applications for performance debugging. In *Proceedings of the International Conference on Computational Science-Part II*, pages 831–840. Springer-Verlag London, UK, 2001.
36. V. Pillet, J. Labarta, T. Cortes, and S. Girona. Paraver: A tool to visualise and analyze parallel code. In *Proceedings of Transputer and occam Developments, WOTUG-18.*, volume 44 of *Transputer and Occam Engineering*, pages 17–31, Amsterdam, 1995. [S.l.]: IOS Press.
37. P.C. Roth, D.C. Arnold, and B.P. Miller. MRNet: A software-based multicast/reduction network for scalable tools. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 21. IEEE Computer Society, 2003.
38. 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.
39. Lucas Mello Schnorr, Guillaume Huard, and Philippe Olivier Alexandre Navaux. Visual mapping of program components to resources representation: a 3d analysis of grid parallel applications. In *Proceedings of the 21st Symposium on Computer Architecture and High Performance Computing*. IEEE Computer Society, 2009.
40. Lucas Mello Schnorr, Guillaume Huard, and Philippe Olivier Alexandre Navaux. Triva: Interactive 3d visualization for performance analysis of parallel applications. *Future Generation Computer Systems*, 26(3):348 – 358, 2010.
41. Lucas Mello Schnorr, Philippe O. A. Navaux, and Benhur de Oliveira Stein. Dimvisual: Data integration model for visualization of parallel programs behavior. In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, pages 473–480, Washington, DC, USA, 2006. IEEE Computer Society.
42. S.S. Shende and A.D. Malony. The TAU parallel performance system. *International Journal of High Performance Computing Applications*, 20(2):287, 2006.
43. 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.
44. F. Song, F. Wolf, J. Dongarra, and B. Mohr. Automatic experimental analysis of communication patterns in virtual topologies. In *Proceedings of the 2005 International Conference on Parallel Processing*, pages 465–472. IEEE Computer Society, 2005.
45. James M. Wilson. Gantt charts: A centenary appreciation. *European Journal of Operational Research*, 149(2):430–437, September 2003.
46. 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.
47. 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.




DOI: 10.1002/cpe.1885

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



DOI: 10.1002/cpe.1885