International Journal of Computer Trends and Technology Volume 68 Issue 11, 53-58, November 2020
ISSN: 2231 – 2803 / https://doi.org/10.14445/22312803/IJCTT-V68I11P107 © 2020 Seventh Sense Research Group®

Original Article

A Hybrid Approach for Fault Tolerance in Datagrid

Senhadji sarra[1], MEGAIZ Samia[2], SADOK Riad Mustapha[3]

[1,2,3] *computer science department, faculty of mathematics and informatics, University of Science and Technology Mohamed Boudiaf, Oran, Algeria*

Received Date: 09 October 2020
Revised Date: 19 November 2020
Accepted Date: 20 November 2020

Abstract - *In recent years, we observe a considerable growth of data that needs to be stored, analyzed, and exploited. In response to these needs, grid systems appear to offer large-scale networks and geographic sharing resources around the world. However, grids are extremely dynamic where nodes are heterogeneous and volatile which increases the probability of failure. Two main solutions handle this problem: masking and no masking technique. For the masking one, the fault and its resolution are hidden from the client and the system still being operational. Contrarily to the no masking solution, the fault can stop the execution for a while until the fault is resolved.*

In this paper, we propose a hybrid solution that combines two fault-tolerance methods, one masking and the other non-masking using respectively recovery and replication techniques.

Keywords - *fault tolerance, recovery, replication, Datagrid*

I. INTRODUCTION

Fault tolerance is a method that allows a system to continue to work, possibly in reduced performance, instead of failing, when one of its components no longer functions correctly. In other words, the system does not stop functioning, whether there is hardware failure or software failure.

A grid is a distributed system composed of many nodes, present in different sites, with different configurations. Besides nodes in the grid are volatile and they can join or leave the system at any time. For this reason, it is necessary to guarantee the continuity of the grid whenever a node disappears.

II. FAULT TOLERANCE DETECTION

The detection of faults in a distributed system is not trivial and represents an essential prerequisite for the implementation of the fault tolerance solution. The study of fault detectors is therefore very important.

Failure detector can be classified as reliable or unreliable depending on the results it produces. [17]

If the output of the failure detector is always accurate it is called a reliable failure detector. An unreliable failure detector is one that provides information that is not necessarily accurate and it may take a very long time for detection of faulty process and produce false results, which means that it is impossible to distinguish a slow process from a failed process.

A. Properties of fault tolerance detector

Fault detectors can make errors in their diagnosis. That's why some properties are considered to verify their efficiency: [17]

- **Completeness**: either the detector may not see some faults
- **Accuracy**: the detector can see faults where there are none. That's means when a process is detected as failed, it has failed.
- **Speed**: Time for the detection of failure should be as shorter as possible. In other words, the time between the occurrence of a failure and its prediction must be small.

B. Fault detection mechanism

Different mechanisms are used for tolerating faults:

- **Pro-active vs. post-active mechanisms**: In pro-active mechanisms, the failure consideration for the grid is made before the scheduling of a job, and dispatched with hopes that the job does not fail. Whereas, post-active mechanisms handle the job failures after it has occurred.

- **Push vs. Pull mechanisms**: In the push model, grid components periodically send heartbeat messages [7] to a failure detector, announcing that they are alive. In the absence of any such message from any grid component, the fault detector recognizes that failure has occurred at that grid component. In contrast, in the pull model, the failure detector sends live-ness requests (“Are you alive?” messages) periodically to grid components [7].

III. FAULT TOLERANCE APPROACHES

In the literature, we distinguish two main approaches for tolerating faults in distributed systems: masking fault approaches and no masking fault approaches.

In the masking solution, the fault and its resolution are hidden from the client and the system still being operational. “Replication” is the most known technique in this type of approach. Contrarily to the no masking solution, the fault can stop the execution for a while until the fault is resolved. “Check to point” is the most used technique to recover the occurred faults.

A. Replication

The job replication and determination of the optimal number of replicas involves many technical considerations. The replication in grids has been studied in [Chtepen et al. 2006]. Several approaches have been used to implement replication in a grid computing environment. In general, replication is classified into static and dynamic models [Chtepen et al. 2006]. The static replication means that, when some replica fails, it is not replaced by a new one. The number of replicas of the original task is decided before execution. While in the case of dynamic replication, new replicas can be generated during run time.

B. Checkpointing

It consists of snapshot records of the entire system state to restart the application after the occurrence of some failure. The checkpoint can be stored on temporary as well as stable storage [11]. However, the efficiency of the mechanism is strongly dependent on the length of the checkpointing interval. Frequent checkpointing may enhance the overhead, while lazy checkpointing may lead to loss of significant computation. [14] [10]

Therefore, various types of checkpointing optimization have been considered by the researchers, e.g., (i) Full checkpointing or Incremental checkpointing (ii) Unconditional periodic checkpointing or Optimal (Dynamic) checkpointing (iii) Synchronous (Coordinated) or asynchronous (Uncoordinated) checkpointing, and (iv) Kernel, Application or User level checkpointing.

IV. RELATED WORKS

We present in this section some works of fault tolerance in grid systems based on replication, checkpointing, and hybrid solutions.

A. Works based on replication solution

Cherian et al. [Cherian et al. 2010] proposed a solution for handling faults in a grid environment. Fault-Tolerance using Adaptive Replication in Grid Computing (FTARG) is an adaptive replication middleware which addresses the fault tolerance of Grid-based applications by providing data replication at different sites. FTARG is an Aneka based Grid middleware especially designed for high-performance Grid-based applications. FTARG enables data synchronization between multiple heterogeneous databases located in the Grid by supporting a variety of synchronization modes.

The author of [5] introduces several dynamic on-line scheduling heuristics that reduce task loss and execution delay resulting from resource failures. The heuristics are based upon task replication and rescheduling of failed tasks. The characteristic of the proposed methods is the relative simplicity and the efficiency with which they are dealing with dynamic grid environments.

[2] addressed the problem of scheduling user jobs in grids so that failures can be avoided in the presence of resource faults. The author employed job replication as an effective mechanism to achieve an efficient and fault-tolerant scheduling system. Most of the existing replication-based algorithms use a fixed number of replications for each job which consumes more grid resources. An algorithm was proposed to determine adaptively the number of job replicas according to the grid failure history. Then another algorithm is proposed to schedule these replicas. The obtained results showed better performance in terms of grid load, throughput, and failure tendency.

Other works can be cited as [15], [6] …..

B. Works based on a checkpointing solution

A classical fault repair has been proposed in [Elnozahy et al. 2002]. The basic principle is to back up the state of the system periodically on reliable and persistent support (e.g. hard disk). In this way, when restarting after a failure, the most recent backup state is restored and the execution resumes its execution before the failure. The global state of a distributed system is defined by the union of local states of all processes belonging to the system.

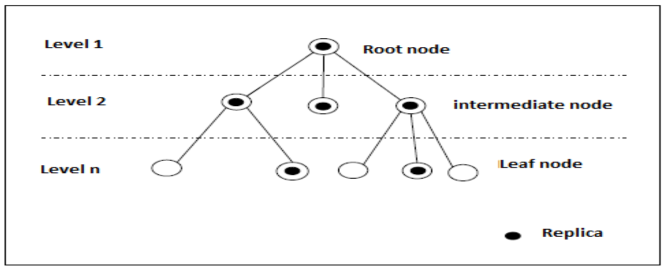


Fig. 1 proposed grid topology

Authors of [8] proposed a grid workflow system (grid-

WFS), a flexible failure handling framework for the grid, which addresses these grid-unique failure recovery requirements. Central to the framework is flexibility by the use of workflow structure as a high-level recovery policy specification. They show how this use of a high-level workflow structure allows users to achieve failure recovery in a variety of ways depending on the requirements and constraints of their applications.

In [9] the author proposed a new strategy named RFOH for fault-tolerant job scheduling in computational grid. This strategy maintains the history of fault occurrence of resources in Grid Information Server (GIS). Whenever a resource broker has jobs to schedule, it uses this information in the Genetic Algorithm and finds a near-optimal solution for the problem.

Other works can be cited as [3], [1], [4] …….

C. Works based on a hybrid solution (replication and checkpointing)

[16] introduced Satin system recently schedules the engrained tasks of a divide-and-conquer application across multiple clusters in a grid. To accommodate long-running applications. They present a fault tolerance mechanism for Satin that has negligible overhead during normal execution while minimizing the amount of redundant work done after a crash of one or more nodes

Another work of [13] proposed two-hybrid fault tolerance techniques (FTTs) that are called alternate tasks with checkpoints and alternate tasks with retry. These proposed hybrid FTTs inherit the good features and overcome the limitations of workflow level FTT and task level FTT. Authors conclude that the alternate task with checkpoint improves the reliability of a grid system more significantly than the alternate task with retry.

The fault tolerance works cited in this section are not exhaustive, but only a few works used replication and checkpointing together to benefit from the advantages of each one. For this reason, we propose a hybrid solution to improve fault tolerance management.

V. PROPOSED APPROACH

In this section, we present in detail our contribution to the fault tolerance problem.

A. Grid topology :

We propose a hierarchical architecture with n levels. As illustrated in fig 1, the root node supervises all descendant nodes. The original data is hosted in the root node and replicated over the intermediate nodes.

B. Proposed solution:

To ameliorate the fault tolerance, we use both replication and recovery solutions.

a) Replication

The replication allows us to reduce the time of transfer of data between nodes and guarantee high availability and reliability. However, replication of data over all nodes of the grid is complex and needs to be controlled whenever a replica is updated. To reduce the number of replicas, we choose a dynamic replication strategy, where the frequently used data are replicated periodically.

Replication algorithm

```
Input:grid configuration file
Output: grid replication file
//root node :
For each node i of the grid do
If node j has minimal transfer
Then if replica k doesn’t exist in node j
Then create replica k in node j
Else overwrite the less frequently
used replica by replica k
End if
End if
End
```

b) Recovery

The recovery technique allows us to replace the root node, whenever it fails, with another node that is closest to the hierarchy.

Recovery algorithm

```
Input:grid configuration file
Output: grid replication file
Ni: root node of the grid
Nj: another node of the grid

The root node Ni replicates data over all his child nodes
If Ni fails then if Nj is not failing with good configuration
and is closest to the root node in the hierarchy
Then replace the root node Ni with Nj.
```

c) Fault tolerance

Once the data is replicated the fault tolerance can be managed. Indeed, when a node fails, the data will be available on another node; so we can get a replica on the nearest neighbor node to minimize the response time.

We used a hybrid solution for fault tolerance. For the root node, a masking solution that consists of replacing the root node with another one having the best configuration is used. For the other nodes, a no masking solution is used and in case of failure, we search the nearest replica.

Fault tolerance algorithm
Considering:
Ni: root node of the grid
Nj, Nk: other nodes of the grid
Masking solution (root node)
If Ni fails then
If Nj id the nearest in the hierarchy and has a good configuration
Then Ni is replaced by Nj
End if
End if
No masking solution (node ≠ root node)
If an access data is requested to Nj and Nj is failed then a replica of data is searched in another node Nk
If Nk is not failing and is near to Nj in term of transfer time then the request is redirected to the node Nk

C. Experimentations:

To evaluate our approach we use the Optorsim tool [12] which is a Grid simulator designed to test dynamic replication strategies.

a) Simulation :

Optimism offers many types of optimizers; we choose the most used ones:

- **SimpleOptimiser** - no replication
- **LruOptimiser** - always replicates, deleting least recently created file
- **LfuOptimiser** - always replicates, deleting least frequently accessed file.

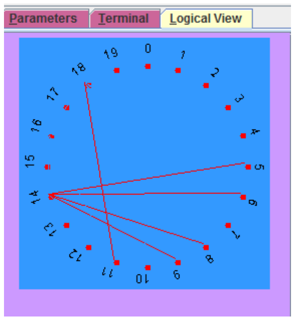
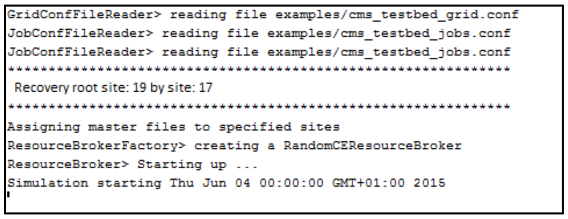


Fig. 2 logical view of replicas

In figure 2 we have an example of a logical view of the replicas hosted in 20 sites.

Masking solution (root site)

The recovery of the root site is performed as described in section **IV. B. 2**.

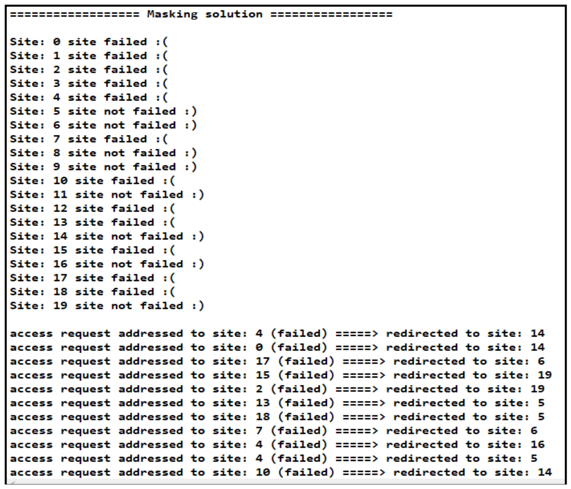


```
GridConfFileReader> reading file examples/cms_testbed_grid.conf
JobConfFileReader> reading file examples/cms_testbed_jobs.conf
JobConfFileReader> reading file examples/cms_testbed_jobs.conf
************************************************************
 Recovery root site: 19 by site: 17
************************************************************
Assigning master files to specified sites
ResourceBrokerFactory> creating a RandomCEResourceBroker
ResourceBroker> Starting up ...
Simulation starting Thu Jun 04 00:00:00 GMT+01:00 2015
```

Fig. 3 No masking solution output

No masking solution (site ≠ root site)

When access is addressed to a failed site, the request is redirected to another site as described in section **IV. B. 1**.



```
================= Masking solution =================

Site: 0 site failed :(
Site: 1 site failed :(
Site: 2 site failed :(
Site: 3 site failed :(
Site: 4 site failed :(
Site: 5 site not failed :)
Site: 6 site not failed :)
Site: 7 site failed :(
Site: 8 site not failed :)
Site: 9 site not failed :)
Site: 10 site failed :(
Site: 11 site not failed :)
Site: 12 site failed :(
Site: 13 site failed :(
Site: 14 site not failed :)
Site: 15 site failed :(
Site: 16 site not failed :)
Site: 17 site failed :(
Site: 18 site failed :(
Site: 19 site not failed :)

access request addressed to site: 4 (failed) =====> redirected to site: 14
access request addressed to site: 0 (failed) =====> redirected to site: 14
access request addressed to site: 17 (failed) =====> redirected to site: 6
access request addressed to site: 15 (failed) =====> redirected to site: 19
access request addressed to site: 2 (failed) =====> redirected to site: 19
access request addressed to site: 13 (failed) =====> redirected to site: 5
access request addressed to site: 18 (failed) =====> redirected to site: 5
access request addressed to site: 7 (failed) =====> redirected to site: 6
access request addressed to site: 4 (failed) =====> redirected to site: 16
access request addressed to site: 4 (failed) =====> redirected to site: 5
access request addressed to site: 10 (failed) =====> redirected to site: 14
```

Fig. 4 Masking solution output

b) Results :

In this section, we present the results of the simulation

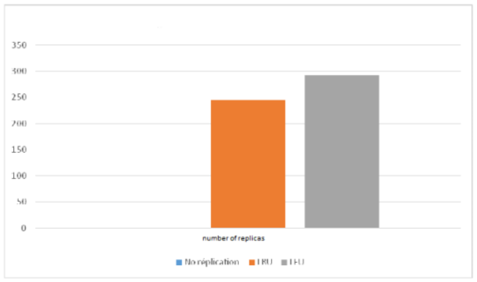


Fig. 5 number of replicas

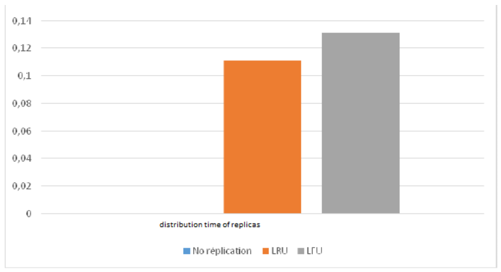


Fig. 6 distribution time of replicas

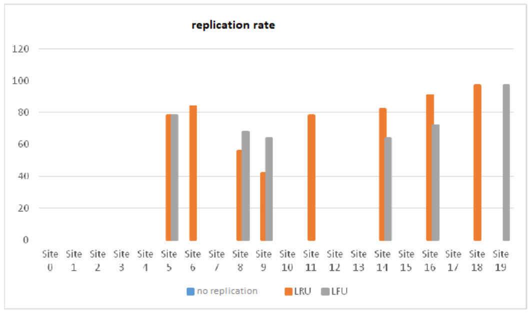


Fig. 7 replication rate

In Fig 5, Fig 6 we have the total number of created replicas and the time consumed for replication.

The rate of replication in each site is shown in Fig 7.

Optimizer 1: no replication

In fig. 9 the number of redirected request is null because there are no replicas in sites, except the root node

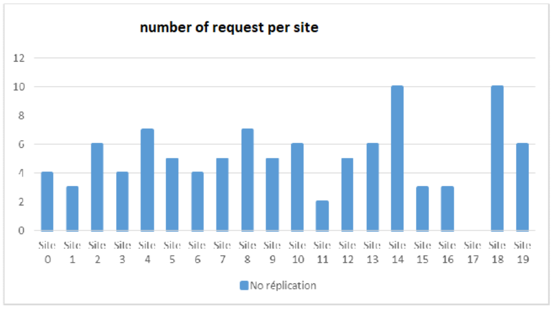


Fig. 8 number of request per site

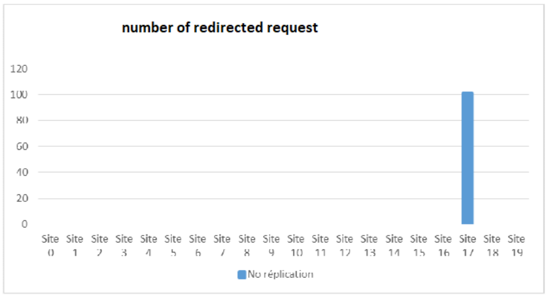


Fig. 9 number of a redirected request

Optimizer 2: LRU

In fig. 10 and fig. 12 we remark that requests addressed to sites that do not have a replica (i.e. site 5, 6, 8, 9, 16, and 18) are redirected to the nearest site to respond to the request.

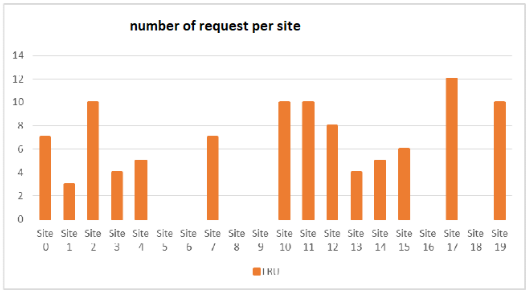


Fig. 10 number of request per site

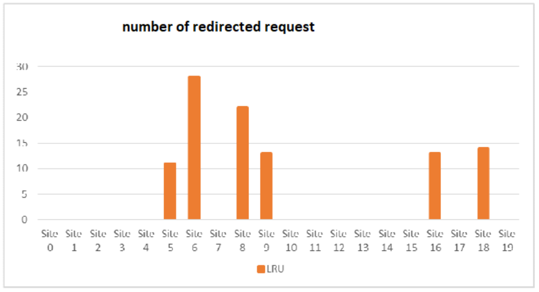


Fig. 11 number of a redirected request

Optimizer 3: LFU

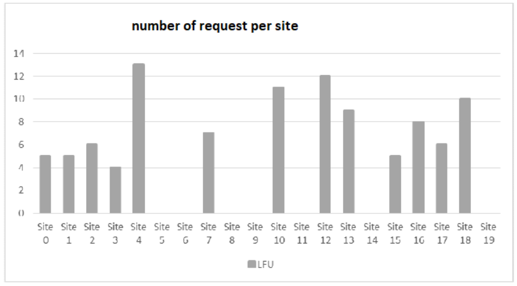


Fig. 12 number of request per site

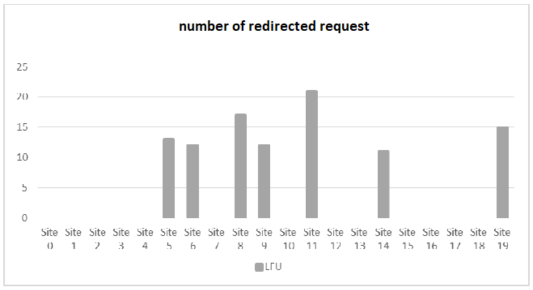


Fig. 13 number of a redirected request

In fig. 12 and fig. 13 we remark that requests addressed to sites that do not have a replica (i.e. site 5, 6, 8, 9, 11, 14, and 19) are redirected to the nearest site to respond to the request.

In no masking solution, replication technique is used to guarantee a high availability where several copies can be saved in different sites offering better response time.

Otherwise, the masking solutions are based on a recovery technique that consists of repairing fault but this solution presents some inconvenient as synchronization

For these reasons, we choose a hybrid solution based on masking and no masking solution using respectively recovery and replication techniques. We simulate our approach with the simulator GRIDSIM and many experiments were presented. The obtained results show a significant gain in terms of fault tolerance and execution time.

In conclusion, some perspectives on our work can be cited:

• Use of recovery check pointing methods or probabilistic methods to recover the state of the system in the event of a failure.

• Adaptation of our approach in a real grid.

REFERENCES

[1] Abawajy, Jemal H. "Fault-tolerant scheduling policy for grid computing systems", 18th International Parallel and Distributed Processing Symposium, 2004. Proceedings. IEEE, 2004.

[2] Amoon, Mohammed. "Fault tolerance in grids using job replication", International Journal of Computing 11.2 (2014): 115-121.

[3] Balazinska, Magdalena, et al. "Fault-tolerance in the Borealis distributed stream processing system." Proceedings of the 2005 ACM SIGMOD international conference on Management of data. 2005

[4] Chandy, K. Mani, and Leslie Lamport. "Distributed snapshots: Determining global states of distributed systems." ACM Transactions on Computer Systems (TOCS) 3.1 (1985): 63-75.

[5] Chtepen, Maria, et al. “Evaluation of Replication and Rescheduling Heuristics for Grid Systems with Varying Resource Availability.” Proceedings of the 18th IASTED International Conference on Parallel and Distributed Computing and Systems, ACTA Press Anaheim, 2006, pp. 622–27.

[6] Erciyes, Kayhan. "A replication-based fault tolerance protocol using group communication for the grid." International Symposium on Parallel and Distributed Processing and Applications. Springer, Berlin, Heidelberg, 2006.

[7] Garg Ritu, Singh Kumar Awadhesh, “Fault Tolerance in Grid Computing: State of the Art and Open Issues,” International Journal of Computer Science & Engineering Survey (IJCSES) Vol.2, No.1, Feb 2011.

[8] Hwang, S., and C. Kesselman. “Grid workflow: a flexible failure handling framework for the grid.” High Performance Distributed Computing, 2003. Proceedings. 12th IEEE International Symposium on (2003): 126-137.

[9] Leili Mohammad Khalil, Maryam Etminan and Far Amir Masoud Rahman, ”RFOH: A New Fault-Tolerant Job Scheduler in Grid Computing”, In Second International Conference on Computer Engineering and Applications (2010).

[10] A. Nguyen-Tuong, “Integrating fault-tolerance techniques in Grid applications”, Ph.D. Dissertation, University of Virginia, August 2000.

[11] Oliner, A.J., Sahoo, R.K., Moreira, J.E., Gupta, M.: “Performance Implications of Periodic Checkpointing on Large-Scale Cluster Systems”, In Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, Washington, 2005.

[12] https://sourceforge.net/projects/optorsim/

[13] Qureshi, K., Khan, F.G., Manuel, P. et al., “A hybrid fault tolerance technique in grid computing system”. J Supercomput 56, 106–128 (2011).

[14] Eric Roman, “A survey of Checkpoint/Restart Implementations”, Lawrence Berkley National Laboratory, CA, 2002.

[15] Townend, Paul, and Jie Xu. "Fault tolerance within a grid environment." Timeout 1.S2 (2003): S3.

[16] Wrzesińska, Gosia, et al. "Fault-tolerant scheduling of fine-grained tasks in grid environments." The International Journal of High-Performance Computing Applications 20.1 (2006): 103-114.

[17] Amin, Zeeshan, Harshpreet Singh, and Nisha Sethi. "Review on fault tolerance techniques in cloud computing." International Journal of Computer Applications 116.18 (2015).

[18] Matarneh, Feras & Matarneh, Rami. “Enhancing Fault-Tolerance in Ring Topology Based on Waiting Queue and Timestamp”. International Journal of Computer Trends and Technology. (2017)

VI. CONCLUSION

In this paper, we dressed the fault-tolerance problem in grid systems. We find two main solutions that handle this problem: masking technique and no masking technique.

For the masking one, the fault and its resolution are hidden from the client and the system still being operational. Contrarily to the no masking solution, the fault can stop the execution for a while until the fault is resolved.

The masking technique indeed appears as the best solution because it assures the continuity of the system, but this solution is too expensive to implement especially when the number of resources grows.