Towards Visualization Scalability through Time Intervals and Hierarchical Organization of Monitoring Data

Lucas Mello Schnorr†‡, Guillaume Huard‡, Philippe Olivier Alexandre Navaux†

†Instituto de Informática - UFRGS
91501-970 - Porto Alegre - Brazil
{lmschnorr,navaux}@inf.ufrgs.br

‡LIG - MOAIS INRIA Project
38000 - Grenoble - France
Guillaume.Huard@imag.fr

Abstract

Highly distributed systems such as Grids are used today to the execution of large-scale parallel applications. The behavior analysis of these applications is not trivial. The complexity appears because of the event correlation among processes, external influences like time-sharing mechanisms and saturation of network links, and also the amount of data that registers the application behavior. Almost all visualization tools to analysis of parallel applications offer a space-time representation of the application behavior. This traditional way of showing timestamp-based information presents the monitoring data with a gantt-chart based representation. This paper presents a novel technique that combine traces from grid applications with a treemap visualization of the data. With this combination, we dynamically create an annotated hierarchical structure that represents the application behavior for the selected time interval. The experiments in the grid show that we can readily use our technique to the analysis of large-scale parallel applications with thousands of processes.

1. Introduction

Grids are often divided into virtual organizations [8] and can be composed by thousands of resources and different network interconnections. The software to manage the different components of the grid must also scale to keep tasks responsiveness to user demands. Together with the number of resources, the size of parallel applications executed over a Grid can also increase. An example of that is the number of processes that belong to the same application. It is not difficult to find applications that can be composed by thousands processes, each one with a number of threads to explore local parallelism.

The behavior analysis of these applications is not trivial. The complexity appears because of the event correlation among processes, external influences like time-sharing mechanisms and saturation of network links, and also the amount of data that registers the application behavior. The reason of the enormous quantity of data can be divided basically in two main factors: the number of monitored entities, such as processes and threads, and how much data is collected to represent the behavior of these entities. A good analysis of this data influences directly the understanding of the application behavior. The perception of global and local patterns is only possible if the developer can analyze, at the same time, the state of all entities.

Besides statistical drawings, almost all visualization tools to analysis of parallel applications offer a space-time representation of the application behavior. This traditional way of showing timestamp-based information presents the monitoring data in two dimensions: the horizontal axis is used to represent time and the other axis is to list processes, threads, and other components of the application. The application analysis through this type of visualization might be negatively influenced when the application is composed by hundreds or even thousands of processes. Another problem is that is hard to compare the behavior of the application’s processes, because there is no way to summarize the events and states that occurred during application execution.

This paper presents a novel technique that can be used to show the behavior of large-scale grid applications. Our approach uses time intervals to dynamically create an annotated hierarchical structure that represents the application behavior for that period of time. We then use the treemap technique [10] to create a visual representation of that structure. The resulting visualization increases the number of monitored entities that can be visualization in the same time, and allow a direct comparison among their behavior. The time slices can be of any amount of time, from microseconds to days, helping the user to see patterns that emerge from the application behavior in different time scales. By moving the time slice over the amount of time the application take to execute, we can observe
the evolution of every process’ behavior by relating it to some aspect of the application, such as a blocking operation or processing.

The rest of the paper is organized as follows. In the next section, we discuss related work, followed by the presentation of our approach in section 3. Implementation details of our prototype are given in section 4. Section 5 presents the results and findings of applying the time-slice technique in the analysis of grid applications. Conclusions and future work are given in section 6.

2. Related Work

The visual scalability [7] defines the capability of a visualization technique to show large amounts of data. Some tools attack this problem by working in the monitoring data, rather than improving the visualization itself. Reduction and selection mechanisms are present in tools such as Pajé [5] and ParaProf [12], decreasing the amount of data to be visualized. Vampir [13] addresses scalability through the use of hierarchical data organization and others techniques [2]. Vampir also shows in its space-time window a general view of all applications components. The developer must then choose which part of this view will be detailed in the rest of the screen space. Jumpshot [19] uses a scalable file format that is used to simplify the visualization of large-scale applications, but it offers no improvements in the way the data is visualized. Paraver [15], Projections [11] and Pablo [16] are tools that were developed to address scalability issues in the visualization and analysis of parallel programs. A common characteristic of all these tools is the use of space-time visualizations to represent the behavior of the applications through time. Even if aggregation and filtering techniques are used, the amount of information that can be visualized in the same time is limited. Our approach avoids this problem by offering an alternative technique to visualize parallel program behavior. Instead of showing states, events and interactions of processes through time as usual tools do with space-time representations, we apply an information visualization technique called treemap to visualize monitoring data. The resulting visualizations allows a different view of the traces, enabling the user to visually compare the behavior of all process through local and global patterns, in different time scales.

3. The Scalable Hierarchical Visualization

This section explains our approach to generate a scalable visualization of the parallel application’s behavior. We first discuss how the monitoring data from distributed systems and applications can be hierarchically organized. We then detail the treemap technique, which is basically a space-filling algorithm to draw hierarchical structures. We end the section by presenting our time-slice algorithm, which creates a hierarchical structure that details the behavior of an application within an interval of time.

3.1. Hierarchical Monitoring Data

Traditional monitoring systems for distributed environments periodically gather data about the behavior of a pre-defined set of entities. This set can contain either resources of the computing system, such as processors and memory, but also entities from a parallel application, like processes and threads. This set of entities can be organized as a hierarchy, where the bottom-level nodes are these entities and the intermediary nodes are used to group the entities based on a logical or location characteristic of the entities.

Figure 1(a) depicts an example where the entities processes and threads are hierarchically organized. The threads are grouped by the process they belong, which is then grouped by the machine where the processes execute, then by cluster and so on. Additional information, such as the CPU utilization can also be present in the tree in order to complement the behavior analysis of the application. In a general way, the nodes present in the figure 1(a) are the types of the entities monitored. When a monitoring system collects real information, it creates instances of these types. Figure 1(a) shows a possible instantiation of the hierarchical organization, where the application is composed by \( n \) processes, grouped by a set of machines, clusters and finally grid.

![Figure 1. Hierarchical organization of monitoring data from a grid application and the corresponding instances of the types.](image)

The types of this hierarchical structure are related to any kind of entity that can be monitored. If, for example, we are monitoring an object-oriented application, the resulting collected data would be composed
by traces from the objects that were instantiated and the methods executed. Further information on this data can be the packages that holds the classes. The resulting hierarchical organization would be a tree having as root a type Package, with a single child of type Class with a child of type Method.

The notion of type hierarchy was implemented and validated in the visualization tool Pajé [6]. Its format is considered generic since it can be adapted to represent virtually any kind of monitoring data. It was applied to the visualization of Java Applications [4], MPI applications and multi-level analysis of parallel applications executed in a cluster [17]. The generic capability of Pajé appears because it uses a hierarchical definition of the data, being able to adapt itself to a broad range of monitoring systems, from the ones focused in the analysis of resources to systems used to trace parallel applications.

Next section presents how the treemap technique can be used to visually represent hierarchical data.

3.2. Treemaps Basic Concepts

The traditional way of displaying hierarchical data is through node-link diagrams [14]. As can be seen in figure 2(a), these diagrams are easy to understand by explicitly showing the relation among the nodes. The problem with this approach appears when we try to visualize large scale trees with thousands of nodes. This happens mostly because they do not explore well the screen space [18].

The treemap technique was proposed in order to solve the scalability problem of hierarchical representations [18]. Instead of drawing nodes and links between them, it uses the whole screen space with a space-filling algorithm. This algorithm recursively divide the screen space following the tree organization. Figure 2(b) shows an example of treemap created based on the the same hierarchical structure depicted in figure 2(a). To this treemap drawing, we consider that each leaf node has a value of 1, so their sizes are the same in the final figure. The parent nodes A, B and C have their values defined based on their children: 6, 3 and 2, respectively. The second level of the tree is highlighted in the treemap by the thicker lines in figure 2(b).

The treemap algorithm passed through several evolutions since its creation. One of them is the Squarified Treemaps [1]. It manages to keep the rectangles shapes as close as possible to squares, facilitating the visualization of the information. Next subsection presents how we find the values that describe the behavior of a parallel application in a given time interval and how we create the visualization of that structure using squarified treemaps.

3.3. The Time-Slice Algorithm

The hierarchical organization of the application may have various levels where the leaves of the tree are the monitored entities that the user is interested in. Usually, they have no quantitative information associated with. The idea is to attribute a value for each of the entities in a way that this information represents its behavior for the selected time slice. After this definition for all entities, we use the treemap algorithm to draw the visualization. We find these values based on two things: the definition of a time interval and a summary of the events of each monitored entity on that time interval.

Different configurations to define the time interval are possible. Its length can be changed dynamically in order to find visual patterns from the data being analyzed. This allows the detection of patterns that might appear in a small slice of time but not in bigger ones. Besides that, the user can move the slice of time being analyzed, allowing the observation of the evolution of every aspect of the entities through time. Any change in the time interval triggers the recalculation of all the values in the hierarchical structure that represents the behavior of the monitored entities.

The summary of events is done by taking into account the interval of time specified and an aspect of the data to be analyzed. This aspect can be the amount of time each entity stayed in a certain state within the interval of time being analyzed, but also the number of singular events of a given type in that time slice. In either case, the objective is to find a numerical value that represents the behavior of each entity. With this, we define a value for all the monitored entities in the selected time slice. The treemap algorithm is then used
to draw the rectangles based on these values. Bigger the values of an entity in the structure, bigger will be the area dedicated to that entity in the visualization.

Figure 3 shows an example with five monitored entities, from A to E, grouped by their location in the computational system, represented by the rectangles denoted by M1 and M2 in the left most part of the figure. Entities A and B are from the one location, and entities C, D and E are from another. The aspect we are analyzing in this example is the one represented by the gray rectangles, which can be a blocking operation or task processing. It represents in a general way a certain state where the entities spent some time during their execution. Every entity’s state has its beginning time represented by $X_t$ and end time by $X_{tf}$, where $X$ is one of the entities. The two vertical lines - $T_i$ and $T_f$ - represent together the interval of time we defined to calculate the values for each of the entities. The arrows in the figure represent interactions among the entities. Each arrow has two singular events: its start and end.

Consider the case shown in figure 3 and the summary of events that takes into account the amount of time each entity spent on the state represented by the gray rectangles. The objective is to define a value for each entity based on the time slice of figure 3 and the amount of time spent on that state. The maximum value for each entity is $T_{tf} - T_i$. Since entity A has the state that we are analyzing with an interval of time greater than the defined time slice, entity A will receive the maximum value. Entity B will receive a value defined by $\frac{B_{tf} - T_i}{T_f - T_i}$, because its beginning is before the beginning of the time slice we are analyzing. Entity C is similar, except that is the end of the state that finishes after the end of time slice: $\frac{C_{tf} - C_t}{T_f - T_i}$. Entity D has the state all within the limits the time slice, so its value is defined by $\frac{Dt_f - D_t}{T_f - T_i}$. If the same state appears more than once in the time axis, their amounts of time must be taken into account. If the state does not appears within the time slice, as the case of entity E, the value defined is 0. Considering all these situations, the final equation to calculate each entity’s value is:

$$X_{val} = \min[T_{tf}X_{tf} - \max T_iX_i]
\frac{T_f - T_i}{T_f - T_i}$$

where $X_{val}$ is the final entity’s value, $X_i$, $X_{tf}$ are the timestamp for begin and end of the state we are analyzing and $T_i$, $T_f$ are the time slice boundaries. If $X_{val}$ is negative, it means there is no state on that period of time, so its value is set to 0.

Table 1 list hypothetical values to the previous example’s variables. Considering these values, we calculate each entity’s final value taking into account the amount of time of the selected state inside the specified time slice and end up with the annotated hierarchical structure of figure 4(a). The values for the nodes on top of the leaves are defined by taking into account the values of their leaves. The hierarchy represented in this figure show the value for the nodes $M1$, $M2$ and $Root$. This structure is then used by the treemap algorithm in order to create the visual representation.

**Table 1. Hypothetical values to an example of treemap creation based on the state’s amount of time.**

<table>
<thead>
<tr>
<th>Variables</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_i - T_{tf}$</td>
<td>5.0 - 10.0</td>
</tr>
<tr>
<td>$At_i - At_{tf}$</td>
<td>4.5 - 10.5</td>
</tr>
<tr>
<td>$Bt_i - Bt_{tf}$</td>
<td>4.0 - 6.0</td>
</tr>
<tr>
<td>$Ct_i - Ct_{tf}$</td>
<td>7.5 - 10.4</td>
</tr>
<tr>
<td>$Dt_i - Dt_{tf}$</td>
<td>6.5 - 7.7</td>
</tr>
<tr>
<td>$Et_i - Et_{tf}$</td>
<td>10.3 - 12</td>
</tr>
</tbody>
</table>

Figure 4(b) shows the hierarchical organization of the same example of figure 3, but here considering the amount of singular events to calculate the value of each entity. The way we do this is similar to the previous explanation, but the time of the events is not take into account. Suppose that arrows indicate a communication between two entities and the end of those arrows are data receptions. If this reception is the
singular event we are analyzing, we count the number of them and define the value for each entity. The leaf nodes of the tree in figure 4(b) show the resulting values, and also to the other nodes of the organization.

![Figure 4](image1.png)

(a) Values defined based on the amount of time.

(b) Values defined based on singular events.

Figure 4. Two situations to calculate the values for the hierarchical organization of monitoring data.

The hierarchical structures of figure 4 are then sent to the treemap algorithm. Its drawing procedures take into account the values for each of the nodes in order to generate the max utilization of the screen space dedicated to represent the structure. The results of these drawings are depicted in figures 5(a) and 5(b). The former shows the treemap created with the hierarchical structure that holds values based on the amount of time. The later shows the treemap created with the number of singular events.

![Figure 5](image2.png)

(a) Treemap sketch created using the amount of time

(b) Treemap sketch created with number of singular events

Figure 5. Treemaps of hierarchies from figure 4.

The time slice technique presents data in a more comparable way. This means that the user can visually and almost instantaneously compare the size of every rectangle. In our previous examples, we show that the rectangle size can be generated in a way that is directly related to the amount of time that each entity stayed in a certain state. By analysing figure 5(a), we can easily see which process spent more time on that state. If this representation is used to analyze parallel application behavior and the state is a blocking operation, the visualization will show which processes spent more time blocked than actually executing. Other types of states and events from the application can be taken into account and combined in the same visualization.

Another possibility is to draw the treemap using only values available up to a certain level of the hierarchy. For example, by removing the leaf nodes of the hierarchy shown at figure 4(a), the drawings would have an aggregated view where the user will be able to analyze with less details, also improving the understanding. This is more easily explained if the M1 and M2 nodes are clusters, and the entities are machines. Instead of having a detailed per machine analysis, the user has direct access to a comparison of the state being analyzed in the cluster level.

4. Triva Prototype

A prototype called Triva was developed in order to evaluate the advantages of the time-slice algorithm to visualize large-scale parallel application’s behavior. Figure 6 depicts its architecture, composed by five modules: DIMVisual Reader, Pajé Simulator and Storage Controller to deal with trace peculiarities; and Time Slice Algorithm and Triva Display to implement our approach and to interact with the user.

![Figure 6](image3.png)

Figure 6. The components of the prototype together with flow of events and user interface commands.

The trace files are read by DIMVisual [17], which can be adapted to read traces in different formats. In our experiments, the DIMVisual was adapted to read trace files from parallel applications written using the KAAPI library [9] and executed in the Grid5000 [3]. As output, the DIMVisual module generates events in the Pajé format. Since the Pajé format is generic [6], this makes the prototype independent of the specific trace format generated by the parallel application (the DIMVisual module is the only trace-dependent part). The Pajé Simulator and Storage Controller are responsible for transforming the flow of events in high-level objects, such as containers, states, singular events. These objects are internally stored as a hierarchical structure, facilitating the implementation of our algorithm in the Time Slice component.

Our technique is implemented in the Time Slice component of figure 6. It is configured by the Triva Display with the time interval to take into account in the algorithm, but also which aspect of the application
should be analyzed. The component requests times-
tamped trace data to the Storage Controller, and the
response is used to attribute values to the hierarchical
structure given by the storage. The component also
calculates the treemap based on the window size of the
application. The Time Slice component generates as
output a hierarchical structure with treemap attributes,
ready to be rendered by the Triva Display.

The Triva Display receives from the Time Slice
component the treemap structure and draw it inside
the application window. New treemap structures can be
requested based on user interaction, such as zooming
in and out in a particular part of the tree, but also
when new trace data is available. Other interaction
techniques are also implemented in the Triva Display,
such as moving the time slice to show the application’s
processes evolution over time, requesting additional
data about each of the processes. Next section presents
our results and findings using this prototype, validating
the time-slice approach.

5. Results

We traced the execution of large-scale coupled appli-
cations in Grid5000. The number of processes involved
range from 200 to 3000 processes, and the number of
used machines goes up to 310 machines. The objective
is to show the visual scalability of the approach, by
showing in the same screenshot the behavior of all the
processes. The hierarchical structure of the monitoring
data is Grid - Cluster - Machine - Process - State.
The processes can be in one of two states: Running
or Rsteal. With this organization, we can see how the
application was partitioned among the clusters and the
number of machines allocated to each of them.

Figure 7(a) shows a visualization of an application
that is composed by 200 processes. They were
executed in 200 machines, one process per machine,
spread in five clusters, denoted by the capital letters
A, B, C, D, E in the figure. Cluster A and B are
from the Grid5000 site nancy, and the others belong
to the site rennes. Each process is represented by a
rectangle, like the one denoted by the letter p. The
amount of time each process stayed in a certain state
is represented by the size of the inner rectangles inside
the process’ rectangle. By offering this global view to
the developer, it is possible to analyze which cluster is
contributing more to the application execution, detect-
ing load imbalance issues.

The three images of figure 7 represent three different
configurations for time intervals of the same application.
The application considered was developed with
the KAAPI library, which uses work stealing tech-
niques to provide load balancing among the processes.

(a) Time-slice corresponding to the start of the execution

(b) to the end of the execution

(c) and to the total time of execution.

Figure 7. Three images showing the behavior of
the same application but in different time intervals.

The normal behavior of the work stealing algorithm
implemented inside KAAPI is the low rate of steal
in the beggining of the application when compared to
the end of application execution, where a high rate
of steal is observed. The first image 7(a), generated
with a time-slice to get only the beggining of the
application, reflects this exact behavior. The processes
spent more time running the tasks than trying to steal
from others processes. The second image 7(b) was generated with a time-slice that correspond to the end of application execution. It reflects the expected behavior with the increasing number of work stealing requests, because of the lack of tasks that preceed the end of execution. The third image 7(c) was created using the total execution time as time-slice. It allows the understand the global behavior of processes and compare their efficiency depending on which cluster they were placed to execute.

Figures 7(a) and 7(c) can also be used to evaluate the efficiency of the work stealing algorithm during the application execution. Each site (nancy and rennes) contributed with the same amount of machines (100 each), but the application was launched from the nancy site, more specifically from the cluster A. Since we used the amount of time to calculate the rectangles size for each process, we can see in figure 7(a) that the are occupied by clusters A and B, from nancy site is bigger than the area from cluster C, D and E, from rennes site, indicating that the amount of time spent by cluster A and B in executing tasks is also bigger. If we see the same application considering all the execution time as time-slice, in figure 7(c), it can be noticed that the amount of time spent by the processes located in each site have become equal.

We executed the application using a greater number of machines and processes in order to show the visual scalability of the approach. In this new execution, the environment was composed by 310 machines, divided in four Grid5000 sites: rennes, sophia and bordeaux (each with 3 clusters) and lille (with 4 clusters). The application was composed by 2900 processes, generating a 400 Megabytes of trace data that registers the work stealing algorithm behavior. Figure 8 depicts the behavior of all these processes at the same screenshot. The bigger rectangles details the clusters involved. Comparing this execution to the previous one, we increased the size of the application in number of processes by 14.5 times, and obtained a visualization that can be also understood increasing the size of the drawings area by only 1.2 time.

6. Conclusion

Large-scale distributed platforms are used today to the execution of applications composed by thousands of processes. The behavior analysis of these applications is often complex because of the number of monitored entities involved and the event granularity. Current visualization schemes applied in this analysis are usually based on gantt-charts.

This paper presented an alternative and novel visualization to the analysis of parallel applications. The technique can be used to show the behavior of large-scale grid applications. Our approach uses time intervals to dynamically create an annotated hierarchical structure that represents the application behavior for that period of time. We then use the squarified treemap technique to represent the structure in the screen. This approach is the first, to our knowledge, to combine treemap techniques to the analysis of parallel applications of distributed systems. The results have validated the approach through multiple analysis and observations of real executions in a Grid.

As future work, we intend to do transformations in the annotated hierarchical structure in order to investigate different organizations of the monitoring data that give more hints to developers. Besides that, we aim to analyze nested states of applications, the communications involved and their impact in the visualization.

References


Figure 8. Visualization of the amount of time each of the 2900 processes stayed in execution of tasks or trying to steal tasks from others processes (the time slice slider is in the bottom of the screen and appears by demand).


