HAL
open science

Visualization and Detection of Resource Usage Anomalies in Large Scale Distributed Systems

Lucas Mello Schnorr, Arnaud Legrand, Jean-Marc Vincent

To cite this version:

Lucas Mello Schnorr, Arnaud Legrand, Jean-Marc Vincent. Visualization and Detection of Resource Usage Anomalies in Large Scale Distributed Systems. [Research Report] RR-7438, INRIA. 2010. inria-00529569

HAL Id: inria-00529569

https://inria.hal.science/inria-00529569v1

Submitted on 26 Oct 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Visualization and Detection of Resource Usage Anomalies in Large Scale Distributed Systems

Lucas Mello Schnorr — Arnaud Legrand — Jean-Marc Vincent

N° 7438

October 15th, 2010

Domaine 3

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--7438--FR+ENG

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA

centre de recherche GRENOBLE - RHÔNE-ALPES

Visualization and Detection of Resource Usage Anomalies in Large Scale Distributed Systems

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

Domaine : Réseaux, systèmes et services, calcul distribué
Équipe-Projet MESCAL

Rapport de recherche n° 7438 — October 15th, 2010 — 31 pages

Abstract: Understanding the behavior of large scale distributed systems such as clouds, computing grids or volunteer computing systems is generally extremely difficult and tedious as it requires to observe a very large number of components over a very large period of time. The analysis of distributed systems generally begins with gathering resource utilization monitoring data through the use of observation tools. This information can then be explored with different analysis techniques to understand the reason behind anomalies that can be present in the system. This paper follows the same two-phase approach but proposes some methods that reveal particularly well suited to the study of very large scale distributed systems. More specifically, in the first phase, we register resource utilization categorized according to application components. The second phase proposes various *ad hoc* different visualization techniques enabling easy navigation through space and time. We demonstrate the efficiency of this approach through the analysis of simulations of the famous volunteer computing BOINC architecture. These simulations rely on the SimGrid framework, to which our analysis techniques have been incorporated. Three scenarios are analyzed in this paper: analysis of the resource sharing mechanism, resource usage of projects that aim at optimizing response time instead of throughput, and the impact of input file size on such an architecture. The results show that our approach allows an easy identification of different types of resource usage anomalies, unfair resource sharing, contention, moving network bottlenecks, and suboptimal resource usage.

Key-words: performance visualization analysis, large-scale distributed systems, volunteer computing, grid computing, cloud computing, resource usage anomalies

* INRIA MESCAL Research Team, CNRS Lig Laboratory, University of Grenoble

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Visualisation et Detection des Anomalies d'Utilisation de Resources en Systèmes Distribuées à Grand Échelle

Résumé : La comprehension du comportement des systèmes distribués de large échelle tels que des systèmes dématérialisés, le calcul des grilles ou des systèmes de calcul volontaires est généralement extrêmement difficile et pénible car il exige l'observation d'un nombre très grand de composants sur une période de temps aussi grande. L'analyse des systèmes distribués commence généralement par recueillir des données de surveillance d'utilisation de ressource par l'utilisation des outils d'observation. Cette information peut alors être explorée avec différentes techniques d'analyse pour comprendre la raison derrière les anomalies qui peuvent être présentes dans le système. Ce document suit la même approche biphasée mais propose quelques méthodes qui se revelent particulièrement bien adaptés à l'étude des systèmes distribués à large échelle. Plus spécifiquement, dans la première phase, nous enregistrons l'utilisation de ressource classée par catégorie selon des composants d'application. La deuxième phase propose divers techniques de visualisation pour permettre une navigation facile dans l'espace et le temps. Nous démontrons l'efficacité de cette approche par l'analyse des simulations de la célèbre architecture BOINC pour le calcul volontaire. Les simulations se fondent sur le cadre de SimGrid, auquel nos techniques d'analyse ont été incorporées. Trois scénarios sont analysés en ce document : analyse du mécanisme de partage de ressource, utilisation de ressource des projets qui visent à optimiser le temps de réponse au lieu de la sortie, et l'impact de la taille de fichier d'entrée sur une telle architecture. L'exposition de résultats que notre approche permet une identification facile de différents types d'anomalies d'utilisation de ressource, partage de ressource injuste, contention, goulot d'étranglement mobiles sur le réseau, et utilisation de ressource suboptimale.

Mots-clés : analyse de visualisation d'exécution, systèmes distribués à grande échelle, calcul volontaire, grille calculant, nuage calculant, anomalies d'utilisation de ressource

1 Introduction

Today's distributed computing systems such as clouds, computing grids or 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. Such analysis is the more difficult part since it generally requires to observe a very large number of components over a very large period of time.

The analysis and debugging of distributed systems consists basically in the deployment of observation tools that collect the dynamic behavior of the system and the applications that execute on top of it. During this collection process or after, visualization tools process the data to provide insights and a performance overview of the system. The data is generally presented either through a set of performance graphs with correlated information, or with different visualization schemes.

We can distinguish between three main categories types of tools to observe a distributed system: monitoring, profiling, and tracing tools.

- **Monitoring** There are several tools today used to monitor the resources of distributed and parallel systems. Some examples include the Ganglia monitoring system [20], the Network Weather Service [33], Monalisa [23] and others [35]. Most of these collection systems gather resource information such as the processor or network utilization.

- **Tracing and Profiling** At the application level, tools are mostly focused on registering local and global states of the program, the amount of application-data transferred in messages, and counters for specific functions. In general, such application-level observation allows the detection of the time spent in communications and actual algorithm execution. Examples of such tools include TAU [29], Scalasca [8], VampirTrace [21], and the MPI standard profiling interface [9, 12]. Such tools can be used either for profiling, i.e., to get statistics about the application, or for tracing, i.e., to log information during the execution without summarizing them. The later is generally much more intrusive and thus does not scale very well.

Merging together the information collected by a resource-level observation and by an application-level observation is extremely difficult as it would require to identify *a posteriori* a particular resource utilization to a particular application component.

Yet, we think that, in the case of large scale distributed systems, where resources are typically shared by several users and programs, abnormal resource usage can only be detected if the information collected about the resources is classified according to application-level components. By doing this, we can understand the impact of applications (or of sub-parts of applications) on the resources and observe if that corresponds to the expected behavior of the application. When such type of analysis is possible, developers clearly spot bottlenecks caused by misplaced processes of the application on the system, or an unexpected behavior from the application on a particular distributed platform. Another possible benefit of such approach is the characterization of resource utilization based on different components of the distributed application. For

instance, if several communications share a given interconnection link, this approach enables to identify which of them consumes most of the interconnection bandwidth. Yet, visualization tools usually represent either the resource utilization or application traces. Although some multi-level performance visualization is already possible [28, 25], there is a lack of tools capable to visualize the resource utilization characterized by application components.

In this paper, we propose efficient techniques for detecting resource usage anomalies in large scale distributed systems. Our approach relies on a simple classification of resource utilization based on application components, such as tasks of a given type, processes of a certain nature, and so on. These components are grouped by categories that are used during the execution of the application to monitor and characterize the utilization of the resources. After the execution, the trace analysis relies on a set of generic but well-suited visualization techniques to detect abnormal behaviors of the application from the point-of-view of resources. Such visualizations enables to identify resource usage anomalies and give hint on which part of the program may be responsible for it.

We validate our anomaly detection approach through the analysis of a simulated BOINC architecture [1, 7] using the SimGrid framework [6] 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 sub-optimal resource usage.

The paper is structured as follows. Next section presents our resource usage analysis approach, considering resource and application-level information. Section 3 presents the framework we used to validate the approach. Section 4 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 *fair 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 impact 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. Section 5 presents related work to our approach and we finish the paper through the conclusion, on Section 6.

2 Resource Usage Analysis Approach

As we explained earlier, the analysis of distributed systems is generally divided in two phases. The first one is the observation of resource utilization. It details the way the data is traced during a period of time. After the end of this first phase, the data is analyzed in the visualization phase. This second phase deals with the analysis of the distributed system behavior from the traces using different visualization techniques. Both phases are described on next subsections.

2.1 Observation Phase

For the first phase, 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. After the categorization of the application components, our approach works by

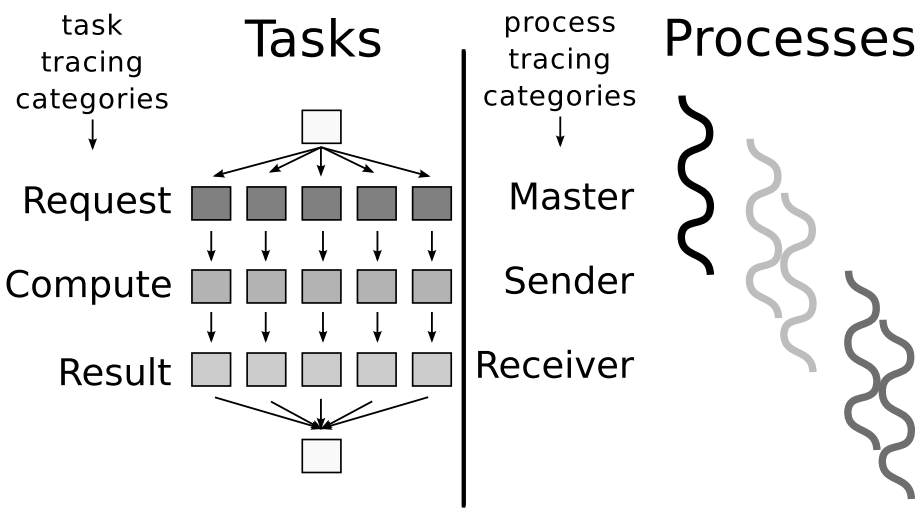


Figure 1: Two examples of tracing categories for application tasks and processes.

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 ρ_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.

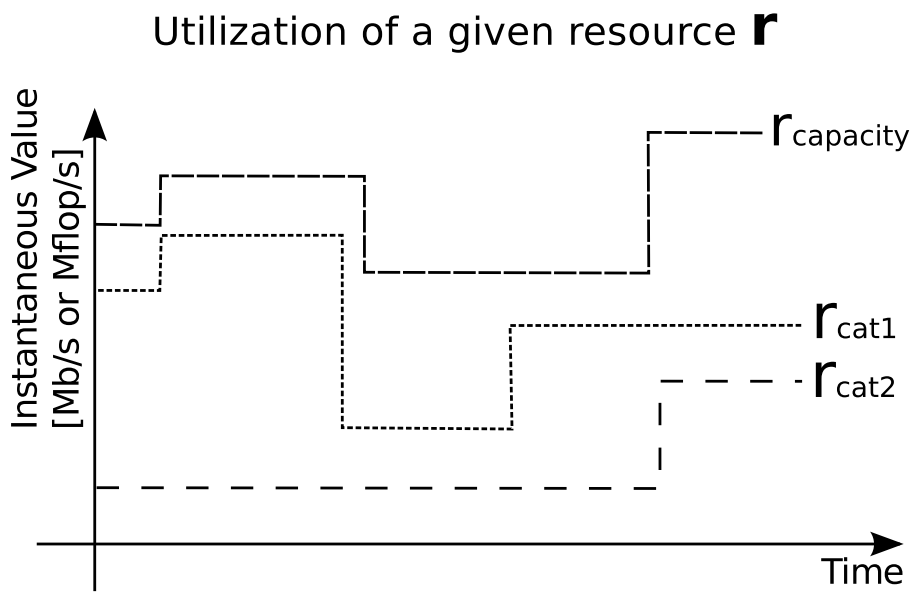


Figure 2: Tracing resource utilization by category.

This process of resource monitoring based on categories is different from the traditional approach, where the traces register for a given period 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 the application-level. This classification of the resource utilization allows a fine-grain control of 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 the resource utilization by categories is less intrusive than a raw profiling (where possibly all the functions are traced), and at the same time more complete than a simple load trace for a processor, for instance. 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 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.

2.2 Visualization Phase

The second phase of our approach consists in analyzing the resources utilization according to the categories used during the observation phase. 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 debugging process.

We detail here the time and space navigation methodologies we adopt in our approach. After this, we detail 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 use a technique known as temporal aggregation. It considers the definition of 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 done in the following way:

$$\rho_{r,c}^{[t1,t2]} = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \rho_{r,c}(t).dt \tag{2}$$

  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 sufficient 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 the 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 ρ by:
$$\rho_{R_i,c}(t) = \sum_{r \in R_i} \rho_{r,c}(t) \tag{3}$$
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.

We use the previous 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 on Gantt-charts, for example. This lack of timeline allows us to use both screen dimensions to draw the components. More details about them are described as follows.

Treemap Representation

The treemap technique [11] 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. To this end, 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 was sufficient in our case study. We plan to investigate this additional capability in future studies.

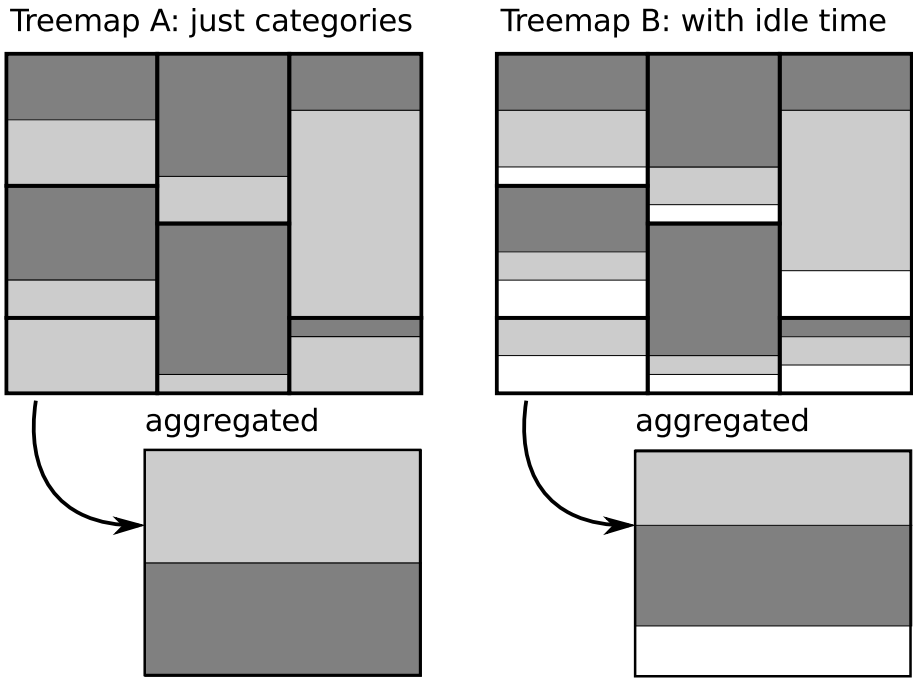


Figure 3: Treemap visualization scheme used to analyze characterized resource utilization, (A) without and (B) with idle time.

Figure 3 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, whether 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 rapidly shows possible anomalies or unexpected behavior. By looking at the left treemap of this figure, we can conclude that some machines were used more for the category represented by the light gray tonalities, at the same time some machines work more for the other category. Such perception is also valid when a larger number of machines is represented on the screen at the same time. We refer the interested reader to [27] for more advanced examples illustrating how scalable this representation can be.

As previously described, we explore spatial aggregation on the treemaps since the the monitoring data in this case is hierarchically organized. The hierarchy is explored to create intermediary values that are on the leaves of the hierarchy. 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 3 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 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 3, 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. 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 4. 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 currently 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.

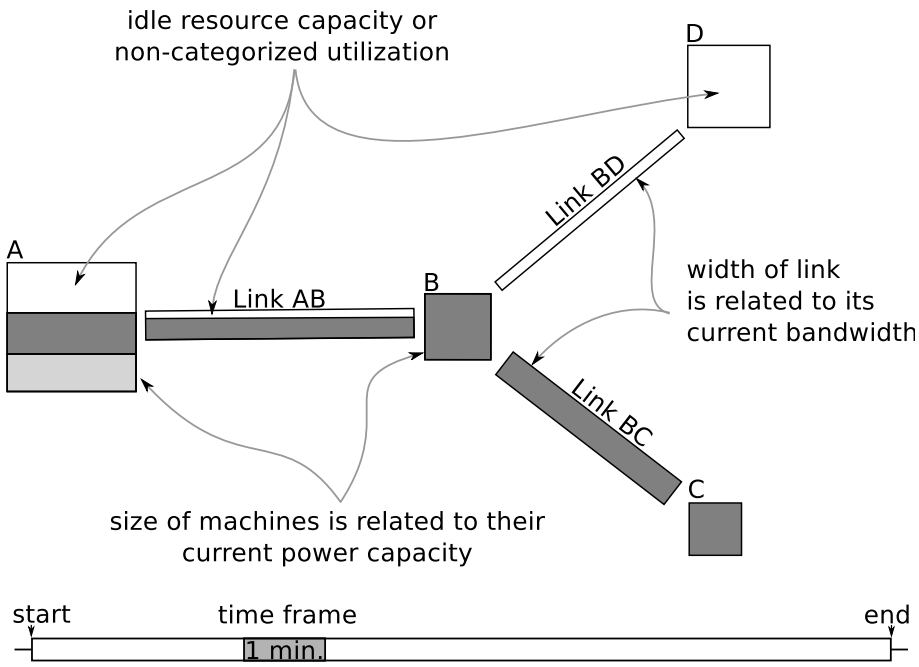


Figure 4: Visualization scheme to analyze the characterized resource utilization traces.

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 4, 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 (resp. their inner filling) evolves according to their average power evolution (resp. the average consumption of each category). Regarding space navigation, we use only the shifting and zooming 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 currently 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 [26], which was adapted to create topological and customized graph representations of resources. On Section 4, we present the detection of anomalies in three scenarios.

3 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 3.1. Then, we explain how we implemented the observation of the interactions and how we obtained the traces from resource utilization in Section 3.2. Last, we give a brief description of the visualization tool used to analyze the traces in Section 3.3.

3.1 Distributed Application Scenario

The scenario used to evaluate the proposed anomaly detection approach is the scheduling of bag-of-tasks in volunteer computing (VC) [2] distributed platforms. We use the BOINC (Berkeley Open Infrastructure for Network Computing) [1] architecture as an example of a volunteer computing platform, depicted on Figure 5. 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 [16]. 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

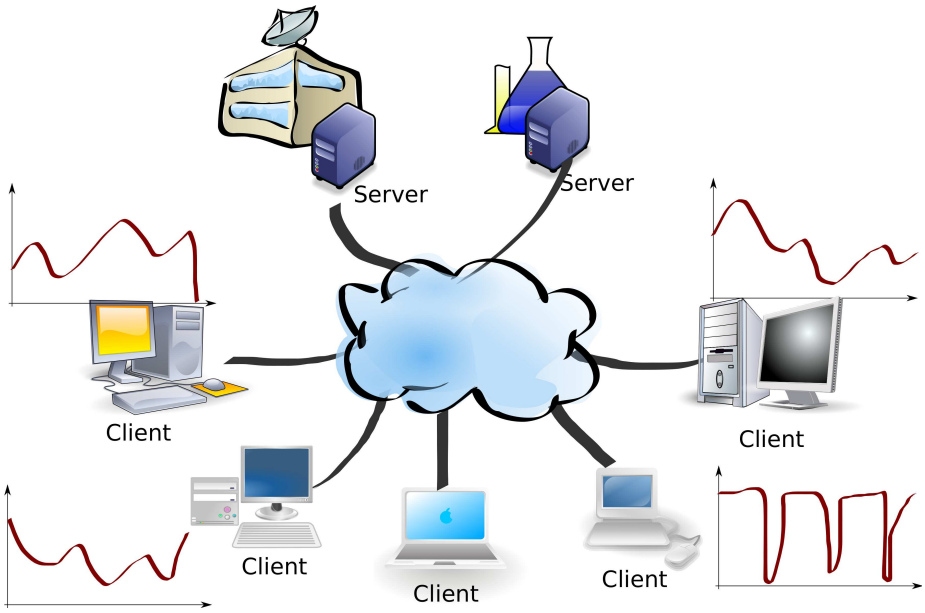


Figure 5: The BOINC architecture with two project servers and a number of clients with varying availability.

implemented through a mix of *earliest deadline first* scheduling and *fair sharing* scheduling based on short term and long term debt [17]. 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 techniques have been proposed for VC applications that consist of small numbers of work units [18, 10, 30]. Another possible extension is to allow the execution of applications that are not only CPU-bound but also incur large data transfers [4].

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.

3.2 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 [7] 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 [7] against the BOINC client simulator designed by the BOINC developer team [1].

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 is that it considers real availabilities traces from BOINC, as defined by the Failure Trace Archive [15]. 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 [6]. 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,

[1] This simulator has been used to evaluate and improve the BOINC client scheduler [17]. 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

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.

3.3 Visualization Analysis with Triva

Triva [26] 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 [27].

As of today, Triva implements the Squarified Treemap [5] 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 representations are used in next section to analyze three different case studies based on the BOINC simulator.

4 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 impact 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.

4.1 Fair Sharing

Setting

For this case study, we create two projects that generate continuous tasks, and two categories for the tracing: *Continuous-0* and *Continuous-1*. The only difference between the two projects is the size of the jobs and their corresponding deadline. The tasks from the *Continuous-0* project are 30 times longer than tasks from the *Continuous-1* project. The deadlines are set to 300 hours and 15 hours for project *Continuous-0* and *Continuous-1*. There were a total of 65 clients[2] whose power and availability traces are taken from the Failure Trace Archive [15]. 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 6 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 6 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.

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

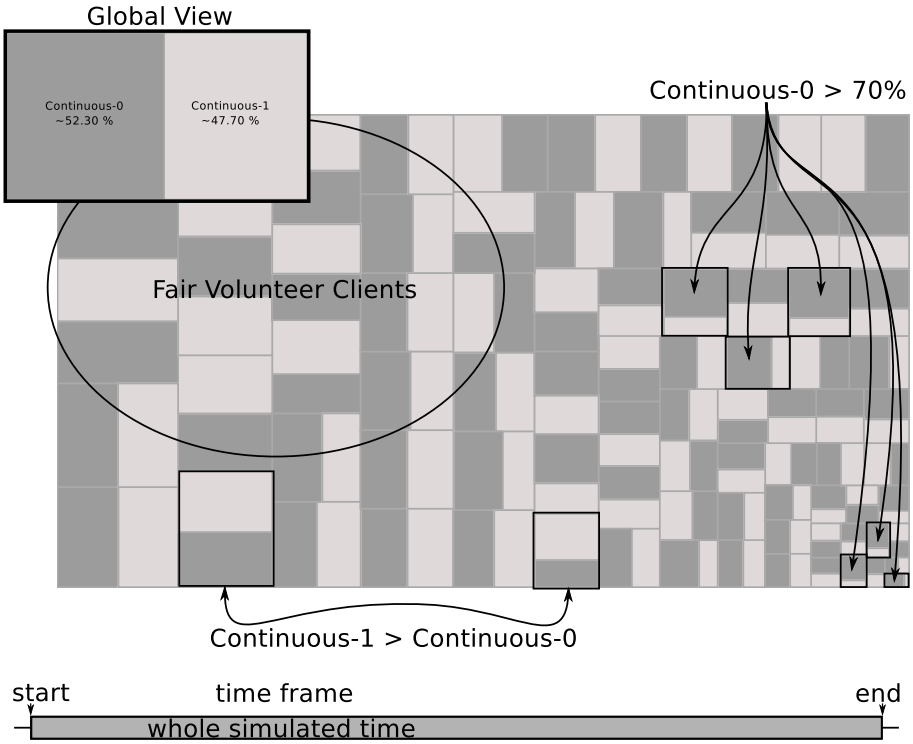


Figure 6: **Fairness anomaly** detected in some BOINC clients. Many clients with a small area (i.e., a small power and/or a low availability) favor application *Continuous-0*.

By analyzing the treemap view, we can notice that smaller clients (occupying a smaller area of the drawing), which reflects less contribution to the work, present such strange behavior. On the other hand, bigger clients, which contributed more to the work, are mostly fair. The size difference between the clients is related to both the power of the hosts and their availability. Since the power heterogeneity is not that large, it can easily be deduced that the smaller clients have very long unavailability periods. Therefore unfairness seemed to be related to unavailability.

This observation was done during the early stages of the development of the BOINC simulator so we informed the developers about this unexpected phenomenon. The origin of the problem, which was identified later on, was related to the algorithm that counts the time worked for each project on the clients. After the problem resolution, we executed again the simulation, with exactly the same parameters and obtained a trace file which was again analyzed with Triva. Figure 7 shows the analysis of this trace file, with the global view of fairness among all clients depicted on its top left corner. Now, the division between the two projects reaches 50.20% for the *Continuous-0* tasks, and 49.80% for the *Continuous-1* tasks, considering the simulation time of ten weeks. The bigger treemap on the figure shows the division by project for each client, allowing the observation that even clients that contribute less are also fair.

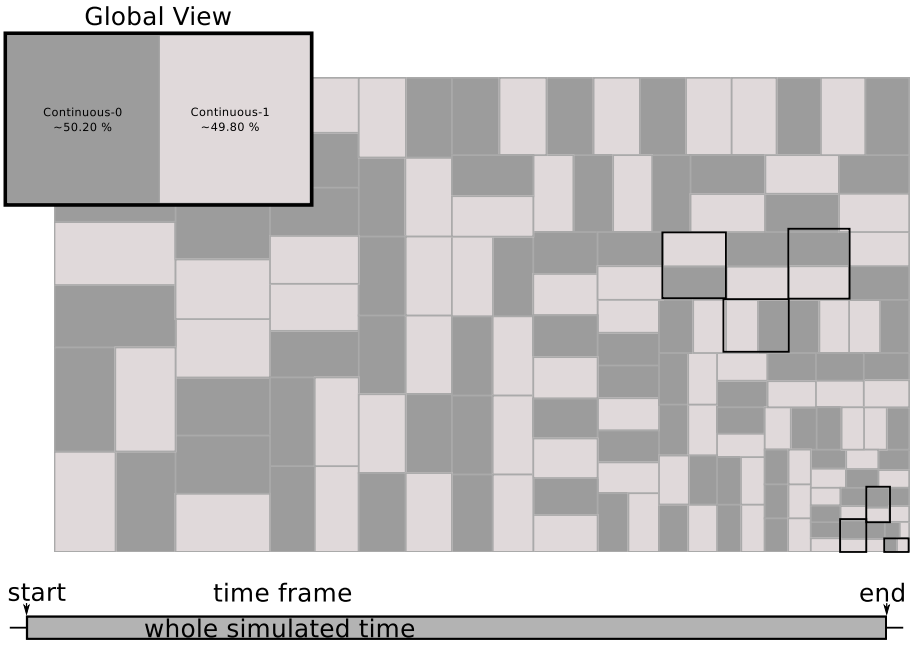


Figure 7: Fairness visualization after the correction on the implementation of the scheduling algorithm

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

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

4.2 Projects Interested in Response Time

Setting

As previously discussed on Section 3, BOINC targets projects with CPU-bound tasks interested in throughput computing. One of the main mechanism that give project servers some control over task distribution is the completion deadline specification. This parameter allows projects to specify how much time will be given to volunteer clients to return the completed task. This is used as a soft deadline specification upon task submission, but above all as a way of tracking task execution. On the client side, this deadline is employed to decide which project to work for. Thus, a client never starts working on an overdue task

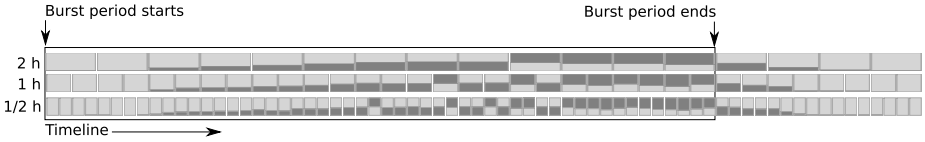


Figure 8: Observing the **slow-start** behavior of volunteer clients giving resources to the burst tasks (shown as dark gray) when the burst server becomes active. Using a 2-hour time frame, we can see that about 18 hours are required for **the burst project** to get more than half of the platform computing power and that it **hardly gets more**. This last observation is surprising as bursts are infrequent and should thus have a temporary high priority compared to the continuous project (clients try to comply to a long-term fair sharing policy). Using a smaller time frame, we observe **oscillations**: the dominant project is alternatively the burst or the continuous project.
Furthermore, we observe that active burst tasks remain in the system after the termination of the burst. This **waste** can be explained by "loose" deadlines and replication.

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 [14]. 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 [30] already implement many of these features.

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

Expected

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

tasks since a long time should get a temporary higher priority than projects whose tasks are available all the time.

This mechanism lead us to expect a slow-start execution of tasks from the project that generates bursts of tasks, from time to time. Such behavior was already anticipated and optimized in other works [10]. What we would like to observe here is the shape of this slow-start.

Observed

To observe the slow-start effect, we decided to observe the system with a treemap view configured to show only aggregated states from all the simulated machines. Figure 8 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 8. It takes from 2 to 3 hours for a client to realize the activity of the *Burst* project server and start the execution of its tasks. According to the view of two hours, it takes 18 hours to the burst tasks consume more than half of the power of the platform, despite its tighter deadlines that indicate that it should be given a higher priority.

Considering that the activity 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 8 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.

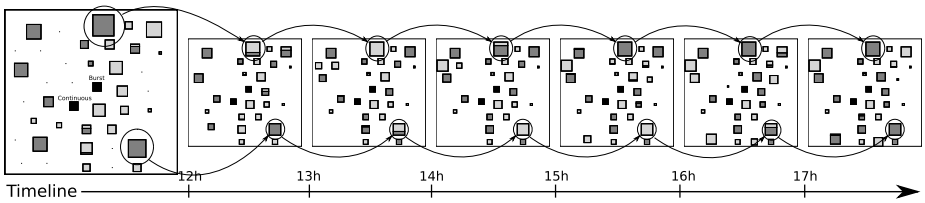


Figure 9: Observation of an anomaly consisting in the cyclic execution of continuous (light gray) and burst (dark gray) tasks on volunteer clients, about 11 hours after the beginning of the burst period. The arrows between the circled hosts help following their evolution. Executing continuous (low priority) 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/2010-tpds.html`) that reveal "**blinking**" hosts (periodic switch between dark and light gray) that caught our eyes.

– ***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 8, 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 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 2.2, 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 9 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 9, 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 may be in this context. As in the first scenario, this issue would probably never have been identified without both the spatial and time aggregation and zooming capabilities of Triva and its ability to seamlessly move from one representation to another.

4.3 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 10 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.

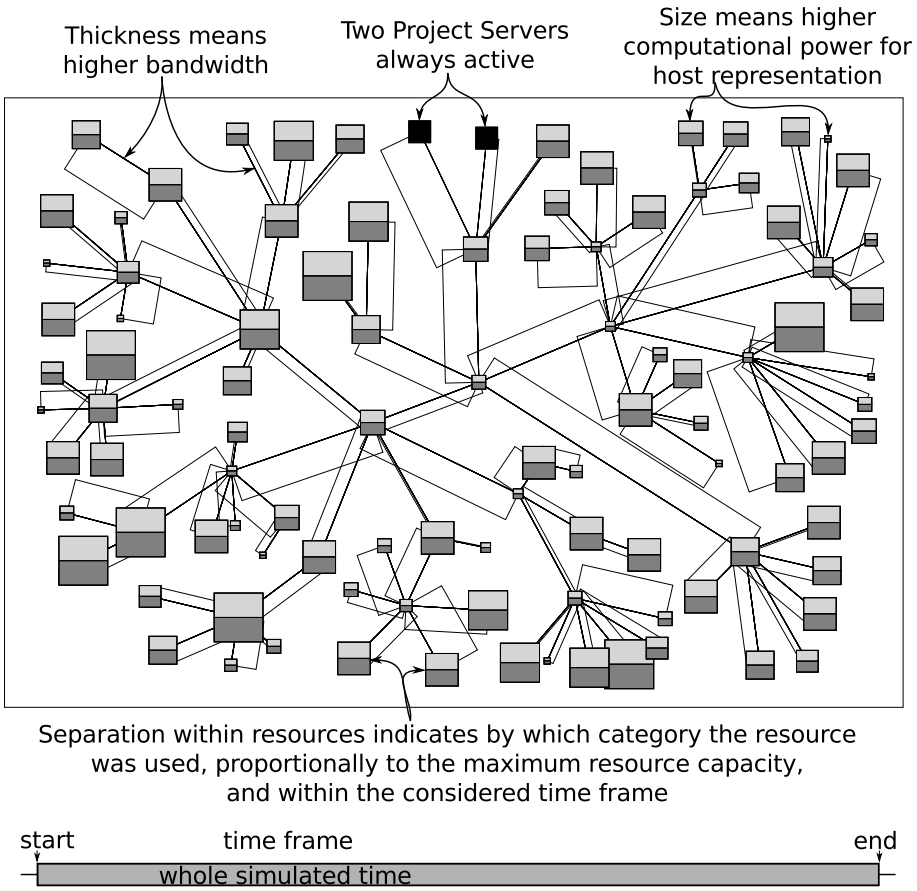


Figure 10: Full resource utilization when BOINC task distribution uses small files as input for volunteer clients. No client exhibit idle time whereas network links are mostly empty.

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 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 10 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 11 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.

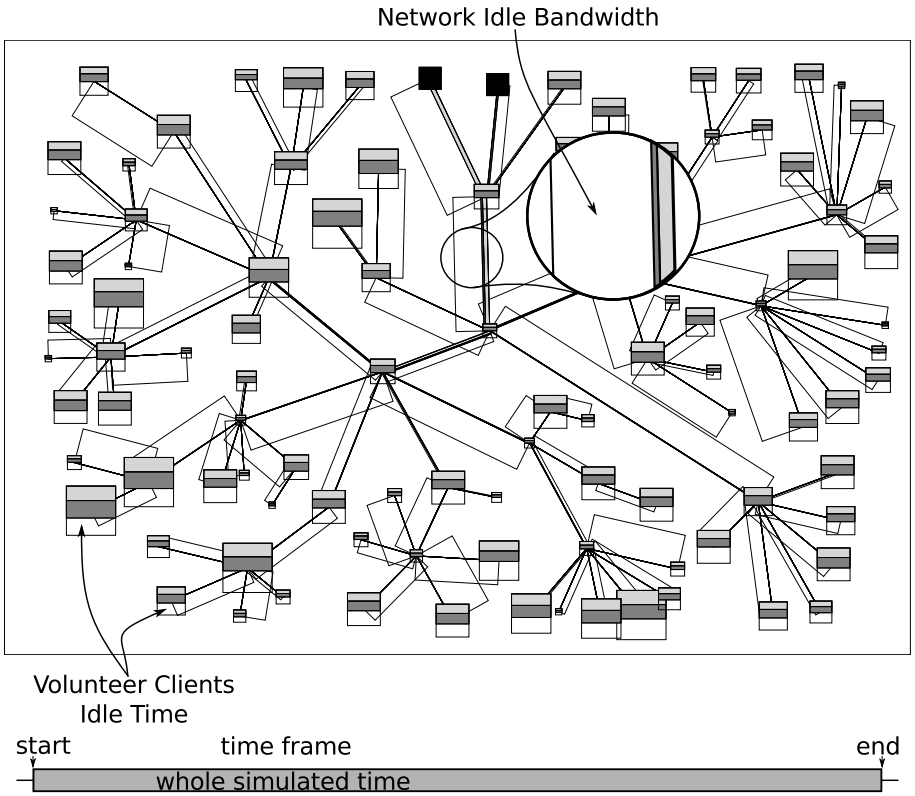


Figure 11: **Resource waste** when BOINC task distribution uses large files as input for volunteer clients. All clients exhibit a large idle time. Surprisingly, network interconnection links around project servers do not appear as bottlenecks and have a rather low usage.

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 12 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 communications. 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.

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 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.

5 Related work

We classified related research in two areas: monitoring systems and visualization techniques. The former details tools and techniques that collect resource utilization data and if they characterize that information. The later presents some visualization techniques that can be used to analyze application traces together with resource utilization, and show how our approach to visualize traces differ from existing solutions.

Monitoring Systems

Most tools for monitoring systems are focused on observing only the resource utilization. That is the case for Ganglia [20], the Network Weather Service [33], Monalisa [23] and many others [35]. They use different distributed collection mechanisms to gather periodic resource utilization from a possibly larger number of resources. The information is generally composed by how much a given resource was used during a time period, without the characterization of the collected data. This means that no associated information is registered about what parallel application or system software caused that utilization. As explained in Section 2, our approach differs because the collected data about resource utilization is characterized, linked to user-defined categories from the application-level. Other tools like Paraver [25], CoSMoS [31] and an approach using Java [24] provide multi-level collection mechanisms to gather data from the different abstraction layers of a computational system. This approach differs from ours mainly because the probes collect monitoring data independently, without correlating during the observation phase the collection with application-level categories.

Visualization Techniques

Many tools and techniques are used to visualize performance monitoring data. Most of them use timeline views derived from Gantt-charts [32], where the behavior evolution of application or system components is depicted. Some examples that provide this type of visualization are Vampir [22], ParaProf [3], Projections [19], among others [34, 25, 13]. The first drawback of using gantt-based charts is related to visualization scalability, where the vertical resolution for observed entities and the use of one screen dimension to represent time limit the amount of data that can be visualized at the same time. Such limitations force the use of other techniques to either filter out the data, or merge it before the visualization. The second issue with timeline-based views is the lack of a topological information in the representation. Since the technique uses one of the dimensions to depict time, a linear representation of an interconnection in the remaining dimension may become hard to understand.

We get around the issues of a timeline-based representation by using temporal aggregation through the definition of time intervals. Such mechanism allows us to use both dimensions of the screen to visually represent the behavior of observed components, without the explicit presence of a timeline in the representation. The visualization scalability is achieved in our approach with treemap representations, adapted to the analysis of characterized resource utilization. On the other hand, our work has a topology-aware representation with customized graphs. Nodes represent hosts and links the interconnection among them. We explore the different visual characteristics of nodes and links to better understand the traces collected in the observation phase. To the best of our knowledge, these representation techniques are not used on other performance analysis tools.

The unique combination of an observation phase that collects characterized resource utilization with a visualization phase that employs novel representation techniques is one of the major contributions of this work.

6 Conclusion

In this paper, we presented an approach to detect resource usage anomalies in large scale distributed systems through the visualization of characterized resource utilization. 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 impact 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 these deadline and replication mechanisms which should have favored them. Thanks to the ability of Triva to represent the state of the system at different spatial and temporal scales, we were able to easily spot this phenomenon and then to identify its origin as the short-term fairness requirements of the client scheduling algorithm.

The third scenario analyzed the impact 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.

The validation of our resource usage anomaly detection approach provided in this paper has shown the benefits brought by the use of characterized resource utilization when combined with a visual analysis of the data. We were able to spot problems on the distributed application scenario that would have been otherwise extremely difficult to detect. As future work, we plan to apply this technique on more classical HPC scenarios and keep on improving the spatial aggregation capabilities of Triva. We also plan to work on the ability to compare two executions and emphasize their differences.

7 Acknowledgment

This work is partially supported by ANR (Agence National de Recherche), project reference ANR 08 SEGI 022 (USS SimGrid).

References

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

[2] David P. Anderson and Gilles Fedak. The computational and storage potential of volunteer computing. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, pages 73–80, Washington, DC, USA, 2006. IEEE Computer Society.

[3] 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.

[4] 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.

[5] 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.

[6] Henri Casanova, Arnaud Legrand, and Martin Quinson. SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In *10th IEEE International Conference on Computer Modeling and Simulation*, March 2008.

[7] Bruno Donassolo, Henri Casanova, Arnaud Legrand, and Pedro Velho. Fast and scalable simulation of volunteer computing systems using simgrid. In *Workshop on Large-Scale System and Application Performance (LSAP)*, 2010.

[8] Markus Geimer, Felix Wolf, Brian J. N. Wylie, Daniel Becker, David Böhme, Wolfgang Frings, Marc-André Hermanns, Bernd Mohr, and Zoltán Szebenyi. Recent developments in the scalasca toolset. In Matthias S. Müller, Michael M. Resch, Wolfgang E. Nagel, and Alexander Schulz, editors, *Tools for High Performance Computing 2009*, 3rd International Workshop on Parallel Tools for High Performance Computing, pages 39–51, Dresden, 2010. ZIH, Springer.

[9] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: portable parallel programming with the message-passing interface*. MIT Press, Cambridge, MA, USA, 1994. ISBN 0-262-57104-8.

[10] 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.

[11] 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.

[12] Rainer Keller, George Bosilca, Graham Fagg, Michael Resch, and Jack J. Dongarra. Implementation and Usage of the PERUSE-Interface in Open MPI. In *Proceedings, 13th European PVM/MPI Users' Group Meeting*, Lecture Notes in Computer Science, Bonn, Germany, September 2006. Springer-Verlag.

[13] Jacques Chassin de Kergommeaux and Benhur de Oliveira Stein. Pajé: An extensible environment for visualizing multi-threaded programs executions. In *Euro-Par '00: Proceedings from the 6th International Euro-Par Conference on Parallel Processing*, pages 133–140, London, UK, 2000. Springer-Verlag.

[14] D. Kondo, A.A. Chien, and H. Casanova. Resource management for rapid application turnaround on enterprise desktop grids. In *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 17. IEEE Computer Society, 2004.

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

[16] 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.

[17] 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.

[18] 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.

[19] Chee Wai Lee, C. Mendes, and L.V. Kale. Towards scalable performance analysis and visualization through data reduction. *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–8, April 2008.

[20] Matthew L. Massie, Brent N. Chun, and David E. Culler. The ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[21] Matthias S. Muller, Andreas Knupfer, Matthias Jurenz, Matthias Lieber, Holger Brunst, Hartmut Mix, and Wolfgang E. Nagel. Developing scalable applications with vampir, vampirserver and vampirtrace. *Parallel Computing: Architectures, Algorithms and Applications*, 38(NIC Series):637–644, 2007.

[22] 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.

[23] H. B. Newman, I. C. Legrand, P. Galvez, R. Voicu, and C. Cirstoiu. Monalisa : A distributed monitoring service architecture. *ECONF*, C0303241:MOET001, 2003.

[24] 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.

[25] 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.

[26] Lucas Mello Schnorr, Guillaume Huard, and Philippe O.A. Navaux. Triva: Interactive 3d visualization for performance analysis of parallel applications. *Future Generation Computer Systems*, 26(3):348 – 358, 2010.

[27] Lucas Mello Schnorr, Guillaume Huard, and Philippe Olivier Alexandre 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*. IEEE Computer Society, 2009.

[28] 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.

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

[30] 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.

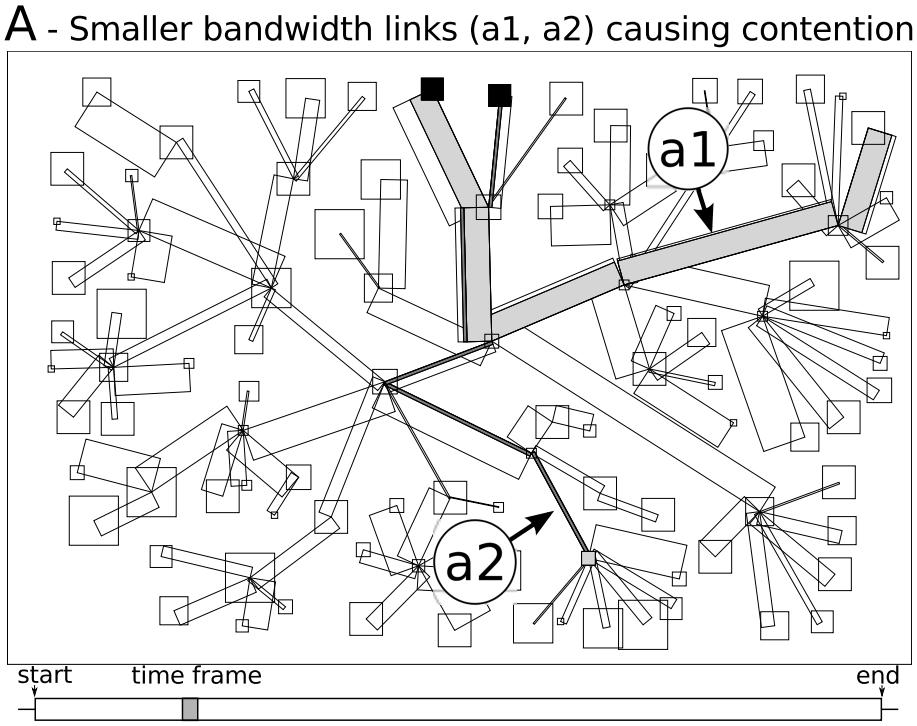
[31] C. Steigner and J. Wilke. Performance tuning of distributed applications with CoSMoS. In *Proceedings of the 21st International Conference on Distributed Computing Systems, ICDCS*, pages 173–180, Los Alamitos, CA, 2001. Los Alamitos: IEEE Computer Society.

[32] James M. Wilson. Gantt charts: A centenary appreciation. *European Journal of Operational Research*, 149(2):430–437, September 2003.

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

[34] Omer Zaki, Ewing Lusk, William Gropp, and Deborah Swider. Toward scalable performance visualization with jumpshot. *Int. J. High Perform. Comput. Appl.*, 13(3):277–288, 1999.

[35] Serafeim Zanikolas and Rizos Sakellariou. A taxonomy of grid monitoring systems. *Future Gener. Comput. Syst.*, 21(1):163–188, 2005.



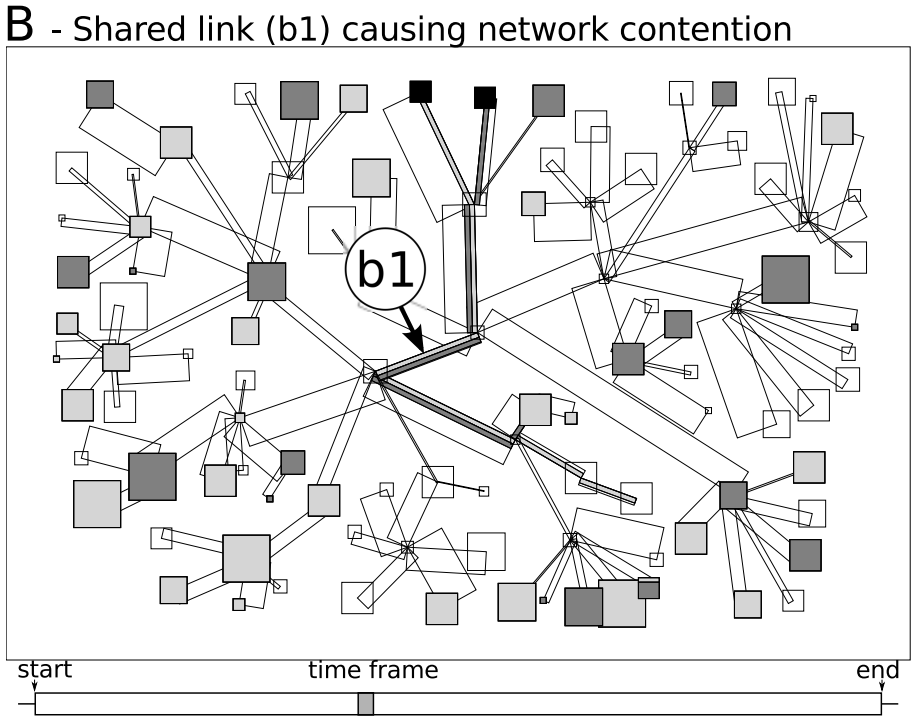


Figure 12: Identification of two types of network contention: screenshot A, caused by smaller bandwidth links (arrows a1 and a2); and B, caused by a shared network link (arrow identified by the marker b1). Therefore, every communication has indeed a bottleneck (often close to poorly connected clients) but these **bottlenecks keep moving** over time, which explains why no bottleneck appears at a large time scale.

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399