Hierarchical Replication Techniques to Ensure Checkpoint Storage Reliability in Grid Environment

Fatiha Bouabache*
fatiha.bouabache@lri.fr

Thomas Herault*
thomas.herault@lri.fr

Gilles Fedak*
fedak@lri.fr

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

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 failures lead to the loss of a whole set of machines, including the more reliable machines used to store the checkpoints in this administrative domain. Thus it is not safe to rely on the high MTBF (Mean Time Between Failures) of specific machines to store the checkpoint images. This paper introduces a new coordinated checkpoint protocol, which tolerates checkpoint server failures and clusters failures, and ensures a checkpoint storage reliability in a grid environment. To provide this reliability the protocol is based on a replication process. We propose new hierarchical replication strategies, with two different degrees of hierarchy, adapted to the topology of cluster of clusters. Our solution exploits the locality of checkpoint images in order to minimize inter-cluster communication. We evaluate the effectiveness of our two hierarchical replication strategies through simulations against several criteria such as topology and scalability .

1. Introduction

High Performance Computing plays an important role in scientific and engineering researches. As the size of HPC 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) is amongst the most frequently used paradigm to write parallel application. However, in traditional implementations, when a failure occurs, the whole distributed application is shutdown and has to be restarted mannualy [24]. A technique to avoid the restart of the application from the beginning is rollback recovery [13] which is based on the concept of checkpoint.

With coordinated checkpoint protocols, all the processes are synchronized and take their image at the same time, building a coherent state and a global image of the system called the snapshot. A snapshot is a collection of checkpoint images (one per process) with the state of the different communication channels [11]. Rollback/recovery protocol imposes that when a failure occurs, all the processes rollback together to the last coherent state. The main advantage of this approach is that the application is not impacted by the protocol between two consecutive checkpoint waves. When processes rollback, 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. Protocols often assume that checkpoint storage relies on special dedicated and reliable machines named Checkpoint Servers (CS).

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 cluster of clusters: companies and universities build large supercomputers by aggregating the resources of several clusters. Using such a Grid, users expect to obtain larger systems more suitable to address the

978-0-7695-3156-4/08 $25.00 © 2008 IEEE
DOI 10.1109/CCGRID.2008.95

complexity of their problems. One of the features of a Grid is its size, orders of magnitude larger than a single cluster. This also means that the probability that one component in the Grid is subject to failure is great. Moreover, the Grid that we consider spans multiple domains and uses Internet to connect between the different clusters. So a set of failures can disconnect a whole cluster, including its reliable components, from the rest of the system. In a single cluster, if the failure hits the switch or the interconnection mechanism, all components set disconnected from the others and the failure may be considered as fatal. In a Grid, however, the amount of resources lost by the failure of a router may be tolerable. But no machine can be considered as reliable anymore.

In this paper, we ensure the checkpoint storage service reliability of coordinated rollback/recovery protocols. We assume that any component in the system can fail, and we handle checkpoint server failures and cluster disconnection. To this purpose, we propose to replicate the checkpoint images over a set of checkpoint servers distributed over the different clusters. We introduce hierarchical replication strategies suited for cluster of clusters and for the hierarchical topology of such environment, called Simple Hierarchical Repliaction SHR and Greedy Hierarchical Repliaction GHR.

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. Section 5 introduces our hierarchical replication strategies. We evaluate performances of our approach and we compare the two different replication techniques in section 6. Last, we conclude in section 7 with a short summary of the presented work and a brief overview of future work.

2. System Model

We consider a HPC Grid made of powerful computer servers. We also consider the Grid environment as an aggregation of C clusters, each cluster i including 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 : client processes that carry out calculation and transfer regularly their checkpoint images to the storage service and checkpoint servers (CS) that insure the checkpoint storage. All checkpoint servers within the same cluster are pooled in a group. The different clusters are linked over front-end machines. Fig.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 failures of clients. Therefore, we propose a solution to handle the checkpoint servers 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. We call a group failure.

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

- We consider a group failure if we loose any connection with the checkpoint servers of this group (e.g.: a front-end failure). We suppose that at most K group failures hit the system, with $0 \leq K \leq C-1$.
- In the case of a group failure, the computation which was executed in this cluster is restarted on a 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. They are parameters to the algorithm.

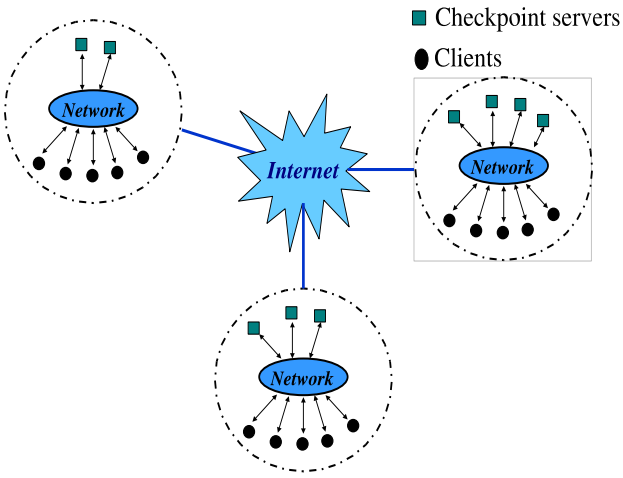

Figure 1. System Architecture

Our solution relies on a distributed checkpoint service. To ensure the checkpoint service reliability, we use a replication protocol. We replicate Checkpoint images over checkpoint servers, so that a valid replica is available even though any checkpoint server failure. 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. At the worse case possible, when $K = C-1$ and in each group i $k_i = n_i - 1$, each CS in the system holds a copy of all replicas.

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 two categories: coordinated checkpointing and uncoordinated checkpointing [4]. The first coordinated checkpointing protocol for distributed applications was proposed by Chandy and Lamport in [11]. 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 [1] try to bound 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 [3]. All thesetechniques assume the ability to store the checkpoint images in a reliable media which is not subject to failures.

Other checkpoint based solutions exist without relying on stable storage, [26] introduces a diskless checkpointing solution. This solution defines a way to perform fast, incremental checkpointing by using N+1 parity, which reduces high memory overhead required by diskless checkpointing methods. However, after a failure, all processors communicate with the parity, which can cause a communication bottleneck. Others distribute the checkpoint images directly in the memory of the computing peers, like for the FT-MPI project [12], or the Charm++ project [30]. 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. [19] 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 [18] and [21]. In [9] 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 by [22] and [18]. [8] introduces a fault tolerance protocol that provides fast restarts. This protocol uses the concepts of message logging and processor virtualization. It does not assume the existence ofreliable 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 [16] and passive replication [25]. Simple replication is not fit to ensure the calculation nodes fault tolerance for high performance computing. Indeed, to tolerate n failures every component must be replicated n times. Thus, the computation resources are divided by n. However, replication mechanism is widely used to ensure the accessibility of data in fault tolerance protocols. [27] 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 our topology. [7] and [6] 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 no coordination between the active replicas, in the symmetric one the active replicas maintain a common global component state.

In the context of data Grid technology, replication is mostly used to reduce access latency and bandwidth consumption. The existing replication strategies proposed in this direction attempt to maximize locality of the file. [17] uses the replication to ensure efficient and fast access to huge distributed data. For that, it introduces a set of replication management services and protocols. The replication decisions are made according to a cost model based on a set of factors such as run-time read/write statistics. [28] presents a dynamic Grid replication strategy which tries to reduce data access time. This paper defines a new form of locality, called network-level locality. Although the required file is not in the site performing job, the replica is located in the site having broad bandwidth to the site of job execution. To reduce access latency, [29] presents the FreeLoader framework, which aggregates unused desktop storage space and I/O bandwidth into a shared cache/scratch space, for hosting large, immutable datasets and exploiting data access locality.

To take the replication decisions, such as when to create a replica and where to store it, the different solutions are based on a set of factors and basically on the locality and the access frequency. In our work, we need neither a replication cost to take the replication decisions, nor a consistency property to keep the different replicas coherent. The replication decisions are made according to our protocol scheme and topology.

4. Checkpoint Storage Protocol

Our checkpoint storage protocol is based on a distributed checkpointing service. To guarantee the reliability of this service we use a replication concept. This protocol proceeds in two phases: the recording phase is responsible for images storage, and the recovery phase executed when a failure occurs on some calculation nodes, and during which the checkpoint images for the last valid wave are downloaded.

4.1. The Recording Phase

The recording phase proceeds in two steps. First (sending step), clients send their images to the checkpoint servers within the same cluster. Second (replication step), those images are replicated amongst the CS group within the local cluster, and in remote clusters.

In order to balance the load between the different CSs, image sending is made in a distributed way. A checkpoint image is split in several parts of fixed size called *chunks*. We call ck_c^j the chunk number j of the checkpoint image of the client c client . During the building of the checkpoint image, the client builds his chunks and sends them to the checkpoint servers of its cluster according to a round robin technique. So the client memorizes a list of checkpoint servers that received at least one of its chunks. At the end, the client keeps a local copy of its checkpoint image, sends to all the checkpoint servers on its cluster the finalize message containing the number of chunks. The image is considered safely stored when the client receives acknowledgments (ACKs) for all its chunks from the checkpoint servers, which means that the replication is finished.

If the client detects a checkpoint server failure before the reception of the corresponding ACK, it selects another server in its group and resends the corresponding chunks. If the client fails during the transfer, the checkpoint wave cancelled, a new resource equivalent is allocated, and the application is restarted from its last checkpoint.

In the second step, chunks are replicated on the checkpoint servers. We consider that a chunk ck is *correctly replicated* in the group i if and only if ck is replicated on $k_i + 1$ servers in this group. According to the assumption on the number of tolerated failures, ck is considered *recorded*, if it is *correctly replicated* in $K + 1$ groups. So, a checkpoint server receiving a chunk from the client has to ensure its recording before sending the ACK to the client.

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 the recovery phase, a consensus is executed between the different CS to define the last valid wave. We define our consensus problem as follows. Each CS proposes the number of its last valid wave, and the goal is to decide the best agreement (the greatest number of checkpoint wave). Fisher et al. [15] proved the impossibility of distributed consensus with one faulty process. This result is essentially based on the fact that it is difficult to determine whether a process has crashed or is just very slow. Therfore, adding failure detectors would solve this problem. chandra et al. [10] show that a weak failure detector w is sufficient to solve consensus in asynchronous systems if and only if $n > 2f$ (n is the number of process in the system and f is the maximum number of failure). According to Kesteloot [23], a stronger failure detectors model allows $f < n$, it requires that at least one correct process is never suspected to be failed. That is why, we use a B model [10] that satisfies the following two properties:

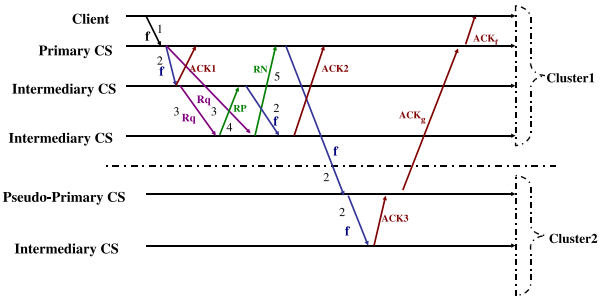

Figure 2. Example of execution

- 1. There is a time after which every process that failed is always suspected by all correct processes.
- 2. There is a time after which some correct process is never suspected by a majority of the processes.

So, we define in each CS s a module that keeps a list l of CS that thinks has failed. Then, in the begining of the consensus, each module sends its list to the authers. A module of s receiving these lists has to update its list. ie. if a CS s' is in the list l and do not exist in one auther received list, then it is removed from the list. Because that is means that s' is simply very slow(according to the property one). According to the property two we can be sure that a consensus will be executed by some correct process.

As several checkpoint wave can be done before failure, the client starts by asking for the number of the last valid wave, and checks whether the image is available locally. Otherwise, the client requests its image from the Checkpoint Servers within its cluster. The client has to receive with the checkpoint wave number, a list of the CSs suspected to be failed. 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, excepted those in the list. 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. Replication Strategies

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. The checkpoint servers are organized on a circular list, where each checkpoint server of a group i has a m-bits identifier from 0 to $(2^m - 1)$. So when a checkpoint server s primary of a certain number of chunks fails, a new server, the next in the list $(s + 1)mod[m]$, is selected to become primary of all the chunks of s.

During the replication step, a CS can play several roles according to the origins of the received chunks. A CS s of a group i which receives a chunk from the client is considered *primary* for this chunk. The CS s 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). If a CS receives a chunk from a CS $s' \neq s$ from another group i' $i' \neq i$, it is considered as a *pseudo-primary* of this chunk in the group i'. It is then responsible to replicate the chunk in the group i' 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.

5.1. Simple Hierarchical Replication Strategy

With Simple Replication Strategy, 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.

5.2. Greedy 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. So when a checkpoint server s primary of a chunk ck fails during the replication a new primary $s' = (s+1)mod[2^m]$ is selected to handle the primary chunks of s. A client detecting the failure of s before the reception of the ck acknowledgement, sends the chunk again to the new primary s'.

The new primary s' will check the replication status before the breakdown. In the replication was started before the failure, s' has already received ck from s, sends a request to collect the acknowledgements from the other CSs to know if they have received the chunk from 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 in case 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.

6. Performance Evaluation

We study our solution using the SimGrid [20] simulator. SimGrid provides the main functionalities for the simulation of distributed applications in heterogeneous distributed environments. It facilitates the research in the area of distributed and parallel application scheduling on distributed computing platforms ranging from simple network of workstations to Computational Grids. 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.

The use of the simulation makes it possible, in the first stage, to validate our solution and to carry out a comparison between the two replication strategies in order to decide which one will be used in the implementation. For each curve presented here, several experimentations have been conducted. Each one with different numbers of clients and servers, and we have observed the same result. The choice of the clients and servers numbers was limited by the actual Simgrid implantation that do not support a great number of components

6.1. Simulation Architecture

One of the main objectives of this study is to examine our protocol behavior in realistic architecture which approaches the structure of the Grid as closely as possible. 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 connec-

tions to manage. However, for the inter clusters connections, we choose a graph much less connected, where each checkpoint server will only have one outgoing connection. For all the experimentation, the links within a cluster are homogenous, as are the checkpoint servers and the clients. The internal communication links are faster than the external communication ones.

6.2. Impact of the Replication

Our goal is to study the potential impact of the replication process,and how to reduce it. To evaluate the two replication strategies, we first investigated the effect of the checkpoint servers number in the system. For doing this, we fixed the clients number $k = 200$ and we varied the CSs number s. Figure 3 shows that the execution time of the checkpoint wave, particularly the replication phase increases considerably and proportionally with the checkpoint servers number. Theoretically, the execution time t of the replication phase is:

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

So when the CS number s increases the execution time of the replication phase increases.

To compare the effect of the SHR versus the GHR, we fixed the clients number and the chunks number 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.4, the best replication strategy depends on the number of checkpoint servers. The GHR does additional checks for the presence of chunks onto the secondary checkpoint servers before each sending. As we give the first priority to the chunks received from clients, and every checkpoint server received data from clients when CSs number is low, the additional checks increase needlessly the execution time, which makes the SHR better than the GHR. However, when the checkpoint servers increases, the GHR 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 size of check messages is smaller than the size of chunks.

6.3. Impact of the Topology

Then, we investigated the clients number scalability, and thus the size of the data to be stored. For that, we fixe the cluster and the checkpoint servers numbers in the system ($c = 1$ cluster and $s = 6$ checkpoint servers), and we vary the clients number. As the CSs servers number is small we used the simple hierarchical replication technique. The first measurement in Fig.5 (the checkpoint wave) presents the

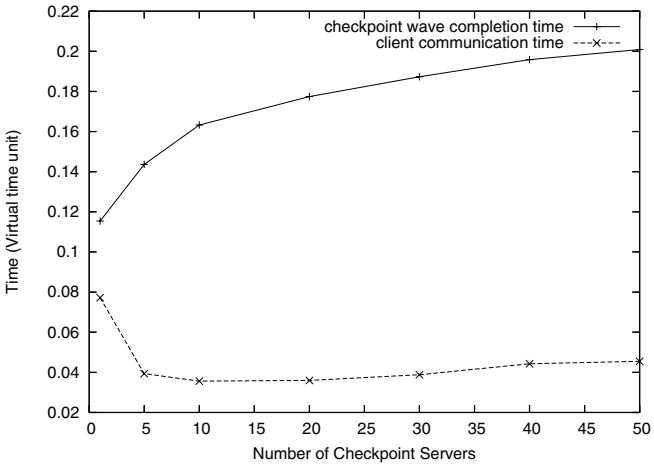

Figure 3. Impact of the Replication

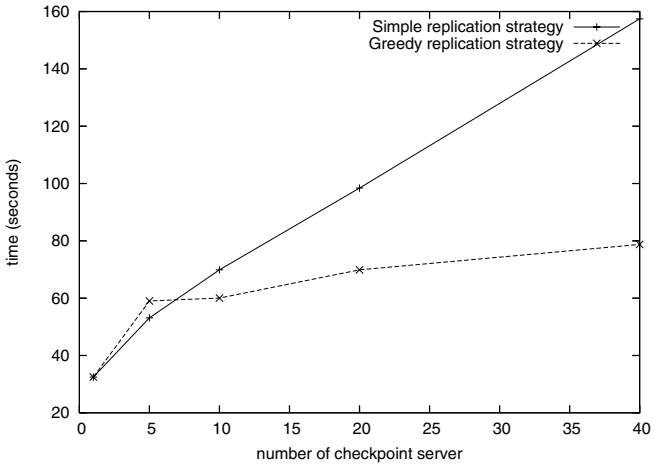

Figure 4. Comparison between hierarchical and simple replication

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 steps of the recording phase (the sending or the replication) influences more the execution time, we isolated the sending one. The corresponding measurement in Fig.5, shows that the execution time of the sending 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 links. So the growing of the checkpoint wave execution time when a clients number increases is caused by the replication step execution time. In theory

the execution time t of the replication phase is:

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

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

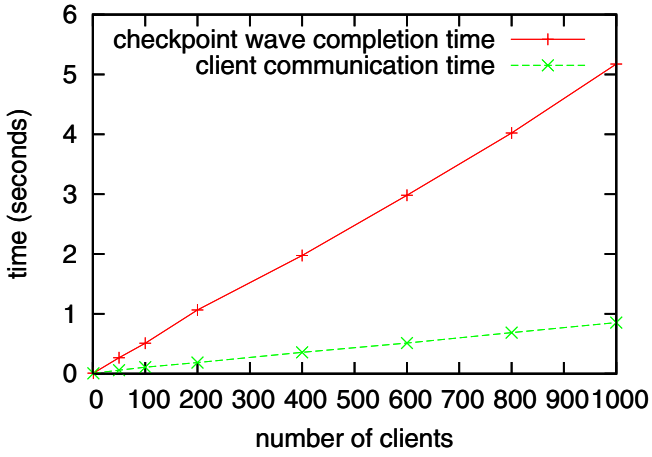

Figure 5. Scalability of the number of clients

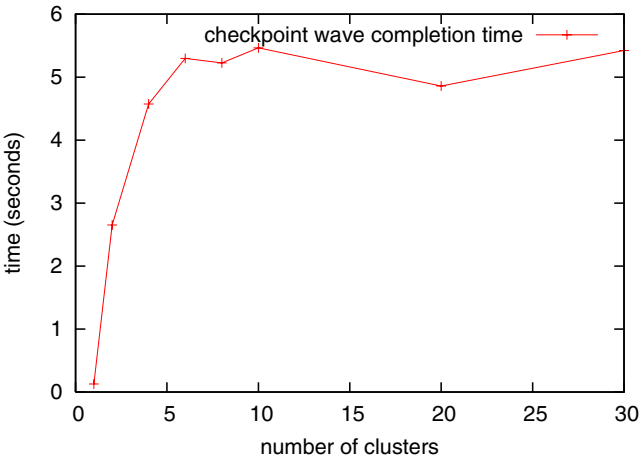

Figure 6. Impact of the topology

The goal of the second experimentation is to evaluate the impact of the network topology, ie. the number of clusters in the Grid. So, we consider a fixed number of clients $k = 100$, a fixed number of checkpoint servers $s = 30$, and we varied 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:

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

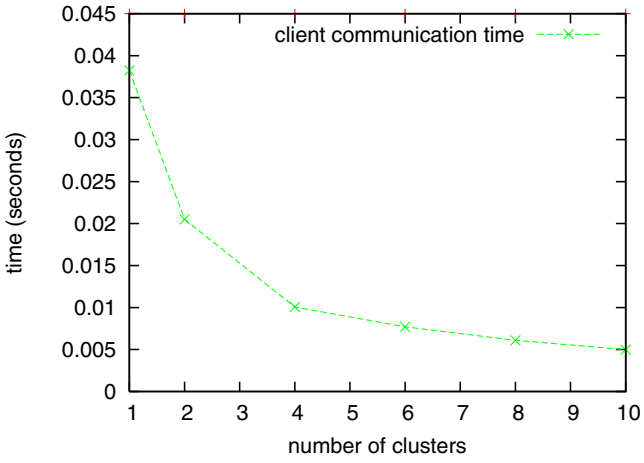

Figure 7. Impact of the topology on the sending step

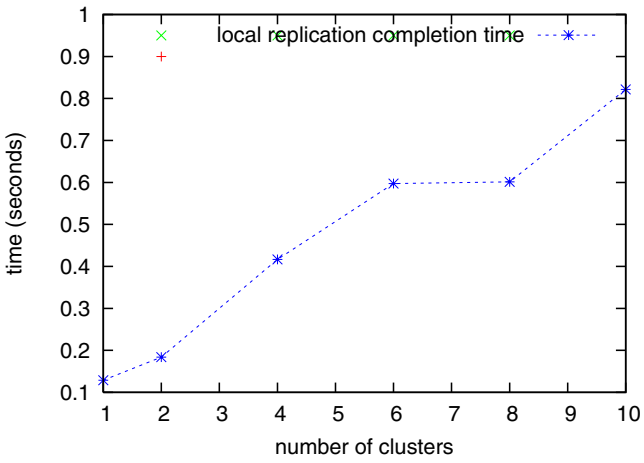

Figure 8. Impact of the topology on the Replication step

The first measurement in Fig.6 presents the result of this second experimentation. We observe that the checkpoint wave execution time increases according to the number of clusters, but with more than 7-10 clusters it becomes stable. As we said before, the internal communication links are faster than the external ones. So when the number of clusters increases the number of external links increases too, which increases the checkpoint wave completion time. But after to a certain number of clusters, the clients and the CS numbers within clusters become so small that the external communications dominate the internal ones, which makes a plateau in the curve. To understand the resulting curve we isolated the recording phase in Fig.7, and the local replication in Fig.8). When the cluster number increases, the number of clients per cluster decreases, and thus the record-

ing phase execution time decreases. However, although the number of checkpoint servers per cluster decreases, the execution time of the local replication increases, this is caused by the overlapping between this phase and the rest of the execution.

Although the execution time of the recording phase should be fixed, increasing the number of clients or decreasing the number of checkpoint servers makes the recording phase more aggressive, in the sense that the size of data to be stored increases or the number of storage devices decreases which causes communication bottleneck.

To confirm our analysis about the external communications and the internal ones, we did a similar experimentation in which we fixed the CSs and the clients numbers per cluster, and we varied the clusters number. The resulting curve Fig.9 shows that the checkpoint wave completion time is proportional to the number of clusters . Which is expected because the number of external communications increases and the number of internal communications within a cluster remain constant.

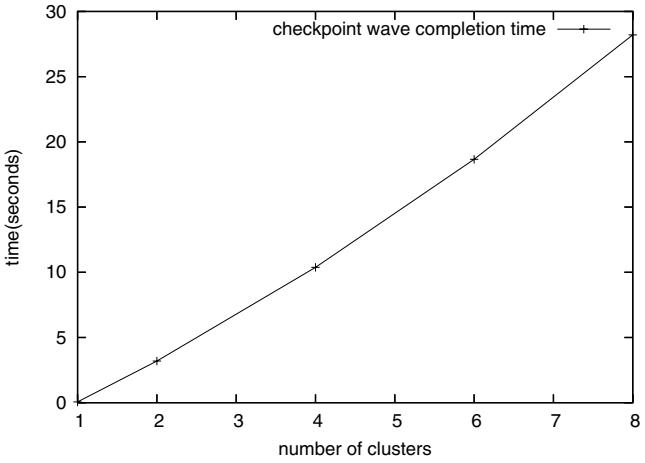

Figure 9. Impact of the topology (2)

7. Conclusions

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. 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 propose a new checkpoint storage protocol for a Grid environment which tolerate checkpoint server failures and clusters failures. To ensure the checkpoint storage reliability we introduced hierarchical replication strategies, SHR and GHR, suited for cluster of clusters. As the topology experimentations show, the external communications are too expensive. With our replication strategies we reduce as far as possible the external communications. After a failure every client will download locally its image whatever the CSs failures. The only case where we will download from another cluster is when we have a cluster failure. We compared, a SHR, where a checkpoint server receiving image from a client uploads this image to each and every one of the checkpoint servers within its cluster; and a GHR one, where checkpoint servers synchronize with each others within its group to ensure the replication. This comparison shows that the strategy choice depends on the system topology, particularly the number of checkpoint servers and the number of client. The different experimentation show that the execution time of the replication phase takes much more time than the recording one. A long time of 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 checkpoint server sends the acknowledgments when it received the data, then it does the replication. Thus we increase the checkpoint wave frequency. If a checkpoint server 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 work, first we will evaluate our approach via an experimentation in a real experimental Grid. Data Grids implement fault tolerant replication of data as a standard feature. The SRB (Storage Resource Broker) manages distributed data in a Grid, it uses a logical name space to represent multiple physical storage systems. So, we think that SRB data Grids provide many of the required features to the implementation of our strategy, and we project to use this middleware, and basically certain services to implement our service with the selected strategy. Then, we plan to propose a new scheduling scheme to improve the performances of our protocol..

References

[1] L. Alvisi, E. Elnozahy, S. Rao, S. Husain, and A. Mel. An analysis of communication induced checkpointing. In *Proceedings of the symposium on fault-tolerant computing*, pages 242–249, 1999.

[2] I. M. Author. Some related article I wrote. *Some Fine Journal*, 99(7):1–100, January 1999.

[3] Baude, Caromel, Delbe, and Henrio. A hybrid message logging-CIC protocol for constrained checkpointability. In *EUROPAR: Parallel Processing, 11th International EUROPAR Conference*. LNCS, 2005.

[4] A. Bouteiller, T. Herault, G. Krawezik, P. Lemarinier, and F. Cappello. 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 for MPI. 1994.

[6] C. L. C. Engelmann, S. L. Scott and X. He. Symmetric active/active high availability for high-performance *Journal of Computers (JCP)*, 2006.

[7] T. L.-S. L. S. C. Leangsuksun, V. K. Munganuru and C. Engelmann. Asymmetric active-active high availability for In *Proceedings of the 2nd International Workshop on Operating Systems, Programming Environments and Computing on Clusters (COSET-2), in conjunction with the 19th ACM*

[8] S. Chakravorty and L. V. Kalé. A fault tolerance protocol with fast faultrecovery. In IPDPS*, pages 1–10. IEEE, 2007.*

[9] S. Chakravorty, C. L. Mendes, and L. V. Kalé. Proactive fault tolerance in MPI applications via task migration. In Y. Robert, M. Parashar, R. Badrinath, and V. K. Prasanna, editors, HiPC*, volume 4297 of* Lecture Notes in Computer Science*, pages 485–496. Springer, 2006.*

[10] Chandra, Hadzilacos, and Toueg. The weakest failure detector for solving consensus. JACM: Journal of the ACM*, 43, 1996.*

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

[12] Z. Chen, G. E. Fagg, E. Gabriel, J. Langou, T. Angskun, G. Bosilca, and J. Dongarra. Building fault survivable MPI programs with FT-MPI using diskless-checkpointing. In Proceedings of the tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 213–223, Chicago, IL, USA, June 2005.*

[13] Elnozahy, Alvisi, Wang, and Johnson. A survey of rollback-recovery protocols in message-passing systems. CSURV: Computing Surveys*, 34, 2002.*

[14] A. N. Expert. A Book He Wrote*. His Publisher, Erewhon, NC, 1999.*

[15] Fischer, Lynch, and Paterson. Impossibility of distributed consensus with one faulty process. JACM: Journal of the ACM*, 32, 1985.*

[16] R. Guerraoui and A. Schiper. Software based replication for fault tolerance. IEEE Computer*, 30(4):68–74, apr 1997.*

[17] E. D. Houda Lamehamedi, Boleslaw Szymanski and Z. Shentu. Data replication strategies in grid environments. In ICA3PP '02: Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing*, page 378, Washington, DC, USA, 2002. IEEE Computer Society.*

[18] C. Huang, O. S. Lawlor, and L. V. Kale. Adaptive MPI. In L. Rauchwerger, editor, Languages and Compilers for Parallel Computing, (16th LCPC'03)*, volume 2958 of* Lecture Notes in Computer Science (LNCS)*, pages 306–322. Springer-Verlag (New York), College Station, Texas, USA, Oct. 2003, Revised Papers 2004.*

[19] C. Huang, G. Zheng, L. V. Kalé, and S. Kumar. Performance evaluation of adaptive MPI. In J. Torrellas and S. Chatterjee, editors, PPOPP*, pages 12–21. ACM, 2006.*

[20] INRIA. Simgrid project. http://simgrid.gforge.inria.fr.

[21] L. V. Kale. The virtualization approach to parallel programming: Runtime optimization and the state of art. In LACSI*, Albuquerque, 2002.*

[22] L. V. Kale and S. Krishnan. CHARM++. In G. V. Wilson and P. Lu, editors, Parallel Programming in C++*, Scientific and Engineering Computation Series, pages 175–214? MIT Press, Cambridge, MA, 1996. chapter 5.*

[23] L. Kesteloot. Fault-tolerant distributed consensus. JACM: Journal of the ACM*, 1995.*

[24] E. Lusk. Fault tolerance in MPI programs, dec 2002.

[25] F. B. S. N. Budhiraja, K. Marzullo and S. Toueg. The primary-backup approach. 1993.

[26] J. S. Plank and K. Li. Faster checkpointing with N +1 parity. In FTCS*, pages 288–297, 1994.*

[27] L. Rilling and C. Morin. A practical transparent data sharing service for the grid. In Proc. Fifth International Workshop on Distributed Shared Memory (DSM 2005)*, Cardiff, UK, May 2005. Held in conjunction with CCGrid 2005.*

[28] Y.-B. K. Sang-Min Park, Jai-Hoon Kim. Dynamic grid replication strategy based on internet hierarchy. In Conference: Grid and cooperative computing*, 2003.*

[29] S. S. Vazhkudai, X. Ma, X. Ma, V. W. Freeh, J. W. Strickland, J. W. Strickland, N. Tammineedi, and S. L. Scott. Freeloader: Scavenging desktop storage resources for scientific data. In SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 56, Washington, DC, USA, 2005. IEEE Computer Society.*

[30] G. Zheng, L. Shi, and L. V. Kale. Ftc-charm++: an in-memory checkpoint-based fault tolerant runtime for charm++ and mpi. In CLUSTER '04: Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pages 93–103, Washington, DC, USA, 2004. IEEE Computer Society.*