ISSN:2231-0711
www.ijcset.net

Survey on Fault Tolerance Techniques on Grid

Geeta Arora
Computer Application, RBIM, Mohali.
mailtogeeta@gmail.com

Dr. Shaveta Rani
Computer Sc. & Engg., G.Z.S.C.E.T, Bathinda.
garg_shavy@yahoo.com

Dr. Paramjit Singh
Computer Sc. & Engg.,G.Z.S.C.E.T, Bathinda.
param2009@gmail.com

***Abstract*— In a grid environment there are thousands of resources, services and applications that need to interact in order to make possible the use of the grid [1] as an execution platform. Since these elements are extremely heterogeneous, volatile and dynamic, there are many failure possibilities, including not only independent failures of each element, but also those resulting from interactions between them. Because of the inherent instability of grid environments, fault-detection and recovery is another critical component that must be addressed. The need for fault-tolerance is especially sensitive for large parallel applications since the failure rate grows with the number of processors and the duration of the computation.In this paper we will discuss the various fault management stratigies that will help to achieve the fault tolerance and is good reference to researcher**.

***Keywords*— Fault Tolerance, Grid Computing, Fault Prevention, Fault Avoidance, Fault Detection, Fault Recovery**

I. INTRODUCTION

Grid computing facilitates coordinated resource sharing and problem solving in heterogeneous, volatile, dynamic and multi-institutional collaborations .Grid computing typically involves using many resources (computer, data, I/O, instruments, etc.) to solve a single, large problem that could not be executed on any one resource. These enable sharing, selection and aggregation of suitable computational and data resources for solving large-scale compute-intensive, data-intensive, collaboration-intensive problems in science, engineering, and commerce and address problems ranging from fault diagnosis in jet engines and earthquake engineering to bioinformatics, biomedical imaging, and astrophysics.

The applications cited above need a coordinated resource sharing, where the sharing is not primarily file exchange but rather direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science and engineering. Thus, this sharing is, necessarily, highly controlled, with resource provides and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what is called a virtual organization (VO). Grid can offer various services [2] such as:

- *Compute services*: CPU cycles by pooling computational power.
- *Data services*: Collaborative sharing of data generated by people, processes and devices such as sensors and scientific instruments.
- *Application services:* Access to remote software services/libraries and license management.
- *Interaction services:* E.learning, virtual tables, group communication and gaming.
- *Knowledge services:*Data minimg and knowledge acquisition,processing and management

II. GRID ARCHITECTURE [2]

The components that are necessary to form a Grid are as follows.

• *Grid fabric*. This consists of all the globally distributed resources that are accessible from anywhere on the Internet. These resources could be computers (such as PCs or Symmetric Multi-Processors) running a variety of operating systems (such as UNIX orWindows), storage devices, databases, and special scientific instruments such as a radio telescope or particular heat sensor.

• *Core Grid middleware*. This offers core services such as remote process management, co-allocation of resources, storage access, information registration and discovery, security, and aspects of Quality of Service (QoS) such as resource reservation and trading.

• *User-level Grid middleware*. This includes application development environments, programming tools and resource brokers for managing resources and scheduling application tasks for execution on global resources.

• *Grid applications and portals*. Grid applications are typically developed using Grid-enabled languages and utilities such as HPC++ or MPI. An example application, such as parameter simulation or a grand-challenge problem, would require computational power, access to remote data sets, and may need to interact with scientific instruments. Grid portals offer Web-enabled application services, where users can submit and collect results for their jobs on remote resources through the Web.

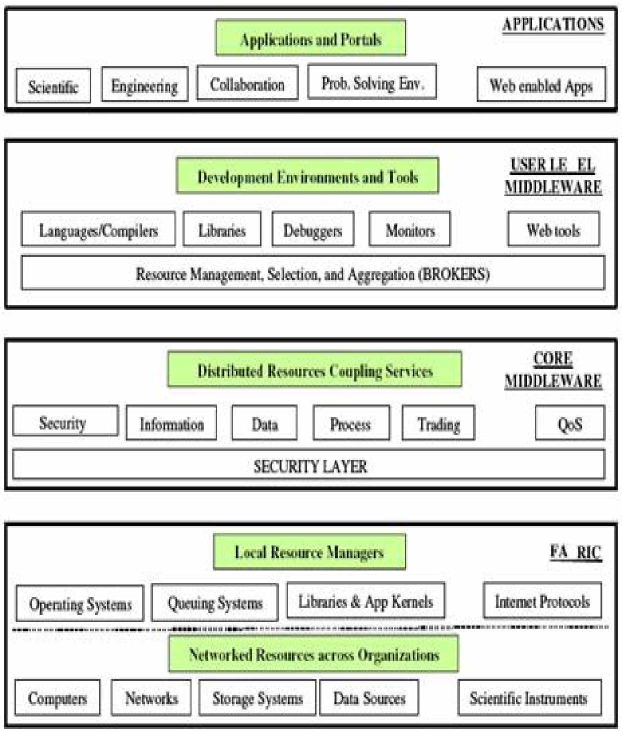

Figure 1: Grid Architecture

III. FAULT MANAGEMENT STRATIGIES

1. FAULT PREVENTION

This is, basically, about how requests about resources are made and how they are permitted.They consider pro-active mechanisms [3] for fault management in which failure consideration for the grid is made before the scheduling of a job, and dispatched with hopes that the job does not fail. The static replication [3] technique in fault tolerance comes under this category in which number of replicas of the original task is decided before execution. Fault tolerance system runs different replicas of same task on different grid resources simultaneously expecting that at least one of them will complete successfully

2. FAULT AVOIDANCE

The system dynamically considers every request and decides whether it is safe to grant the resource it at this point, the system requires additional apriori information regarding the overall potential use of each resource for each process. It includes Information aware Scheduling [4], having the information about resource availability and task execution time, the scheduler can determine which resource is able to complete the task without failure. Roughly speaking, there are two types of information that can be exploited by a scheduler, namely the information about the characteristics of the tasks, and the information about the characteristics of machines. The activity of a scheduler may be abstractly described as consisting of two steps:
1) Selection of the next task to be executed, and
2) Selection of the machine on which it will be executed.
We will start by considering task selection first, and then we will move to machine selection. In particular, we will discuss two task selection policies (called LongET and ShortET) and four machine selection policies (called NoKnow, CpuKnow, AvailKnow and AllKnow) that, when combined, give rise to 8 different scheduling algorithms. Our task selection policies will assume the knowledge of the execution time of tasks on the reference machine, while our four machine selection policies will assume no knowledge, the knowledge of the computing power, of the next failure time, and of both of them for each machine, respectively

3. FAULT DETECTION

In order to detect occurrence of fault in any grid resource two approaches can be used: the push or the pull model. In the push model [3], grid components periodically send heartbeat messages 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. It then implements appropriate measures dictated by the predefined fault tolerance mechanism. In contrast, in the pull model [3] the failure detector sends live-ness requests periodically to grid components. The heart beating technique [3] can further classified into 3 types:

- *Centralized Heart beating* - Sending heartbeats to a central member creates a hot spot, an instance of high asymptotic complexity.
- *Ring Based Heart beating* - along a virtual ring suffers from unpredictable failure detection times when there are multiple failures, an instance of the perturbation effect.
- *All-to-all Heart beating* - sending heartbeats to all members, causes the message load in the network to grow quadratically with group size, again an instance of high asymptotic complexity

4. FAULT RECOVERY

They consider post-active mechanisms [3] which handle the job failures after it has occurred.

4.1 Task level fault tolerance techniques [5]:

Task level techniques refer to recovery techniques that are applied at the task level to mask the effect of faults irrespective of fault types Task level FTTs include the following:

4.1.1 Retry—Retry technique for fault tolerance is the simplest technique being used. After a failure it retries the task on the same grid resource regardless the cause of failure up to some threshold value with the expectation that there will be no failure in successive attempts.

4.1.2 Alternate resource—The alternate resource technique works just like the retry technique except it retries on an alternate resource rather than retrying on the same resource again and again

4.1.3 Checkpoint—The checkpoint technique periodically saves the state of an application. On failure it moves the task to another resource and starts the execution from the last saved checkpoint.

4.1.3.1 Uncoordinated or coordinated checkpoint [3]
Uncoordinated Checkpoint: In this approach, each of the processes that are part of the system determines their local checkpoints independently of the other processes though it may lead to domino effect (processes may be forced to rollback up to the execution beginning). During restart, these checkpoints have to be searched in order to construct a consistent global checkpoint.

Coordinated Checkpoint: In this approach, the Checkpointing is orchestrated such that the set of individual checkpoints always results in a consistent global checkpoint. This minimizes the storage overhead, since only a single global checkpoint needs to be maintained on stable storage. Algorithms used in this approach are blocking and nonblocking

4.1.3.2 Full Checkpoint or Incremental checkpoint [3]

A full checkpoint is a traditional checkpoint mechanism which occasionally saves the total state of the application to a local storage. However, the time consumed in taking checkpoint and the storage required to save it is very large. Incremental checkpoint mechanism was introduced to reduce the checkpoint overhead by saving the pages that have been changed instead of saving the whole process state. In the incremental checkpoint scheme, the first checkpoint is typically a full checkpoint. After that, only modified pages are checkpointed at some predefined interval. When large numbers of pages get modified another full checkpoint is taken. In order to recover the application, we will load a saved state from the last full checkpoint and load the changed pages from each incremental checkpoint following the last full checkpoint. This results in more expensive recovery cost than the recovery cost of the full checkpoint mechanism.

4.2 Workflow level fault tolerance techniques [5]:
Workflow level FTTs change the flow of execution on failure based on the knowledge of task execution context. Workflow level FTTs are classified as follows:

4.2.1. Alternate task—is similar to retry technique. The only difference is that it exchanges a task with different implementation of same task with different execution characteristics on failure of the first one.

4.2.2 Redundancy—the redundancy technique requires different implementations of same task with different execution characteristics which run in parallel as opposed to task level replication technique, where same tasks are replicated on different grid resources.

4.2.3. User defined exception handling—In user defined exception handling technique user specifies the particular treatment to workflow of a task on failure.

4.2.4 Rescue workflow—the rescue workflow technique allows the workflow to continue even if the task fails until and unless it becomes impossible to move forward without catering the failed task.

4.3 Hybrid Fault tolerance techniques [5]:

4.3.1 Alternate task with retry

The simplest solution to this problem is to hybrid a task level FTT and workflow level FTT. This is to avoid failures at both task level and workflow level separately. Alternate task with retry is a hybrid of “alternate task” FTT at workflow level and “retry” FTT at task level. After the failure of an alternate task, alternate task with retry FTT simply retries the failed alternate task on the same resource up to a certain threshold value which is in our case three. In this way we can overcome failures of a system to a certain extent.

4.3.2 Alternate task with checkpoint

In this FTT we choose task level checkpoint FTT with workflow level alternate task FTT in order to minimize the failures. When a task fails the first time, alternate task manager finds an appropriate alternate task and forwards it to the job dispatcher. The job dispatcher submits the alternate task to the same resource. When the alternate task fails the first time, the checkpoint manager applies checkpoints to the task and forwards it to the job dispatcher. The job dispatcher submits it to the same grid resource. When the alternate task fails again, the checkpoint manager retrieves the intermediate results of the last saved checkpoint from the checkpoint information server (CIS) and forwards the incomplete gridlet with intermediate results to the job dispatcher.

The job dispatcher in turn submits the incomplete gridlet and intermediate results to another suitable grid resource

5. FAULT IGNORANCE

Ignore the problem and imagine the fault never will never occur. Also, as we may still hope that the application executes successfully

CONCLUSION

In the light of above survey, fault tolerance plays an important role in order to achieve availability and reliability of a grid system. The performance [of defferent techniques is evaluated in different conditions, on different parameters such as throughput, turnaround time, waiting time and transmission delay. Because of the simplicity of implementation, retry and alternate resource techniques are being mostly used as compared to replication and checkpointing techniques. Replication provides better reliability and improved response time. The waiting time of ‘alternative task’ techniques is high it is due to the resubmission of slow task, but it works well under high workloads and with different percentages of faults injected in a system.The reason for good performance of ‘alternative task’ is to takes less network delay as compare to other FTTs. On the other hand when we have low workload and with high percentage of faults in a system ‘checkpointing’ give better results than alternative task and other FTTs. Our comparative study will help other researchers in order to understand the behavior and performance of different FTTs for Grid computing environment.

REFERENCES

[1] Chopra Inderpreet, *Fault Tolerance in Computational Grids*, ME Thesis ,Thapar University, May 2006

[2] Baker Mark, Buyya Rajkumar, Laforenza Domenico , *Grids and Grid technologies for wide- area distributed computing*, Volume: 32, Issue: 15, John Wiley & Sons,1437-1466 ,2002

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

[4] Anglano Cosimo, Brevik John and et al., *Fault-aware scheduling for Bag-of-Tasks applications on Desktop Grids*, IEEE/ ACM International Conference on Grid Computing, 28-29 Sept. 2006.

[5] Manuel Paul, Khan Fiaz Gul and et al., *A hybrid fault tolerance technique in grid computing system*, Springer Science Business Media, Vol 56, No. 106-128, January 2010.

[6] I. Foster, C. Kesselman and et al., *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, Lecture Notes in Computer Science, 2001

[7] Latchoumy P , Khader Abdul Sheik P, *Survey of fault tolerant techniques for grid*, International Journal of Computer Science & Engineering Survey (IJCSES) Vol.2, No.4, November 2011.

[8] Townend Paul, Xu Jie, *Fault Tolerance within a Grid Environment*, Proceedings of the UK e-Science, 2003.

[9] Khan Fiaz Gul, Qureshi Kalim and et al., *Performance evaluation of fault tolerance techniques in grid computing system*, Science Direct, Vol 36 ,No 1110-1122 ,May 2010

[10] Buyya Rajkumar, Murshed Manzur , *GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing*, Concurrency and computation :Practice and Experience , 1175–1220 , February 2002

[11] Dabrowski Christopher, *Reliability in Grid Computing Systems*, Wiley InterScience National Institute of Standards and Technology, 100 Bureau Drive, Stop 8970, Gaithersburg, MD 20899-8970, U.S.A.,2009.

[12] Yi Sangho, Kondo Derrick and et al., *Using Replication and Checkpointing for Reliable Task Management in Computational Grids*, High Performance Computing and Simulation , 125 – 131,June 28 2010-July 2 2010,

[13] Chopra Inderpreet, Singh Maninder, *Automating the fault tolerance process in Grid Environment*, International Journal of Computer Science and Information Security, Vol. 8, No. 7, October 2010.

[14] Manuel Paul, Qureshi Kalim and et al., *Adaptive check pointing strategy to tolerate faults in economy based grid*, Springer Science Business Media, Vol 50,No. 1–18, October 2008.

[15] Canonico Massimo, *Scheduling Algorithms for Bag-of-Tasks Applications on Fault-Prone Desktop Grids*, 2005.

[16] Nguyen-Tuong Anh, *Integrating Fault-Tolerance Techniques in Grid Applications*, August 2000.