How to Run Scientific Applications over Web Services

Diego Puppin Nicola Tonellotto
Domenico Laforenza
Institute for Information Science and Technologies
ISTI - CNR, via Moruzzi, 56100 Pisa, Italy
email: {diego.puppin, nicola.tonellotto, domenico.laforenza}@isti.cnr.it

Abstract

Today, the task of running and coordinating a scientific application across several administrative domains is extremely complex. As an example, the most popular tool for scientific applications, MPI, is not designed to address firewall limitations or data heterogeneity, even if its extensions deal with some of these problems.

In this paper, we design a new approach to run a scientific application in a distributed environment, when data and computing power are scattered across the Web: Web Services can be used to tunnel computation and data migration.

We show that a very simple mapping exists between MPI primitives and the Web Service infrastructure. We are currently designing a framework, based on Web Services, which will implement the main MPI primitives: this way an MPI application could be run on any platform supporting Web Services.

Keywords: *Web Services, distributed computing, MPI.*

1. Introduction

Computational Grids are re-shaping the way scientific problems are faced. Scientists are not anymore bound to a single machine (or a single cluster of machines) and a single source of data: they use machines that are distributed across several institutions in the world, and they need to access data in remote repository.

When computational scientists are given the opportunity to access the Grid and its abundance of resources, they will be able to run faster code with more reliable data; they will be able to exploit instruments that are not located in their laboratory; they will be able to cross-verify different sources of information.

Traditional programming tools, MPI among them, are not designed to manage with these complex issue: their programming model refers to a single set of machines and a very coherent data structure.

A way to deal with this is to move toward a Service Oriented Architecture (SOA) [1], a computing paradigm that considers services as building blocks for applications. Web Services (WSs) represent one of its implementations. One of the most interesting features of WSs is that, when wrapped as http data, requests to and responses from any Web Service can cross, without problems, firewalls, differences in implementation languages or in operating systems.

This is why we discuss about the opportunity of using Web Services to implement parallel applications, as an alternative to MPI. Clearly, MPI will be hard to overcome in term of performance on local machines and local data, but we want to show that MPI applications can be easily mapped, with good results, on a Web Service framework: this will give the programmer the opportunity to access a wide array of computing and data resources.

The rest of the paper is structured as follows. After a review of related work, we discuss a new methodology to map distributed application to a set of Web Services. Then, we show the results of our first experiment. Finally, we conclude and show directions of future work.

2. Related Work

As discussed in the introduction, MPI is a widely used tools for scientific applications. There is a wide interest in overcoming some of its limitations so to make it more suitable to run in distributed environments.

In order to connect distinct private machine clusters, PACX-MPI [2] uses a two-level communication: MPI-based within the cluster, and TCP/IP-based across clusters. Two daemons, sitting on publicly visible nodes, mediate the two types of communications. MPICH-G2 [3] uses Globus services to cross firewall boundaries, so to connect clusters within different, secure administrative domains. An extension to MPICH-G2, called MPICH-GP [4] and still under development, uses the Linux NAT address translation, along with a user-level proxy, to make two firewall-protected clusters, with private IP address, to cooperate as a single MPI machine, with limited overhead.

Issues of QoS and performance predictability raised attention among MPI researchers. MPICH-GQ [5] set an interesting research direction, and showed the potential of an extension of MPI that addresses QoS. Data security is dealt with by MPICH-GS [4]: in this project, data are encrypted using a cluster-specific key.

The interest to SOA to run scientific applications is also growing. Some recent works showed how to wrap whole MPI applications as Web or Grid Services, so they could be linked to other pieces of software through a standard framework. An interesting approach is taken in [6], where authors show an effective way to include a legacy MPI algorithm into a larger application using the OGSA standard, by creating a surrounding Grid Service. Also, whole parallel applications are wrapped as Web Services in recent works [7, 8].

This paper suggests a different approach: we want to use WSs as a way to connect computation kernels, so to implement a distributed applications, able to communicate across firewalls and differences in hardware and operating system. Instead of wrapping a whole application as a service, we build it out of services.

Our approach moves in the direction of using services, available around the Internet, as building units for an application.

3. Mapping MPI to Web Services

Web Services (WS) are a way to make data and computational services available on the Internet: using the WS standard, machines connected across the Internet can interact with each other to perform complex activities. A simple `http` connection is usually enough to connect to a server and to use the services it offers. One of the goals of this architecture is to give a standard interface to new or existing applications, i.e. to offer a standard description to any computational service, which can be stored in public repositories.

The main messages in the WS standard are the request to and the replies from the services. They are carried around using the *Simple Object Access Protocol (SOAP)* [9], used in the communication and data exchange between a WS and a client. WS-oriented frameworks (see Figure 1) support this kind of communications, by implementing the infrastructure to communicate over the Internet with SOAP messages, and by offering a *procedure-call abstraction* to service invocation: a programmer can perform a request to a WS as simply as calling a procedure.

The WS standard defines a variety of communication patterns, including rendezvous and one-way communications. We are designing a mapping from the main MPI primitives to the WS framework (see Table1): `MPI_init` will be mapped to a set of coordination messages to the WSs, `MPI_send` to a one-way communication, `MPI_recv` to a rendezvous message, that forces a WS to send the needed message back.

More complex primitives will also be supported, even if they will be initially implemented using the basic ones: e.g. `MPI_scatter` will be replaced by a set of `MPI_send`.

MPI_init. `MPI_init` is responsible for verifying the status of the workers in the MPI pool, and to give a unique ID to each of them. This task can be easily performed by a set of synchronization messages, one to each WS used in the application, carrying the unique ID.

MPI_send. Using one-way communications, we can implement non-blocking messages. The sender can transfer data to every other WS, in a very simple way. The WS will receive the message as soon as it is available for listening. Blocking communications can be implemented with RPC abstraction.

MPI_recv. `MPI_recv` is performed simply by accepting requests from other entities. Also, one service can force another WS to send messages, by using a locking communication that asks for data.

The new process we envision is as follows:

- A legacy MPI application is linked to a supporting library, so that MPI communications are mediated by WS communications.
- The resulting code, is wrapped into a Java class.
- Its interface is extracted (with Java introspection) and converted to a suitable XML representation for deployment.
- The class is deployed within a WS container as an independent service.
- `MPI_init` will connect all running WSs and will give them a unique ID.
- The coordinated WSs will run the application.

As will be said below, probably the biggest performance overhead is given by the switch from MPI to SOAP messages. SOAP messages have to pay a high cost in order to be inter-operable: the payload is not sent in a binary form, but in a more portable, verbose way. For instance, doubles are converted to ASCII, and they take one byte (character) per figure, rather than (roughly) one byte every three figures in the binary form. They are then converted back by

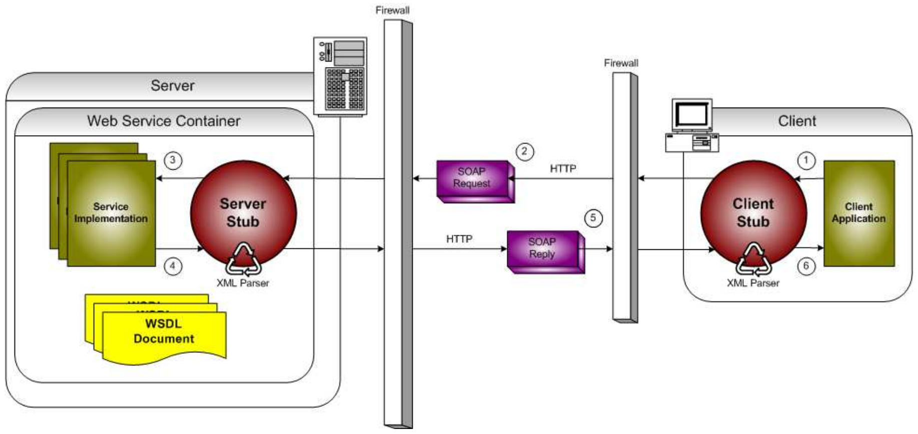

Figure 1. Web Service architecture

MPI_init	initiation messages to WSs (synchronization needed)
MPI_send	one-way or rendezvous communication from the master to the WS
MPI_recv	rendezvous communication, to force the WS to send the message back

Table 1. Simple mapping of basic MPI primitives to WSs

the receiver. This enlarges greatly the message size, and also is cause of a big slowdown in the parameter passing (discussed in detail in [10]).

Many researchers are exploring ways to reduce the overhead introduced by SOAP. The most interesting effort in this direction is being performed by W3C, which is investigating the XML-binary Optimized Packaging protocol (XOP), which allows binary transmission of data between services when possible [11].

We are planning to use several strategies to limit this problem. One can be to use XOP or to compress messages (using zlib) to reduce this effect. Another could be to post the data in some Web accessible URL, and then let the WS retrieve them from there.

4. Initial Experimentation

In this paper, we upgrade the results of our initial experiments, which we presented in [12]. While we work at our MPI mapping to WS, we manually ported a simple MPI application (a farm-like computation) to a WS-based solution. We rewrote part of the code to connect to the WSs, which were running the heavy computational part of the problem, and to coordinate communications from and to them.

A simple Java class of 20 lines, implementing the numerical kernel, was simply transformed into a WS, and then used by our client to perform a distributed computation. We could run our application on our private cluster, and then across the Internet with no modification.

In the WS version, the master, running on the cluster front-end, invoked the services of the farm workers: each worker appears as a WS running on a different machine. The master, with the roles of dispatcher and collector, is implemented as a client application. The master spawns a thread for each worker, with the responsibility of waiting for the responses and submitting new values to it. This is needed in order to have asynchronous message.

In [12], we showed that creating a WS is relatively simple: a Java class has to be developed, with the code implementing the numerical kernel. Then, it has to be *deployed* into Tomcat. With our new framework, we plan to make the porting process even simpler. A MPI-WS library will need to be linked to the MPI code. At that point, MPI messages will be mapped to WS communications. All MPI process, including the master, will be running as WSs, and the message patterns will be more symmetrical (with no preference for Process 0).

In this paper, we present upgraded results. We were able to add a large number of machines to our pool of Grid resources, so to scale the application to 65 workers. Using the abundance of resources we had, our application was able to run very close to MPI, with only 50% overhead: while MPI took about 6 seconds, the WS-based application, using resources across the world, took less than 9 seconds (see Figure 2 and 3). In this version, the application run on 25 machines located in San Diego (UCSD), 33 machines located in Cambridge, MA (MIT) and 7 machines located in

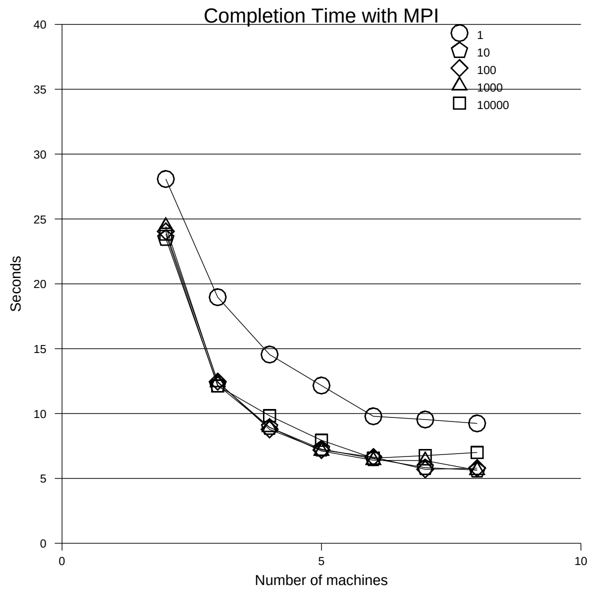

Figure 2. Performance of the MPI benchmark (adapted from [12])

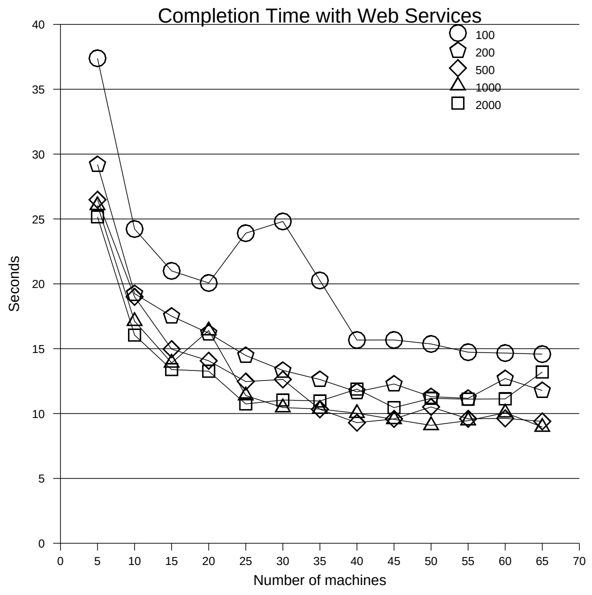

Figure 3. Performance of the Ws-based application, varying granularity

our laboratory in Pisa. Remote machines are not accessible from within MPI, because they are behind a firewall that filters out MPI messages.

We are working at limiting the effect on performance of a number of factors: difference in performance from Java to C; overhead in message marshalling and unmarshalling (heavier for SOAP than MPI); overhead of the Web Service container (communication is not mediated in MPI); the use of the procedure-call paradigm, which can slow down the control flow, because there is the need for multiple threads, waiting for replies from the services.

Several strategies must be exploited:

- To improve numerical performance, heavy kernels should be implemented in C and connected to the WS framework (usually, running in Java), via the Java Native Interface.
- Simpler WS frameworks than Apache are available. By working with a stripped container, we will be able to reduce the framework overhead.
- To limit the overhead introduced by RPC, and its active waiting, we are planning to use framework such as Homa [13] or ProActive [14], which implement the concept of *future*: outstanding results from procedure calls that are automatically managed by the framework.

4.1. Moving to WS from Other Frameworks

Transition to WS-based programming is much easier with modern development environments such as Java 2 Enterprise Edition (J2EE) and Microsoft .NET. With them, it is very easy to wrap an existing application into a WS: a programmer can run a prototype of his/her application by bridging the communication to remote machines with Web Services.

5. Conclusions and Future Work

Modern scientific applications are complex programs: they often exploit heterogeneous tools, compare data from different sources, run on machines spread across the world. To tackle this complexity, the Service Oriented Architecture, with its implementation in the Web Services, seems a viable approach.

MPI, a very accepted standard for parallel applications, is trying to adapt to Grid-oriented solutions with a variety of extensions, none of which has yet the flexibility and generality needed by Grid programmers.

On the other side, WSs are emerging as a standard for distributed, Internet-oriented applications. As we showed in our previous work, legacy applications can be easily wrapped into a WS interface, and made available to any client on the Internet. Here, we presented a methodology which simplifies the task of porting an MPI application to a WS-based solution. We are designing a mapping from MPI primitives to WS-based communication. In our plans, porting should become as easy as linking a library that implements MPI communications over WSs.

Clearly, we do not want to say that MPI can be substituted by WSs, but that the SOA can be a way to develop complex distributed applications running across the Internet. Using the abundance of resources available on the Grid, we could run an intensive numerical application, originally running with MPI on a local cluster, on remote machines, with a limited overhead (50%). If a cluster is not available or data are stored on remote machines, this is a very small price to pay to be able to run the application.

6. Acknowledgments

We want to thank the group led by Dean Tullsen at UCSD, and the COMMIT group led by Saman Amarasinghe at MIT, for letting us run our tests on their machines. This work has been partially supported by the MIUR GRID.it project (RBNE01KNFP) and the MIUR CNR Strategic Project L 499/97-2000.

References

[1] Papazoglou, M.P., Georgakopoulos, D., eds.: Service-Oriented Computing. Volume 46 (10) of Communications of ACM. (2003)

[2] Beisel, T., Gabriel, E., Resch, M.: An extension to mpi for distributed computing on mpps. In: Recent Advances in PVM and MPI, LNCS (1997) 75–83

[3] Karonis, N., Toonen, B., Foster, I.: MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. JPDC **63** (2003) 551–563

[4] Kwon, O.Y.: Mpi functionality extension for grid. Technical report, Sogang University (2003) Available at: http://gridcenter.or.kr/ComputingGrid/file/gfk/Oh-YoungKwon.pdf.

[5] Roy, A.J., Foster, I., Gropp, W., Toonen, B., Karonis, N., Sander, V.: Mpich-gq: Quality-of-service for message passing programs. In: Proceedings of Supercomputing 2000, Dallas, Texas, United States (2000)

[6] Floros, E., Cotronis, Y.: Exposing mpi applications as grid services. In: Proceedings of EuroPar 2004, Pisa, Italy (2004)

[7] Gannon, D.: Software component architecture for the grid: Workflow and cca. In: Proceedings of the Workshop on Component Models and Systems for Grid Applications, Saint Malo, France (2004)

[8] Balis, B., Bubak, M., Wegiel, M.: A solution for adapting legacy code as web services. In: Proceedings of the Workshop on Component Models and Systems for Grid Applications, Saint Malo, France (2004)

[9] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H.F., Thatte, S., Winer, D.: Simple object access protocol (soap) 1.1. Technical report, W3C (2003) Available at: http://www.w3.org/TR/SOAP/.

[10] Chiu, K., Govindaraju, M., Bramley, R.: Investigating the limits of soap performance for scientific computing. In: Proceedings of HPDC 11, IEEE (2002) 246

[11] Web Consortium (W3C): The xml-binary optimized protocol (2004) Available at: http://www.w3.org/TR/xop10/.

[12] Puppin, D., Tonellotto, N., Laforenza, D.: Using web services to run distributed numerical applications. In: Proceedings of EuroPVM-MPI 2004, Budapest, Hungary (2004)

[13] Gautier, T., Hamidi, H.R.: Automatic re-scheduling of dependencies in a rpc-based grid. In: Proceedings of the 2004 International Conference on Supercomputing (ICS), Saint Malo, France (2004)

[14] ProActive: The proactive environment (2005) Available at: http://www-sop.inria.fr/oasis/ProActive/.