Towards the Performance Visualization of Web-Service Based Applications

Marian Bubak[1,2], Wlodzimierz Funika[1], Marcin Koch[1], Dominik Dziok[1], Allen D. Malony[3], Marcin Smetek[1], and Roland Wismüller[4]

[1] Inst. Comp. Science, AGH, Krakow, Poland
[2] ACC CYFRONET-AGH, Krakow, Poland
[3] Dept. of Computer and Information Science, University of Oregon, Eugene, USA
[4] Fachgruppe BVS - Universität Siegen, Siegen, Germany
Phone: (+48 12) 617 44 66; Fax: (+48 12) 633 80 54
{bubak, funika, smetek}@uci.agh.edu.pl, malony@cs.uoregon.edu, ivn@icslab.agh.edu.pl, domin@student.uci.agh.edu.pl, roland.wismueller@uni-siegen.de

Abstract. In this paper we present an approach to building a monitoring environment which underlies performance visualization for distributed applications. Our focus is to make the J-OCM monitoring system and the TAU-Paravis performance visualization system to collaborate. J-OCM, based on the J-OMIS interface, provides services for on-line monitoring of distributed Java applications. The system uses J-OCM to supply monitoring data on the distributed application, whereas TAU-Paravis provides advanced visualization of performance data. We managed to integrate J-OCM into TAU/Paravis by developing additional software providing access to the monitor and transformation of raw monitor data into performance data which is presented with 3-D charts. At the end we present an extension, which introduces Web Service monitoring into the integrated environment.

Keywords: performance visualization, monitoring tools, OMIS, TAU, web service.

1 Introduction

The ability to monitor the execution of a distributed application and to measure its performance is a key issue in designing and deploying such applications [1]. A standard approach assumes a kind of pre-execution instrumentation of the source code. During execution the instrumented code generates monitoring information which are stored and presented to the user either in semi-on-line or off-line mode. Such an approach has many limitations: it requires often source recompilation, does not work with applications that execute very long, does not allow to control the execution of the application. As a result, there is a distinct need for performance analysis systems that can perform on-the-fly monitoring and allow for application control and interaction at run-time. In this paper we present our approach to building a monitoring environment and also the way of extending it to support the monitoring of Web Services which become an increasingly popular technology of distributed programming.

Our approach is based on the integration of J-OCM [2], a flexible monitoring system into the TAU-Paravis visualization package [3]. Up to now there were no tools which used J-OCM for performance analysis, so it was impossible to fully use this on-line monitoring system for performance analysis. We use J-OCM to supply on-the-fly monitoring data from a distributed application, while TAU-Paravis provides advanced visualization of performance data. As J-OCM and TAU-Paravis use different data models, our work is aimed at the development of additional packages to make them cooperate.

The paper is organized as follows: Section 2 introduces the main features of J-OCM. Next, Section 3 gives some details on SCIRun and TAU. Section 4 presents the concept and structure of the integrated monitoring environment. Next, Section 5 explains a way of extending the environment to support Web Services monitoring, followed by the features of 3-D performance visualization in Section 6. Then we give a short overview of related work. Finally, we sum up the work done and show some plans for further research.

2 J-OCM Monitoring System

J-OCM is a monitoring system for Java applications, compliant with the OMIS specification [4] extended by a support for Java, in form of the J-OMIS extension [5]. The idea of OMIS (On-line Monitoring Interface Specification) is to separate the functionality of a monitoring system from monitoring tools. The OMIS specification defines an interface that is an intermediate layer between them. The communication is based on the request-reply mechanism realized as a set of *services* while the processing of events uses the event-action paradigm. OMIS enables convenient access to performance objects like classes, methods, threads, or web services; they are identified by tokens. All performance object types, observable by the monitor, form an *objects hierarchy.* The OMIS concept allows multiple monitoring-based tools like profilers, debuggers, etc. to use a single monitoring system at the same time.

The J-OMIS specification, which underlies J-OCM, extends OMIS to match the monitoring of Java applications. It introduces new, specific for Java types of objects and services, to form an object hierarchy relevant to Java. It also divides the new object hierarchy into two kinds of objects – execution ones: nodes, JVMs, threads, and application ones: interfaces, classes, objects, methods. Each object has its own set of services divided to three groups: information services - to provide information about objects, manipulation services - to allow to change objects' states, and event services to trigger some actions when matching events occur. Event services are used by tools to program the monitoring system for getting specific data from a monitored application or manipulating it. J-OCM (see Fig. 1) is implemented as an extension to the OCM monitoring system [6].

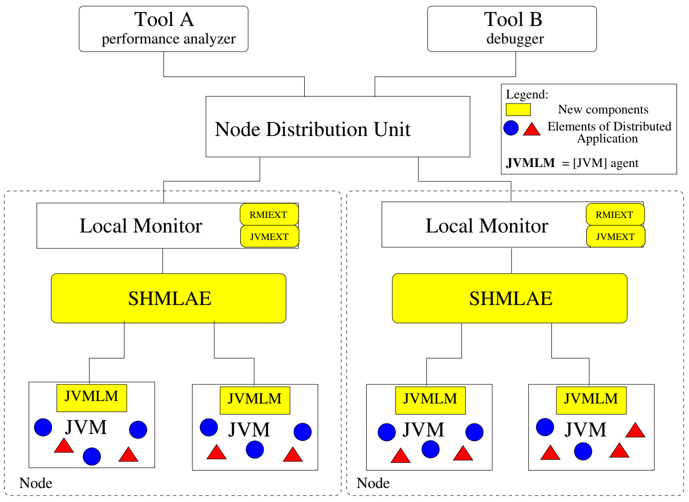

Fig. 1. Architecture of J-OCM

Recently, J-OCM was extended to enable the monitoring of web-service based applications written in Java. Some modifications have been introduced to the J-OMIS interface specification. The extension provides means for accessing web services and forwarding web-service specific events.

3 SCIRun and TAU

Within our work we are integrating J-OCM with TAU-Paravis visualization package. Paravis, which is a part of the TAU(Tuning And Analysis Utilities) monitoring environment , is developed at the University of Oregon in Eugene [7, 8]. Paravis introduces advanced 3-D visualization into TAU.

TAU-Paravis is built within SCIRun, a powerful Problem Solving Environment [9]. It is developed at the University of Utah, as an open source software. SCIRun can be used widely for solving various scientific problems. It consists of the modules that allow to perform complicated computations, data transformations, and provide advanced visualization. An application built within SCIRun is composed from the modules which can be connected one to another through pipes carrying data. A module usually gets input data, performs some computations on it and sends the results to another module. Through various configurations of modules the user can solve many complicated scientific problems.

The concept of providing monitoring data by J-OCM to a tool required additional software to be written. We had to develop a new package in SCIRun to make TAU/Paravis work with J-OCM. The new set of modules is responsible for: programming the J-OCM monitoring system, receiving the events from it, selecting a performance object to be monitored, controlling the execution of application, and processing the data which are passed to TAU-Paravis.

4 Design of Monitoring Environment

The new monitoring environment can be divided into two parts: a monitoring subsystem and an SCIRun compliant tool. The tool is responsible for the programming monitoring activities, handling monitor events, measuring and visualizing. The tool runs inside SCIRun and consists of several packages. Its most important components are the TAU package and JOCM package. We focused on the second one. The JOCM package contains several SCIRun modules, data types and ports definitions. They provide access to the monitor and produce a TAU compatible data structure on the output. The structure is a 3-D matrix. Having passed such matrices TAU can be used to provide matrix specific operations on the data.

Fig. 2 presents an overview of the system architecture. J-OCM and SCIRun are two separate systems which communicate using the J-OMIS interface. SCIRun *access modules* execute J-OCM specific services and handle responses and events. In principle, it is possible to use any monitoring system other than J-OCM, which complies to the J-OMIS interface. A monitoring system can be developed or evolve separately without influencing the whole environment.

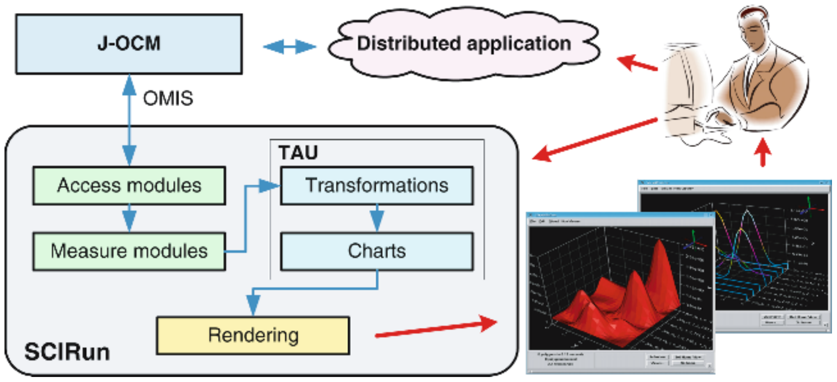

Fig. 2. System architecture

In the JOCM package we define the following kinds of objects which can be monitored: *Node* (physical machine), *JVM* (Java Virtual Machine), *Class* (Java class), *Method* (class method), and *Thread* (Java thread)

Each performance object has its representation within J-OCM in form of *token*. Objects form a hierarchy with nodes on the top and methods and threads at the bottom. Each performance object is associated with a specific SCIRun access module. The monitoring environment can be easily configured by placing modules and connecting them according to the objects hierarchy. Each access module can be attached to one or more J-OCM tokens. It is possible to attach multiple modules to the same token.

The access modules perform monitor programming, but they also handle events from J-OCM. The module on top of the hierarchy receives all event notifications and passes them down until the event comes to the module which is intended to handle it. Each access module has also a performance output which can be connected to some measurement module's input port.

The second category of modules contains *measurement modules*. They are responsible for building measurements from monitoring events and gathering measurement results. Measurements are connected to some metrics. We have defined the following metrics:

– *method execution time* (aggregate or momentary, inclusive or exclusive, in context of a thread)
– *thread status over time*
– *garbage collector activity* (execution time, released memory size)

The "method execution time" can be aggregated at the class level. In this case, the "class execution time" is a sum of all class methods execution times. The monitoring of thread status provides important information about the activity of a thread. This information can be useful in solving the issues of thread synchronization.

The measurement modules gather performance data and transform it into 3-D matrix structures. These can be passed to TAU modules which perform some additional transformations like aggregation or scaling. The results are being used to produce 3-D charts which are finally rendered by an SCIRun built-in rendering module.

The functionality of the system is much broader that performance visualization only. It can also take control over an application execution. Now this ability is limited to threads only. The user can make use of this feature to change a thread status during the execution, without modifying source code.

5 Web Services Monitoring

The Web Services approach to building distributed applications is getting more and more popular. The advantages of using web-based components are very difficult to overestimate. Such components have well-defined interfaces and can be easily accessed. An issue when using Web Services is their performance, due to the use of XML-based communication protocols.

Debugging and optimizing a single Web service is quite an easy task. Problems occur when Web services start to interact while forming a working web application. In this case the ability to do performance analysis is extremely important. It is still very difficult to find a complete solution for the performance monitoring and visualization of web services.

Following the extension done to J-OCM, we have developed additional modules for accessing the new types of performance objects. These modules are:

– *Web Service access module*
– *Web Service's port access module*
– *Port's operation access module*

The Web Service access module can be connected to the Node access module. In this way the new modules extend the existing modules hierarchy (it is related to the extended J-OCM tokens hierarchy).

We have defined relevant metrics:

- *Resources usage* (CPU, memory)
- *Communication* (throughput, latency, reply time)
- *Requests* (frequency, SOAP message processing time, message size)
- *Run-time specific* (operation execution time)

These metrics define the new measurements which require new modules for gathering the measurement results. The important thing is that the output of these modules is still TAU's 3D matrix. As a result the visualization engine is invariant to changes.

Within this concept, a very important part of the work are extensions to the monitoring system. J-OCM must be able to manage a web service's distribution which differs from that on a cluster of nodes supported by J-OCM. Since nodes can be grouped into sites, the monitoring system architecture must comprise *Service Managers* above Local Monitors. Service Managers are managed by Service Distribution Unit instead of the Node Distribution Unit as it was in the cluster-oriented version of J-OCM.

6 Performance Visualization

The advantages of 3-D visualization over 2-D are obvious. 3-D charts are much more readable and can provide more information at the same time. One of the most important advantages of TAU is the ability to perform such visualization in real time. The user is enabled to see what performance problems occur during an application's execution. The user is enabled to access various performance

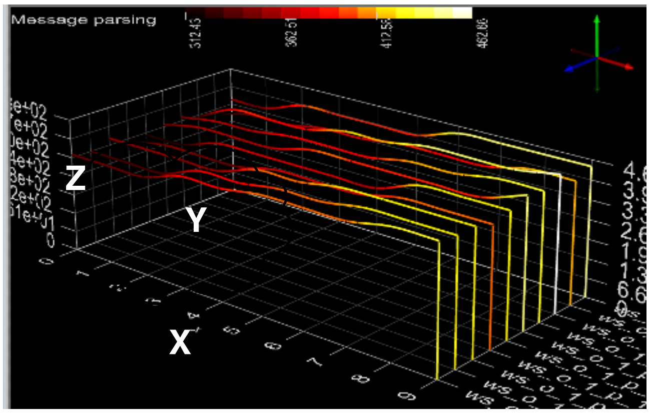

Fig. 3. Summarized message parsing time (X-time; Y-web services; Z-measured value)

data on different objects at the same time. In this case it is much easier to figure out dependencies between performance phenomena (e.g. bottlenecks).

In Fig. 3 we show an example performance visualization session of several similar web services ("Hello world"). The monitoring environment is configured to measure the SOAP messages parsing time. Each "Hello world" Web service is located on a different physical machine. The client application randomly sends requests to the Web services. With the tool we can observe the dynamics of increase in the aggregate parsing time over the execution.

7 Related Work

There are a large number of monitoring tools for Java distributed programs. Commercial tools like AmberPoint Express[1] for .NET platform or open source ones like Apache jMeter[2] for Java provide for the user a lot of useful features. They support advanced visualization and can point at application bottlenecks. However, each of them runs as a client application. They test Web Services by sending requests and counting the response time or number of fails. They can also inspect SOAP packages. However, an issue is that they do not allow the user to get insight into what really happens inside the Web Service.

Our goal is to overcome this constraint and to provide that our approach can point at the part of the Web Service (initialization, request processing, operation invocation or response) responsible for the performance problem. Moreover, the system under discussion supplies more advanced visualization and can be easily extended by new functionality.

8 Conclusion

Performance visualization is important for several reasons. Efficient application programming requires efficient techniques for performance analysis and visualization. When dealing with distributed systems the need in such a functionality is even stronger.

It is still difficult to find comprehensive open source solutions for performance monitoring, analysis and visualization of web services. Most of available tools are limited to the SOAP messages analysis. Our approach is to attach the monitoring tool to the Web service container and obtain from it needed monitoring data.

The system under discussion is open source and offers a functionality required in case of the monitoring of distributed Java applications and also web services implemented in Java. It uses a powerful scientific visualization environment - SCIRun which is widely used due to it's open architecture. As a result, the system's functionality can be easily extended by adding new modules.

As any monitoring system, J-OCM induces some overhead into the monitored application performance, due to the local agents which may do dynamical instrumentation of the monitored application. In other cases, agents use the JVM

[1] [http://www.amberpoint.com/solutions/express.shtml]

[2] [http://jakarta.apache.org/jmeter]

built-it Tool Interface which influences the performance to a small extent. The results of overhead measurements will be presented in the final version of the paper. As well we will provide the results of scalability research.

One of the important issues of the whole environment we will work on in further research is to optimize the resources usage needed by SCIRun (memory, CPUs, efficient graphics accelerator), which is crucial in large distributed applications.

The web page of the monitoring environment is being worked on.

Acknowledgements

This research was partially supported by the KBN grant 4 T11C 032 23.

References

1. M. Gerndt, Automatic performance analysis tools for the Grid, Concurrency and Computation: Pract. Exper, Vol. 17, pp. 99-115, 2005
2. W. Funika, M. Bubak, M.Smętek, and R. Wismüller. An OMIS-based Approach to Monitoring Distributed Java Applications. In Yuen Chung Kwong, editor, Annual Review of Scalable Computing, volume 6, chapter 1. pp. 1-29, World Scientific Publishing Co. and Singapore University Press, 2004.
3. TAU's 3-D Profile Visualizer - ParaVis, University of Oregon, Computer and Information Science http://www.cs.uoregon.edu/research/paracomp/tau/tauprofile/dist/paravis/
4. Ludwig, T., Wismüller, R., Sunderam, V., and Bode, A.: OMIS – On-line Monitoring Interface Specification (Version 2.0). Shaker Verlag, Aachen, vol. 9, LRR-TUM Research Report Series. 1997 http://wwwbode.in.tum.de/˜omis/OMIS/Version-2.0/version-2.0.ps.gz
5. Bubak, M., Funika, W., Wismüller, R., Mętel, P., Orłowski. Monitoring of Distributed Java Applications. In: Future Generation Computer Systems, 2003, no. 19, pp. 651-663. Elsevier Publishers, 2003
6. R. Wismüller, J. Trinitis and T. Ludwig: A Universal Infrastructure for the Runtime Monitoring of Parallel and Distributed Applications. In Proc. Euro-Par'98, Southampton, UK, September 1998, LNCS 1470, pp. 173-180. Springer-Verlag, 1998
7. A. D. Malony, S. Shende, and R. Bell, "Online Performance Observation of Large-Scale Parallel Applications", Proc. Parco 2003 Symposium, Elsevier B.V., Sept. 2003.
8. A. D. Malony, S. Shende, R. Bell, K. Li, L. Li, and N. Trebon, "Advances in the TAU Performance System," Chapter, "Performance Analysis and Grid Computing," Kluwer, Norwell, MA, 129-144, 2003.
9. C. Johnson, S. Parker, "The SCIRun Parallel Scientific Computing Problem Solving Environment" Proc. Ninth SIAM Conference on Parallel Processing for Scientific Computing, 1999. http://software.sci.utah.edu/scirun.html