Programming and Coordinating Grid Environments and Applications

Cristina Ururahy Noemi Rodriguez

Computer Science Department – PUC-Rio
{ururahy,noemi}@inf.puc-rio.br, http://www.inf.puc-rio.br

Abstract

The heterogeneous and dynamic nature of Grid environments place new demands on models and paradigms for parallel programming. In this work we discuss how ALua, a programming system based on a dual programming language model, can help the programmer to develop applications for this environment, monitoring the state of resources and controlling the application so that it adapts to changes in this state.

1 Introduction

Parallel programming in distributed memory environments must deal with issues such as heterogeneity, fault tolerance, and non determinism in communication. Coordination models [1] support the separation of concerns of the computation itself from those of cooperation and communication between computational components, often proposing distinct languages for programming these two activities.

Currently, parallel programmers are facing new issues related to harnessing the computing power available through large-area networks. Grid initiatives and projects [2, 3] seek to create a distributed computing infrastructure for highly demanding scientific and engineering applications. Grids present a highly heterogeneous and dynamic configuration, in contrast to conventional parallel machines. One important technique for dealing with these variations is to use *adaptive strategies*, allowing the program to react dynamically to changes in the environment. Besides, it often does not make sense to have to stop an application and begin processing all over again just because it was launched with an inadequate configuration.

In light of these requirements, the use of an adequate language for coordinating a parallel application gains new importance. In this paper, we discuss ALua, a distributed programming system based on C and Lua, an interpreted language, and the flexibility that this system can bring to Grid environments. One of the important characteristics of an interpreted language is allowing for interactivity: With an interpreted coordination language, the programmer may use a "coordination console" to monitor and control the application.

In the next section we present the ALua model. Section 3 presents the work we have been doing using ALua for Grid environments and applications. Finally, in Section 4 we draw our conclusions.

2 ALua

The ALua proposal is to use a dual programming model for parallel applications, where ALua acts as a linking element, allowing pre-compiled parts of the program to be executed in different computers. In this context, applications are divided in two parts, kernel and configuration, usually written in different languages. The kernel implements the basic components of the system and is usually written in a compiled statically typed language, such as C or C++. The configuration part, that is usually written in an interpreted language, connects these components defining the final shape of the application [4]. Using this model, we can build flexible applications without compromising their performance [5].

ALua [5] is an event-driven communication model for parallel distributed applications, based on the interpreted language Lua [6]. An important feature of an interpreted language is the support for executing

dynamically created chunks of code. In ALua, messages are chunks of code that will be executed by the recipient. This provides a very simple yet very powerful communication mechanism [5]. There is only one asynchronous communication primitive in ALua, `send`, that sends a chunk of code to another process. There is no equivalent to a `receive` primitive. Each process has a Lua interpreter and an event loop, that manages network and user-interface events. The arrival of a message is treated as a network event. This communication model is very flexible: A programmer can use it to perform simple tasks, such as calling a remote function, but she can also use it for much more complex tasks, such as remotely changing the algorithm a process is executing. In the context of long-term parallel applications, and using an interactive console, this is a powerful possibility that allows the programmer to redefine the application behavior dynamically.

The user interface is a console, where the user can enter arbitrary Lua commands. Each line the user types generates an event. Through simple commands the user can inspect variables (`print(var)`), change variable values (`var = exp`), send messages to other processes (`send(receiver, msg)`), or even run a program (`dofile("progname")`).

An important characteristic of ALua is that it treats each message as an atomic chunk of code, handling each event to completion before dealing with the next one. This means that there is no internal concurrency in an ALua process. In our experience this is not a hindrance. As we discuss in [5], the use of the event-driven paradigm, as of any other programming paradigm, leads programmers to create specific program structures. Messages must typically be small, non-blocking chunks of code. If an application requires larger actions, a process can always resort to sending a message to itself, as a means of breaking up its code in non-atomic parts, therefore allowing other messages to be received in between. On the other hand, as pointed out in [7], the lack of concurrency greatly simplifies many aspects of distributed programming, since there is no need for synchronization inside one process.

3 The ALua Model: Flexibility for the Grid

Among the main Grid technologies, the mechanisms that the *Globus toolkit* [3] offers stand out not only for being used in several sites, but also for the independence of its services, what allows us to select only the services that are relevant to the Grid we want to use. But, at the same time that Globus is very popular for the amount and independence of the services it offers, it becomes very complex to use so many different services and libraries all together.

In this work, we are investigating the use of ALua for monitoring and developing parallel applications for the Grid. Our main goal is to use ALua to build not only a distributed parallel programming model for the Grid, but also a tool for developing, monitoring and adapting Grid applications and monitoring the Grid itself. In the works we developed so far, the ALua model showed to be highly flexible, bringing advantages not only to the final shape of the applications, but also to their development process. ALua allows, for example, the rapid development of application prototypes, and many times there is no need to replace this prototype, because we do not observe a considerable performance loss in the application.

In the Grid computing context, a tool like ALua becomes very important, once the Grid has a dynamic configuration in its definition. The ALua model can be used to perform a bunch of different tasks. For example, the interpreted nature of ALua can be useful for interactively controlling applications with large execution times. The programmer can use the console to redefine the behavior of a function on the fly to instrument the application.

We have integrated ALua with existing Grid communication and resource management mechanisms so that we can dynamically analyze the behavior of applications and the Grid. Through an ALua console we can monitor the Grid nodes that are part of a specific computation, as well as their resources. Also, we can send code that can change the behavior of a node, adapting an application dynamically, without the need to define what this adaptation will be in advance.

We are also working on the support for dynamic adaptation of Grid applications. Previously, we had worked on adaptation in the context of CORBA applications [8] and we are applying some results of that work to the ALua model for the Grid.

For the monitoring and adaptation infrastructure, we are using Globus services for resource allocation management, security infrastructure and direc-

tories. With the meta directories service (MDS), we can find out not only the nodes available for executing an application, but also their characteristics and resources. The idea is to make these services available to the ALua programmer, so that she can use the infrastructure that is already available in the Grid platform in a much more flexible and dynamic way, in the applications as well as in a management console. We believe that accessing these services through ALua, together with the flexibility the ALua model provides, greatly simplifies a programmer's life.

We made an experiment in which we linked the Lua library with a few libraries from Globus: GRAM and DUROC (Globus Resource Allocation Management), and LDAP (used in the Meta Directories Service). Then, we built the MDSmonitor, a monitor mechanism based on [8]. A monitor is a program that has a *timer* and that from time to time verifies the state of a list of properties of user defined resources. For each monitored property, the user can register a callback, that is responsible for performing the adaptation itself. It can be written in Lua or C and can use other libraries, such as MPI. We used the LDAP binding to Lua to query the dynamic information that the MDS provides.

The use of Lua greatly simplifies the access to Globus services and libraries. With Lua we were able to build concise interfaces that are very light for the programmers who are using few resources, but still offer a lot of flexibility for the ones who need more complex tasks.

Now we are studying the applicability of the ALua model to control and reconfigure applications that use other communication libraries. Because of its popularity, we have a special interest in MPI for the Grid, such as the mpich-g2 [9].

4 Final Remarks

We have discussed on-going work using ALua's flexibility to allow the programmer to monitor and control resource usage on the Grid. Although the tools available in the Globus toolkit provide an extensive set of facilities, their integrated use demands a lot of effort.

Our goal is to provide an unified, flexible and interactive programming interface. This interface will allow the programmer to deal with the dynamic and heterogeneous nature of the Grid and make effective use of the available resources. We believe that the use of ALua together with Globus libraries and services provides a very flexible and powerful environment for the Grid.

This work is supported by CNPq-Brazil.

References

[1] G. Papadopoulos and F. Arbab. Coordination models and languages. In Marvin V. Zelkowitz, editor, *Advances in Computers*, 46:329–400. Academic Press, Aug. 1998.

[2] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organization. *The International Journal of High Performance Computing Applications*, 15(3):200–222, Fall 2001.

[3] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.

[4] J. Ousterhout. Scripting: Higher level programming for the 21st century. *IEEE Computer*, 31(3):23–30, March 1998.

[5] C. Ururahy, N. Rodriguez, and R. Ierusalimschy. ALua: flexibility for parallel programming. *Computer Languages*, 28(2):155–180, Dec. 2002.

[6] R. Ierusalimschy, L. H. Figueiredo, and W. Celes. Lua—an extensible extension language. *Software: Practice and Experience*, 26(6):635–652, 1996.

[7] L. Bic, M. Fukuda, and M. Dillencourt. Distributed computing using autonomous objects. *IEEE Computer*, 29(8):55–61, Aug. 1996.

[8] A. L. de Moura, C. Ururahy, R. Cerqueira, and N. Rodriguez. Dynamic support for distributed auto-adaptive applications. In *Proceedings of AOPDCS (held in conjunction with IEEE ICDCS 2002)*, pages 451–456, Vienna, Austria, July 2002.

[9] I. Foster and N. Karonis. A grid-enabled MPI: Message passing in heterogeneous distributed computing systems. In *Proceedings of 1998 SC Conference*, Nov. 1998.