# Hierarchical Checkpoint Protocol in Data Grids

Rahma Souli-Jbali, Minyar Sassi Hidri, Rahma Ben Ayed

***Abstract*—Grid of computing nodes has emerged as a representative means of connecting distributed computers or resources scattered all over the world for the purpose of computing and distributed storage. Since fault tolerance becomes complex due to the availability of resources in decentralized grid environment, it can be used in connection with replication in data grids. The objective of our work is to present fault tolerance in data grids with data replication-driven model based on clustering. The performance of the protocol is evaluated with Omnet++ simulator. The computational results show the efficiency of our protocol in terms of recovery time and the number of process in rollbacks.

***Keywords*—Data grids, fault tolerance, chandy-lamport, clustering.

## I. Introduction

DATA grid provides services for supporting the discovery of resources and enables computing in heterogeneous storage resource.

Since complex scientific problems in science and engineering run for a long time, it becomes important to make them resistant to failures in the underlying hardware and infrastructure. The computation cannot complete if any node failure encountered. Therefore, fault tolerance has become necessity.

The loss of a node in the grid can have influence on the operation of the grid system and cause losses replicas of a given that exist on these nodes. In this context, fault tolerance becomes an obligation to ensure the proper functioning of the distributed system.

Several methods have been proposed to handle failures in a widely distributed system such as grid computing [1]-[3]. We mention essentially:

- **Replication (masking):** It uses multiple copies of the same component or process on different machines. Therefore, when a component has failed with the failure, it may be masked by one of the copies. The problem of this method is how to maintain a strong consistency between the copies. There are various replication strategies: Active [4]-[6], passive [4]-[6], semi-active [5]-[7] and coordinator/cohorts [7]. All replication mechanisms rely on the first two replication techniques.
- **The rear cover:** A recovery point is a set of elements such as memory status which allows it to restart process in the event of a node failure [1]-[3].

Approaches based on replication are well suited for fault tolerance to protect the data. But in the case of failure nodes based-solutions, backups checkpoints seem smarter.

Rahma Souli-Jbali, Minyar Sassi Hidri and Rahma Ben Ayed are with the University of Tunis El Manar Ecole Nationale d'Ingénieurs de Tunis BP. 37, Le Belvédère 1002, Tunis, Tunisia (e-mail: rahma.souli@enit.rnu.tn, minyar.sassi@enit.rnu.tn, rahma.benayed@enit.rnu.tn).

Recently, we proposed to combine data replication and job scheduling while using MapReduce-driven clustering to place replicas in grid. With the proposed strategy, scheduling is computed using information provided by the replicas which are established based on information provided by the scheduler [8]. The simulation and experiment results demonstrate that the clustering-driven replication strategy can reduce the data access time, the job scheduling time and the number of active sites for replication according jobs' frequency and databases' size.

In this paper, we present a hierarchical checkpoint protocol in data grids with data replication-driven model based on clustering. So, in case of breakdown, the impact of error would remain confined in the nodes of the same cluster. The fault tolerance protocol combined two pessimistic-based ones: the log-based protocol founded on the transmitter applied to the inter-clusters' messages and the generalization founded on both the not blocking transmitter and the coordinated checkpointing of Chandy-Lamport [9].

The rest of the paper is organized as follows: In Section II, we present the different mechanisms of hierarchical recovery. Section III presents a new fault tolerance technology. Section IV presents the computational results made to validate the proposed protocol. Section V concludes the paper and presents a preview of future work.

## II. Related Work

Logging techniques require a periodic backup of the local process conditions, and recording all messages received after establishing local checkpoints on a stable support.

In the case of process failure, it is restarted from the last checkpoint, and all messages received after the latter are returned in FIFO (First In first Out) order. All logging techniques require that the state of a process recoverable always compatible with the state of other processes. They must also ensure that no orphan processes, which make the calculation of the global state incoherent [6]. However, these rear covers protocols are not completely suitable for all types of grids. Indeed, each protocol has more advantages than another, depending on the application and environment. So, we focused our analysis on hierarchical protocols in correspondence with the hierarchical architecture of computing grids and supported on these rear cover protocols.

Several researchers proposed hierarchical fault tolerance techniques based on rear cover protocols including:

- **Coordinated Hierarchical Recovery point (CHC):** It is designed for networks like the Internet. Experimental studies have been done on a network consisting of four clusters of eight knots. They assume that the cluster nodes

are *safe*, and connected via a broadband network equal to 10 Mbps and the network speed between clusters is equal to 1Mbps [10]. Only one fault occurs at a time and no fault occurs during recovery. They distinguish, during a session establishment checkpoint, three roles between different processes: Initiator, Leader, Follower. The initiator is a process that initiates a checkpoint backup session. The leader is a process chosen in each cluster and coordinates activities within the cluster, in agreement with the Offeror process instructions. The follower is a process that represents the rest of the processes, they follow the instructions of their leader.
This protocol takes place in two steps:

  – The implementation of the coordinated limited to the cluster checkpoint algorithm. The processes are blocked and saved a consistent global state during this step.
  – The implementation of of a coordinated checkpoint, but the leaders are the only participants, with the initiator that acts as a coordinator.

- **Hierarchical Causal Logging (HCL):** This protocol uses a proxy network that stores in the cache the retrieved information. While ensuring the benefits of the standard causal protocol logging, it adds the benefit of proxies exponentially reduces the size of data to follow in the causality footsteps [11].
  The authors have also shown that the use of proxies significantly reduces the overhead bandwidth led by causal information.
- **Hybrid Protocol (HP):** It is designed for applications by code coupling. The protocol combines a technique coordinated dump checkpoint within a cluster and a point protocol recovery induced communication between clusters [12]. Unlike hierarchical protocols described above, the new protocol supports mistakes simultaneous. Experimental studies have shown that if we limit the inter-cluster communications, it limits the number of checkpoints forced. Therefore, it is necessary to adjust the period of retention between non forced checkpoints for an application whose communications are intense. It is also demanding much set the frequency to trigger the garbage collector (expensive network congestion) and storage cost.
- **Process Group (PG):** It uses log-based messages mechanisms. Unlike checkpointing protocols that require all processes to go back, logging protocols avoid the overall restoration replaying messages offending processes only [13]. The backup logging introduces a premium on memory because each message exchanged must be registered in a stable. This is why the authors proposed a new strategy based on an organization of group process that reduces memory usage during execution without fault. Each cluster represents a unit in recovering and saving messages.
  The principle is that only the inter-clusters' messages are stored in the log, while only for intra-cluster messages determinants are kept to determine their order of receipt in case of failure. In this case, only the fault process and other processes in the same group running the recovery procedure [14]. The inter-clusters' messages are simply replayed since they are stored in stable storage. This system represents a significant improvement over the frequent access to a stable support for saving messages in the log.
  In [15], they used the same process for clusters and provides a new rear cover protocol combining a checkpoint coordinated within clusters and message logging mechanism based on the issuer between clusters.

In [6], they established a comparative study that was used to select two protocols adapted to the grid architecture:

- **Intra-cluster: The coordinated backup protocol non-blocking for Chandy and Lamport** This protocol works under the assumption of FIFO while using markers. Indeed, at the time of backup checkpoints, each process safeguards its local state and sends on all output channels for a marker that other neighbors are informed process checkpoint actualization. All processes that receive this message for the first time recorded his state and broadcasts the marker by continuing in the same way [9].
- **Inter-cluster: The pessimistic logging based on Johnson transmitter (PL):** There are three message-logging categories: Optimistic, pessimistic and causal. In optimistic protocol (OL), the author supposed that the logging of a message on reliable support will be complete before a failure occurs. In fact, during the execution of the process, the determinants of messages are stored in volatile memory, before being saved periodically on stable support. The storage of stable memory is asynchronous. Induced latency is then very low. But, a failure may occur before the messages are saved on stable storage support. It is clear that this leads to a loss of information stored in volatile memory of the process down and to an orphaned the messages sent by this process.
  The pessimistic technique was designed on the assumption that a failure may occur after any non-deterministic event. The determinant of each message is recorded stable support before interacting with the system. When a process writes a non-deterministic event in a stable, it does not continue the execution until after receiving an acknowledgment. In this technique, you can always retrieve the status of each process This characteristic has three main advantages:
  – Sending messages to the outside by the process can be performed without using a special protocol.
  – In case of failure, the process relaunched at the most recent checkpoint, and replay the messages stored in the log.
  – Recovery is easy, since the effects of failure are limited only to the failed process.

  The main drawback of this technique is the timing that causes degradation in system performance. Several methods have been established to minimize the timings as the approach proposed in [16].

## III. Hierarchical Checkpoint and Log-Based Protocol in Data Grids

One of the main goals of our work is to present a model of checkpoint in data grids.

### A. Problem Statement and Model Overview

We present a protocol which combines two very known algorithms and used in the distributed systems: the pessimistic log-based protocol founded on transmitter [16] and the coordinated checkpointing of Chandy-Lamport (CL) [9].

Since applications were installed in the distributed systems communicate by passage of messages or the main actor is the process which exchanges messages by using two procedures at least (send and receive). Our study focused on these two types of application.

We are going to make, first of all, a detailed description of our protocol.

In [6], the author made a comparative study of all possible combinations between both algorithms of Chandy-Lamport and the pessimistic log-based protocol founded on transmitter in hierarchic architectures. In the first combination, called *Message Logging Message Logging*, a pessimistic log-based protocol is used to save inter-clusters' and intra-cluster messages. In the second combination, called *Message Logging Chandy-Lamport*, the pessimistic log-based protocol founded on Johnson's transmitter [16] is used to inter-clusters' messages and the protocol of not blocking coordinated maintenance of Chandy and Lamport (CL) [9] is used to intra-cluster messages.

The results of experiments presented in [6] showed that the combination *Message Logging Message Logging* is adapted to not very communicating applications. The combination *Message Logging Chandy-Lamport* gives a better performance for the applications of broadcasting exchanging a big number of messages. However, they are strongly communicating applications, the system save only the inter-cluster messages with the pessimistic log-based protocol.

Our protocol is inspired of jobs of [6] who offers a new protocol of hierarchical adaptive checkpoint protocols based on the combination *Message Logging Chandy-Lamport* (see Fig. 1).

We use the pessimistic log-based protocol founded on the transmitter which is applied to inter-clusters' messages. In intra-clusters, the pessimistic log-based protocol founded on the not blocking transmitter and the coordinated checkpointing of Chandy-Lamport are folded up according to the messages' frequency.

As this protocol is based on the regrouping of nodes in cluster, for fault process, we believe to reproducing the same model inside clusters by applying a clustering algorithm for the training of clusters during the execution of applications. In case of breakdown, the impact of error therefore, would remain confined in the nodes of the same cluster.

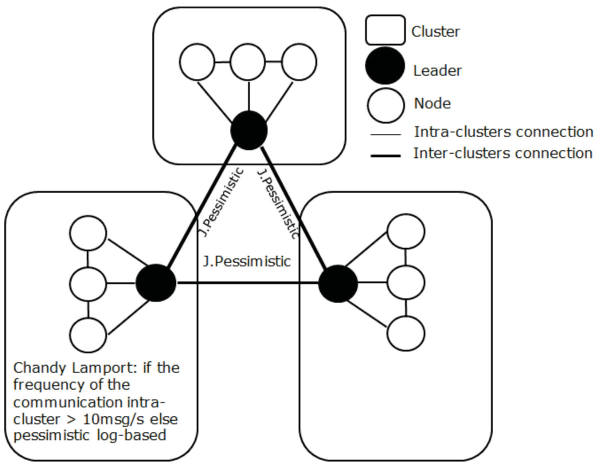

Cluster
Leader
Node
Intra-clusters connection
Inter-clusters connection
J.Pessimistic
J.Pessimistic
J.Pessimistic
Chandy Lamport: if the frequency of the communication intra-cluster > 10msg/s else pessimistic log-based

Fig. 1 Hierarchical checkpoint protocol in Data Grids

### B. Notations

To better present the formal approach to our model, the following notations are used:

- $p_i$: the $i^{th}$ process.
- $cInP_i$: state of the input channel of the $i^{th}$ process.
- $cOutP_i$: state of the output channel of the $i^{th}$ process.
- *numSeq*: number of sequences relating to a process incremented in every reception of message by this process.
- *idClust*: the cluster identifier.
- *idP*: the process identifier.
- $P^{Rep}$: binary indicator to indicate if the process made a checkpoint.
- *ListMsg*: array of messages sent by process.
- *Memvolatil*: array of messages not yet saved.
- $nbMsgclust_i$ : number of messages exchanged in the $i^{th}$ cluster.
- *dateEmissMsg*: sending date of the message.
- *dateRecepMsg*: receiving date of the message.
- *freqmsgs*: frequency of intra-cluster messages.
- *TXsauv*: time after saving a state of process.
- *FreqExPRep*: frequency of the non-blocking coordinated checkpoint protocol execution.
- *MaxNbMsgEmi*: threshold of the number of sent messages.
- *FreqMaxIntraMsg*: threshold of the frequency of intra-clusters' messages.
- *JP*: pessimistic log-based protocol founded on the transmitter.
- *CL*: Chandy-Lamport protocol.
- *MsgRec*: message of the checkpoint request in case of breakdown.
- *MsgCtrl*: message sent by a process to inform the other processes to perform a checkpoint.
- *msg*: exchanged message.
- *ack*: acknowledgement message.
- *act*: acquittal message.
- $MC_i$: center of the the $i^{th}$ cluster.
- *idPtc*: identifier of the last carry out intra-cluster protocol.

The objective of our protocol is not only to reduce the cost of the pessimistic log-based protocol during execution without error, but also to control a too long procedure of checkpoint in case of breakdown.

We have chosen to fix a threshold of frequency of intra-cluster messages (*FreqMaxIntraMsg*) in the execution of the application. Under a frequency less than *FreqMaxIntraMsg*, they are going to apply the combination of the *pessimistic log-based pessimistic log-based* and all messages which are in the grid are saved with their determiners by using this checkpoint protocol.

To minimize the number of messages which will be used in case of breakdown, processes make independent points of checkpoint when the number of broadcast messages reached a threshold (*MaxNbMsgEmi*) equal to 50. To pass our protocol in combination with *pessimistic log-based Chandy-Lamport* protocol, it is necessary to fix a threshold to the maximum of number of messages played again by processes belonging to the same cluster to degrade the performances of applications. Experiments allowed to fix this threshold to ten messages in second (10 msg/s). If the frequency of communications intra-cluster becomes more than this threshold, the log of these messages is unlocked. Therefore, to save the states of processes, it is the Chandy-Lamport protocol that will be used inside clusters. This protocol will be thrown every one hundred and twenty second (*FreqMaxIntraMsg*). Fault can happen after the execution of the pessimistic log-based protocol (see PLP procedure) within cluster or after the execution of the protocol of non-blocking coordinated checkpoint of Chandy-Lamport (see ICB procedure).

**Procedure** Pessimistic log-based protocol

```
Procedure PLP ($msg$, $nbMsgclust_i$, $idClust(Sender)$,
  $idClust(Receiver)$)
{
1   Move $msg$ in the volatile storage
2   Send $msg$
3   if (($idClust(Sender) = idClust(Receiver)$ and $msg <> ack$
      and $msg <> MsgCtrl$))
    {
4       $nbMsgclust_i \Leftarrow nbMsgclust_i + 1$
5       if ($freqmsgs <= FreqMaxIntraMsg$)
        {
6           Lock the execution
7           $numSeq \Leftarrow numSeq + 1$
8           Send $act$
9           Save $msg$
        }
    }
}
End
```

The protocol carried out is saved in *idPtc* attribute of the center node of the cluster. The *idPtc* takes by default value "*JP*".

To perform a recovery, for ($idPtc =$ "*JP*"), it is necessary to execute the last checkpoint and played again messages received by processes after this checkpoint in the same order where they were sent using the numbers of sequences and determiners (see procedure $RRP_i$).

**Procedure** Initiate a coordinated backup

```
Procedure
  ICB ($freqmsgs$, $FreqMaxIntraMsg$, $TXsauv$, $FreqExPRep$)
{
1   if (($freqmsgs > FreqMaxIntraMsg$) and
      ($TXsauv = FreqExPRep$))
    {
2       Choose any process
3       Send $MsgCtrl$ to $MC$
    }
}
End
```

**Procedure** Reception of a covering request in case of process *i* breakdown

```
Procedure $RRP_i$ ($process$, $idPtc$)
{
1   if ($process = MC_i$)
    {
2       if ($idPtc = CL$)
        {
3           Send $MsgRec$ to other centers of clusters
4           Send $MsgRec$ to process of his cluster
        }
5       else
        {
6           if ($idPtc = JP$)
            {
7               Send $MsgRec$ to other centers of clusters
            }
        }
    }
8   else
    {
9       Send the determiner $MC_i$
10      Send $MsgRec$ by $MC_i$ to process which have a
          determiner including $P_i$
    }
11  Receive $MsgRec$ by process $i$
12  if ($process <> MC_i$)
    {
13      Execute $stateProc_i$
14      Play again the messages contained in memory of$P_i$
15      Play again the messages defined in determiners
    }
}
End
```

In another way, to save a message *msg* sent by a broadcasting process *E* to receive process *R*, it is necessary that *E*, firstly, keep *msg* in its volatile storage. And when *R* receives this message, it updated the number of sequences *NumSeq*, then sends an acknowledgement of receipt *ack* in *E*. In the reception of the message *ack* by *E*, *E* adds *NumSeq* in *msg* and sends an *act* in receiving process *R*. Contents and determiners of inter-clusters and intra-cluster messages are available at the level of their transmitters and receivers.

In our protocol, all activities within clusters are coordinated by their centers. In the case of recovery, centers recover the determiners of messages sent in process or cluster. Then, they send these determiners in processes who need the recovery. For $idPtc =$ "*CL*", all processes of the cluster containing process breakdown take back their execution in the last saved point of recovery in last no coordinated checkpoint. The messages received after the last point of recovery are also played again similarly than in the first case where $idPtc = "JP"$. The protocol of coordinated checkpoint of Chandy-Lamport works under the hypothesis of FIFO channels (on a channel all messages are received in order where they were sent). In the execution of the system, any process in a cluster can initiate the calculation of the global state of the system.

Our objective is to save the local status of all messages which circulate. Therefore, if $P1$ the originator of the checkpoint, $P_1$ is going to record the local, state and sends on all the output channels a $MsgCtrl$ message. If a process receives $MsgCtrl$ for the first time, it save in its turn its local state and broadcasts the message. In that case and as channels are FIFO, the state of the channel of this process is empty because, process received all messages issued before the reception of the message $MsgCtrl$, and messages received after the message $MsgCtrl$ are not part of the global state. Otherwise, the state of the input channel relating to the broadcasting process of this message $MsgCtrl$ is equal to messages received between the recording of last local state and the reception of the message.

The algorithm 1 describes the different steps of our protocol.

**Algorithm 1:** Hierarchical checkpoint protocol in Data Grids (HC)

```
Require: $idClust$, $msg$, $freqmsgs$, $FreqMaxIntraMsg$, $nbMsgclust$,
    $numSeq$, $idP$, $TXsauv$, $FreqExPRep$, $etatProc$.
Ensure: State of process, $cInP_i$, $cOutP_i$, $daterep$, $idPtc$
 1: Save the determiner of $msg$ by calling PLP($msg$, $nbMsgclusti$,
    $idClust(Sender)$, $idClust(Receiver)$).
 2: if $msg = ack$ then
 3:   Add $numSeq$ to $msg$
 4:   Send $msg$ to volatile memory
 5:   Send $act$
 6:   Lock the communication
 7: end if
 8: Initiate a coordinated checkpoint if the frequency 10 msg > s
    by calling ICB($freqmsgs$, $FreqMaxIntraMsg$, $TXsauv$,
    $FreqExPRep$)
 9: if $msg = MsgRec$ then
10:   Call $RRP_i(process, idPtc)$
11: else
12:   if $msg = determinant$ then
13:     Send the determiners to $P_i$
14:   end if
15: end if
16: if $msg = MsgCtrl$ then
17:   if $(process = initiator)$ and $(P^{Rep} = False)$ then
18:     Make checkpoint
19:     Send $msg$ to $MC$
20:   else
21:     if $(process <> MC)$ and $(P^{Rep} = False)$ then
22:       Make checkpoint
23:       Send $MsgCtrl$ to $MC$
24:     end if
25:   end if
26: end if
27: if $(process = MC)$ and $(P^{Rep} = False)$ then
28:   Make checkpoint
29:   Broadcast $MsgCtrl$ in cluster
30: end if
31: if $(freqmsgs <= FreqMaxIntraMsg)$ and
    $Size(ListMsg) > MaxNbMsgEmi$ then
32:   Make checkpoint
33: end if
34: Make checkpoint
35: Send to volatile memory the state of the process, $cInP_i$, $cOutP_i$,
    $dateChkpt$ and the $idPtc$
36: $P^{Rep} \Leftarrow True$
```

## IV. Computational Results

We compare the performance of the proposed protocol with Chandy-Lambort (CL), the coordinated checkpoint (CHC), causal (HCL), optimistic logging (OL) and pessimistic logging protocol (PL) (presented in Section II) in grid environment.

We use the Omnet++ simulator [17]. The cluster is configured with 50 nodes. For the grid configuration, 50 nodes were uniformly spread in 5 clusters. The intra-cluster delay is fixed to 0.1 ms and the inter-cluster delay is fixed to 100 ms. Our tests were carried out with 50 application processes. Messages between processes were randomly generated.

Fig. 2 shows the grid configuration in OMNeT ++.

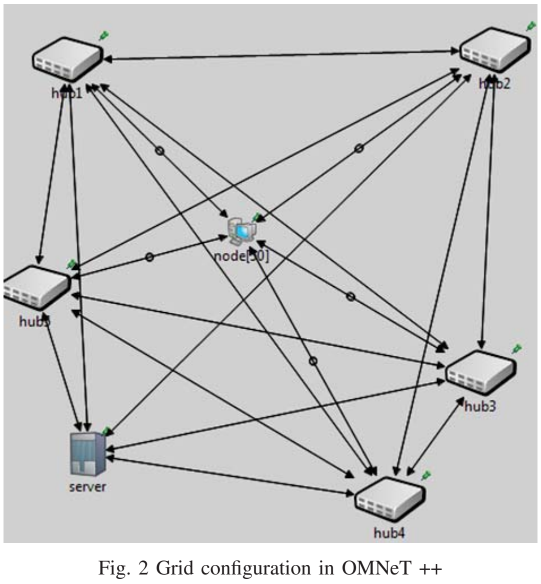
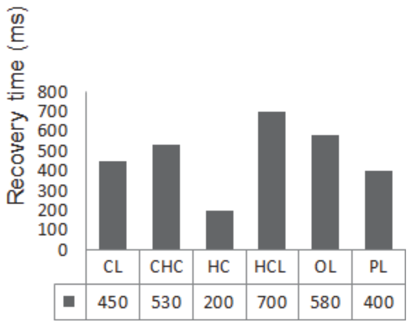


Fig. 2 Grid configuration in OMNeT ++

### A. Recovery Time

The recovery time relies on the number of checkpoints maintained by the protocol and the number of process in rollbacks. In coordinated checkpoint protocol (CHC) and pessimistic logging (PL), recovery is simplified because the system is rolled back only to the most recent checkpoint. In fact, if the faulty node has no dependencies with nodes of other cluster nodes, the fault is confined to the cluster node's fault. So all the nodes of the grid do not perform the recovery procedure. Consequently, if the inter-clusters' communications are intensive, the recovery time increases in the case of causal (CL) and optimistic logging (OL) and was decreased by remarkable way in the case of our hierarchical checkpoint protocol (HCL) (see Fig. 3).



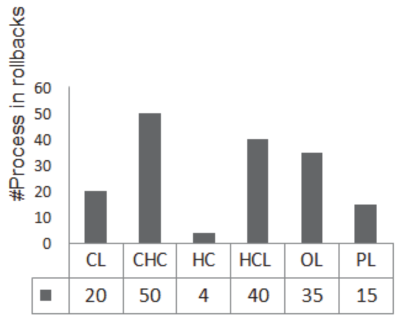
| | CL | CHC | HC | HCL | OL | PL |
|---|---|---|---|---|---|---|
| ■ | 450 | 530 | 200 | 700 | 580 | 400 |

Fig. 3 Recovery time

### *B. Number of Processes in Rollbacks*

For coordinated checkpoint protocol (CHC), all processes must resume during recovery. The logging protocol (CL) reduces the number of rollbacks. This number is minimal in pessimistic protocol (PL) since only faulty processes need to be rolled back. For the other logging protocol (OL), this number depends on the information stored in backups and in the main memory of correct processes.

For our hierarchical checkpoint protocol (HC), only the process in the same cluster except the center must resume during recovery (see Fig. 4).



| | CL | CHC | HC | HCL | OL | PL |
|---|---|---|---|---|---|---|
| ■ | 20 | 50 | 4 | 40 | 35 | 15 |

Fig. 4 Number of process in rollbacks

## V. Conclusion

In this paper, we proposed a fault tolerance model in grid computing. We first looked for related work on hierarchical checkpoint protocols, and we have established a comparative and detailed analysis of different strategies. We concluded that the combination of different protocols may have an important performance in grid computing. The pessimistic log-based protocol founded on the transmitter is applied to inter-clusters messages. In intra-cluster, the pessimistic log-based protocol founded on the transmitter and the not blocking coordinated checkpointing of Chandy-Lamport are folded up according to the messages' frequency. Seen that the real target environments upon which our study relates are widely distributed, it may be difficult to conduct long-term reproducible experiments in such context. We have therefore chosen to use a simulator to test our protocol. Indeed, simulation is a useful way to test solutions before their validation in a real distributed environment. In addition, the use of a simulator gives us the possibility to control all the parameters of the simulated platform, which may be difficult or impossible in a real platform. Currently, the proposed model is being deployed in Omnet++. As our protocol is based on the nodes' clustering, for a large number of processes in rollbacks, we can apply the same procedure in cluster to reduce the number of rollbacks processes. This opens on one of our future work.

## References

[1] O. Marin, "The darx framework: Adapting fault tolerance for agent systems," Ph.D. dissertation, Université de Have, 2003.

[2] B. Hamid, "Distributed fault-tolerance techniques for local computations," Ph.D. dissertation, Université Bordeaux I, 2007.

[3] F. Reichenbach, "Service snmp de dtection de faute pour des systmes rpartis," Ph.D. dissertation, Ecole polytechnique de Lausane, 2002.

[4] M. Wiesmann, F. Pedone, and A. Schiper, "A systematic classification of replicated database protocols based on atomic broadcast," in *3rd Europeean Research Seminar on Advances in Distributed Systems*, 1999.

[5] X. Besseron, "Tolérance aux fautes et reconfiguration dynamique pour les applications distribuées à grande échelle," Ph.D. dissertation, Université de Grenoble, 2010.

[6] N. M. Ndiaye, "Techniques de gestion des dé faillances dans les grilles informatiques tolé rantes aux fautes," Ph.D. dissertation, Université Pierre et Marie Curie, 2013.

[7] S. Drapeau, "Un canevas adaptable de services de duplication," Ph.D. dissertation, Institut National Polytechnique de Grenoble, 2003.

[8] R. Souli-Jbali, M. S. Hidri, and R. B. Ayed, "Dynamic data replication-driven model in data grids," in *39th Annual Computer Software and Applications Conference, COMPSAC Workshops 2015, Taichung, Taiwan, July 1-5, 2015*, 2015, pp. 393–397.

[9] Chandy and Lamport, "Distributed snapshots : Determining global states of distributed systems," *ACM Transactions on Computer Systems*, vol. 3, no. 1, pp. 63–75, 1985.

[10] H. S.Paul, A. Gupta, and R. Badrinath, "Hierarchical coordinated checkpointing protocol," in *International Conference on Parallel and Distributed Computing Systems*, 2002, pp. 240–245.

[11] K. Bhatia, K. Marzullo, and L. Alvisi, "Scalable causal message logging for wide-area environments," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 3, pp. 243–250, 2003.

[12] S. Monnet, C. Morin, and R. Badrinath, "Hybrid checkpointing for parallel applications in cluster federations," in *3rd Workshop on Resiliency in High Performance Computing (Resilience) in Clusters, Clouds, and Grids*, 2004, pp. 773–782.

[13] E. Meneses, C. L. Mendes, and L. V. Kale, "Team based message logging : Preliminary results," in *4th IEEE ACM International Symposium on Cluster Computing and the Grid*, 2010.

[14] J.-M. Yang, K. Li, W.-W. Li, and D.-F. Zhang, "Trading off logging overhead and coordinating overhead to achieve efficient rollback recovery," *Concurrency and Computation: Practice and Experience*, vol. 21, no. 3, pp. 819–853, 2009.

[15] A. Guermouche, "Nouveaux protocoles de tolrance aux fautes pour les applications du calcul haute performance," Ph.D. dissertation, Université Paris-Sud, 2011.

[16] D. B. Johnson and W. Zwaenepoel, "Sender based message logging," in *The Seventeenth Annual International Symposium on Fault-Tolerant Computing*, 1987, pp. 14–19.

[17] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, 2008, pp. 60:1–60:10.