Using Web Services to Run Distributed Numerical Applications

Diego Puppin, Nicola Tonellotto, and Domenico Laforenza

Institute for Information Science and Technologies
ISTI - CNR, via Moruzzi, 56100 Pisa, Italy
{diego.puppin,nicola.tonellotto,domenico.laforenza}@isti.cnr.it

Abstract. MPI is a *de facto* standard for high performance numerical applications on parallel machines: it is available, in a variety of implementations, for a range of architectures, ranging from supercomputers to clusters of workstations. Nonetheless, with the growing demand for distributed, heterogeneous and Grid computing, developers are hitting some of its limitations: e.g. security is not addressed, and geographically distributed machines are difficult to connect.
In this work, we give an example of a parallel application, implemented with the use of Web Services. Web Services represent an emerging standard to offer computational services over the Internet. While this solution does not reach the same performance of MPI, it offers a series of advantages: high availability, rapid design, extreme heterogeneity.

1 Introduction

The emergence of computational Grids is shifting the interest of the computational scientists. Their applications are not anymore (or not only) CPU-bound numerical kernels, but can be seen as collections of services. This causes a growing need for ways to connect heterogeneous tools, machines and data repositories. In this direction, there is a growing demand for a standard, cross-platform communication and computational infrastructure.

At the present day, MPI is a *de facto* standard for the development of numerical kernels: parallel applications have been using MPI for years; several MPI implementations are available for a variety of machines; binding to many programming languages are available. Nonetheless, MPI incurs in communication problems, very difficult to overcome, when the application is spread over remote machines, when firewalls are present, when machine configurations are different. There have been some experimental works to overcome these limitations, but so far there is no agreement on a common solution.

On the other side, there is a growing interest toward the Service Oriented Architecture (SOA) [1], a computing paradigm that considers services as building blocks for applications, and Web Services (WSs) are one of its implementations. A WS is a specific kind of service, identified by a URI, whose description and transport utilize open Internet standards. 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, it is not our intention to state that Web Services are candidates for substituting MPI for scientific applications: rather, we want to show that WSs can be fruitfully used in an area where typically MPI is used, when there is a stronger need for flexibility and interoperability among application components. We want to face this comparison from many different points of view: ease of programming, performance, availability.

The rest of the paper is structured as follows. In the next section, we give an overview of related work. Then, we introduce Web Services and their features. In section 4, we compare these two approaches by using a very simple benchmark application. Lastly, we conclude by showing some open research directions.

2 Related Work

There is an emerging attention to the convergence of MPI parallel software and the Service-Oriented Architecture. An interesting approach to this is taken in [2], 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 [3, 4].

Our work differs in that we are trying to use WSs as workers for a numerical application, rather than wrapping the whole application to be used as a service.

On the other side, there is interest in overcoming some limitations of MPI so to use it to program distributed applications:

– the internal nodes of a private cluster often do not have a static public IP, and cannot connect directly to the nodes outside the cluster; this way, they cannot participate to the *MPI world* of an application running remotely;
– if a machine is behind a firewall, MPI messages could be filtered out by it.

Several solutions have been proposed. In order to connect distinct private machine clusters, PACX-MPI [5] 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 [6] uses Globus services to cross firewall boundaries, so to connect clusters within different, secure administrative domains. An extension to MPICH-G2, called MPICH-GP [7] 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.

3 Web Services

Web Services (WS) are a way to make data and computational services available on the Internet: using the WS standard, machines connected through the Internet can interact with each other to perform complex activities. A simple http

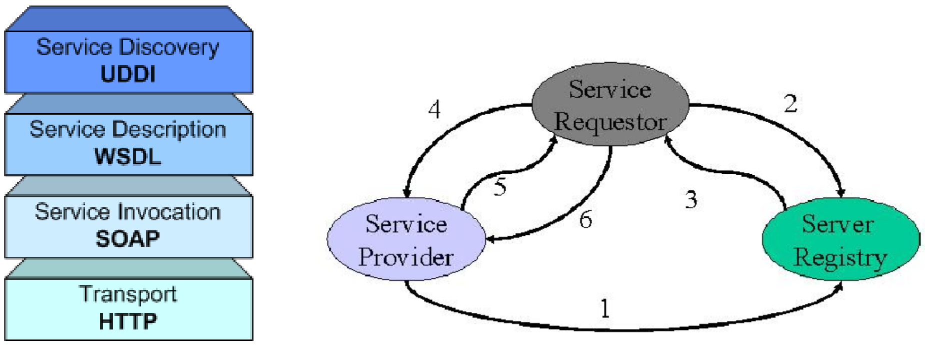

Fig. 1. A logical representation of messages hierarchy for Web Services (left). Web Service life cycle (right).

connection is usually enough to connect to a server and 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 WS standard is designed around three main types of messages, based on XML (see Figure 1): *Web Service Description Language (WSDL)* [8] is used to describe the service implementation and interface; *Simple Object Access Protocol (SOAP)* [9], used in the communication and data exchange between a WS and a client; *Universal Description, Discovery and Integration (UDDI)* [10], used to register and discover public services.

Figure 1 shows the typical scenario of Web Service utilization. The Web Service is first published using a UDDI repository (1). When a Web Service consumer is looking for a specific service, it submits a query to the repository (2), which answers back with the URL of the service that best matches the query (3). By using this URL, the Consumer asks for the WSDL describing the needed service (4); then, by using this WSDL description (5), it invokes the service (with a SOAP message) (6). The Web Service Consumer and Web Service provider have to share only the description of the service in a WSDL standard format. So, Internet access and support for SOAP message are all that is needed to use a Web Service.

The use of a WS is simplified by WS-oriented frameworks, which implement the infrastructure to communicate over the Internet with SOAP messages, and offer a *procedure-call abstraction* to service invocation (see Figure 2).

4 Experimental Results

We compared MPI and WSs with a simple client/server (task farm) application: workers perform an intensive computational loop on floating-point values, sent in blocks (with varying granularity) by the master, which then collects the results. Clearly, this limited experimentation is not exhaustive of the matter.

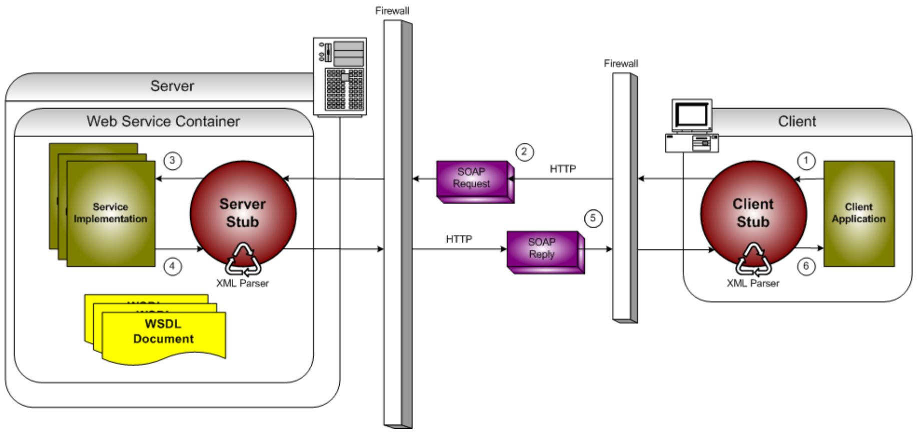

Fig. 2. Web Service architecture.

Rather, it wants to be an initial evaluation of the difference of the two discussed methodologies.

We performed two experiments. First, to understand the overhead related to the use of WSs, our applications were tested on our local cluster, using its internal private network. It is built up of 8 nodes, each of which with a dual 2 GHz Intel Xeon processor, 1 GB RAM memory. On the internal network, all nodes have a unique static IP that can be used by applications.

Second, we launched our WS-based application across the Internet: we installed our services on remote machines in San Diego. They have a (slower) 1.4 GHz AMD Athlon K6 processor, 1 GB RAM memory. They are not accessible with MPI because they are sitting behind firewalls.

MPI Implementation. Our farm application was implemented very simply with MPI: a master is responsible for distributing sequences of floating-point numbers among slaves, which are running on the internal nodes; then, it waits for answer, and gives new data to the idle workers. Our tests were performed using MPICH-1.2.5. Its implementation is a single C file, 218-line long, and can be run with a single shell command.

WS Implementation. To test the WS implementation, we installed a web server on each of the internal nodes of the cluster, and we used their internal IP to connect to them. We also linked our service to the web servers running on the remote machines in San Diego.

We used Apache Tomcat (V4.0) and Apache Axis. Tomcat is an open-source Web Application container developed with Java. Axis allows the developer to use Java classes as WS, by intercepting SOAP messages and by offering a procedure-call abstraction.

The master, running on the cluster front-end, invokes the services of the farm workers: each worker appears as a WS running on a different machine. The

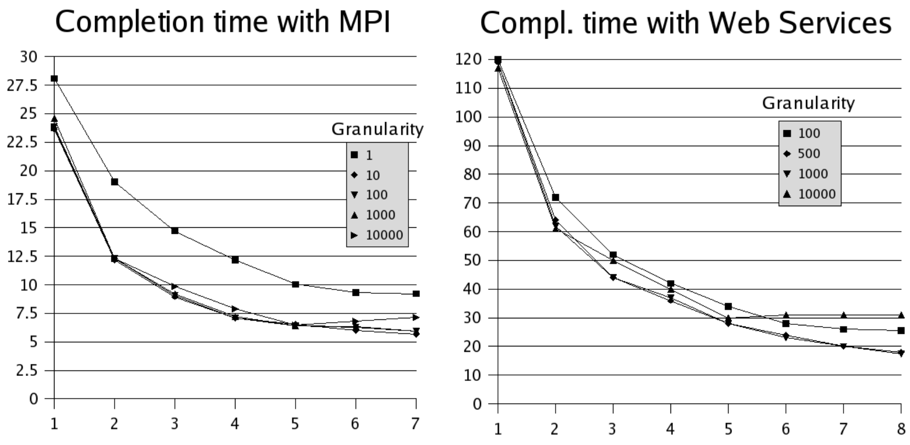

Fig. 3. Completion time of a MPI farm (left) and a WS farm (right), with different granularity, varying the number of workers, on our local cluster. Time in seconds.

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.

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. We developed a set of scripts that perform most of this task automatically, using *ssh*. An MPI programmer interested in using WSs to implement his/her application, should focus mainly on translating its algorithm so to use a client/server (procedure-call) paradigm. We are interested in using one-way WS communications to mimic MPI messages: in the future, porting an MPI applications to WSs should be much simpler.

The WS implementation of our benchmark is composed of three Java files: the implementation of the master (318 lines), the worker interface (6 lines), and the worker implementation (20 lines). Its length is motivated by the verbosity of Java, and by the need of creating new threads for each worker. Also, the information about the topology of the computing environment is encoded within the master, while, in the MPI application, it is part of the environment configuration.

4.1 Programming Complexity and Performance

The MPI application behaves as expected, with good scalability, as shown in Figure 3(left). The best speedup is obtained with granularity 100.

The WS-based version, running on the local cluster, has an overhead ranging from 3x to 4x. This is due to a number of factors, including 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). Also, the use of the procedure-call paradigm can

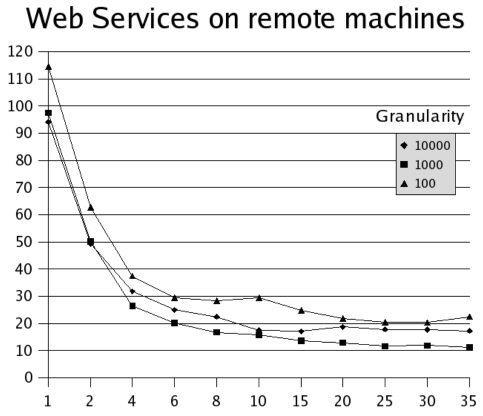

Fig. 4. Completion time of a WS farm when using remote machines. Time in seconds.

slow down the control flow, because there is the need for multiple threads, waiting for replies from the services. Recent advancements in RPC-based scheduling (e.g. [11]) could be useful in this context too.

Our WS-based application was able to use the computing services offered by a set of remote machines located in San Diego. We were able to use a pool of up to 35 workers. Using a total of 25 workers, with the best choice of granularity, we completed our application in 11.59 seconds, that is about twice as much as the best timing we had running MPI locally (5.87 seconds, with 7 workers).

In this example, the overhead related to moving from an MPI implementation to a WS implementation is somewhat limited, and can be acceptable if firewalls and difference in operating systems or programming languages prevent the use of MPI. Using WSs, we could connect to a set of (slower) remote machines, not reachable with MPI, and had an acceptable overall performance (2x slower).

4.2 Communication Overhead

MPI messages are very compact, and highly optimized: the MPI envelope is very small compared to the payload, as it stores very limited information, such as sender's and receiver's ID, message tag and MPI communicator number, totaling 16 bytes of data. Also, the array of double-precision numbers is sent in a very compact binary way, 8 bytes per element (see Table 1(a)).

On the other side, SOAP has 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 – doubles are converted to ASCII, and then converted back by the receiver. This enlarges greatly the message size (see Table 1(b)), and also is cause of a big performance overhead (discussed in detail in [12]).

Many researchers are exploring ways to reduce the overhead introduced by SOAP. The most interesting effort in this direction is being performed by W3C,

Table 1. Message size (bytes) for MPI (a) and WS (b), varying granularity.

Granularity	In data	Out Data	MPI request	MPI reply	Overhead
1	8	8	24	24	200%
10	80	80	96	96	20%
100	800	800	816	816	2%
1000	8000	8000	8016	8016	0.2%
10000	80000	80000	80016	80016	0.002%

(a)

Granularity	In data	Out Data	SOAP Req.	SOAP Resp.	Overhead
100	800	800	2692	4181	236% - 422%
500	4000	4000	11992	18981	199% - 374%
1000	8000	8000	23494	37482	193% - 368%
10000	80000	80000	230496	370483	188% - 363%

(b)

which is investigating the XML-binary Optimized Packaging protocol (XOP), which allows binary transmission of data between services when possible [13].

5 Conclusions and Future Work

While MPI is a strong, efficient, very accepted standard for parallel applications, there is a growing need for more general solutions, especially when distributed/Grid applications are to be developed. A variety of MPI extensions have been proposed, each addressing a specific MPI limitation, but so far there is no general agreement on a standard solution.

On the other side, WSs are emerging as a standard for distributed, Internet-oriented applications. Legacy applications can be easily wrapped into a WS interface, and made available to any client on the Internet. In this work, we showed how a computationally intensive application can be performed by using WSs as workers. While the performance is clearly lower, it can be a choice when firewalls are present, when operating systems and programming languages are different.

Also, for a variety of frameworks (.NET, J2EE...), it is very easy to wrap an existing application into a WS: this can be a way to do rapid prototyping of the parallel/distributed version of legacy software. Here we showed that a simple Java class of 20 lines, implementing a numerical kernel, can be simply transformed into a WS, and then used by a client to perform a distributed computation. We could run our application on our private cluster, and then across the Internet with no modification.

Future work includes a comparison of some of the available MPI extensions with WSs in a real distributed application, a detailed analysis of the overhead introduced by the use of WSs and of the suitable granularity for applications built as collections of services. Also, we are interested in implementing the MPI protocols over WSs: the MPI world could be emulated by a pool of coordinated services, communicating directly with one another.

Acknowledgements

We want to thank Massimo Serranò for his help with the experimental setting, and the group led by Dean Tullsen at UCSD 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. Floros, E., Cotronis, Y.: Exposing mpi applications as grid services. In: Proceedings of EuroPar 2004, Pisa, Italy (2004) To appear.
3. 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)
4. 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)
5. 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
6. Karonis, N., Toonen, B., Foster, I.: MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. JPDC **63** (2003) 551–563
7. Kwon, O.Y.: Mpi functionality extension for grid. Technical report, Sogang University (2003) Available at: http://gridcenter.or.kr/ComputingGrid/ file/gfk/OhYoungKwon.pdf.
8. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web services description language (wsdl) 1.1. Technical report, W3C (2003) Available at: http://www.w3.org/TR/wsdl.
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. Bryan, D., *et al.*: Universal description, discovery and integration (uddi) protocol. Technical report, W3C (2003) Available at: http://www.uddi.org.
11. 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)
12. Chiu, K., Govindaraju, M., Bramley, R.: Investigating the limits of soap performance for scientific computing. In: Proceedings of HPDC 11, IEEE (2002) 246
13. Web Consortium (W3C): The xml-binary optimized protocol (2004) Available at: http://www.w3.org/TR/xop10/.