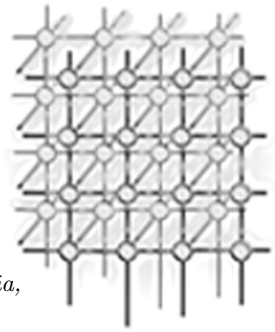
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2009; **00**:1–16 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

MPI Support on Opportunistic Grids based on the InteGrade Middleware



M. C. Cardoso and F. M. Costa[1] *,†

[1] *Institute of Informatics, Federal University of Goiás, Campus Samambaia, Goiânia-GO, 74.690-815, Brazil*

SUMMARY

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 MPI-2 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.

KEY WORDS: Grid computing; opportunistic grids; Message-Passing Interface; parallel applications

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 [1]. 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

*Correspondence to: Instituto de Informática, Universidade Federal de Goiás, Campus Samambaia, Goiânia-GO, 74.690-815, Brazil
†E-mail: fmc@inf.ufg.br
Contract/grant sponsor: CNPq–Brazil; contract/grant number: 55.0895/2007-8
Contract/grant sponsor: FAPEG–Goiás–Brazil; contract/grant number: Chamada 02/2007
Contract/grant sponsor: FINEP–Brazil; contract/grant number: 01.08.0166.00

Received 01 March 2009
Revised 10 May 2009

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 for running 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 re-implementing parallel programming libraries so that they use the grid middleware services instead of their own native services to manage application execution.

In this paper, we present MPICH-IG, an implementation of the MPI-2 standard [2] for opportunistic grids. MPICH-IG is based on MPICH2 [3], which has a modular architecture that allows reuse of several of its components, as well as clean replacement of those components related to the management of application execution, essential to effectively use the resources of a computing grid. The main feature of MPICH-IG is the ability to run native MPI applications on resources that are used in an opportunistic way. This is complemented by fault tolerance mechanisms that enable individual tasks of an MPI application to be recovered after failure, based on previously taken checkpoints. MPICH-IG is built on top of the InteGrade platform, an object-oriented grid computing middleware based on CORBA (Common Object Request Broker Architecture) [4] and aimed at opportunistic grids that are built using general purpose, non-dedicated, computers [5].

The paper is structured as follows. Section 2 discusses the architecture of MPICH2, with emphasis on the features that enable its use to implement MPICH-IG. Section 3 reviews the overall architecture of the InteGrade grid computing middleware, with a focus on application execution management. Section 4 presents the architecture and implementation of MPICH-IG, including its fault-tolerance features, while Section 5 discusses some evaluation results and Section 6 considers related work. Section 7 presents final remarks and considerations.

2. MPI AND MPICH2 ARCHITECTURE

The Message Passing interface (MPI) has become the *de facto* standard for parallel applications programming [6]. It provides a comprehensive set of functions for the management and communication of parallel processes, also enabling a level of distribution transparency to the developer. The first version of the MPI standard was released in 1995 and contains functions for point-to-point communication (both blocking and non-blocking), definition and management of process groups, group communication, and query about the execution state of processes. The second version of the standard, MPI-2, released in 1997, augmented the interface with two extra set of functions, respectively for the support to distributed shared memory and for the dynamic spawning of parallel processes [2].

Several implementations of the standard exist, among them the open source MPICH2 [3]. One of the main characteristics of MPICH2 is its modular, layered, architecture, which clearly

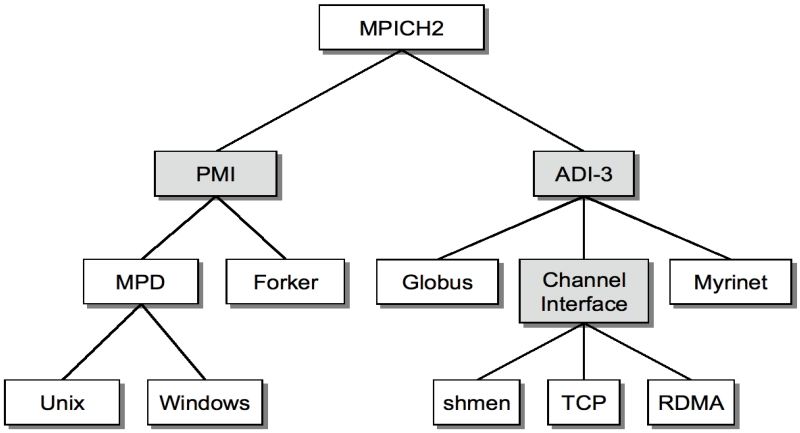


Figure 1. The layered structure of MPICH2.

separates the implementation of the high level protocols and functions of MPI-2 from the low-level mechanisms used for interprocess communication and process management. This makes it easier to port MPICH2 to different 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 MPI-2 functions [7]. 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. The layered architecture of MPICH2 is presented in Figure 1, which shows the structure of ADI-3, the latest implementation of the ADI.

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*, which in turn may be implemented using mechanisms such as shared memory, Remote Direct Memory Access (RDMA) and TCP sockets. All higher-level communications features are implemented in terms of these low-level primitives.

Another important module of the architecture, also shown in Figure 1, is represented by 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 Multipurpose 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. Another common implementation of the PMI interface is the *forker* process management system, which simply starts MPI tasks as processes on a single machine. In this work, we propose an altogether different implementation of the PMI, based on InteGrade's mechanisms for managing application execution, as seen in Section 4.

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 MPI-2 [8]. 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 MPI-2 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, Bulk Synchronous Parallelism (BSP), 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 in [5].

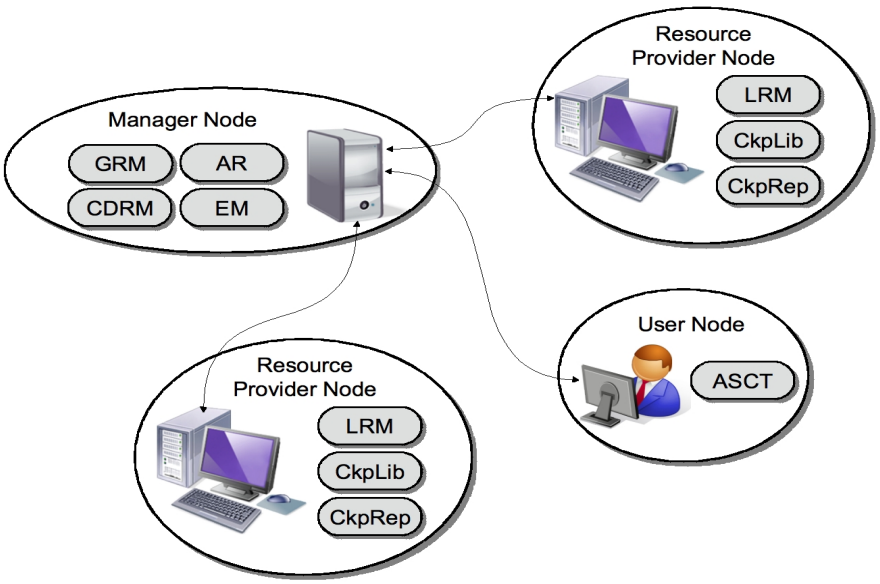


Figure 2. Software components in an InteGrade cluster.

InteGrade architecture is based on CORBA [4], which is the communications middleware that enables interaction among InteGrade components. This includes the use of standard CORBA services, most notably the trading service, which is used to discover instances of the components (see below) that make up the grid support infrastructure. The basic structural unit of an InteGrade grid is the cluster, which may contain both dedicated and shared (i.e., used by other applications) computing 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. An InteGrade grid is thus comprised of a potentially large federation of clusters, which may be structured in a hierarchical way, further contributing to its scalability, as described in [5]. As shown in Figure 2, 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 of InteGrade, with respect to the implementation of MPICH-IG, are described next. The checkpoint-related components, as, which are of particular importance in this work, are further described in [9].

- 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 computer 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): this component maintains information about the checkpoint repositories (CkpRep) of a cluster. Such information is provided to the checkpointing library (CkpLib) when it needs to store or retrieve application checkpoints.
- CkpLib (Checkpoint Library): enables checkpointing and recovery of the applications running on a given resource provider node. The checkpoint library interacts with CDRM to obtain the list of repositories available to store checkpoints in the cluster. Importantly, its architecture enables the decoupling of the actual library used to generate checkpoints and the mechanism used to store and retrieve such checkpoints. In this way, InteGrade is able to support a variety checkpoint generation strategies.
- CkpRep (Checkpoint Repository): also called ADR (Autonomous Data Repository), this component stores checkpoint data and enables them to be retrieved when the application needs to be recovered.
- EM (Execution Manager): manages application execution by monitoring the nodes where applications are run. It sends notifications when the application finishes, and coordinates the reinitialization process when an application fails.
- ASCT (Application Submission and Control Tool): GUI-based application used to submit, monitor and collect the results of application execution.

These components communicate using two CORBA-compliant ORBs, namely JacORB [10] for the components written in Java, and OiL (ORB in Lua) [11, 12] for those written in Lua or C/C++. MPICH-IG, described in the next section, is implemented as a set of components that extend and complement this original InteGrade architecture. In particular, the checkpoint and recovery components of InteGrade, originally designed for BSP applications, have been adapted, together with MPICH2, in order to provide such fault-tolerant behavior for MPI applications as well. MPICH2 has also been modified in order to seamlessly use the application management protocols implemented by the GRM and EM components.

4. MPICH-IG

As seen above, the approach we adopted for the implementation of MPICH-IG was to modify an existing MPI-2 library, MPICH2, instead of significantly 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.

Figure 3 shows how MPICH-IG fits in the InteGrade architecture as a layer on top of which MPI applications run. As the figure suggests, as far as InteGrade is concerned, the MPI support provided by MPICH-IG is considered as part of the grid application. In fact, when launching an MPI-based grid application, the whole package comprised by MPICH-IG and the application itself is fetched from the application repository and put to run on each of the selected grid nodes. As discussed in Section 5, this represents an initial overhead in terms of the overall application execution time. Nevertheless, it has the advantage of not requiring the machines

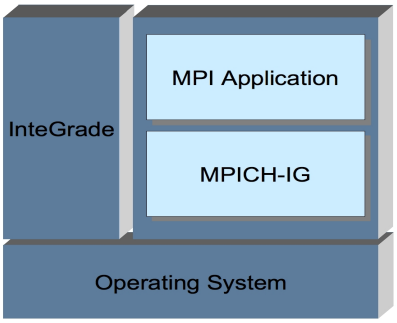


Figure 3. Layered structure for the integration of MPICH-IG and InteGrade.

on the grid to be previously prepared to support MPI applications. All that is required from those machines is the usual InteGrade support. As a further consequence of this approach, InteGrade becomes independent of the particular MPI-2 implementation in use. For instance, we could replace MPICH2 with LAM/MPI [13] without having to change InteGrade (LAM MPI would have to be adapted instead).

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 (the original implementation of the CI) and MPD (the multipurpose daemon, original implementation of the PMI), as shown in Figure 4. As the figure (part b) suggests, InteGrade is used as the support for process management, enabling grid resources to be transparently scheduled to run MPI applications. The communications channel, in turn, is still based on TCP sockets for the sake of efficiency (the changes introduced by IG-Sock are mostly concerned with instrumenting the channel for application checkpointing and recovery). A CORBA-based implementation of the channel is considered for future work, enabling better interoperability and the use of CORBA's firewall/NAT traversal mechanisms for inter-cluster communication.

Regarding changes to the InteGrade middleware, the approach followed by MPICH-IG requires only a minor modifications to the Execution Manager (EM) and ASCT components. The EM has been changed in order to enable it to collect connection information from the MPI application's processes during the synchronization phase (right after application launch). Such information is provided by each MPI processes in the form of *key:value* pairs, which

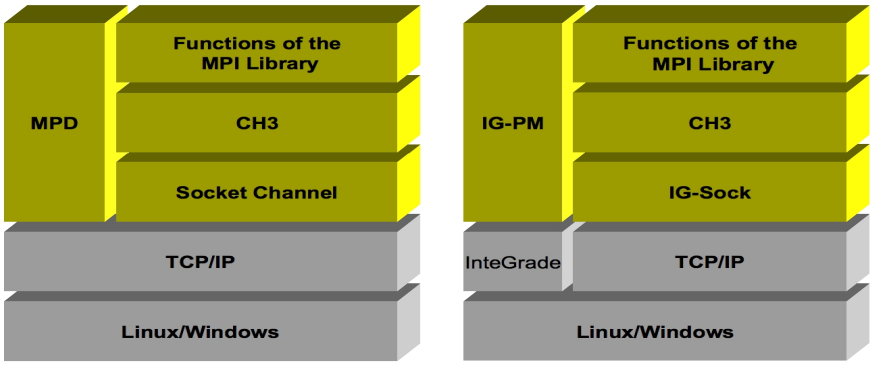


Figure 4. Differences between the standard implementation of MPICH2 (a) and MPICH-IG (b).

contain, among other things, the IP address and port number that identify the process. This information, however, is treated as opaque data by the EM, which simply relays it to each and every process comprising the MPI application. Finally, the change to the ASCT component consists simply in adding an extra option in its GUI (Graphical User Interface) to allow the user to request the execution of MPI applications, as shown in Figure 5

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 in order 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 daemon. However, MPD does not deal with resource heterogeneity and failures, which are common in 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:

- 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);

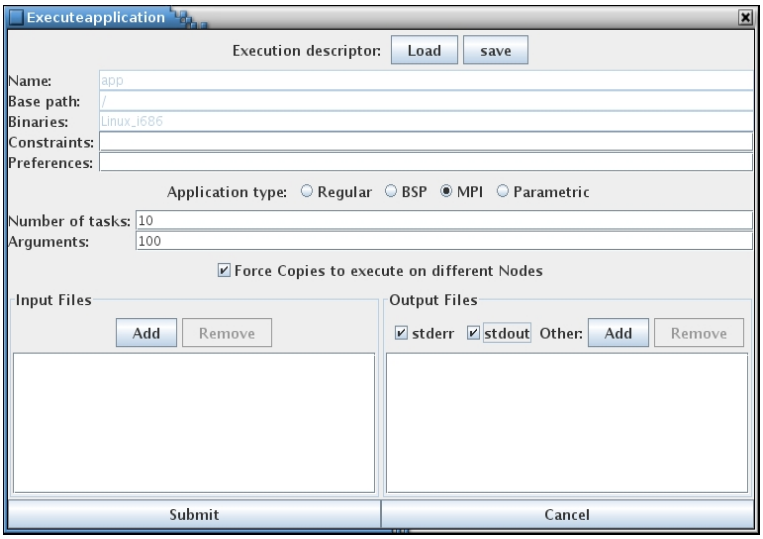


Figure 5. ASCT GUI interface with new option for MPI applications.

- 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 about 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 ASCT to 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 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 6 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-

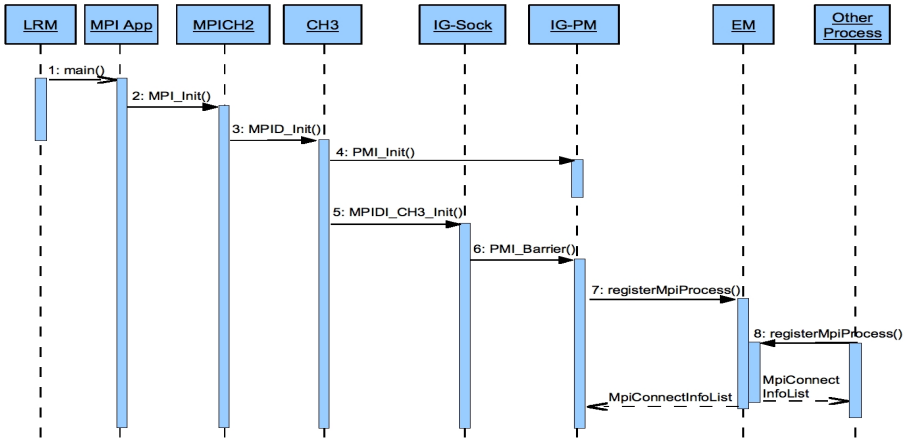


Figure 6. Execution of an MPI application on InteGrade.

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 of the EM is the only implementation change that is required on InteGrade. It 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

In an opportunistic grid, resources are non-dedicated and can fail independently from each other, which may compromise long-running MPI applications. Support for 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 [14], for the creation of checkpoints based on a distributed snapshot protocol to determine consistent cuts among the application's processes [15].

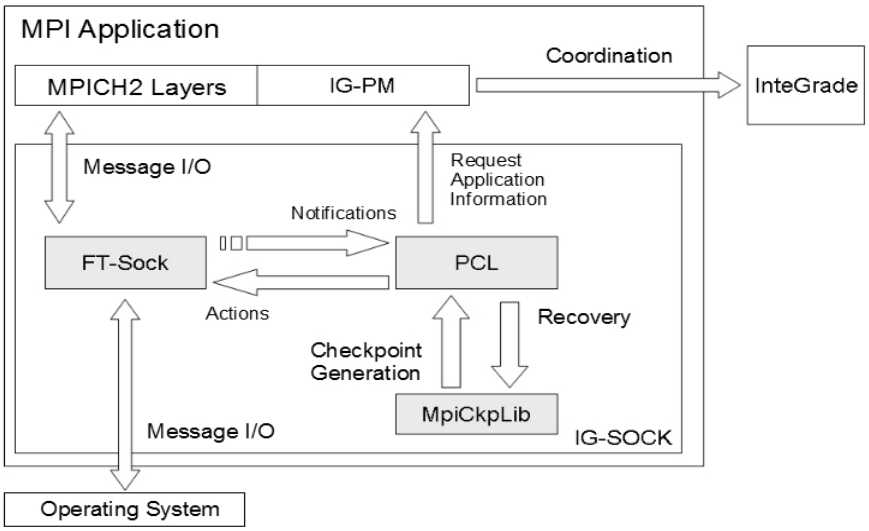


Figure 7. Architecture of the IG-Sock module.

MPICH-Pcl enables checkpointing of processes that communicate over a sockets channel. Checkpoints include not only the local state of the processes themselves, but also the state of the communications channel. IG-Sock extends MPICH-Pcl with features that are specific to InteGrade: to ability to determine when to generate checkpoints; checkpoint storage; and application recovery after failures. Currently, IG-Sock employs a blocking protocol to obtain global checkpoints, although we plan to re-implement it with a more efficient non-blocking protocol in the near future.

The architecture of IG-Sock is composed of three components, as shown in Figure 7:

- 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 protocol to find consistent cuts based on the notifications it receives from FT-Sock and on the distributed snapshot protocol [15]. It also uses IG-PM to obtain information about the application, notably the location to store checkpoints and the interval for checkpoint generation.
- MpiCkpLib: this is the actual implementation of the checkpoint strategy; the two main options are system-level, currently implemented by wrapping the Berkeley Lab Checkpoint/Restart (BLCR) library [16], and application-level checkpointing (not

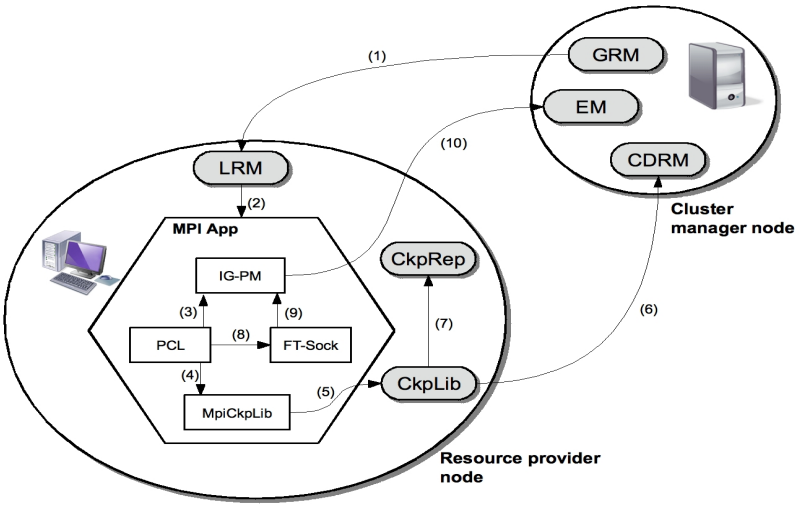


Figure 8. The application recovery protocol of MPICH-IG.

currently implemented). This component is also responsible for reconstructing the application from a previously stored checkpoint, as seen next.

As a result MPICH-IG has a modular implementation of checkpoints, meaning that each of these components can be implemented in different ways, with different protocols and mechanisms, without compromising the other components. For instance, the checkpoint generation protocol can be changed by replacing BLCR with a different (e.g., application-level) checkpoint strategy. As another example, the implementation of FT-Sock can be replaced with a CORBA-based channel, provided that the latter also sends the usual notifications to the PCL component.

4.3. Recovery Protocol

The application recovery protocol is illustrated in Figure 8. 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 the 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).

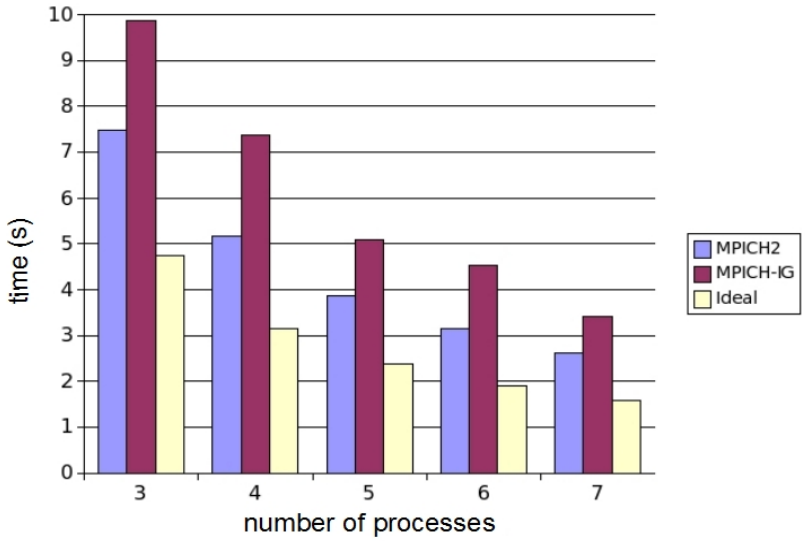


Figure 9. Comparing the execution time for multiplying two 1000x1000 matrices on MPICH2 and MPICH-IG.

5. EVALUATION

In this section we provide a performance comparison of MPICH-IG and MPICH2. Due to the fact that both use a socket-based 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. In contrast, in MPICH2, the application binaries must be previously installed on each node.

A visible advantage of MPICH-IG is the use of InteGrade's scheduling to transparently select 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 9 and 10 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. To put the results in perspective, the figures also show a hypothetical case (called "ideal") when the execution time does not include the overhead of managing application execution (i.e., considering the same set of tasks, but with all tasks running independently of each other).

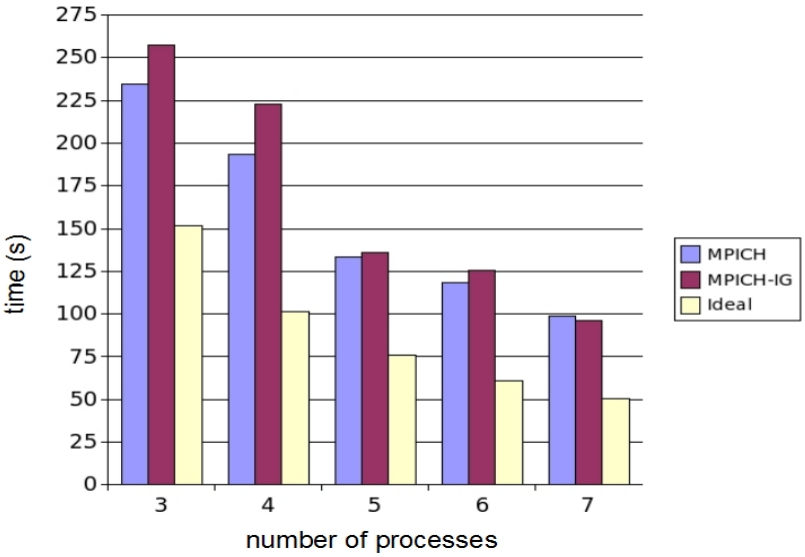


Figure 10. The same experiment for an input size of 3000x3000.

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 remains 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 computing nodes. This becomes clear in the second experiment (Figure 10), in which the computation of the larger instance size diminishes the effect of the initial overhead. This latter result is particularly promising as computer grids are typically used to run long-running applications.

Further experiments are part of our ongoing work and include an evaluation of the checkpointing and recovery mechanisms, as well as the use of MPICH-IG to execute long-running applications in more realistically sized grids.

6. RELATED WORK

MPICH-G2 [17] is a port of MPICH to run on GTK4 (Globus Toolkit version 4) 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 [18] 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.

A few MPI implementations focus specifically on fault-tolerance and recovery. One example is MPICH-V [19], 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. Another example is the fault-tolerance extension of Open MPI [20]. It takes advantage of the modular, framework-based, architecture of Open MPI to enable the seamless integration of different alternative strategies for checkpoint and recovery. In this respect it pursues a similar goal as MPICH-IG. Nevertheless, Open MPI is meant for use in dedicated clusters, as opposed to opportunistic grid settings.

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 performance evaluation was carried out by comparing execution times of a simple application in MPICH-IG and MPICH2. The results have shown that the main overhead of MPICH-IG is due to the initial steps of application execution (application staging), which are not present in MPICH2. We believe that for long-running applications the relative importance of this overhead will become less significant. Such overhead can also be compensated by the ability to run MPI applications over a potentially larger set of resources (resulting from the opportunistic harvesting of idle, non-dedicated, machines) than it would be possible in more conventional MPI settings, usually restricted to local dedicated clusters. 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 MPI-2, 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 new tasks that were created after the initial scheduling of a grid application. Finally, we aim to implement other kinds of communication channels, notably one based on CORBA to enhance interoperability in heterogeneous environments.

The current release of MPICH-IG can be obtained from http://www.integrade.org.br, as part of the InteGrade distribution.

REFERENCES

1. Foster I, Kesselman C. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers: San Francisco, 2003.
2. Huss-Lederman S. MPI-2: Extensions to the message passing interface. *Technical Report*, MPI Forum July 1997.
3. Gropp W. MPICH2: A new start for MPI implementations. *Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Springer-Verlag: London, UK, 2002; 7.
4. OMG. *The Common Object Request Broker Architecture (CORBA/IIOP)*. Object Management Group, Needham, MA, rev. 3.1 edn. January 2008.
5. Goldchleger A, Kon F, Goldman A, Finger M, Bezerra GC. InteGrade: Object-Oriented Grid Middleware Leveraging Idle Computing Power of Desktop Machines. *Concurrency and Computation: Practice and Experience* March 2004; **16**(5):449–459.
6. Gropp W, Lusk E, Skjellum A. *Using MPI: Portable Parallel Programming with the Message Passing Interface*, vol. 1. The MIT Press, 1994.
7. Thakur R, Gropp W, Lusk E. An abstract-device interface for implementing portable parallel-I/O interfaces. *Proc. of the 6th Symposium on the Frontiers of Massively Parallel Computation*, Annapolis-MD, 1996; 180–187.
8. Foster I, Geisler J, Gropp W, Karonis N, Lusk E, Thiruvathukal G, Tuecke S. Wide-area implementation of the Message Passing Interface. *Parallel Computing* 1998; **24**(12–13):1735–1749.
9. de Camargo RY, Kon F, Cerqueira R. Strategies for checkpoint storage on opportunistic grids. *IEEE Distributed Systems Online* 2006; **7**(9):1.
10. Brose G, Vogel A, Duddy K. *Java Programming with CORBA, Third Edition*. John Wiley & Sons, Inc.: New York, NY, USA, 2001.
11. Distributed Systems Group. OIL – The Lua Object Request Broker. http://oil.luaforge.net/ (Accessed: August 27, 2009).
12. Maia R, Cerqueira R, Kon F. A middleware for experimentation on dynamic adaptation. *ARM '05: Proceedings of the 4th workshop on Reflective and adaptive middleware systems*, ACM: New York, NY, USA, 2005.
13. Burns G, Daoud R, Vaigl J. LAM: An Open Cluster Environment for MPI. *Proceedings of Supercomputing Symposium*, Toronto-Canada, 1994; 379–386.
14. MPICH-V. MPICH-V: MPI implementation for volatile resources. http://mpich-v.lri.fr/ (Accessed: August 27, 2009).
15. Chandy KM, Lamport L. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems* Feb 1985; **3**(1):63–75.
16. Hargrove PH, Duell JC. Berkeley Lab checkpoint/restart (BLCR) for Linux clusters. *Proceedings of SciDAC 2006 – Scientific Discovery Through Advanced Computing*, Denver, 2006; 494–499.
17. Karonis NT, Toonen B, Foster I. MPICH-G2: A grid-enabled implementation of the message passing interface. *Journal of Parallel and Distributed Computing* May 2003; **63**(5):551–563.
18. Woo N, Jung H, Shin D, Han H, Yeom HY, Park T. Performance evaluation of consistent recovery protocols using MPICH-GF. *EDCC*, *Lecture Notes in Computer Science*, vol. 3463, Cin MD, Kaniche M, Pataricza A (eds.), Springer, 2005; 167–178.
19. Bosilca G, Bouteiller A, Cappello F, Djilali S, Fedak G, Germain C, Herault T, Lemarinier P, Lodygensky O, Magniette F, *et al.*. MPICH-V: toward a scalable fault tolerant MPI for volatile nodes. *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, IEEE Computer Society Press: Los Alamitos, CA, USA, 2002; 1–18.

20. Hursey J, Squyres JM, Mattox TI, Lumsdaine A. The design and implementation of checkpoint/restart process fault tolerance for Open MPI. *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE Computer Society, 2007.