MPI Support on Opportunistic Grids based on the InteGrade Middleware *

Marcelo de Castro Cardozo
Fábio M. Costa
Institute of Computing - Federal University of Goiás
Campus 2 - UFG - 74690-815 - Goiânia-GO, Brazil
{fmc,mcastro}@inf.ufg.br

Abstract

The Message Passing Interface (MPI) is a popular programming model for parallel applications. Support for MPI in grid middleware is important for the widespread use of grids for parallel programming. This enables existing parallel applications to be executed on large-scale grids, as opposed to being restricted to local clusters. In the specific case of opportunistic grids, the use of idle computing power from non-dedicated computers further adds to the range of resources that can be used. In this paper we present MPICH-IG, an implementation of the MPI2 standard on top of the InteGrade grid middleware. Existing MPI applications can be run unmodified, while taking advantage of the InteGrade scheduler to harvest available computing power from the grid. In addition, fault-tolerance of MPI applications is achieved through a checkpointing mechanism, which allows applications to be resumed after failures of particular grid nodes.

1 Introduction

One of the main motivations for the advent of grid computing was the ability to provide high performance computing using widely available resources [4]. Arguably, grids are strong candidates as execution environments for parallel applications as they may be used to provide vast amounts of computing power in a cost-effective way, without requiring dedicated infrastructure. Opportunistic grids go one step further by enabling such a scenario in a general purpose computing environment where idle resources are harvested and aggregated to run distributed applications.

Nevertheless, opportunistic grids pose extra complexity to the running of parallel applications when compared to dedicated cluster environments. In particular, this type of grid is very dynamic, which means that a number of services have to be provided, among them: resource location and scheduling; recovery from failures; security; and heterogeneity management. Although these services are common in grid computing middleware, it must be possible for the parallel applications programmer to use them in a transparent way, i.e, without requiring modification of the source code. This can be achieved by modifying or reimplementing parallel programming libraries so that they use the grid middleware services instead of their own native services to run the applications.

In this paper, we present MPICH-IG, an implementation of the MPI2 standard for opportunistic grids. MPICH-IG is based on MPICH2 [11], which has a modular architecture that allows reuse of several of its components, as well as clean replacement of those related to the management of application execution, essential to effectively use the resources of a computing grid. MPICH-IG is built on top of the InteGrade platform, a CORBA-based object-oriented grid computing middleware aimed at opportunistic grids based on general purpose, non-dedicated, computers [5].

2 MPI and MPICH2 Architecture

The Message Passing interface (MPI) has become the *de facto* standard for parallel applications programming [7]. Its current version, MPI2, was released in 1997 and defines a comprehensive set of functions for point-to-point communication, management of process groups, group communication, query about the execution state of processes, support for distributed shared memory, and dynamic spawning of processes [11].

Several implementations of the standard exist, among them the open source MPICH2 [1]. One of the main characteristics of MPICH2 is its modular, layered, architecture, which clearly separates the implementation of the high level protocols and functions of MPI2 from the low-level mechanisms used for interprocess communication and process management. This makes it easier to port it to different

*This work was partly supported by CNPq-Brazil, grant number 55.0895/2007-8.

platform infrastructures. In this respect, the main element of the architecture is the Abstract Device Interface (ADI), which specifies the operations required from the low-level communications protocol to support the MPI2 functions [6]. Through different implementations of the ADI, it is possible to seamlessly port (and optimize) MPICH2 to different hardware and software platforms, which are called *devices* in the MPICH2 terminology.

A number of implementations of the ADI exist, aimed at different platforms, such as Globus and Myrinet. There is also a generic implementation, called CH-3, which was developed with the goal of further minimizing the set of operations that must be re-implemented in order to port the library to different platforms. CH-3 defines a lower layer, called Channel Interface (CI), which specifies basic primitives for interprocess communication, such as *send*, *receive* and *select*. All higher-level communications features are implemented in terms of these low-level primitives.

Another important module of the architecture is the Process Manager Interface (PMI). It defines functions to manage communication among parallel processes, such as process location, configuration of the communications environment (e.g., binding IP addresses and port numbers to processes), dynamic creation of processes, and collection of computation results. The most common implementation of the PMI is the Multiporpose Daemon (MPD), which runs on each machine that hosts MPI processes. When the user requests the execution of an MPI parallel application (through the mpiexec command), the involved MPDs exchange the required information to enable communication among the application's processes. The MPDs also collect the output of the application processes and send it to the the process that requested the execution of the application.

2.1 Parallel Applications on the Grid

Computational grids have been proposed as an environment to run parallel applications, mostly due to the fact that they interconnect large amounts of computing resources and their ability to transparently schedule and allocate the resources that are most appropriate to run the application's processes. Scavenging or opportunistic grids go one step further by allowing resources to be harvested from general-purpose, non-dedicated, machines, contributing to the cost/benefit relation. On the other hand, problems of resource location, communication and fault-tolerance become more complicated due to the need to deal with the typical heterogeneity of such grids and the fact that machines may leave the grid when their resources are requested by the local user, causing resource failures.

In order to deal with such problems, parallel programming environments must be able to negotiate with a diverse set of resource managers to run the applications on an integrated set of resources that span different administrative domains. This metacomputing capability, however, is not present in standard parallel programming environments like MPI2 [3]. A solution to this problem is to modify the parallel programming library to replace its process management mechanisms with those of the grid middleware. This maintains the original programming model and enables transparency to the parallel applications developer, as well as the ability to run legacy parallel programs on the grid. This approach is adopted in our implementation of MPI2 on top of the InteGrade middleware.

3 Overview of InteGrade

The InteGrade project aims at building an object-oriented grid middleware to use the idle capacity of non-dedicated computing resources in an opportunistic setting. One of the major design principles of InteGrade is to provide support for a rich set of parallel programming models, among them: Bag-of-Tasks (BoT), BSP (Bulk Synchronous Parallelism), and MPI. In this section, we give a brief overview of the architecture of InteGrade, focusing on the elements that were relevant for the integration of the MPI programming model and library. Further details can be found elsewhere [5].

InteGrade architecture is based on CORBA [12], which is the communications middleware that enables interaction among InteGrade components. This includes the use of standard CORBA services such as naming and trading. The basic structural unit of an InteGrade grid is the cluster, which may contain both dedicated and shared (i.e., used by other applications) compute nodes, along with user nodes (from which grid application execution is requested) and a manager node, which manages and schedules the computing resources of the cluster. Each node in a cluster runs a set of InteGrade components (i.e., CORBA objects), which execute the meta-computing tasks of the grid. The main components with respect to the implementation of MPICH-IG are:

- GRM (Global Resource Manager): runs on the manager node and is responsible for resource allocation and application scheduling;
- LRM (Local Resource Manager): runs on each compute node and monitors resource usage levels, sending periodic notifications to the GRM. It also accepts and processes requests to run grid applications on the node.
- AR (Application Repository): stores grid applications in executable form, making them available so the LRMs can run them.
- CDRM (Checkpointing Data Repository Manager):

provides to the checkpoint library the location of the repositories where to store checkpoints in the cluster.

- CkpLib (Checkpoint Library): enables checkpointing and recovery of the applications on a given node.
- CkpRep (Checkpoint Repository): stores checkpoint data and provides such information when the application needs to be recovered.
- EM (Execution Manager): manages application execution by monitoring the nodes where they are run. It sends notifications when the application finishes, and initiates recovery actions upon failures.
- ASCT (Application Submission and Control Tool): GUI-based application used to submit, monitor and collect the results of application execution.

These components communicate by using two CORBA-compliant ORBs, namely JacORB[1] for the components written in Java, and OiL (ORB in Lua)[2] for those written in Lua or C/C++. The MPICH-IG library, described in the next section, is implemented as a set of components that complement this original InteGrade architecture.

4 MPICH-IG

As seen above, the approach we adopted for the implementation of MPICH-IG was to modify an existing MPI2 library, MPICH2, instead of heavily modifying the implementation of InteGrade itself. This is enabled by the architecture of MPICH2, which encourages portability by requiring only the re-implementation of its lowest, platform-dependent, layers.

In order to adapt MPICH2 to run on InteGrade, two of its interfaces had to be re-implemented: the *Channel Interface* (CI) and the *Process Management Interface* (PMI). The former is required to monitor the sockets channel to detect and treat failures through a mechanism of coordinated checkpointing and recovery. The latter is necessary to couple the management of MPI applications with InteGrade's Execution Manager (EM), adding functions for process location and synchronization, as well as to store checkpoints. These interfaces are implemented by the modules IG-Sock and IG-PM, respectively, replacing the corresponding modules of MPICH2 (Socket Channel and MPD), as shown in Figure 1.

4.1 IG-PM

On a grid, the several processes that compose a parallel application may be geographically dispersed, requiring mechanisms to publish and discover the necessary information to establish communication among them. The processes also need to synchronize their execution for the purposes of coordination. In MPICH2, this service is provided by the MPD. However, the MPD does not deal with resource heterogeneity and failures, which are common on opportunistic grids. Thus, MPICH-IG replaces MPD with the IG-PM component, which besides determining application termination and collecting the results (with the help of the LRMs), performs the following functions:

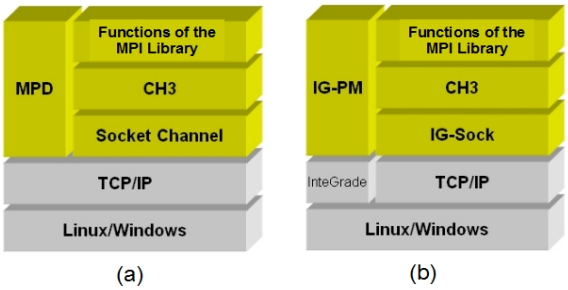

Figure 1. Differences between the standard implementation of MPICH2 (a) and MPICH-IG (b).

- loads, from the LRM, the necessary information to enable execution of the processes of an MPI application; this information is passed to the LRM by the GRM when requesting application execution) and comprises the process rank (unique id), the number of processes, and a numeric value representing the checkpoint interval (in seconds);
- synchronizes and locates application processes: the IG-PM running on each node must send to InteGrade's EM connection information (IP address and port number) of the local processes; in return, it receives, also from the EM, similar connection information of all other processes; and
- stores and recovers checkpoints: for this purpose, IG-PM uses the CkpLib component of InteGrade, which implements a given checkpointing and recovery strategy.

The application execution protocol for MPICH-IG applications is similar to that for BSP applications [5]. Its first steps basically consist of the user submitting the application via the ASCT to the GRM, which uses resource availability information to select the nodes to run the application and informs the EM to start managing it. The GRM then sends requests to the LRMs on the selected nodes to dispatch the

[1] http://www.jacorb.org
[2] http://oil.luaforge.net

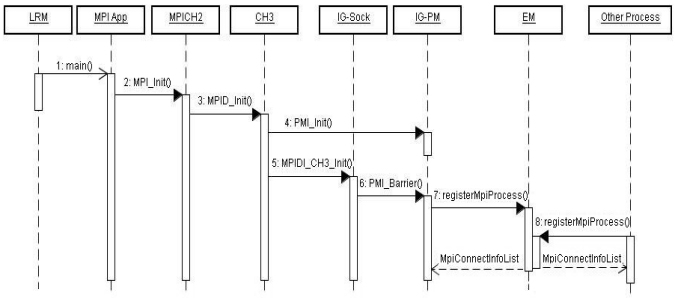

Figure 2. Execution of an MPI application on InteGrade.

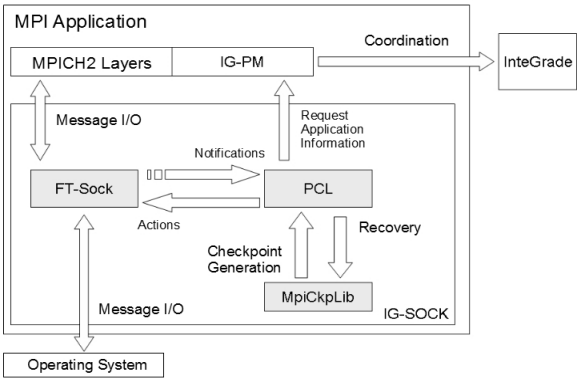

Figure 3. Architecture of the IG-Sock module.

application processes. The LRMs in turn fetch the application executable from the AR and the input data from the ASCT, before dispatching the local application process and notifying the execution to the ASCT. The next step, however, is specific to the execution of MPI applications, and is used to publish MPI-specific connection information to the several processes that compose the parallel application. This step is detailed in the sequence diagram of Figure 2 and described next for one particular node involved in the execution of the parallel application.

Once the LRM launches an application (1), the location and synchronization process is initiated on each participating process by the *MPI_Init* and *MPID_Init* calls (2, 3), through which MPICH2 initializes its control variables, buffers and data structures. CH3 then calls IGPM's *PMI_Init* function (4) to load application execution information. It then calls IG-Sock's *MPIDI_CH3_Init* (5), which initializes the necessary TCP/IP socket data structures. IG-Sock initiates a server socket and records its IP address and port number. It then calls IG-PM's *PMI_Barrier* (6) to synchronize the execution of the local process with the other processes that make up the parallel application. IG-PM then uses InteGrede's EM to publish the connection information received from the local MPI process by calling *registerMpiProcess* (7, 8). The EM waits until all processes of the MPI application are registered in this same way in order to publish all the collected connection information back to all of them. This modification takes advantage of EM's object-oriented architecture, which enables different application management strategies to be implemented by inheriting from the standard implementation.

4.2 IG-Sock and Rollback Recovery

In an opportunistic grid, resources are non-dedicated and can fail independently from each other, which may compromise long-running MPI applications. Support for some form of fault-tolerance is thus of paramount importance for MPICH-IG, which uses an approach based on checkpointing and recovery to avoid that applications need to be restarted from scratch after failures. The IG-Sock module implements a checkpoint strategy based on the MPICH-Pcl version of MPICH-V, a fault-tolerant implementation of MPI [10], for the creation of checkpoints based on distributed snapshots [2].

MPICH-Pcl enables checkpointing of processes that communicate over a sockets channel, including the state of the channel. It is extended by IG-Sock with functionality that is specific to InteGrade: to define when to generate checkpoints, to store checkpoints, and to recover the application after failures. Currently, IG-Sock employs a blocking protocol to obtain global checkpoints, although we plan to extend it with a more efficient non-blocking protocol.

The architecture of IG-Sock is composed of three components, as shown in Figure 3:

- FT-Sock: a re-implementation of MPICH2's channel interface; although it still uses TCP/IP sockets, it extends the original implementation to generate notifications when messages are sent and received, when connections are established, and when communication failures occur. These notifications are sent to the PCL component to coordinate the creation of checkpoints.
- PCL: implements the checkpoint protocol based on the notifications from FT-Sock and on the distributed snapshot protocol. It also uses IG-PM to obtain information about the application, notably the location to store checkpoints and the interval for their generation.
- MpiChpLib: this is the actual implementation of the checkpoint strategy; the two main options are system-level (currently implemented using the BLCR library [8]) and application-level checkpointing. This component is also responsible for reconstructing the application from a previously stored checkpoint, as seen next.

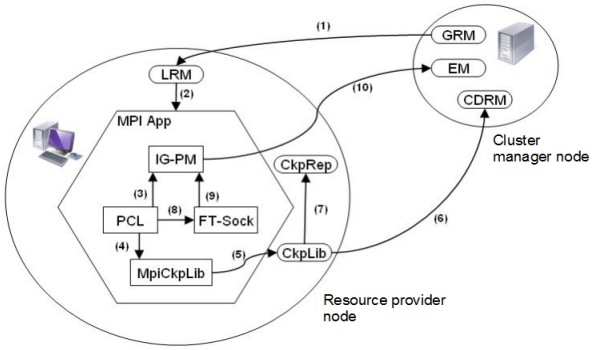

Figure 4. The application recovery protocol of MPICH-IG.

4.3 Recovery Protocol

The application recovery protocol is illustrated in Figure 4. In InteGrade, when an application fails and has to be recovered, it is re-scheduled for execution in a similar way as in the application execution protocol (steps 1 and 2), except that application state is recovered from a checkpoint using the checkpointing services of InteGrade (CDRM, CkpLib and CkpRep), in steps 5-7, instead of being initialized from input data provided by the ASCT. The components of IG-Sock are responsible for determining that it is a recovery of the application instead of its first launch ever. Finally, connection information is obtained from the EM by the FT-Sock component (steps 8-10).

5 Evaluation

In this section we provide a performance comparison of MPICH-IG and MPICH2. Due to the fact that both use a sockets channel for inter-process communication, we did not observe a major performance difference. In fact, most of the overhead of MPICH-IG is concentrated on the initial steps of application execution, due to the fact that the application binaries need to be fetched from the application repository before being launched locally on each machine.

A visible advantage of MPICH-IG is the use of InteGrade's scheduling to transparently selects the most appropriate resources, managing their allocation and use across the application's lifetime. In contrast, in MPICH2, the user explicitly needs to define which nodes to use, as well as to ensure that the application binaries are available on them. In addition, the ability to use widely dispersed nodes contributes to make larger amounts of resources available than would be possible in a centralized cluster.

Figures 5 and 6 show a comparison of the execution times for a parallel matrix multiplication application running on MPICH2 and MPICH-IG (without checkpointing) with two different input sizes. The figures also compare both results with the "ideal" execution time, i.e., when the application does not require interprocess communication.

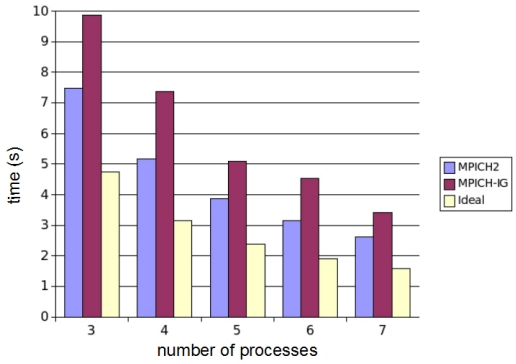

Figure 5. Comparing the execution time for multiplying two 1000x1000 matrices on MPICH2 and MPICH-IG.

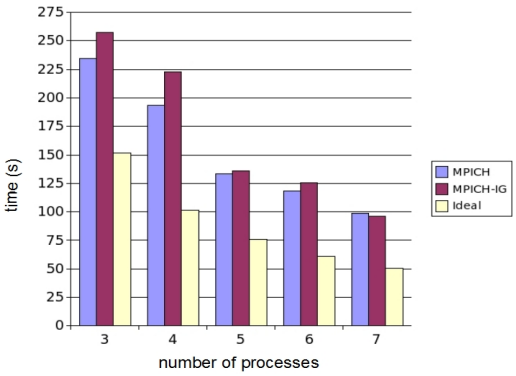

Figure 6. The same experiment for an input size of 3000x3000.

As can be seen, an MPICH-IG application usually takes longer to execute, although the absolute difference tends to become smaller as the number of processes rises (the percent difference seems to remain roughly the same though). The overhead is mainly due to the application execution protocol of MPICH-IG, as seen above, which requires the transfer of application binaries to the compute nodes. This is evidenced by the second experiment (Figure 6), in which the computation of the larger instance size tends to diminish the effect of the initial overhead.

6 Related Work

MPICH-G2 [9] is a port of MPICH to run on GTK4 grids. It is based on MPI-1 and uses GTK4's services for authentication, authorization, resource allocation and network I/O, as well as for the creation, monitoring and control of processes. Similarly to MPICH-IG, it re-implements the ADI interface. It also enables MPI applications to run across multiple organizational domains and enables the programmer to select the most appropriate communications topology for the application. Its main disadvantage is the absence of a recovery service, which means that applications have to be restarted from scratch in case of failures.

MPICH-GF [13] is an extension of MPICH-G2 to provide transparent checkpoint-based recovery. It uses a fixed checkpointing mechanism, based on TCP sockets and system-level checkpoints. This limits the degree to which MPI applications can be run on an heterogeneous grid. In contrast, MPICH-IG's modular architecture enables checkpoint strategies to be seamlessly implemented and replaced.

Other MPI implementations focus specifically on fault-tolerance and recovery. One example is MPICH-V [10], with its two versions, MPICH-Pcl, which uses blocking checkpoints, and the more efficient MPICH-Vcl, which uses non-blocking checkpoint. However, MPICH-V only runs on homogeneous clusters, although we used MPICH-Pcl as part of the checkpointing solution of MPICH-IG.

7 Final Remarks

MPICH-IG enables the execution of legacy MPI applications on opportunistic grids based on InteGrade. Its architecture and implementation are based on MPICH2, which has been adapted to use the resource management mechanisms of the grid and augmented with services that are required in the grid environment, notably an automatic mechanism to recover applications after failures. The modular architecture of MPICH-IG (which was influenced by the architecture of MPICH2) enables a clear separation between process management, communication and recovery from failures. As a result, new communication channels can be seamlessly added to MPICH-IG, such as one based on CORBA to deal with more heterogeneous grids.

A preliminary performance evaluation was carried out by comparing execution times in MPICH-IG and MPICH2. The results have shown that the main overhead of MPICH-IG (in the current implementation) is due to the initial steps of application execution (application staging), which are not present in MPICH2. This means that for long-running applications, the relative importance of this overhead becomes insignificant. Furthermore, the presence of a checkpointing mechanism enables more efficient recovery from failures since applications do not have to be restarted from scratch.

As future work, we plan to replace the blocking checkpoint protocol, currently provided by the PCL module, with a non-blocking checkpointing mechanism to optimize the generation of checkpoints. This implementation will demand an extension of the architecture, including a new module that will intercept received messages. We also plan to provide support for the complete set of functions of MPI2, as MPICH-IG currently does not provide support for the dynamic creation (spawning) of application processes. This implementation will require InteGrade to be extended with a protocol to support the scheduling of dynamic process. Finally, we aim to implement other kinds of communication channels, notably one based on CORBA to enhance interoperability in heterogeneous environments.

References

[1] D. Ashton, W. Gropp, E. Lusk, R. Ross, and B. Toonen. MPICH2 design document. *Draft, Mathematics and Computer Science Division, Argonne National Laboratory*, Oct 2003.

[2] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, Feb 1985.

[3] I. Foster, J. Geisler, W. Gropp, N. Karonis, E. Lusk, G. Thiruvathukal, and S. Tuecke. Wide-area implementation of the message passing interface. *Parallel Computing*, 24(12–13):1735–1749, 1998.

[4] I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, San Francisco, 2003.

[5] A. Goldchleger, F. Kon, A. Goldman, M. Finger, and G. C. Bezerra. InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines. *Concurrency and Computation: Practice and Experience*, 16(5):449–459, March 2004.

[6] W. Gropp, E. Lusk, D. Ashton, R. Ross, R. Thakur, and B. Toonen. MPICH abstract device interface. *Reference Manual, Mathematics and Computer Science Division, Argonne National Laboratory*, May 2003.

[7] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*, volume 1. The MIT Press, 1994.

[8] P. H. Hargrove and J. C. Duell. Berkeley Lab checkpoint/restart (BLCR) for linux clusters. *In Proceedings of SciDAC 2006*, Jun 2006.

[9] MPICH-G2. MPICH-G2 for Globus Toolkit. http://www3.niu.edu/mpi/, Jan 2008.

[10] MPICH-V. MPICH-V: MPI implementation for volatile resources. http://mpich-v.lri.fr/, Feb 2008.

[11] MPIForum. MPI-2: Extensions to the message-passing interface. *Technical Report*, Apr 1997.

[12] OMG. CORBA – common object request broker architecture. http://www.corba.org/, Jan 2008.

[13] N. Woo, H. Jung, D. Shin, H. Han, H. Y. Yeom, and T. Park. *Performance Evaluation of Consistent Recovery Protocols Using MPICH-GF*, pages 167–178. EDCC, Lecture Notes in Computer Science, Springer, Nov 2005.