Triva: Interactive 3D Visualization for Performance Analysis of Parallel Applications✩

Lucas Mello Schnorr∗a,b, Guillaume Huardb, Philippe O. A. Navauxa

aInstituto de Informática, Federal University of Rio Grande do Sul - Brazil
bINRIA Moais research team, CNRS LIG Laboratory, Grenoble University - France

Abstract
The successful execution of parallel applications in grid infrastructures depends directly on a performance analysis that takes into account the grid characteristics, such as the network topology and resources location. This paper presents Triva, a software analysis tool that implements a novel technique to visualize the behavior of parallel applications. The proposed technique explores 3D graphics to show the application behavior together with a description of the resources, highlighting communication patterns, the network topology and a visual representation of a logical organization of the resources. We have used a real grid infrastructure to execute and trace applications composed of thousands of processes.

Key words: Grid Computing, Parallel Application Analysis, Parallel Program Visualization, Behavioral Visualization

1. Introduction
A Grid [1] is an example of distributed system, having as main characteristics the heterogeneity, the dynamism, and the scalability. Parallel applications that execute in grids must deal with these different characteristics to obtain a good execution performance. Issues that must be faced by developers are different levels of parallelism, high-latency links and variable or

✩Sponsored by grants from CAPES and CNPq.
*Corresponding author

Email addresses: lmschnorr@inf.ufrgs.br (Lucas Mello Schnorr), Guillaume.Huard@imag.fr (Guillaume Huard), navaux@inf.ufrgs.br (Philippe O. A. Navaux)
limited bandwidth in the network interconnection. Most of these characteristics are especially important for parallel applications that perform frequent communications between processes.

The understanding of parallel application’s behavior is an important task when developing for the grid. This task is especially useful to help the developer to adapt the application to the available resources. This tuning usually results in improved execution performance, along with a better use of the resources. To observe the behavior of applications, developers use mechanisms that collect traces during run time and then perform an analysis of the collected data, in an offline or online fashion. Although significant improvements have been made during recent years to obtain a better analysis, the handling of all the trace data is often complex because of the large amount of resources involved, the complex network interconnection and the number of processes of the application under investigation.

Visualization is a way to perform the analysis of parallel applications. The most traditional way of visualizing application behavior is through an adaptation of Gantt charts [2], also known as space-time diagrams. This technique lists the application’s components in one of the two axis and presents the time in the other. Examples of tools that provide this kind of visualization are Pajé [3, 4], Vampir [5] and others [6, 7, 8]. Many of these tools were adapted to observe the behavior of parallel applications that execute on grid systems, but they generally keep rendering the same space-time representation. The problem with this technique when applied to grids is that it is hard to represent the complex network topology of the infrastructure. Since the network can impact the behavior of a parallel application, this type of information should be included in the analysis. Besides, traditional tools usually fail to show the communication pattern of the application in contrast with the network topology, a thing that can be useful if the developer needs to adapt the application to the execution environment.

This paper presents a novel visualization technique for the analysis of parallel applications. Our approach uses three dimensions where two of them are used to render the application’s components, resources and their interconnections, and the third dimension is used to show the objects’ evolution over time [9], including communications. We implemented this approach in the prototype Triva, with a set of three different techniques that render the dimensions reserved to depict application and resource details. These techniques respectively show the communication pattern of parallel applications, the network topology in contrast with the communication pattern, and
a hierarchical representation of resources computed from application traces. Results show that with the three types of configurations created in this work, the 3D technique is better suited to the analysis of parallel applications along with their execution environment characteristics. The technique is especially useful for applications in which processes communicate more frequently, although it can also be used to visualize work balance of loosely-coupled parallel applications.

The next Section surveys related work. The following presents Triva, its visual conception, the component model and the three different configurations developed for it. The implementation decisions of the software prototype used to validate the approach are then presented, showing its architecture and the modules developed. Next Section presents the results obtained with the prototype during the analysis of traces collected in the grid. The last Section concludes the paper and presents future work.

2. Related Work

There is considerable work developed to support the performance analysis of parallel applications. Some of them already make use of three-dimensional rendering. OpenMosix [10], for example, has a tool called 3dmosmon that dynamically shows resource usage using 3D images, generated by a OpenGL-based client monitor. The base is used to place monitored resources and the vertical axis represents the metric currently being analyzed. The vertical size of each visual object corresponds to the value of the metric for that object. GridPP Real Time Monitor [11] is another example of 3D visualization tool that shows resources location distributed over a globe. The resources are placed in the visualization base following their geographical location. The tool also exports data to the KML format, enabling the visualization using the Google Earth tool. Both approaches differ from ours because their 3D view does not use the vertical axis to represent the application evolution over time. Furthermore, they do not enable users to relate application with platform topology. Nevertheless, as we will see in the further sections, the global map in GridPP can also be applied to our model by creating a new configuration technique for the visualization base.

Virtue [12] is another tool that offers a 3D visualization. This tool connects to Autopilot [13] to receive its monitoring data and helps the performance analysis by trying to enhance rendering with human sensory capabilities. It offers three types of 3D visualization: wide-area geographic
display, whose concept is similar to the one present in GridPP, where nodes are placed following their geographic location; time-tunnel display, showing a cylinder where the internal part of the cylinder is used to represent processors state evolution over time and chords illustrate cross-process interactions; and call-graph display, which shows in a 3D space the functions calls of a program. The time-tunnel display is similar to our approach, since it has one axis to represent time. However, the other two dimensions in the time-tunnel are used to place processes in a circular view. Our approach explores these dimensions to show different information including topological data from application and resources.

Paradyn [8] is a visualization tool that has a 3D-Trace/Space/Time diagram. The visualization is able to show processes stack traces of a parallel job to detect odd behaviors. This visualization has three data dimensions: stack trace, space and time but the resources are visualized using 2D graphics.

Some visualization tools have been developed to support performance analysis of parallel application with a possible resource correlations representation, such as Paraver [7], Vampir [5], TAU’s ParaProf [14] and Pajé [4]. All these tools have in common the use of two dimensional graphics to present the monitoring data. Nevertheless, none of them combine in the same visualization the timeline evolution of a parallel application and a two dimensional topological representation of the resources, as we do in our approach.

Some related work addresses the problem of trace analysis by proposing techniques that automatically detect behavior connected to common problems [15, 16]. Meanwhile, an automatic analysis is not always sufficient: when a parallel application exhibits an irregular or an unknown problem, human interaction becomes essential. That is why we are proposing, with Triva, a complementary tool to help the human analysis of traces.

The treemap visualization is already used to observe monitoring data from distributed environments. Coviz [17], for instance, is a Grid visualization tool developed for PlanetLab. It shows different aspects of resources, including their usage, in a web interface. Another example is the visualization of workloads [18], where developers can explore application traces to visualize its overall performance. GridMaps [19] shows the status of different sites in the Worldwide Large Hadron Collider Computing Grid. It uses different colors to represent the status of the resources in the visualization. In these approaches, the generated visualization has a focus on the resources view or the application view. Our approach uses a combination of both to give developers an application visualization enhanced with used resources.
Besides, our 3D view has a time axis in which users can select different time intervals to observe particular parts of the application behavior.

3. Triva: the 3D Visualization

This Section is organized as follows. We start by describing the visual conception of the 3D visualization, detailing its components and how application traces are mapped into the 3D view. Then, we explain the abstract model that is used to deal with the monitoring data and generate the 3D visualizations. The main component of the model, the entity matcher, is detailed in the next Subsection, together with its three configurations developed for the presentation of the idea.

3.1. Visual Conception

Our 3D approach uses three dimensions where two of them are used to organize the application’s components and resources. The third dimension, normally depicted vertically in the visualization, is the timeline. Figure 1 depicts how the approach looks like. The main difference with the space-time approach is the use of an additional dimension to represent the monitored entities. Instead of simply listing vertically the components as the space-time technique, the two dimensions can be used to construct more useful representations of these components. For example, they can be used to illustrate the communication pattern of the parallel application, but also the network topology involved in the execution. We call these two dimensions the visualization base in the remaining of this paper. In our work, we propose three types of configurations for the visualization base (see section 3.3), but additional techniques can easily be adapted.

On the left part of Figure 1, there is an example of the use of 3D approach to represent application traces. The states of the processes are represented in the 3D visualization as vertical bars. They are placed on top of the visualization base. The different states along the time axis of a certain process are represented by different colors. Each state representation is placed vertically according to its start and end timestamps. Communications can be represented as arrows or lines within the 3D environment, connecting two or more processes that communicate. They can be used to represent any type of message-passing communications, including collective operations. Using their position in the timeline, the user is capable to identify the direction of the communication between two processes. In the examples presented in
this paper, we use only lines to represent communications among processes, since the focus here is the 3D representation model. The right part of the same figure shows a different point of view, located on the top of the visual objects. This point of view enables the observation of the communication pattern of the application.

3.2. Model Overview

To create a 3D visualization, the trace data collected from the application execution must pass a series of transformations. We define here an abstract component model where these transformations are detailed. Figure 2 depicts the overall organization of the model. As input, the model uses two types of information: the trace files from the monitored application and a configuration file that holds the resource description of the execution environment used by the parallel application. The first input is handled by the Trace Reader component (box A in Figure 2), responsible for the transformation of traces into abstract objects and high level representations of application components. These representations can be a state of a process during some period of time (blocked, executing some function, and so on). They can also represent a communication and, in this case, contain its source and destination along with the related timestamps. The location data that is registered in each event is also propagated to these objects. Synchronization issues [20] and trace-dependent transformations [21] are all resolved by the component
A. As output, the trace reader component sends the objects to the extractor component (B), through a flow of visual objects ordered by their timestamps.

Figure 2: Abstract Component Model of the 3D approach, with the three different configurations for the visualization base.

The objective of the extractor component is to select from the flow of visual objects the information needed by the entity matcher module. This information is composed by entities of the parallel application, such as processes and threads, and the interactions among them, such as communications. The extractor also redirects to the visualization component (D) all the information it receives from the trace reader.

The visualization base is configured by the entity matcher module (C). In this work, we have explored three different configurations for it: one that shows the communication pattern of the application; another that shows this pattern combined with the network topology of the execution environment and the last one is the combination of application traces with a logical organization of the resources. The entity matcher chooses one of these configurations based on the resource description and the selected objects sent by the extractor component. Then, it sends the appropriate flow of objects to one of the three sub-components of the entity matcher: C.1, C.2 or C.3, as can be seen in Figure 2. The details of how the entity matcher and its sub-components work are described in the next Subsection.

The 3D visualization is created by the component D based on the flow of events that comes directly from component A through the extractor and the base configuration generated by the sub-components of the entity matcher.

3.3. The Entity Matcher

The entity matcher component is responsible for setting up the visualization base of the 3D approach. It performs its task by taking into account

...
the selected objects from the extractor and a resource description. This description is given to the component in one of two possible formats: either as a hierarchical structure describing the logical organization of the computation system, or as a graph describing the network topology of the execution environment. With the application traces and these resources descriptions, we have developed three configurations for the visualization base.

Case 1: Parallel Application’s Communication Pattern

The first configuration for the visualization base of the 3D approach shows the communication pattern of the application. The extractor component (see Figure 2), selects from the flow of visual objects the monitored entities and the communications among them. This selection is represented in the left most part of the Figure 3. The entity matcher acts by merging this information into a graph that represents the communication pattern for the selected objects. The graph creation is dynamic and based solely on the arrival of new monitoring data through the flow of events. This graph can highlight particular performance problems of the application, like bottlenecks or improper communication patterns. For instance, it can help the developer to adapt its application to a particular communication pattern, such as master/slave or divide-and-conquer models. Another advantage is that the application developer can see if some part of the application is overloaded with too many communications in a small period of time, increasing bottleneck effects. The graph is then sent to the visualization component, which draws the graph in the visualization base and the evolution of the application’s components in the vertical axis of the 3D environment.

![Figure 3: Entity matcher configured to generate the communication pattern of the application, based on the amount of processes and the communications.](image-url)

The example of Figure 3 illustrates the generation of the communication pattern. The component has as input 10 processes, from P0 to P9, and a set
of communications among them. As output, we can see a ring-like communication, among the processes from P5 to P9, an all-to-one communication among processes from P0 to P4 and a one-to-one communication between P4 and P5. This communication pattern can change dynamically depending on the selected objects given to the entity matcher component.

Case 2: Network Topology combined with Communication Pattern

The second case for the visualization base is the combination of the network topology and the communication pattern of the application. Figure 4 depicts this situation, where the entity matcher receives as input the network topology (bottom part of the figure) and the application data selected by the extractor. The application process must have in this case location information that defines where they were executed. This is needed because the matcher needs to combine them with the resource description. As output, the component generates two graphs: one that represent the network topology itself, and another that is rendered on top of the first, showing the communications among the processes for the selected objects.

![Figure 4: Entity matcher can receive the network topology as resource description, creating as output the communication pattern over the network interconnection.](image)

Figure 4 shows the same example of Figure 3, but with the network topology description as an additional input for the entity matcher. Each process has a resource associated with it, from R0 to R4. The network topology connecting the resources is on the bottom part of the Figure. The right part of the figure shows a visual representation of the output, composed of network topology representation, with straight lines representing the interconnections, and processes on top of the resources they used during the execution. Communications among processes are represented by the arrows.
with dashed lines. This output is sent to the visualization module to be rendered in the visualization base of the 3D scene. The position of the processes in the visualization base will then be used by the visualization module to render timestamped events in the vertical axis. Through this combination, we are able to understand the application behavior taking into account the network interconnection of the execution system.

The developer can benefit from this configuration in the visualization matching the communication pattern of the application to a specific network interconnection. With this match, the application can better exploit the network, avoid concurrent communications and improve the number of parallel communications that can happen at the same time. Moreover, if the network topology is given with bandwidth and latency information, the developer is enabled to adapt the application so it obtains the highest bandwidth for the processes that communicate more data and the smallest latency for the processes that exchange short messages.

**Case 3: Logical Organization and the Communication Pattern**

The third configuration is a combination of the communication pattern of the application and a logical organization of the resources. The input to the sub-component of the entity matcher in this case is the same as in case 2. But for the resource description, we use a resource hierarchical description instead of using a graph to describe resources. Figure 5 shows the same previous example, but having as input a hierarchical structure where the resources are grouped by their location. In the Figure, the resources R0 to R4 were grouped according to a hypothetical organization by clusters C0 and C1 and then by grid. This structure can be customized in the model to represent other types of organization, such as administrative domains or middleware dependent structures.

There are many ways of graphically representing a hierarchical organization. In this work, we have used the treemap concept [22]. This technique works by using nested rectangles to represent tree-structured data. On the right of Figure 5, we show an example of a treemap created using the hypothetical hierarchical structure given to the entity matcher module. Each rectangle represents a resource and its size is directly related to the number of processes on top of it. The dashed arrows are the communications rendered on top of that treemap and reflect the communication pattern of the application. This output is sent to the visualization component, which is responsible for drawing in the visualization base of the 3D scene the treemap.
created by the entity matcher. An important characteristic of this configuration is that the entity matcher can be adapted to configure the treemap using other characteristics from the traces, such as the number of communications, the time spent by the monitored application executing a certain function, and so on.

The visualizations obtained with this technique in the visualization base can highlight important parts of the application in contrast with the resources. For example, it can be used to see resource usage and the load balancing of the application by configuring the treemap to show the functions that do the processing part of the application. The same situation can be applied in order to observe which processes communicate more or stay blocked more time due to message-passing.

4. Triva Implementation

We implemented the component model presented in the previous Section inside a prototype called Triva – ThRee dimensional and Interactive Visualization Analysis. The configurations for the visualization base, presented in section 3.3, were implemented as well. The main objective of this implementation is to validate our model and to enable the analysis of traces from parallel applications executed on the Grid using the 3D representation. Figure 6 depicts the overall organization of the prototype, composed of modules that transform the trace data into visual objects. The only trace-dependent part of the prototype is the one represented on the left of the Figure, it is handled by the DIMVisual Integrator. We first describe the two types of input the prototype can receive: the trace files and the resource description.
Then, we describe the implementation of each module.

Figure 6: Triva Implementation Layout

4.1. Input Description

The first type of input, represented in Figure 6, is a set of files with events that were registered during the execution of the monitored parallel application. In our implementation, we use traces generated by the KAAPI library [23] and by an instrumented version of the OpenMPI communication library. KAAPI applications are composed of a set of tasks and the dependence among them. During the execution, it uses work stealing techniques to provide load balancing for the application. With the current implementation of KAAPI, we were able to execute applications composed of thousands of processes in the Grid’5000 platform [24]. The KAAPI reader used in our prototype is able to read all the traced activities of the work stealing algorithm, such as the start and end of a steal, and the result of a steal attempt (successful or not). Information indicating the stealer and the victim during a work stealing request are also read from the trace files.

The second type of input is the resources description. For the network topology description we use the dot’s Graphviz format [25]. For the hierarchical structure, we use the property list format [26], either in pure text or in the XML format. No matter which of them is given to the prototype, both are treated by the Base configurator module.

4.2. Modules Description

In Figure 6, the white modules are existing libraries and tools we used in the prototype. The other modules in gray were implemented according to the model and the entity matcher behavior described in the previous Section. On the left of the vertical separation of the figure, we detail the trace-dependent modules. The existing DIMVisual Integrator [21] is a tool to integrate traces
from different data sources. We implemented a module, the KAAPI reader, that is incorporated in DIMVisual. As output, the Integrator generates a flow of timestamped events that represents the application behavior. This flow is received by the TrivaPajeReader module, which we implemented following the internal protocol of Pajé [4]. The responsibility of the TrivaPajeReader is to transform the flow of events in textual representations using the Pajé file format [4]. These representations are sent to the existing Pajé filters that create the visual objects that will be used in the subsequent modules.

The Pajé filters are the same as those used by the Pajé Visualization Tool [3]. Their implementation takes into account several issues like scalability and response time regarding requests made from the user interface. The first of the filters, PajeEventDecoder, handles the input generated by the TrivaPajeReader and prepares it for the next module. The PajeSimulator is responsible for the transformation of the events into visual objects. This transformation consists in the creation of a hierarchical structure of traces, using the basic types of Pajé. This structure, which represents the same information from the trace files, is optimized for the visualization, and stored in the StorageController.

Until now, the flow of events has been transferred from the trace files to the Triva prototype. This flow stops in the StorageController, where the internal representation of the traces is stored in memory. In the right most part of Figure 6, the interactions among the modules work in a two-way fashion. The interactions from right to left are the requests for new data. They are mostly triggered by user commands or changes in the configurations given as resource description. The interactions from left to right are the responses for the requests generated by the visualization.

The TrivaView module represents the implementation of the Extractor component from the model represented in Figure 2. Every time the StorageController answers a request for new data coming from the DrawManager, it intercepts the response and select the links and the representations for the monitored entities, such as processes and threads. The resulting selection is sent to the BaseConfigurator module, which implements the entity matcher from our 3D model. The BaseConfigurator acts depending on this selection and the resource description given as input. If a tree is given to this module, it uses our implementation of the squarified treemap algorithm [27] to create the visualization base. Otherwise, it combines different algorithms provided by the Graphviz library to generate a data structure of the graph. The last module of the prototype is the DrawManager, responsible for creating the
3D visualization. Our implementation of this module uses the Ogre3D library [28] to render the 3D scene and the visualization base configuration generated by the BaseConfigurator.

5. Results

The results are presented through screenshots of our prototype Triva. To obtain these screenshots, we executed a Fibonacci parallel application on the Grid’5000 platform to collect trace files of different sizes in terms of data and number of processes involved. This type of application is representative experience for algorithms that are decomposed recursively, such as the ones used in combinatorial optimization. The traces from these executions are sent to the Triva prototype, to be visualized in the 3D scene.

The Section is organized as follows. We start by a general description of Grid’5000, including the resources and network interconnections involved. Then, we show the first results obtained with Triva to visualize an application communication pattern. The mapping of this pattern on top of the visual representation of the network topology is presented in the next Subsection. The Section ends with the results obtained with the squarified treemap in the visualization base.

5.1. Platform Description: Grid’5000

The Grid5000 infrastructure [24] is a grid composed of clusters that offers developers an execution environment of 5000 processors/cores. The clusters are geographically located in France, but spread over 9 different sites. Optical fiber links are used to connect the sites, and different technologies are used to connect intra-site computers. The infrastructure is known as a lightweight grid, with limited heterogeneity, but strong hierarchical interconnection. The limited heterogeneity appears because the resources of this grid are from different homogeneous clusters. The strongly hierarchical interconnection is present because of the number of hops involved to interconnect distinct clusters. In this grid, all machines can connect to all others directly, without the need of software routers.

In terms of software, most of Grid5000 machines runs a traditional Linux operating system flavor installed as default, along with a set of basic tools to perform the execution. We installed the KAAPI library configured to trace the execution of applications developed with it. The installation has been made in the user directory of a dedicated account to perform the experiments.
5.2. Application Communication Pattern

The first configuration for the visualization base is the communication pattern of parallel applications. We used two parallel applications to generate different patterns to be visualized. The first application is a basic MPI application used to compare our approach to Gantt charts. The MPI application is composed of five processes that exchange messages repeating a round-robin pattern 10 times. The second application is a Fibonacci solver developed using the KAAPI library [23]. Both applications have been traced with instrumented versions of the communication libraries. In the specific case of the Fibonacci application, the traces have been collected using built-in trace levels offered by the KAAPI library. These tracing facilities are very convenient to extract more or less detailed views from any KAAPI application execution.

Finally, for the sake of demonstrating our prototype capabilities on other topologies, we automatically generated peculiar synthetic traces. These traces provide an overview of the behavior of our prototype when tracing an application specifically designed for a lightweight grid.

The first experiment compares Gantt Chart visualization with the three-dimensional approach. The application considered is a five processes MPI application that when executed exchange messages in a round-robin fashion, repeating the pattern 10 times. The tracing was performed through an instrumented version of the MPI library that is capable of registering events. Figure 7(a) shows the visualization of the program behavior using a Gantt Chart generated by the Pajé visualization tool. Communications are represented by arrows and the different rectangles with various gray tones show the different states of each process.

Using our 3D prototype, we first visualized the same traces using a linear layout. As shown in Figure 7(b), the traditional 2D view is obviously a subset of the visualizations we can obtain with the 3D approach. In the Figure, we have plunged the 2D view into a 3D space. As in the 2D case, each line interconnecting two bars in the 3D view represents a combination of MPI_Send and MPI_Recv.

A significant advantage of the 3D approach is related to the visualization of communications: they can be represented without crossing links and their topology is thus outlined. Using traditional 2D visualization systems, communications, usually represented as arrows, often cross over other visual elements. For instance in Figure 7(a), due to the ring constituted by the communications made by the application, some arrows cross the state line of all
Figure 7: Mapping Gantt-charts to the 3D visualization.

When dealing with a large amount of data, the visual perception of such communication patterns can become tedious. This is usually not the case when using the 3D approach. Figure 7(c) shows the same application’s trace along with a proper choice of elements layout and camera position. In this Figure, no communication link cross over another visual element, and the communication pattern appears clearly.

Figure 8: Different communication patterns visualized with Triva.

Figure 8(a) shows the visualization of a synthetic trace. The example
has 20 nodes, divided among 4 sites. The lines connecting the processes represent successful work stealing requests (work stealing and the associated traces is provided by the library). We can notice that the application passes through an inter-site work stealing phase at the beginning and at the end of its execution. In the middle of the execution, the work stealing remains local (which is rather good). This example shows that the visualization can help in the process of refining work stealing algorithms.

The parallel application analysis can be increasingly more complex when communications follow more elaborated patterns. Some parallel applications use different communication algorithms in different levels of the parallel architecture (shared memory, inter-process communication, TCP communications, ...). This happens often in Grid architectures when applications use shared memory within computers and message-passing communication between two or more administrative domains of the Grid. Figure 8(b) shows a simple synthetic parallel application composed of 12 processes being executed in three different administrative domains. The processes are grouped geographically according to their domain and communications between domains pass through a front-end node. A similar situation is presented in Figures 9(a) and 9(b), but with different communication patterns. Figure 9(a) is a visualization of a parallel application involving 50 processes, divided equally among 5 different sites. The visualization clearly shows the ring communication pattern inside each site and a star communication among processes that work as front-ends. The visualization can help, in this case, to detect possible bottlenecks caused by a specific communication pattern.

Our rendering model has been designed to render interactive traces: evolution of the application can be monitored in real time. In its current state, our prototype, only handles post mortem traces. Nevertheless, the rendering process iterates through time (at selectable pace) and it is possible to pause the events processing to analyze the application state at some peculiar point of the execution. The user can also select a time slice and the prototype reconstructs the visualization eliminating irrelevant objects.

Figure 9(b) shows a larger scale master-slave parallel application, composed of 600 nodes distributed equally over 6 sites. In this application, there are two levels of communication: the first level relates to communications inside each site where there is a process that acts as an application’s front-end for that site; the second level relates to front-ends communications between the application’s master processes. The 3D visualization provides a clear view of these two levels and enables the developer to identify communica-
tion or load unbalances. This problem detection is eased by the combined rendering of processes topology and their state evolution through time.

5.3. Network Topology versus Application Communication Pattern

Figure 10 shows two views of the execution of a KAAPI parallel application, composed by 110 processes running on three sites of Grid5000. There is one machine per process, with one of the three sites with 50 machines and the other two with 30. Each process is represented by one vertical bar and lines represent the communications among them. The bars are placed according to the location where the process has been executed. The graph in the visualization base reflects the sites interconnection. The use of different algorithms to define the position of the resources in the visualization base together with camera moves in the 3D space provides a rich set of various views to analyze the monitoring data. With them, the user can see the mapping between the set of processes belonging to the application and the resources reserved for them. This mapping changes every time a new application is under investigation.

Another example of analysis of applications communications with the network interconnection is depicted in Figure 11. The application here is composed by 660 processes running on 5 clusters of Grid’5000. Clusters are represented in the visualization base using rectangles, with their sizes directly related to the total number of machines in each cluster. The interconnection is represented by the thicker lines in the base, while the other links represent
application communications. The developer can benefit from this view by matching the communication pattern of the application to the specific network interconnection dedicated to the application execution. Thicker lines can also be used to represent different bandwidth capabilities of the network links, enabling the developer to understand the behavior of groups of processes that communicate more frequently. The two images differ by their observer position and their time scale extent.

The top image of figure 11 also presents, through the dashed circles and line, the identification of a possible bottleneck in the application execution. The dashed circle depicts in the visualization the communications between clusters B and D. The dashed line indicates that all these communications pass through the two routers and the links that interconnect these clusters. This type of visualization helps the identification of possible bottlenecks, since we are also able to notice that the communications between B and C share at least one link with communications from clusters B and D. The developer is able, through the 3D visualization, to detect a better placement of processes among the available clusters, in order to minimize such bottlenecks and improve the application performance.

5.4. Resource Treemap customized by Application Monitoring Data

The third configuration for the visualization base is the use of a squarified treemap, as can be seen in Figure 12. The images of this figure show the same 110 processes application from Figure 10, but using the squarified treemap visualization implemented in the entity matcher module. The image on the left shows a broader view of the application during the start of its execution.
Figure 11: Two views of an application with 660 processes, having the visualization base rendered with the resources graph.

The image on the right shows a different angle of the same visualization. Each rectangle in the visualization base represents a machine allocated to the execution of the application, with its size indicating the number of work
stealing events registered by the process running on top of that machine. The view shows that some processes receive more steal requests than others (which is natural as work data is initially contained in a single process). By using this type of visualization base together with a time selection, the user can thus see load balancing issues in specific time intervals. Moreover, the prototype is configurable: the user can choose what kind of information will be used to plot this summary in the visualization base. This results in multiple views that help the analyst to identify behavior patterns.

Figure 13 shows the scalability of the approach, through a representation of 2900 processes on the same screen, rendered in two views: one on top of the resource graph (on the left) and the other on top of a treemap based on the number of work stealing requests. In the resource graph base, we can see the 13 participating clusters, interconnected by 4 routers in the middle of the scene.

Figure 12: Different views of the same application of Figure 10, but with a base configured to hold the treemap with the work stealing controlling the size of each rectangle.

6. Conclusion

A successful analysis of parallel applications behavior must take into account information from the execution environment. Since this operation is usually complex, a common way to tackle this problem is to visualize traces from parallel application executions. This paper presented a novel 3D-based visualization technique for the analysis of parallel applications. Our approach can be used to merge resource interconnection characteristics into the application analysis through visualization. The paper presented the visual conception of the 3D approach, the model overview and a detailed description of
the entity matcher. The three possible configurations developed in this work to validate the approach are the visualization base showing the communication pattern of parallel applications, the resource interconnection versus the application communication pattern and the resources visualization through squarified treemaps with the applications components on top of it.

One of the main findings of the approach is the importance of the correlation between network topology and application traces. This can be noticed through the analysis of different scenarios using real KAAPI applications executed in large-scale allocations of Grid’5000. The Triva implementation have been used for these analysis with different representations for the resources, such as a graph or a hierarchy. We have shown that their use enable a better understanding of the application behavior, because they enable to visualize several issues related to parallel application execution, such as load unbalance of the application or the lack of locality in communications. Furthermore, the treemap visualization for the base gives a simple way to present application traces summaries taking advantage of its customization capabilities. Graph representations of the resources were used to analyze the parallel applications in contrast with the network topology. The results show the benefits of our approach by giving to the developer a synthetic view of which communications channels are used by the application during its execution.

As future work, we aim at enhancing user interaction mechanisms of our Triva prototype, especially the ones related to the 3D visualization. We intend also to develop visual data reduction mechanisms to make sense out of very large data set visualizations.
References


URL http://threedmosmon.sourceforge.net/

URL http://gridportal.hep.ph.ic.ac.uk/rtm/


