A DISTRIBUTED AND REPLICATED SERVICE FOR CHECKPOINT STORAGE

Fatiha Bouabache
Laboratoire de Recherche en Informatique
Universite Paris Sud-XI, 91405 ORSAY, FRANCE.
fatiha.bouabache@lri.fr

Thomas Herault
INRIA Futurs/Laboratoire de Recherche en Informatique
Universite Paris Sud-XI, 91405 ORSAY, FRANCE.
thomas.herault@lri.fr

Gilles Fedak
INRIA Futurs/Laboratoire de Recherche en Informatique
Universite Paris Sud-XI, 91405 ORSAY, FRANCE.
fedak@lri.fr

Franck Cappello
INRIA Futurs/Laboratoire de Recherche en Informatique
Universite Paris Sud-XI, 91405 ORSAY, FRANCE.
fci@lri.fr

Abstract As High Performance platforms (Clusters, Grids, etc.) continue to grow in size, the average time between failures decreases to a critical level. An efficient and reliable fault tolerance protocol plays a key role in High Performance Computing. Rollback recovery is the most common fault tolerance technique used in High Performance Computing and especially in MPI applications. This technique relies on the reliability of the checkpoint storage, most of the rollback recovery protocols assume that the checkpoint servers machines are reliable. However, in a grid environment any unit can fail at any moment, including components used to connect different administrative domains. Such a failure leads to the loss of a whole set of machines, including the more reliable machines used to store the checkpoints in this administrative domain. It is thus not safe to rely on the high MTBF of specific machines to store the checkpoint images.

This paper introduces a new protocol that ensure the checkpoint storage reliability even if one or more Checkpoint Servers fail. To provide this reliability the protocol is based on a replication process. We evaluate our solution through simulations against several criteria: scalability, topology, and reliability of the nodes. We also compare between two replication strategies to decide which one should be used in the implementation.

Keywords: high performance computing,fault tolerance,replication,rollback recovery

1. Introduction

High Performance Computing has an important role in scientific and engineering researches. As the size of High Performance Systems increases continuously, the average time between failures becomes increasingly small. So Fault Tolerance becomes a critical property for Parallel applications running on these systems. MPI (Message Passing Interface) paradigm is actually the most used to write parallel applications. However, in traditional implementations, when a failure occurs, the whole distributed application is shutdown and restarted [1]. To avoid this, many solutions have been proposed, but the most used is Rollback Recovery [2]. Rollback recovery is based upon the concept of a checkpoint. A checkpoint describes the state of one or more components of the system at a given time of its execution. These checkpoints are built from images of processes and the state of communication channels. During its execution, the system takes checkpoints according to a scheduling policy. When a failure occurs, some processes rollback to their last images. The fault tolerance protocol must ensure that the system is in a coherent state which allows it to continue its execution. With coordinated checkpoint protocols, all the processes are synchronized and take their images at the same time, by building a coherent state and a global image of the system called a snapshot. A snapshot is a collection of checkpoint images (one per process) with the state of the different communication channels [3]. When a failure occurs, all the processes must rollback together to the last coherent state, so the checkpoint images of all the processes must be available simultaneously. Usually, checkpoint images are kept for the two last checkpoint waves in order to spare storage resources. If the checkpoint images are not available, the rollback technique fails. These protocols often assume that Checkpoint storage is made by special dedicated and reliable machines named Checkpoint servers.

A grid is an infrastructure consisting of the aggregation of several distributed resources, usually from different administrative domains. There are many kind of grids, and we focus in this study on the cluster of clusters: companies and universities build large supercomputers by aggregating the resources of different clusters. Using such a grid, users expect to obtain larger systems more suitable to address the complexity of their problems. One of the features of a grid is its size, orders of magnitude larger than a single cluster. Moreover, a grid spans multiple domains and is characterized by a topology including few interconnection points linking many components. In a single cluster, if the failure hits the switch or the interconnection mechanism, each component is disconnected from the others and the failure may be considered as fatal. If one of the interconnection point fails, a whole cluster is lost for the rest of the system, including its most reliable components. So, no machine can be considered as reliable anymore. In a grid, however, the amount of resources lost by the failure of a router may be tolerable.

In this paper, we introduce a distributed checkpoint storage service of coordinated Rollback Recovery Protocols suited for clusters of clusters. It addresses the issues related to the Grid Model: to ensure the checkpoint storage reliability, even though one or more checkpoint servers fail, we use a replication process.

We compare two different strategies of replication named simple and hierarchical. The paper is organized as follows. Section 2 presents the Grid and failure models we consider. Section 3 presents the related works. Section 4 introduces our protocol for distributed checkpoint storage. We evaluate performances of our approach and we compare two different replication techniques in section 5. Last, we draw our conclusions in Section6.

2. System Model

We consider a High Performance Grid made up of powerful computer servers. We also consider the grid environment as an aggregation of C clusters, each cluster i includes N_i machines. To store the checkpoint images, we define in each cluster a set of checkpoint servers. Thus, in a cluster, we have two kinds of processes. Clients processes that carry out calculations and regularly transfer their checkpoint images to the storage service; and checkpoint servers (CS), which maintain the checkpoint storage. All checkpoint servers within the same cluster are pooled in a group. The different clusters are linked over front-end machines. Figure 1 illustrates the architecture of our system.

We assume that any component of the system can fail at any time, and we consider that there exists a coordinated checkpoint protocol which handles the clients failures. Therefore, we propose a solution to handle the checkpoint server failures to ensure the storage service reliability even when a checkpoint server fails. We consider two types of behaviors:

- a failure may hit a checkpoint server in a cluster.
- a failure may hit the cluster's front-end machine, or a set of failures disconnects a whole cluster from the rest of the grid. For the clusters which remain connected, all the components of the cluster fail simultaneously.

To increase the protocol flexibility, we make the following assumptions :

- We consider a group failure if we lose a connection with the checkpoint servers of this group (e.g.: a front-end failure). We suppose that it cannot be more than K group failures, with $0 \leq K \leq C - 1$.
- In the case of a group failure, the computation which was executed in this cluster get restarted on new one.
- We suppose that for a number of checkpoint servers n_i in group i, $0 \leq i \leq C - 1$, at a given moment, there cannot be more than k_i checkpoint servers failures, $0 \leq k_i < n_i$.

These numbers are fixed according to the mean time between failures in the system. Our solution relies on a distributed checkpoint service. To ensure the reliability of this service, we use a replication protocol. We replicate Checkpoint images over checkpoint servers, so that a valid replica is available even though one or more checkpoint servers fail. To tolerate k_i failures in a group i, $0 \leq i \leq C - 1$, we must have at least $k_i + 1$ replicas in this group. To tolerate a group failure, we also replicate the checkpoint images outside the cluster which hold them. So, to tolerate K group failures, with $0 \leq K \leq C - 1$, we replicate the checkpoint images over $K + 1$ different groups.

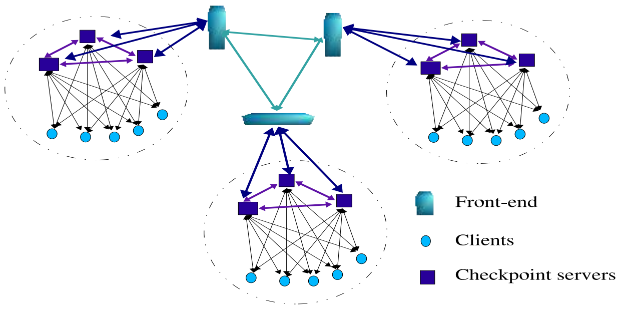

Figure 1. System Architecture

3. Related Works

In checkpoint-based protocols, during the execution the computation state is periodically saved. Then when a failure occurs, the computation is restarted from the last saved state. Checkpoint based protocols can be classified into three categories: coordinated checkpointing, uncoordinated checkpointing, and message logging [4]. The first coordinated checkpointing protocol for distributed applications was proposed by Chandy and Lamport in [3]. This solution assumes that all the channels are FIFO and any process can decide to initiate a checkpoint wave. This algorithm is implemented in many fault tolerant message passing libraries, such as LAMMPI [5], MPICH-V [4]. Other techniques like Checkpoint Induced Communication [6] try to limit the size of the coordinated set to build the global coherent snapshot. This technique has also been implemented in other fault-tolerant libraries, like the proactive communication library [5]. All these techniques assume the ability to store the checkpoint images in a reliable media which is not subject to failures.

Other checkpoint based solutions exists without relying on stable storage, [8] introduces a disk-less checkpointing solution. This solution defines a way to perform fast, incremental checkpointing by using N+1 parity, which reduces high memory overhead required by disk-less checkpointing methods. However, after a failure, all processors communicate with the parity, which can cause a communication bottleneck. Also, the solution is based on the parity machine which should never fail. Others distribute the checkpoint images directly in the memory of the computing peers, like for the FT-MPI project [9], or the Charm++ project [10]. However, storing the checkpoint image in the memory of the other processes implies either to use twice the memory necessary for the application or remove the transparency assumption and to use user-driven serialization of the checkpoint image. [11] describes disk-based and memory-based checkpointing fault tolerance schemes. The goal of this solution is to automate the checkpointing and the restarting of the tasks, and thus to avoid writing additional code. These schemes are based on the works presented in [12] and [13]. In [14] a new solution based on the assumption that some failures are predictable is introduced. It pro-actively migrates execution from processors suspected to fail. This solution is based on processor virtualization and dynamic task migration ideas provided on [15] and [12]. [16] introduces a fault tolerance protocol that provide fast restarts. This protocol uses the concepts of message logging and processor virtualization. It does not assume the existence of reliable component that never fails.

The goal of the replication services is to keep the states of the different replicas coherent, by implementing the adequate primitives. The two major classes of replication techniques ensuring this consistency are: active replication [17] and passive replication [18]. Simple replication is not adequate for high performance computing. Indeed, to tolerate n failures every component must be replicated n times. Thus, the computation resources are divided by n. Replication is however a mechanism used to ensure the accessibility of data in fault tolerance protocols. [19] considered distributing generic data on the grid using distributed hash tables, and evaluated the efficiency of this approach for storing checkpoint images for fault tolerance. However, this technique is not focused on the coordinated checkpoint protocols, which induce a peak overload on the EDG network, and we believe that hierarchical techniques are more suited than DHTs for this kind of topology. [20] and [21] introduce solutions to ensure availability of some failures points (e.g. the head node of a cluster architecture) using redundancy. These solutions are based on the asymmetric and symmetric Active/Active High availability. Active/Active High availability means that several replicas are active in the same time. Wherease in the asymmetric one there is not any coordination between the active replicas, in the symmetric one the active replicas maintain a common global component state.

4. Checkpoint Storage Protocol

Our checkpoint storage protocol, based on a distributed checkpointing service and a replication process, proceeds in two phases. The recording phase, responsible for images storage, and the recovery phase executed when a failure occurs on some calculation nodes.

4.1 The Recording Phase

The recording phase proceeds in two steps. First, clients send their images to the CSs within the same cluster. Second, those images are replicated amongst the CS group within the local cluster, and in remote clusters. In order to improve the performances, image sending is made on a distributed way. A checkpoint image is split in several parts of fixed size named *chunks*. We call f_c^j the j^th chunk of the *c*th client checkpoint image. During the building of the checkpoint image, the client builds his chunks and sends them to the CSs of its cluster. The client memorizes a list of CSs that received its chunks. At the end, the client keeps a local copy of its checkpoint image, then it sends to all the CSs on its cluster the finalize message which contains the chunks number. The image is considered safely stored, when the client receives acknowledgments (ACKs) for all its chunks. If the client detects a CS failure before the reception of the corresponding ACK, it selects another CS and resends the corresponding chunks. If the client fails during the transfer, the checkpoint wave get cancelled, a new resource equivalent is allocated, and the application is restarted from its last checkpoint.

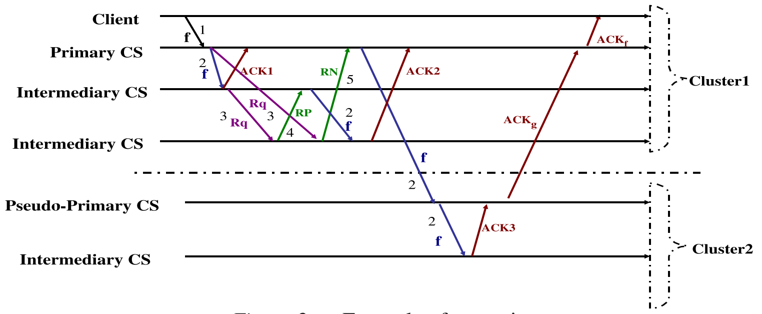

Figure 2. Example of execution

In the second step, chunks are replicated on the CSs, we consider that a chunk f is *correctly replicated* in the group i if and only if f is replicated on $k_i + 1$ servers in this group. According to the assumption on the number of tolerated failures, a chunk is considered *recorded*, if it is *correctly replicated* in $K + 1$ groups.

4.1.1 Replication Strategies. .

1. Simple Replication Strategy: We have adapted the passive replication technique : each checkpoint server receiving a chunk ck_c^j from the client c becomes primary of this chunk, and must ensure its replication. When a checkpoint server s primary of a certain number of chunks fails, a new server is selected to become primary of all the chunks of s.

During the replication step, a CS s can play several roles according to the origins of the received chunks. First, receiving a chunk from the client, s is considered *primary* of this chunk. It is responsible of the correct replication of this chunk in its group and on K different groups before sending the acknowledgement to the client (ACK_f in figure 2). Second, if it receives a chunk from a CS $s' \neq s$ from another group $i' \neq i$, it is considered as a *pseudo-primary* of this chunk in its group. It is then responsible to replicate the chunk in its group and to send the acknowledgement to the primary s' (ACK_g in figure 2). The last role, *intermediary* is played by a CS when it receives a chunk from another CS within its group. In this case, the CS sends directly the acknowledgement to the primary or to the pseudo-primary ($ACK1$, $ACK2$, and $ACK3$ in figure 2). During the replication step, the chunks received from clients have the greatest priority, than those received from the other CSs, and finally those received from the other clusters.

With the Simple Replication Strategy (SRS), the primary CS does the replication over all the other CSs of its group, then over the other groups. Then each *pseudo-primary* does the replication over all the other CSs of its group. So a CS s receiving a chunk ck from a client or from another group sends it to $(s + i) mod[2^m], 1 \leq i < 2^m$. With this technique, an *intermediary* CS has no active role in the replication process.

2. Hierarchical Replication Strategy: To accelerate the replication process, we introduce another strategy. Its goal is that each CS in the system has an active role, including the *intermediary* ones. For that we define for each CS s a

set of CSs with identifiers $\{s, s+2^0, s+2^1, \cdots, s+2^{m-1}\}$ called *children*. Fig.2 presents a diagram of an execution of the replication step with this strategy. The primary server of a chunk ck_c^j replicates it on the *children* servers which constitute the first level of replication, then, each CS receiving this chunk must replicate it over its own *children* servers, carrying on that way until all the CS have received the chunk. To avoid replicating a chunk twice on the same CS, a request is sent before each replication (the third step in Fig.2).

During the execution of a checkpoint wave, two cases may happen : 1) the execution finishes without any CS failure, and 2) some checkpoint servers fail before the end of the wave. If a primary CS s fails during the replication, a new primary s' is selected to handle the primary chunks of s. A client detecting the failure of s before the reception of its acknowledgement, resends the chunk to s'. The CSs are organized on a circular list, so when a primary CS fails the new primary is simply the next in the list. s' will check the replication status before the breakdown. In case the replication was started before the failure, it sends a request to collect the acknowledgements from the other CSs to know if they have received the chunk by the last primary. When a CS in the same group receives this request, it acknowledges the previous reception of the chunk, or asks for it if it has not received it before. When a CS from another group receives this request, it checks the previous reception of the chunk, then it verifies if a correct replication was made in its group before sending an acknowledgment to the primary, otherwise, it asks for it.

At the end of the recording phase the CS has to check if all the clients of the same distributed application have correctly recorded their images, then validate locally the checkpoint wave.

4.2 The Recovery Phase

In the beginning of this phase, a consensus is executed between the different CSs to define the last valid wave: each CS proposes the number of its last valid wave, and the goal is to arrive at an agreement. As several checkpoint wave can be done before failure, the client starts by asking for the last valid wave, and checks whether the image is available locally. Otherwise, it requests its image from the CSs within its cluster. As for the recording, recovery is done in a distributed way: the client sends its request of recovery to all the CS of its group, then a CS receiving the request provides chunks of which is primary. Finally, once all the chunks are recovered, the image is reconstituted, and the client is restarted.

5. Performance Evaluation

We study our solution using the SimGrid [22] simulator. SimGrid provides the main functionalities for the simulation of distributed applications in heterogeneous distributed environments. We particularly use MSG, the first distributed programming environment provided within SimGrid. It allows us to study the different heuristics of the issues before the implementation. It makes it possible, in the first stage, to validate our solution and especially to carry out a good comparison between the two replication strategies.

Running a simulation with Simgrid requires as input two files in XML formats. These files do not only describe the simulation parameters and dynamics (e.g. links and machines failures) but also the network topology. We suppose that the Checkpoint Servers of a group are connected between them through a complete graph. The number of CS is small, so we will have a realistic number of connections to manage. However, for the inter clusters connections, we choose a graph much less connected, where each CS will only have one outgoing connection. For all the experimentation, the links within a cluster are homogenous, as are the CSs and the clients.

5.1 Impact of the Topology

We first investigated the impact of the clients number, and thus the size of the data to be stored. For that, we fixed the cluster and the CSs numbers in the system ($c = 1$ cluster and $s = 6$ CSs), and we varied the clients number. The first curve in Fig.3, the checkpoint wave, presents the wave execution time according to the number of clients. We notice that the execution time is proportional to the number of clients. This is not surprising since more clients means a larger quantity of data to store and to replicate, and thus the wave of checkpoint takes more time. To identify which one of the two checkpoint phases influences the execution time, we isolated the recording phase. The corresponding measurement in Fig.3 shows that the execution time of the recording step increases slowly. This is expected, because in theory this step is executed in a parallel way and it takes xl/N time unit (where x is the number of chunks per client, l the size of a chunk, and N the link capacity) whatever the clients number is . In practice, the observed increasement is due to the saturation of the communication bottleneck. So the growing of the checkpoint wave execution time when a clients number increases is caused by the replication phase execution time. In theory the execution time t of the replication phase is:

$$t = kxl(\frac{1}{N} - \frac{1}{sN})$$

Thus when the clients number k increases the execution time of the replication phase increases proportionally.

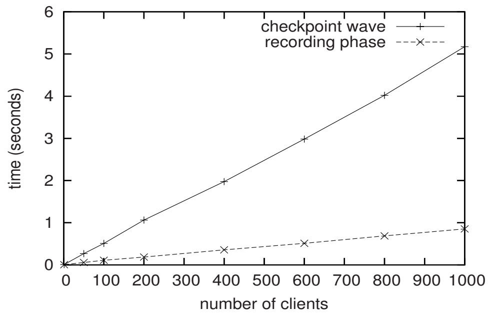

Figure 3. Scalability of the checkpoint

The goal of the second experimentation is to evaluate the impact of the network topology. In order to do this, we consider a fixed number of clients

$k = 100$, a fixed number of CSs $s = 30$, and we make vary the number of clusters c. Thus, there is k/c clients and s/c servers in each cluster; every client has x chunks of size l. The links have a capacity of N MB/s within a cluster and N' MB/s between clusters. Theoretically, the checkpoint wave over c clusters takes the time t defined so:

$$t = \frac{xl}{N} + \frac{2xkl}{N} + \frac{xkl}{sN} + \frac{xkl}{N'} - \frac{1}{c}(\frac{xkl}{N} + \frac{xkl}{N'}) - \frac{cxkl}{sN}$$

The curve resulting from this equation is presented in Fig.4.

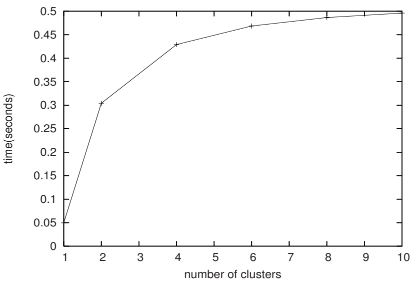

Figure 4. Impact of the topology and comparison with the thoeretical result

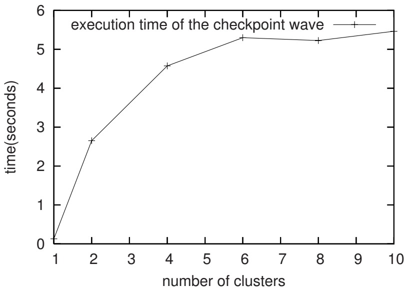

Figure 5. Impact of the topology

The first curve, Fig.5, presents the result of this second experimentation. To better understand the result we isolated the recording phase (the second curve, Fig.6), and the local replication (the last one, Fig.7). When the cluster number increases, the clients number per cluster decreases, and thus the recording phase execution time decreases. However, although the number of checkpoint servers per cluster decreases, the execution time of the local replication increases, because there is overlapping between this phase and the rest of the execution, which reduces the global execution time of the checkpoint wave.

5.2 Impact of the Replication

To evaluate the two replication strategies, we first investigated the effect of the CSs in the system and thus the effect of the replication. For doing this, we fixed the clients number $k = 200$ and we varied the CSs number s. Figure 8 shows that the execution time of the checkpoint wave, particularly the replication phase increases considerably and proportionally with the CSs number. Theoretically, the execution time t of the replication phase is:

$$t = \frac{kxl}{N} - \frac{kxl}{sN}$$

So when s increases the execution time of the replication phase increases too.

To compare the effect of the hierarchical replication versus the simple one, we fixed the clients and the chunks numbers per client, and we varied the CSs number. Then we launch two series of executions with the two strategies. These experiments are carried out to decide which replication strategy will be used in the implementation. As we can see in Fig.9, the best replication strategy depends on the number of CSs. The hierarchical one does additional checks

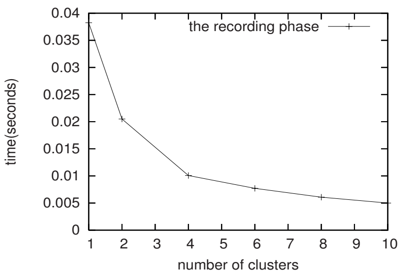

Figure 6. Impact of the topology on the recording phase

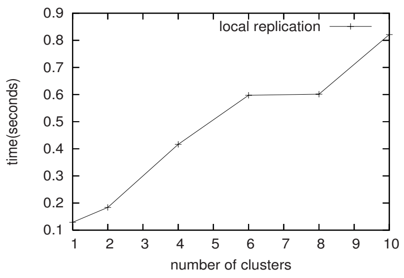

Figure 7. Impact of the topology on the replication phase

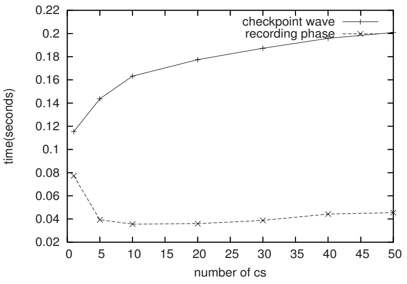

Figure 8. Impact of the Replication.

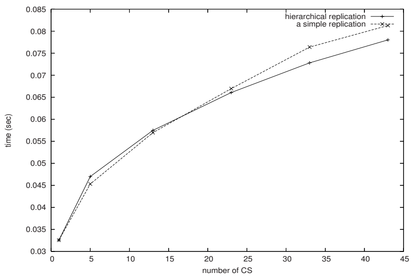

Figure 9. Comparison between hierarchical and simple replication.

for the presence of chunks onto the secondary CSs before each sending. As we give the first priority to the chunks received from clients, and every CS received data from clients when Css number is low, the additional checks increases needlessly the execution time, which makes the simple replication better than the hierarchacal. However, when the CSs number increases, the hierarchical replication allows overlapping of communications to secondary CSs, and so the acceleration of the replication phase. We observe that when the simple replication is better, the difference is small because the checks message size is smaller than the chunk size. Although the execution time of the recording phase should be fixed, increasing the number of clients or decreasing the number of CSs makes the recording phase more aggressive, in the sense that the size of data to be stored increases or the storage devices number decreases which causes communication bottleneck.

6. Conclusion

An efficient and reliable fault tolerance protocol plays a key role in High Performance Computing and especially in MPI applications. Rollback recovery is the most used technique in such environments. To ensure a high level of fault tolerance, the rollback recovery techniques rely on the availability of checkpoint images at rollback time. Usually, rollback/recovery protocols often assume that Checkpoint storage is made by special dedicated and reliable machines named Checkpoint servers. In a grid, however, no machine can be considered as reliable anymore, since even machines with a high MTBF are located inside a cluster which may be entirely disconnected from the rest of the grid.

In this work, we introduced a distributed checkpoint storage service of coordinated Rollback Recovery Protocols suited for clusters of clusters. It addresses the issues related to the Grid Model: to ensure the checkpoint storage reliability, even though one or more checkpoint servers fail, we use a replication process.

We compared two replication strategies, a simple direct strategy, where a CS receiving image from a client uploads this image to each and every one of the CSs; and a hierarchical one, where CSs synchronize with each others to ensure the replication. This comparison shows that the strategy choice depends on the system topology, particularly the CSs and the clients numbers.

The different experimentations show that the execution time of the replication phase takes much more time than the recording one. A long time of the checkpoint wave execution decreases the checkpoint wave frequency. To avoid this we propose to consider the checkpoint wave as done when the recording phase is finished. So, a CS sends the acknowledgments when it received the data, then it does the replication. Thus we increase the checkpoint wave frequency. If a CS fails before the end of the replication, and some data is lost, we cancel this step, and we consider the last wave for which the replication is successfully finished.

For the future, first we will evaluate our approach via an experimenation in a real experimental grid. Then, we would like propose a new scheduling scheme and a new replication strategy that improve the performances of our protocol.

References

[1] W. Groop and E. Lusk, Fault Tolerance in MP Programs. *OAI-PMH server at cs1.ist.psu.edu*, 2002

[2] E. N. Elnozahy et al.A survey of Rollback-Recovery Protocols in Message-Passing Systems, Journal "CSURV: Computer Surveys", volume 34, 2002.

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

[4] A. Bouteiller et al.Mpich-v: a multiprotocol fault tolerant mpi. *International Journal of High Performance Computing and Applications, 20(8):319?333, fall*, 2006.

[5] G. Burns, R. Daoud, and J. Vaigl. *LAM: An open cluster environment forMPI*, 1994.

[6] L. Alvisi et al.*An analysis of communication induced checkpointing. In Proceedings of the symposium on fault-tolerant computing, pages 242?249*, 1999.

[7] F. Baude et al.A hybrid message logging-cic protocol for constrained checkpointability. In Proceedings of EuroPar2005, LNCS, 2005.

[8] James S. Plank and Kai Li, Faster Checkpointing with N+1 Parity, 24th International Symposium on Fault-Tolerant Computing, Austin, TX, June, 1994, pp 288–297.

[9] Z. Chen et al.Building fault survivable MPI programs with FT-MPI using diskless-checkpointing. In Proceedings of the tenth ACM SIGPLAN Symposium on (PPoPP), June 2005.

[10] G. Zheng, L. Shi, and L. V. Kale. Ftc-charm++: an inmemory checkpoint-based fault tolerant runtime for charm++ and mpi. In Proceedings of the IEEE International Conference on Cluster Computing, USA, 2004. IEEE Computer Society.

[11] C. Huang et al.Performance evaluation of adaptive MPI. PPOPP 2006: 12-21

[12] L. V. Kale and S. Krishnan. Charm++: Parallel programming with message-driven objects. In Wilson, G.V., Lu, P., eds.: Parallel programming using C++. MIT Press (1996) 175-213.

[13] L. V. Kale. The Virtualization approach to Parallel Programming: Runtime Optimization and the State of Art. In LACSI 2002, Albuquerque, October 2002.
[14] S. Chakravorty, C. L. Mendes, and L. V. Kalé, Proactive Fault Tolerance in MPI Applications Via Task Migration. HiPC 2006: 485-496
[15] L. V. Kale and S. Krishnan. Charm++: Parallel programming with message-driven objects. In Wilson, G.V., Lu, P., eds.: Parallel programming using C++. MIT Press (1996) 175-213.
[16] S. Chakravorty and L. V. Kalé, A fault tolerance Protocol with Fast Fault Recovery, Accepted for publication at IPDPS 2007.
[17] R. Guerraoui and A. Schiper. Software based replication for fault tolerance. IEEE Computer, 30(4):68?74, Apr. 1997.
[18] N. Budhiraja et al.The primary-backup approach, Dec. 01 1993.
[19] L. Rilling and C. Morin. A practical transparent data sharing service for the grid. In Proc. Fifth InternationalWorkshop on Distributed SharedMemory (DSM 2005), Cardiff, UK, May 2005. Held in conjunction with CCGrid 2005.
[20] C. Leangsuksun et al.Asymmetric active-active high availability for high-end computing. In Proceedings of (COSET-2), in conjunction with the 19th ACM International Conference on Supercomputing (ICS), Cambridge, MA, USA, 2005.
[21] C. Engelmann et al.Symmetric active/active high availability for high-performance computing system services. Journal of Computers (JCP), 1(8), 2006.
[22] INRIA. Simgrid project. http://simgrid.gforge.inria.fr.