Message Passing over Windows-based Desktop Grids

Carlos Queiroz, Marco A. S. Netto, Rajkumar Buyya

Grid Computing and Distributed Systems Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{carlosq, netto, raj}@csse.unimelb.edu.au

ABSTRACT

Message Passing is a popular mechanism used to enable inter-process communication in parallel and distributed computing. Many complex scientific and engineering applications that are executed on clusters have been developed based on this communication model. Due to the huge amount of computing power being wasted in desktops, there is an increasing interest in using these computers to execute complex applications. However, most of the current middleware systems are aimed at executing only embarrassingly parallel applications, i.e. with no inter-process communication. Moreover, most existing middleware systems are based on Linux. Nevertheless, it is well-known that the desktop machines in the world are predominantly based on the Windows operating system. Hence, in this work we present the design, implementation and evaluation performance of a Windows-based implementation of two message passing models, Message Passing Interface (MPI) and Bulk Synchronous Parallel (BSP), over the Alchemi Grid computing framework.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; D.1.3 [**Programming Techniques**]: Concurrent Programming—*Parallel Programming*

Keywords

Message Passing Interface, Bulk Synchronous Parallel Model, Parallel Computing, Desktop Grids

1. INTRODUCTION

Idle desktop machines offer considerable computing power that can be harnessed to execute complex applications. Popular projects such as SETI@home, BOINC@home, FightAIDS@home, Distributed.net, and Folding@home have come up with new ways of make use of these idle resources. However, these systems primarily target parameter sweep applications, and have no support for inter-process communication.

Message Passing is a mechanism to enable inter-process communication in parallel and distributed computing. Many complex scientific and engineering applications that have been developed to execute on cluster machines are based upon such a communication model. These applications impose several challenges due to inter-process communication. For example, a machine failure may compromise the entire application and differences between performance of various machines have to be carefully handled to minimise the influence on the final result. Problems such as scalability, security, scheduling, are also need to be considered for message passing applications. In order to tackle the challenges of using desktop machines for distributed computing, middleware systems must provide simple and efficient mechanisms for development of applications, and also have to simplify the provisioning of the computational resources. Currently, desktop Grid technologies, such as Bayanihan [5], InterGrade [3], BSP-G [6], and PUBWCL [1], either do not support the Windows environment or do not provide message passing programming libraries. Another problem is that message passing libraries, such as MPICH-V [2], do not make use of various advanced Grid middleware services.

This work overcomes this limitation by implementing support for two well-known message passing models – the Message Passing Interface (MPI) and the Bulk Synchronous Parallel (BSP) model, over Alchemi which is a Grid computing framework for the Windows platform [4].

2. MESSAGE PASSING OVER ALCHEMI

Relying on Grid middleware brings several benefits. Some of these include: security, resource discovery, user and data management and scheduling. In order to leverage such functionalities, we chose to develop our libraries on top of Alchemi [4], a Windows-based desktop Grid computing framework implemented on Microsoft .NET platform. Alchemi is designed to be user friendly without sacrificing power and flexibility. It provides the run-time machinery and a programming environment (API), which is required to construct desktop Grid applications. It also supports cross-platform application submission via a web-service interface. Alchemi is based on the master-worker model, where a manager is responsible for coordinating the execution of tasks sent to the executors (desktop machines).

The key features supported by Alchemi are Internet-based clustering of desktop computers without a shared file system, federation of clusters to create hierarchical coopera-

MGC'06, November 27, 2006 Melbourne, Australia Copyright 2006 ACM 1-59593-581-9...$5.00.

tive grids, dedicated or non-dedicated (voluntary) execution by clusters and individual nodes, grid thread programming model (fine-grained abstraction), and a Web Services interface to support a grid job model (coarse-grained abstraction) for cross-platform interoperability.

Most of the BSP and MPI implementations have bindings that can be used in C, C++ and FORTRAN languages. Developers working with these implementations must use low level data types and provide several parameters, such as array of bytes, array size, and array data type for some functions. As our implementation is developed on top of the .NET framework, all these parameters can be reduced to a single object. Another advantage of our approach is that the application developers can use high level languages supported by the .NET, such as C#, C++, J++, Visual Basic, and Python.

3. MESSAGE PASSING IMPLEMENTATION

In this initial implementation we provide the basic functions for the MPI library and all functions for the BSP model. Some of these functions are very similar in both MPI and BSP. Among the functions we have those to initialise and finalise the environment, to receive and send messages, to get information on the number of processes and their identification, and also to execute barrier and broadcast.

Note that even though these functions are very similar, they have some peculiarities for each model. For instance, the *MPI_Send* function allows defining the type of the message but the *bsp_send* does not. Taking this into account we have designed a single core architecture that is used by both environments. The *send* and *receive* (*move*) functions, as well as, *rank* (*pid*), *init* (*begin*), *finalise* (*end*), and so forth are implemented in a core library and a wrapper is used to keep up with differences in the function signatures for each specification that was implemented.

The steps to execute message-passing applications on Alchemi are basically the same to execute the parameter sweep applications – which are currently supported by Alchemi. However, the users now make use of the runners (*MPIRun* and *BSPRun*) responsible for loading the message passing libraries. Figure 1 summarises the process to execute the parallel applications in an Alchemi grid. There are basically, four steps:

1. The user submits an application to the Alchemi Manager (in our case an MPI or a BSP application using *MPIRun* or *BSPRun* respectively) by specifying the number of machines and the application.

2. The Alchemi Manager selects the nodes to run the application.

3. The Executors (Alchemi nodes) start to run the application (in our case the nodes communicate with each other).

4. The results are sent back to the user, through the Manager, that saves them in files, one for each process.

4. PERFORMANCE EVALUATION

In order to evaluate our MPI and BSP implementation on Alchemi, we setup an environment, composed of 9 desktop machines, and compared our libraries with MPICH over Cygwin. The matrix multiplication application was chosen

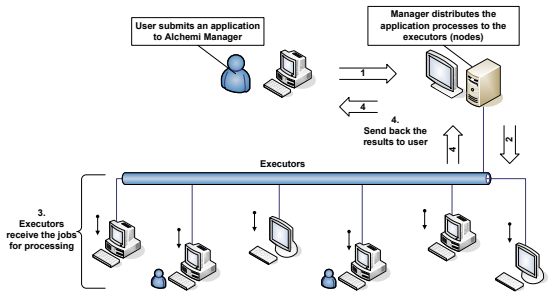

Figure 1: Flow diagram for executing message passing applications on Alchemi middleware in a Desktop Grid.

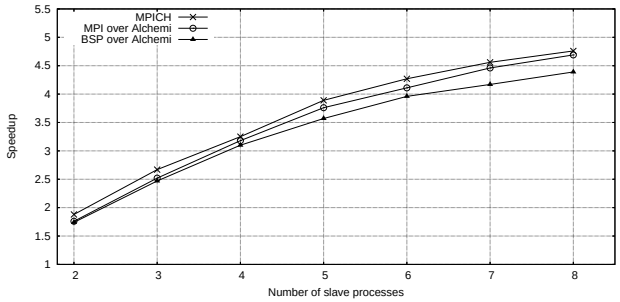

Figure 2: Speedup for Matrix with 2000x2000 elements.

for this evaluation. The results in Figure 2 show that our implementation performs similar to MPICH. Therefore we argue that the BSP and MPI implementations over Alchemi are a very attractive alternative to develop message passing applications over desktop machines due to the several facilities provided by our environment.

5. REFERENCES

[1] O. Bonorden *et al.* A web computing environment for parallel algorithms in java. *Scalable Computing: Practice and Experience*, 7(2):1–14, 2006.

[2] G. Bosilca *et al.* MPICH-V: toward a scalable fault tolerant MPI for volatile nodes. In *Proceedings of the ACM/IEEE SC*, pages 1–18, Baltimore, USA, 2002.

[3] A. Goldchleger *et al.* InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines. *Concurrency and Computation: Practice and Experience*, 16:449–459, March 2004.

[4] A. Luther *et al. Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework, High Performance Computing: Paradigm and Infrastructure.* Wiley Press, New York, USA, 2005.

[5] L. F. G. Sarmenta and S. Hirano. Bayanihan: building and studying web-based volunteer computing systems using java. *Future Generation Computer Systems*, 15(5–6):675–686, 1999.

[6] W. Tong *et al.* A parallel programming environment on grid. In *Proceedings of the ICCS*, volume 2657, pages 225–234. Springer, 2003.