Application-Level Fault-Tolerance Solutions for Grid Computing

Daniel Díaz, Xoán C. Pardo, María J. Martín, Patricia González
Computer Architecture Group
University of A Coruña
ddiax@udc.es, pardo@udc.es, mariam@udc.es, pglez@udc.es

Abstract

One of the key functionalities provided by Grid systems is the remote execution of applications over the Grid. We present a research proposal on fault-tolerance mechanisms for the execution of message-passing parallel applications on the Grid. An architecture called CPPC-G is proposed, consisting of a set of services built on top of the Globus Toolkit. The CPPC (Controller/Precompiler for Portable Checkpointing) framework is used to insert checkpointing instrumentation into the application code. CPPC-G services will be in charge of the submission and monitoring of the application's execution, management of checkpoint files generated by CPPC-enabled applications, and detection and automatic restart of failed executions.

1. Introduction

The need of organizations to have access to computing resources beyond their own, to share data and computing power with other organizations without losing local control over their own resources, and to maximize the use of these resources, has gave rise to the field of Grid Computing.

A Grid system [7] coordinates resources that are not subject to centralized control, using standard, open, general-purpose protocols and interfaces, to deliver nontrivial qualities of service. There are several middleware tools that can provide the infrastructure for a Grid system. One example of such Grid middleware is the Globus Toolkit [8]. It follows the Open Grid Services Architecture (OGSA) [9], and provides services for remote execution, data transfer, and Grid-level scheduling.

A Grid can be harnessed for the execution of long and complex applications that would tax the resources of a single organization. Participants in a Grid can find on it computing resources suitable for their needs and remotely execute the application on these resources. They can also distribute the execution over several nodes of the Grid. The presence of Grid-level schedulers can simplify these tasks.

As the number of machines of a distributed system increases, so does the failure rate of the global system. This is not a problem while the mean time to complete an application execution remains well under the Mean Time To Failure (MTTF) of the underlying hardware. Unfortunately, that is not always true for applications with large execution times. Under these circumstances, some kind of fault tolerance mechanism becomes essential to ensure that not all the computation done is lost in machine failures. This is especially true for Grid contexts, where the reliability of remote machines can not be taken for granted and, furthermore, communication problems may also arise.

The availability on the Grid of machines with parallel hardware, and the distributed nature of the Grid itself, makes most of the compute-intensive applications executing over the Grid be parallel programs, more precisely message-passing programs. Achieving fault-tolerance for parallel/distributed applications involves a set of additional concerns, such as more complex checkpoint management and ensuring data consistency.

2. Research proposal

The heterogeneity of the components represents a challenge for the remote execution of applications over the Grid. This challenge is also faced by any fault-tolerance mechanism implemented over the Grid. For example: if an application fails, perhaps because the computing node executing it goes down, how can we ensure that we can successfully restart the application on a different computing node, possibly with different hardware, over the Grid? And even if we restrict our selection to those nodes with compatible hardware/software, how can we reliably identify such nodes over the Grid?

The Grid heterogeneity problem also affects transient data files created by some fault-tolerance mechanisms, such as checkpointing. Checkpoints created by an application in some particular machine should allow the restarting of the application on a different machine with different hardware.

The management of transient data created by a faul-
tolerance mechanism—such as checkpoint files—is also an open question. The end user is not interested in them per se, but only as a means for completing the application’s execution. So, ideally, their management should be as automated and transparent to the user as possible.

Our proposal consists of a set of Globus-based grid services for the fault-tolerant execution of message-passing parallel applications.

The services will leverage the functionality of CPPC [18], a library for the checkpointing of message-passing parallel applications. This library performs distributed checkpointing, that is, each process in the parallel application generates its own sequence of checkpoint files independently from the other processes. After an execution failure, a consistent set of checkpoint files—one for each process—is selected and execution continues from them. One important characteristic of CPPC is that the generated checkpoint files are portable: they are written in a portable format and, furthermore, they only include portable application state. Non-portable state is recovered during restarts through controlled re-execution of the application code.

The proposed services will be built on the checkpointing capabilities of CPPC, providing services for the automatic handling of checkpoint files generated by CPPC-enabled applications being executed on the Grid. Checkpoints generated on the local execution machine will be replicated on a safe backup location, so that if the execution machine goes down, the checkpoint files will be still accessible.

The user will not need to specify the location of checkpoint repositories; they will be automatically located by our services in a manner transparent to the user. The checkpoint metadata generated for each execution will also be collected and processed. Checkpoint files will be automatically deleted once they aren’t needed anymore, either because the application finished successfully, or because more recent checkpoint files have rendered the previous ones obsolete.

Some mechanism for automatic selection of suitable computing resources on the Grid will be provided, and also for automatic restart of failed applications. Failures in the CPPC applications and in the services themselves would be detected. Apart from heterogeneity and data management, there’s also the problem of how to reliably detect execution failures over the Grid, ensuring that we can both detect failures and avoid false positives with reasonable accuracy. The ways in which a set of services for fault-tolerant execution of applications could be made resilient to errors (not only of the applications whose execution is managed, but also of the services themselves to some degree) should also be studied. Some form of replication will be used to ensure the robustness of the architecture.

The proposed services will be built on top of the standard Globus services, like WS-GRAM [13] for execution management and Reliable File Transfer [12] for data movement. Reimplementation of already existing functionality should be avoided. For example, leveraging existing Grid-level schedulers and resource brokers for node selection and execution restart should be considered.

The availability of a robust set of Grid services for the fault-tolerant execution of message-passing parallel programs would make the execution over the Grid of these kinds of complex applications more reliable, widening the range of problems for which Grid computing could be consistently harnessed.

More organizations could use the Grid to efficiently access and utilize parallel hardware available on remote sites, and also to offer their own clusters for the execution of compute-intensive parallel programs, thereby avoiding idle periods on the hardware they own.

Besides, the Grid does not only present the traditional fault tolerance use for checkpoint techniques, but also shows a new use-case for the checkpoint-and-restart techniques: opportunistic computing, where the computational power is provided by the idle resources of all participants. When a resource is not used by its owner, it is said to be available state. Otherwise it is said to be in not available state. To be non intrusive to the resource owners, a machine only can be used when it is in available state. In these environments the resources change very often from the available state to the not available one, being then necessary to provide checkpoint mechanisms to ensure execution ends.

Also, sophisticated error detection techniques for Grid services have a reach beyond remote execution of applications: all kinds of Grid services could benefit from them.

Finally, automated checkpoint management touches on the wider field of replication, access, and management of data on Grids.

3. Related work

Important effort has been made in developing middleware to provide Grids with functionalities related to application execution. However, support for fault tolerant execution is either lacking or limited. WS-GRAM [13], the execution manager of the Globus Toolkit, handles file transfer before and after an application’s execution, but offers no handling of the checkpoint files generated while the execution is underway. Other fault tolerance-related functionalities are absent, too. GridWay [6] is a Grid-level scheduler for use with the Globus Toolkit (interfacing with other Grid systems is also possible). It handles the search for nodes that are suitable to the user’s needs. It allows for a form of distributed computing, in the form of task work flows, but it is not geared for compute-intensive message passing parallel applications. It offers checkpointing support for indi-
vidual processes, but the checkpointing of message-passing parallel applications, with its additional requirements, is not supported.

Several approaches for the implementation of fault tolerance in message-passing applications exist. MPICH-GF [21] is checkpointing system based on MPICH-G2 [16], a Grid-enabled version of MPICH. It handles checkpointing, error detection, and process restart in a manner transparent to the user. But, since it is a particular implementation of MPICH, it can’t be used with other message-passing frameworks. The checkpointing is performed at a data segment level, thus generating non-portable files. To achieve global consistency, MPICH-GF uses process coordination, which is a non-scalable approach.

MPICH-V2 [2] is another checkpointing system implemented as a MPICH driver. As with MPICH-GF, it works at the segment level and generates non-portable files. Global consistency is achieved through message logging, another non-scalable approach.

CLIP [4] is a Libckpt-based [17] implementation for Intel Paragon architectures. Thus, file portability is not a concern in this context. It uses process coordination to deal with global consistency, flushing message buffers and storing them along with the state file. The disadvantage of CLIP is that it is tied to Intel Paragon architectures.

CoCheck [19] was developed for the Condor system [20]. It works at the segment level, thus discarding file portability. It uses process coordination to achieve global consistency, thus being a non-scalable approach.

$C^3$ [3] is a checkpoint compiler that works at the variable level, implemented over the MPI library. Thus the code is independent of the MPI implementation. It uses a newly developed protocol based on the Chandy-Lamport protocol [7] to achieve global consistency. Scalability of this protocol is under study. Portability is not one of its goals and, although it does not store any internal MPI state, enforces data to be recovered on the same virtual location than on the original execution to achieve pointer consistency, thus making impossible architectural changes and file portability.

CPPC [18] is a checkpointing tool focused on the insertion of fault-tolerance into long-running message-passing applications. It is designed to allow for execution restart on different architectures and/or operating systems, also supporting checkpointing over heterogeneous systems, such as the Grid. It uses portable code and protocols, and generates portable checkpoint files while avoiding traditional solutions such as process coordination or message-logging, which add an unscalable overhead. The checkpointing is performed at the variable level. CPPC is not tied to any particular message-passing framework. We have selected CPPC as the basis for our own project due to these qualities of portability, scalability, and compatibility with multiple message-passing frameworks.

There have been a number of initiatives towards achieving fault-tolerance on Grids. The Grid Checkpoint and Recovery (GridCPR) Working Group [10] of the Global Grid Forum is concerned with defining a user-level API and associated layer of services that will permit checkpointed jobs to be recovered and continued on the same or on remote Grid resources. A key feature of Grid Checkpoint Recovery service is recoverability of jobs among heterogeneous Grid resources.

The CoreGrid checkpointing work group of the CoreGrid Network of Excellence has proposed [14] an alternative Grid checkpointing architecture. The difference with the GridCPR proposal is that the latter assumes that the checkpointing tool should be a part of the application, and would be tightly connected to various Grid services such as communication, storage, etc. In the CoreGrid proposal, checkpoints are system-level and external to the applications.

MIGOL [15] is a fault-tolerant and self-healing grid middleware for MPI applications built on top of the Globus Toolkit. MIGOL supports the migration of MPI applications by checkpointing and restarting the application on another site. However, as for now the current version of the middleware depends of locally stored checkpoints, which have to be accessible after an execution failure to enable auto-recovery. No checkpoint replication is performed. This means that if the machine goes down or becomes otherwise inaccessible, application execution must again start from the beginning.

Regarding checkpoint file storage, work has been done for opportunistic Grids [15]. More precisely on InteGrade, a project for leveraging the computing power of idle workstations. It would be interesting to study checkpoint storage and management for other kinds of Grids.

4. Preliminary Work

A proof-of-concept set of services has already been developed, which is called CPPC-G. The architecture of the system is shown in Figure 1.

The dark circles represent services which are part of CPPC-G, the white circles represent standard Globus services. In the top-right part of the diagram we see a computing node on the Grid where a CPPC-enabled application is being executed, which generates checkpoint files in the local file system.

The FaultTolerantJob service is the one the users contact to request the fault-tolerant execution of a CPPC-enabled application. When it receives a request, it first allocates a CkptWarehouse resource for managing the checkpoint metadata, and proceeds to obtain a suitable execution node by querying an scheduler service. It then starts a checkpointed execution on the allocated node, and moni-
In case the execution fails, FaultTolerantJob requests another node from the scheduler and tries to restart the application there from the last saved state.

SimpleScheduler is a service which aggregates information about nodes on the Grid on which the service for checkpointed execution CkptJob has been deployed and is active. The information is obtained by querying the standard Globus Monitoring and Discovery System (MDS) [11]. Asides from the identity of the nodes, SimpleScheduler also maintains information about a set of tags associated to each node, and allows to select nodes using simple boolean queries over those sets of tags.

CkptWarehouse is a service which aggregates metadata about the checkpoint files generated by the execution of a CPPC-enabled application. Metadata such as which process generated which file, the sequence number of the file among those generated by the process, the URL on which a replica of the checkpoint file is accesible, and so on. Based on that metadata, CkptWarehouse determines from which sets of checkpoint files to restart the application in case of failure. Also, it detects which checkpoint files have become obsolete, and deletes them.

CkptJob is the service which allows for the checkpointed execution on the Grid of CPPC-enabled message-passing parallel applications. It delegates on Globus WS-GRAM the execution of the application while adding the needed checkpoint handling functionality. When dealing with application restarts after a failure, CkptJob depends on CkptWarehouse to obtain a consistent set of checkpoint files from which to restart execution. Conversely, every checkpoint file that is generated by the application is eventually registered on a CkptWarehouse resource by CkptJob. Upon activation, CkptJob registers itself with the Globus MDS service of the local machine, to allow other users and services (particularly SimpleScheduler) to find it. The registration information includes a series of tags specified by the administrator, which could be used to specify the hardware/software configuration of the local computing resource.

The last service in CPPC-G is StateExport. This is a service closely associated to CkptJob. It is tasked with detecting the checkpoints generated by the CPPC-enabled applications. After a checkpoint has been detected, it is replicated on a safe backup location specified beforehand by the user. Once the replication process is completed, StateExport notifies CkptJob, which in turn registers the replicated checkpoint file with CkptWarehouse.

The modular architecture of CPPC-G allows the user to ignore the higher-level services (FaultTolerantJob, SimpleScheduler) and invoke CkptJob directly, if he wishes perform the monitoring and eventual restarts by himself.

More details about the CPPC-G architecture can be found in [5].
5. Future work

The work done until this point is only the first step in our research. CPPC-G services are currently very basic; they will serve as a springboard for future research and development. Currently, the user must find and specify a safe backup location for the checkpoints; this should be handled by the CPPC-G services themselves. The monitoring of the applications is done by periodically polling the state of the computing resource, more sophisticated heuristics for determining failure states should be employed. Also, the services which form CPPC-G are not resilient to errors; some form of replication -combined with a persistence mechanism- should be employed to make them more robust.

Another further work would be the integration of CPPC-G with other Grid systems -not necessarily based on Globus- like OurGrid [1]. Grids following this framework differ from Globus Grids in that the system is seen as a peer-to-peer network of sites, where access is granted based on the amount of computing power offered for sharing, instead of requiring explicit administrative permission. OurGrid is oriented towards bag-of-tasks applications, that is, applications composed of a set of independent tasks that neet no communication among themselves during execution.

CPPC-G services will provide a fault-tolerance mechanism for parallel message-passing applications which Globus currently lacks. Built on top of already existing Globus services and having a modular architecture, CPPC-G will not require changes in the standard Globus services, and will give users ample choice about which subset of CPPC-G’s functionality they wish to use.

CPPC-G should make easier to resort to the Grid for the execution of long and complex parallel applications. It’s modularity and emphasis on transparency to the user -for example, the automatic selection of checkpoint backup sites- are desirable characteristics for future Grid services.

References