SURVEY ON FAULT TOLERANCE IN GRID COMPUTING

P. Latchoumy[1] and P. Sheik Abdul Khader[2]

[1]Research Scholar, Department of Information Technology, BSA University, Vandalur, Chennai, Tamil Nadu, India.

lak75dhana@gmail.com

[2] Professor & Head, Department of Computer Applications, BSA University, Vandalur, Chennai, Tamil Nadu, India.

hodca@bsauniv.ac.in

ABSTRACT

Grid computing is defined as a hardware and software infrastructure that enables coordinated resource sharing within dynamic organizations. In grid computing, the probability of a failure is much greater than in traditional parallel computing. Therefore, the fault tolerance is an important property in order to achieve reliability, availability and QOS. In this paper, we give a survey on various fault tolerance techniques, fault management in different systems and related issues. A fault tolerance service deals with various types of resource failures, which include process failure, processor failure and network failures. This survey provides the related research results about fault tolerance in distinct functional areas of grid infrastructure and also gave the future directions about fault tolerance techniques, and it is a good reference for researcher.

KEYWORDS

Fault Tolerance, Dependability, Checkpoint-Recovery, Redundancy, Scheduling, Advance Reservations, Workflow, Service Level Agreement (SLA), Agent-Oriented, Co-allocation and Load Balancing.

1. INTRODUCTION

Grid computing is a form of distributed computing that involves coordinating and sharing computational power, data, and storage and network resources across dynamic and geographically dispersed organizations [1]. Management of these resources becomes complex as the resources are geographically distributed, heterogeneous in nature, owned by different individual or organizations with their own policies, have different access models, and have dynamically varying loads and availability.

To achieve the promising potentials of computational grids, the fault tolerance is fundamentally important since the resources are geographically distributed. Moreover the probability of a failure is much greater than in traditional parallel computing and the failure of resources affects

job execution fatally. It is therefore to investigate the fault tolerance techniques for grid computing. Fault tolerance is the ability of a system to perform its function correctly even in the presence of faults and it makes the system more dependable. Fault tolerance is the survival attribute of computer system: describe the function of fault tolerance "to preserve the delivery of expected services despite the presence of fault caused errors within the system itself, errors are detected and corrected, and permanent fault are located and removed while the system continues to deliver acceptable service". The fault tolerance service is essential to satisfy QOS requirements in grid computing and it deals with various types of resource failures, which include process failure, processor failure and network failures. This may lead to job/resource failure, violating timing deadlines and Service Level Agreement (SLA), and degraded user expected QOS.

1.1. Dependability Taxonomy

Fault tolerance makes to achieve system dependability. Dependability is related to some QoS aspects provided by the system, it includes the attributes like reliability and availability. The dependability taxonomy is shown in figure 1.

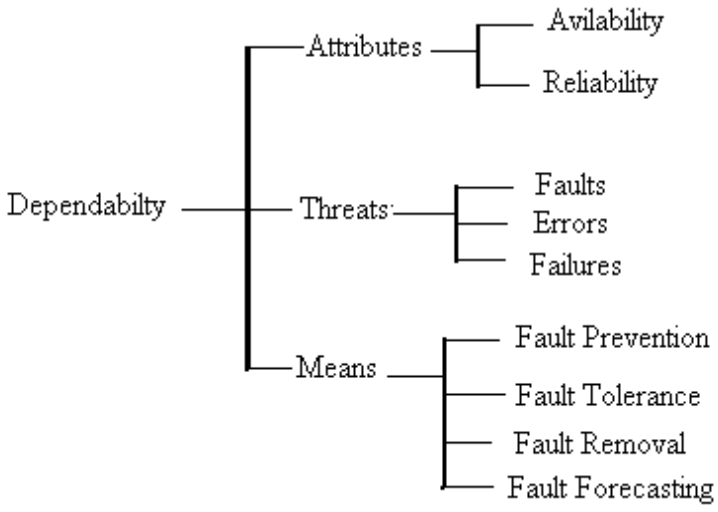

Figure 1. Dependability Taxonomy

Reliability indicates that a system can run continuously without failure. A highly reliable system is the one that continues to work without any interruption over a relatively long period of time. Availability means that a system is immediately ready for use. Fault tolerance techniques are often used to increase the availability and reliability.

Reliability R (t): It is probability that the system has been up continuously in the time interval [0,t]. Reliability is closely related to Mean Time to Failure (MTTF) and Mean Time between Failures (MTBF). MTTF is the average time the system operates until a failure occurs, whereas the MTBF is the average time between two consecutive failures. The difference between the two is due to the time needed to repair the system following the first failure. Denoting the Mean Time to Repair by MTTR, we obtain MTBF=MTTF+MTTR.

Availability A (t): It is the average fraction of time over the interval [0,t] that the system up. The availability can be calculated from MTTF, MTBF, and MTTR as follows: A=MTTF/MTBF=MTTF/ (MTTF+MTTR).

It is possible for a low-reliability system to have high availability: consider a system that fails every hour on the average but comes back up after only a second. Such a system has an MTBF of just 1 hour and, consequently, a low reliability; however, its availability is high: A=3599/3600=0.99972.

To date, developing methods to ensure reliability of grid resources have largely meant developing methods for fault tolerance. Fault tolerance consists of: (1) detecting faults and failures in grid resources and (2) recoveries to allow computations to continue.

Threats are classified as faults, Errors and failures. A fault (or failure) can be either a hardware defect or a software / programming mistakes (bug). An error is a manifestation of the fault/failure/bug. Hardware faults can be classified into permanent, transient, intermittent, benign or malicious [in figure 2].

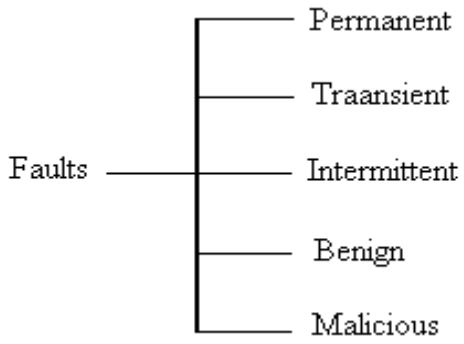

Figure 2. Fault Taxonomy

A permanent fault reflects the permanent going out of commission of a component. A transient fault is one that causes a component to malfunction for some time; it goes away after that time and the functionality of the component is fully restored. An intermittent fault never quite goes away entirely; it oscillates between being quiescent and active. When the fault is quiescent, the component functions normally; when the fault is active, the component malfunctions. A fault that just causes a unit to go dead is called benign. Such faults are the easiest to deal with. For more insidious are the malicious faults that cause a unit to produce reasonable-looking, but incorrect, output, or that make a component: act maliciously" and send differently valued outputs to different receivers.

In an agent oriented, pro-active fault tolerant grid framework was used in which faults are divided into six classes: (a) Hardware Faults: CPU, memory, storage, (b) Application and OS Faults: memory leaks, resource unavailable, (c) Network Faults: node failure, link failure, packet

loss, (d) Software Faults: Un-handles exception, unexpected input, (e) Response Faults: Value faults and (f0 Timeout faults. These are further divided into different sub classes, where agents deal with individual faults proactively. Agents maintain information about hardware conditions, executing process memory consumption, available resources, network conditions and component mean time to failure. Based on this information and critical states, agent enables the grid system to tolerate faults [5].

It has been observed that omission, interaction and timing faults are more prevalent in grid computing. Omission faults will arise when resources become unavailable. Interaction faults may be due to different services supporting different protocols, security incompatibilities, and policy problems. Timing faults will arise when a service may block another service due to time-out.

The paper is organized as follows. Section 2 briefly describes the different services used to detect failures in any grid resources; Section 3 presents the fault tolerance strategies, mechanisms, techniques like checkpointing and replication. Section 4 deals about fault tolerant networks, Section 5 provide how fault management is done in distinct functional areas of grid infrastructure and Section 6 presents findings and conclusions.

2. FAILURE DETECTION SERVICES

As far as fault detection in any resource of grid is concerned, there are two main services: pull model and push model as described in [10]. In the ***pull model***, different grid components are responsible for sending periodic signals to a fault detector. In the absence of any such signal 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 the ***push model***, it is the responsibility of the fault detection component to send periodic signals to the different grid components. Further, the fault detection component is responsible for detecting different failures such as network failures, node crashes, and process failures and processing out the faults.

3. FAULT TOLERANCE STRATEGIES

Fault tolerance can be achieved by following strategies: ***Fault masking*** is the process of preventing faults in a system from introducing errors. It is basically used to hide the occurrence of faults and prevent faults from resulting in errors. Fault masking is used in systems that cannot allow even momentary erroneous results to be generated. Example: Error correcting memories and majority voting. ***Reconfiguration*** is the process of eliminating faulty component from a system and restoring the system to some operational state. Following are the approaches to reconfiguration: 1. Fault detection is the process of recognizing that a fault has occurred. Fault detection is often required before any recovery procedure can be initiated. 2. Fault location is the process of determining where a fault has occurred so that an appropriate recovery can be initiated. 3. Fault containment is the process of isolating a fault and preventing the effects of that fault from propagating throughout the system. 4. Fault recovery is the process of regaining operational status or remaining operational via reconfiguration even in the presence of faults.

3.1. Mechanisms

Fault tolerance mechanism in the grid environment can be divided into two main types: proactive and post-active. In ***proactive*** fault tolerance mechanism the failure consideration for the grid is made before the scheduling of a job, and dispatched with hopes that the job does not fail. On the other hand, the ***post-active*** mechanism handles the job failure after it has occurred [10].

3.2. Techniques

As in distributed systems generally, recovery methods in grid systems rely on exploitation of redundancy. Redundancy is the key for fault-tolerance. There can be no FT without redundancy! There are two types of redundancy: temporal redundancy and spatial redundancy. Temporal redundancy involves repeated attempts to restart failed resources or services. Spatial redundancy attempts to take advantage of multiple copies of computing resources. Fault tolerance in computer system is achieved through redundancy in hardware, software, information, and/or computations. Such redundancy can be implemented in static, dynamic, or hybrid configurations. Redundancy is the incorporation of extra components in the design of a system so that its function is not impaired in the event of a failure. That is addition of extra information/resources/time beyond what is needed for normal system operation.
There are four forms of redundancy: Hardware Redundancy, Software Redundancy, Information Redundancy, and Time Redundancy. Hardware faults are usually dealt with by using hardware, information, or time redundancy, whereas software faults are protected against by software redundancy [33].

Hardware Redundancy: It is provided by incorporating extra hardware into the design to either detect or override the effects of a failed component. Extra hardware is added to override the effects of a failed component. For example, instead of having a single processor, we can use two or three processors, each performing the same function. By having two processors, we can detect the failure of a single processor; by having three, we can use the majority output to override the wrong output of a single faulty processor. This is an example of static hardware redundancy, the main objective of which is the immediate masking of a failure. A different form of hardware redundancy is dynamic redundancy, where spare components are activated upon the failure of a currently active component. A combination of static and dynamic redundancy techniques is also possible, leading to hybrid hardware redundancy. Hardware redundancy can thus range from a simple duplication to complicated structures that switch in spare units when active ones become faulty.
Software Redundancy: It is the addition of extra software, beyond what is needed to perform a given function, to detect and possibly tolerate faults. It independently produces two or more versions of that software (preferably by disjoint teams of programmers) in hope that the different versions will not fail on the same input. These forms of design diversity will ensure that not all copies will fail on the same set of input data.

Information Redundancy: It is the addition of extra information beyond that required to implement a given function. Here, extra bits (called check bits) are added to the original data bits so that an error in the data bits can be detected or even corrected. Information redundancy often requires hardware redundancy to process the additional check bits.

Time Redundancy: It uses additional time to perform the functions of a system such that fault detection and often fault tolerance can be achieved. The systems exploit time redundancy through re-execution of the same program on the same hardware.

The use of redundancy can provide additional capabilities within a system. But, redundancy can have very important impact on a system's performance, size, weight, power consumption, and reliability.
Both temporal and spatial redundancy is used in grids. However, because grid systems inherently provide redundant computing resources, spatial redundancy has been a focus of fault-tolerance research. Three techniques that emphasize spatial redundancy exist: (1) checkpointing, or periodically saving the state of a process running on a computing resource so that, in the event of resource failure, it can resume on a different resource; (2) replication, or maintaining a sufficient number of replicas, or copies, of a process executing in parallel on different resources so that at least one replica succeeds; and (3) rescheduling, or finding different resources to rerun failed tasks. Note that (1) and (3) involve operations that repeat over time and are therefore temporally redundant as well. Fault tolerance techniques in grid systems are commonly achieved with checkpoint-recovery and job replication on alternative resources, in cases of system outage.

3.2.1. Checkpoint-Recovery

Checkpoint-recovery depends on the system's MTTR. It periodically saves the state of the application on stable storage, usually a hard disk. After a crash, the application is restarted from the last checkpoint rather than from the beginning. There are three checkpointing strategies. They are coordinated checkpointing, uncoordinated checkpointing, and communication-induced checkpointing. In ***coordinated checkpointing***, processes synchronize checkpoints to ensure their saved states are consistent with each other, so that the overall combined, saved state is also consistent. In contrast, in ***uncoordinated checkpointing***, processes schedule checkpoints independently at different times and do not account for messages. ***Communication-induced checkpointing*** attempts to coordinate only selected critical checkpoints [24].
Some of the ***issues*** are: At what level (user or kernel) should we checkpoint: what are the pros and cons of each level? How transparent to the user should the checkpointing process be? How many checkpoints should we have? At which points during the execution of a program should us checkpoint? How can we reduce checkpointing overhead? The problem of determining the optimal number of checkpoints is known as the checkpoint placement problem and its objective is to minimize the expected total execution time, ckeckpointing storage type and location, and checkpointing frequency.

3.2.2. Replication

Replication depends on the availability of alternative sits to run replicas. In grid resource replication, multiple grid resources simultaneously perform an identical computation and maintain identical state. The goal of replication is to ensure that at least one replica is always able to complete the computation in the event others fail. In some cases, one replica may be designated as a primary copy for purposes of external interaction, whereas others assume the role of backups.
Although replication method is widely used as a fault tolerance technique but number of backups is a main drawback. Number of backups increase drastically as coverage against number of fault increases. As the number of backup increase management of these backups is

very costly. Fusion based techniques overcome this problem. It is emerging as a popular technique to handle multiple faults. Basically it is an alternate idea for fault tolerance that requires fewer backup machines than replication based approaches. In fusion based fault tolerance a technique, back up machines is used which is cross product of original computing machines. These backup machines are called as fusion corresponding to the given set of machines. Overhead in fusion based techniques is very high during recovery from faults. Hence this technique is acceptable if probability of fault is low. Two aspects of research in resource replication are considered: (1) Algorithms for determining optimal (or near-optimal) placement of replicas in order to increase fault tolerance and (2) Methods for synchronizing replica states to ensure their consistency.
Some of the ***issues*** *are:* degree of replication, replica on demand, consistency, and number of replicas as a reaction on changing system properties (number of active resources, resource failure frequency and system load).

4. FAULT-TOLERANT NETWORKS

Fault tolerance in networks is achieved by having multiple paths connecting source to destination, and/or spare units that can be switched in to replace the failed units.

4.1. Network Measures

The following measures express the degradation of the dependability and performance of a computer network in the presence of faults. ***Reliability****:* Network reliability R (t) at time t, as the probability that all the nodes are operational and can communicate with each other over the entire time interval [0, t]. ***Path reliability /terminal reliability***, as the probability that an operational path has existed for this source-destination pair during the entire interval [0, t]. ***Bandwidth:*** the maximum rate at which messages can flow in a network. It usually degrades as nodes or links fail in a network. ***Connectability*****:** Expected number at time t of source-destination pairs which are still connected in the presence of a failure process.

4.2. Common Network Topologies and Their Resilience

1. ***Crossbar Network****:* In crossbar networks, the failure of any switchbox will disconnect certain input-output pairs. Redundancy can be introduced to make the crossbar fault tolerant. We add a row and a column of switchboxes and augment the input and output connections so that each input can be sent to either of two rows, and each output can be received on either of two columns. If any switchbox becomes faulty, the row and column to which it belongs are retired, and the spare row and column are pressed into service. *2.* ***Rectangular Mesh Network:*** A conventional two-dimensional rectangular mesh network is unable to tolerate any faults in any of its nodes without losing the mesh property. The fault tolerant mesh network includes spare nodes that can be switched in to take the place of any of their neighbors that have failed [33].

4.3. Fault-Tolerant Routing

The objective of a fault-tolerant routing strategy is to get a message from source to destination despite a subset of the network being faulty [33]. The basic idea is simple: if no shortest or most convenient path is available because of link or node failures, reroutes the message through other paths to its destination. The examples for fault tolerant routing are hypercube fault-tolerant routing and origin-based routing in the mesh. The basic idea of hypercube fault-tolerant routing is to list the dimensions along which the message must travel and then traverse them one by one.

As edges are traversed, they are crossed off the list. If, because of a link or a node failure, the desired link is not available, then another edge in the list, if any, is chosen for traversal. If no such edges are available (the message arrives at some node to find that all dimensions on its list are down), it backtracks to the previous node and tries again. In origin-based routing in the mesh the faulty regions are known in advance.

5. GRID FAULT MANAGEMENT IN DISTINCT FUNCTIONAL AREAS

Resource-level fault tolerance techniques involve the application of standard fault tolerance techniques in each and every one of the resources in the system. The heterogeneous and non-dedicated nature of the system increase complexity. Resource level failures like node crashes, link failure, events such as a machine turning unexpectedly off or the temporary loss of a network link are clearly regarded as faults.
Service-level fault tolerance, on the other hand, deals with system-wide policies aiming to increase dependability of the services provided. The Quality-of-Service (QoS) is the key factor. Dependability of a system is the ability to avoid service failures that are more frequent and more severe than is acceptable.

5.1. Scheduling

The basic grid model generally composed of a number of hosts, each composed of several computational resources, which may be homogeneous or heterogeneous. The four basic building blocks of grid model are user, resource broker, grid information service (GIS) and lastly resources. When user requires high speed execution, the job is submitted to the broker in grid. Broker splits the job into various tasks and distributes to several resources according to user's requirements and availability of resources. GIS keeps the status information of all resources which helps the broker for scheduling.
Job Scheduling: Job scheduling is the mapping of jobs to specific physical resources, trying to minimize some cost function specified by the user. This is a NP-complete problem and different heuristics may be used to reach an optimal or near optimal solution. Effective computation and job scheduling is rapidly becoming one of the main challenges in grid computing and is seen as being vital for its success.

Resource Scheduling: The grid resource scheduling process can be defined as the process of matching a query for resources, described in terms of required characteristics, to a set of resources that meet the expressed requirements. To make information available to users quickly and reliably, an effective and efficient resource scheduling mechanism is crucial. Generally grid resources are potentially very large in number with various individual resources that are not centrally controlled. These resources can enter as well as leave the grid systems at any time. For these reasons resource scheduling in large-scale grids can be very challenging.

5.1.1 Fault Tolerant Scheduling

This strategy maintains history of the fault occurrence of resource in Grid Information Service (GIS). Whenever a resource broker has job to schedule it uses the Resource Fault Occurrence History (RFOH) information from GIS and depending on this information use different intensity of checkpointing and replication while scheduling the job on resources which have different tendency towards fault. Further, it increases the percentage of jobs executed within specified

deadline. Using checkpoint techniques, this strategy can make grid scheduling more reliable and efficient [13].

This algorithm can also use RFOH information in Genetic Algorithm to schedule the job with optimized resource [14]. It can reduce the probability of selecting the resources that have more fault occurrence history. Therefore we have a reliable scheduling and kind of fault tolerance. In this paper we presented a new GA for reliable job scheduling in the Grid. This algorithm uses RFOH information which is maintained in FOHT. The using of this information causes the reduction of selecting chance of the resources which have more failure probability.
Grid Tuple-Space (GRIDTS) is a decentralized and fault-tolerant grid infrastructure [12], in which the resources pick the tasks to execute, instead of using a centralized scheduler. The communication is made using a tuple space, benefiting from it being decoupled in time and space. It combines different fault tolerance techniques-checkpointing, transaction, replication- to provide fault-tolerant scheduling.

5.2. Agent Oriented Fault Tolerant (Mobile Agent)

In [6], mobile agents are used for providing fault tolerance. The mechanism is named as MAG (Mobile Agents Technology for Grid Computing Environments). Here, fault tolerant components are developed as mobile agents to provide fault tolerance. Mobile agents form a multiagent society. Agent maintains information about hardware conditions, executing process memory consumption, available resources, network conditions and component mean time to failure. Based on this information and critical states, agent enables the grid system to tolerate faults.

5.3. Fault Tolerance in Advance Reservations

Advance Reservation is a process of requesting various resources for use at a later time. It provides an efficient method to guarantee the availability of resources to users and applications at the required time. It increases the probability of admission of the job and this planning of resources in advance allows users to gain concurrent access for their applications to be executed in parallel (i.e., in co-allocation environment).
In [7] , failure recovery not only has to handle already active jobs, but also those which are admitted but not yet started, inactive jobs. This means that the affected inactive jobs have to remap in advance to another matching resource. During execution any node failure occurs the active affected jobs are assigned to other resources using remapping technique in which the downtime is used, but it is increased the number of terminated jobs. The remapping of admitted but not yet active jobs is essential in order to reduce the number of terminated jobs and hence the overall resource utilization is increased.
Link Failure can be recovered by rerouting flows based on the actual load of the network. Once a failure on the network is notified, the affected flows are mapped onto alternative paths using downtime-independent strategy [9].
One job's overtime (exceeding its booked time) may lead to a serious of job's abnormal termination (Process Failure). It will affect resource utilization of later jobs. The system can redirect later inactive jobs to other nodes transparently when the active job is detected that may exceed its booked time [11]. Hence it decreases the abnormal termination ratio and improve system throughput.

5.4. Fault Tolerant Load Balancing

Load balancing is a technique to enhance resources, utilizing parallelism, exploiting throughput improvisation, and to cut response time through an appropriate distribution of the applications. This model [19] can checkpoint tasks in a consistent state and migrate those tasks immediately to other lightly balanced nodes using intra-cluster and intra-grid load balancing strategy. It uses Fault detector that detects the occurrence of resource failures and fault manager that guarantees that the tasks submitted are completely executed using available resources. In this load balancing model, if worker node N_{ij} fails, then at First level**:** Fault Detector (FD) of ith cluster detects the Nij failure and reports it to its Fault Manager (FM) and FM decides whether to start or not a local task migration operation from N_{ij} to rest of its worker nodes and at Second level: Tasks of N_{ij} are transferred to under loaded clusters. (Minimal communication cost for transferring tasks).

5.5. Fault Tolerance and Recovery of Workflows

Grid workflow as defined as the orchestration of a set of atomic tasks processed at distributed resources in a well-defined order to accomplish a large and sophisticated goal. Currently, Directed Acyclic Graph (DAG) has been extensively used in scientific computational workflow modeling. In a grid environment, workflow execution failure can occur for various reasons: the variation in the execution environment configuration, non-availability of required services or software components, overloaded resource conditions, system running out of memory, and faults in computational and network fabric components. Grid workflow management systems should be able to identify and handles failures and support reliable execution the presence of concurrency and failures. Hwang et al. [32] divided workflow failure handling techniques into two different levels, namely task-level and workflow-level may alter the sequence of execution in order to address the failures. Hwang and Kesselman proposed three different techniques. (i) The alternate task technique executes another implementation of a certain tasks if the previous one failed. (ii) The redundancy technique executes multiple alternative tasks simultaneously. (iii) The user-defined exception handling allows the users to specify a special treatment for a certain failure of a task in workflow. At Workflow level, failures can occur in data movement or infinite loops in dynamic workflows. Incorrect or not available input data could also produce faults.

5.6. Fault Tolerance in Resource Co-Allocation

One of the promises of Grid Computing is to enable the execution of applications across multiple sites. Some of these applications require coordinated access to resources managed by utonomous entities. This coordinated access is known as resource co-allocation. ***Co-allocation*** is the process of allocating resources from multiple providers in a coordinated manner. A computational grid is typically composed of several sites from geographically distributed organizations Parallel jobs should be scheduled to spread to more than one sites in order to run simultaneously on several sites without considering the resource limitation from one single site. The grid scheduling algorithm should be capable of coordinating these resources from different sites [30].

5.7. Fault Tolerance Mechanism for SLA-Aware Resource Management

A Service Level Agreement (SLA) is a powerful instrument for describing all expectations and obligations in the business relationship between service customer and service provider. It does not only cover questions and regarding the required resources, but also applies to issues like QoS, fault tolerance, or the measurement of SLA compliance. In addition, it encompasses the price for resource consumption, respectively the penalty fee for breaking the agreement. This system allows the Grid use to negotiate on SLAs, assuring the adherence with agreed SLAs by means of application-transparent checkpointing, and migration. Features like advance reservations, diffuse requests, and negotiation protocols are mandatory to realize SLA-aware RMS (Resource Management System). RMS provides uniform and transparent access to large pools of heterogeneous resources [19].
In Failure-Aware Grid resource Management system the Virtual Resource Manager (VRM) which supports QoS by means of SLAs [7]. In this work it addresses the problem of remapping reservation to other resources when the originally selected resource fails. It mainly focuses on those jobs that are scheduled to a failed resource and not yet started its execution, which is so called in-active jobs. Instead of dealing with fault tolerance of active jobs which usually requires checkpointing and migration. It computes a remapping interval during which it remaps those jobs that are assigned to a faulty resource and are inactive to some other resource in advance before it begins its execution. The checkpointing algorithms used in this paper are based on this concept and thus are cooperative (adaptive) heuristics.

5.8. Fault Tolerance in Optical Grid

In optical grid [21], an application is modeled as DAG. Each task can be viewed as a checkpoint. If a grid resource fault happens, the application can be continued from the interrupted task instead of execution from the beginning of the application. The grid resources including computational, storage and visualization devices are connected by optical network with optical switches and optical links. When a fault happens to a grid resource or an optical link, the execution of task on the resource or the data communication in the optical link is interrupted and stopped. The fault is independent from each other. There is a fault manager in the optical grid system, and all the faults can be detected as soon as they occur. In this paper the rescheduling policy to achieve fault-tolerance in optical grid.
During the execution, if a fault occurs, the fault manager is able to collect the fault information and the execution manager is able to collect the application's execution information. Based on the information of the fault and application, it can reschedule the unfinished tasks and data communication. If an execution of a task is interrupted by a resource fault, it will have to be executed again, but for an interrupted data communication it can be continued from the interrupted point instead of from the beginning of the data. For unfinished tasks, it assigns it to the resource with minimal finish time. If an interrupted task is rescheduled to another resource, all the data it needs will transferred to the resource. The rescheduling algorithm schedules the unfinished part of the application based on the fault information, so it will be able to find a better schedule for the application with less execution time.

6. FINDINGS AND CONCLUSIONS

The study has surveyed the progress in making grid systems more reliable. It has found that, to date, efforts to make grid systems more reliable have centered on developing methods for fault

tolerance. Efforts toward improving fault tolerance have focused on distinct functional areas of grid computing, including computational hardware and software resources, user applications and workflows, scheduling, advance reservation, co-allocation, SLA, load balancing, infrastructure and management services, and grid networks. Generally, work has progressed differently in these areas, and in each area, different problems remain to be solved.
Hence, new fault detection methods, client transparent fault tolerance architecture, on demand fault tolerant techniques, economic fault tolerant model, optimal failure prediction system, multiple faults tolerant model and self adaptive fault tolerance framework have been proposed to make the grid environment is more dependable and trustworthy.

REFERENCES

[1] Ian Foster and Carl Kesselman, S.T., (2001) "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", Intl J. Supercomputer Applications, 15: 200-222.

[2] I. Foster and C. kesselman, (2004) "The Grid: Bluprint for a New Computing Infrastructure, 2nd Edition, and pp: 748.

[3] R. Medeiros, W. Cirne, f. Braso;erop amd J. Sauve, (2003) " Faults in Grids: Why are they so bad and What can be done about it?, In Proceedings of the Fourth International Workshop on grid Computing.

[4] Weissman, J.B, (1999) "Fault Tolerant Computing on the Grid", pp: 351-352, HPDC.

[5] Mohammad Tanvir Huda, Heinz and W. Schmidt, Ian D. Peake , (2005) "An Agent Oriented proactive Fault-tolerant framework for Grid Computing", In Proceedings of the First IEEE International Conference on e-Science and Grid Computing, pp.304-311.

[6] Rafael Fernandes Lopes and Francisco Jos´e da Silva e Silva, (2006) "Fault tolerance in a Mobile Agent Based Computational grid", In Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid Workshops.

[7] Lars-Olof Burchard, Cesar, A. F. De Rose, Hans-Ulrich Heiss, Barry Linnert, and Jorg Schneider, (2005) "VRM: A Failure-Aware Grid Resource Management System", In Proceedings of the 17th IEEE International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'05) 1550-6533.

[8] Buyya R, (2002) " Economic-based distributed resource management and scheduling for grid computing Ph.D. Paper, Monash University, Melbourne, Australia.

[9] Lars-Olof Burchard, Barry Linnert and Joerg Schneider, (2005) "Rerouting Strategies for Networks with Advance Reservations", Proceedings of the First IEEE International Conference on e-Science and Grid Computing 0-7695-2448-6.

[10] Nazir, B. and Khan, T., (2006) "Fault Tolerant job Scheduling in Computational Grid. Emerging Technologies, In IEEE International Conference on Emerging Technologies, Volume, Issue, 13-14.

[11] Libing Wu1, Chanle Wu1 , Jianqun Cui , Huyin Zhang1 and Gang Ye, (2006) " An Overtime-tolerance Strategy for Advance reservation",in Proceedings of the Seventh international Conference on Parallel and distributed Computing, Applications and Technologies.

[12] Fabio Favarim, joni da silva Fraga, lau Cheuk Lung and Miguel Correia,(2007) "GRID TS: A New Approach for Fault-Tolerant Scheduling in Grid Computing", In International Symposium on Network Computing and applications, pages: 187-194.

[13] Leyli Mohammad Khanli, (2010) "Reliable Job Scheduler using RFOH in Grid Computing", Journal of Emerging Trends in Computing and Information Sciences, Vol. 1, No. 1.

[14] Leili Mohammad Khanli , Maryam Etminan and Far Amir Masoud Rahmani , (2010) "RFOH: A New Fault Tolerant Job Scheduler in Grid Computing", In Second International Conference on Computer Engineering and Applications .

[15] P. Stelling, I. Foster, C. Kesselman, C. Lee and G. Von Laszewski, (1998) "A fault detection service for wide area distributed computations", In Proceedings of 7thIEEE symposium on High Performance Distributed Computing.

[16] Babar Nazir, Kalim Qureshi and Paul Manuel, (2009) "Adaptive checkpointing strategy to tolerate faults in economy based grid", J Supercomput , 50: 1-18.

[17] Lars-Olof Burchard', Matthias Hovestadt', Odej Kao', Axel Keller' and Barry Linnert',(2004)"The virtual Resource Manager: An Architecture for SLA-aware Resource Management", in IEEE International Symposium on Cluster Computing and the Grid.

[18] Matthias hovestadt, (2005) "fault tolerance Mechnisms for SLA-aware Resource management", in International conference on Parrallel and Distributed Systems.

[19] B. Yagoubi and M. Medebber, (2007) "A Load Balancing Model for Grid Environment", IEEE.

[20] Elvin Sindrilaru, Alexandru Costan , Valentin Cristea , (2010) "Fault Tolerance and Recovery in Grid Workflow Management Systems", In International Conference on Complex, Intelligent and Software Intensive Systems.

[21] Y. Wang, Y. H. jin, W. Guo, W. Q. Sun, W. S. hu and M. Y. Wu, (2007) " Joint Scheduling for Optical Grid Applications", Journal of Optical Networking, Vol. 6, pp. 304-318.

[22] J. H. Abawajy, (2004) " Fault-Tolerant Scheduling Policy for Grid Computing Systems", In Proceedings of the 18th International Parallel and Distributed Processing Symposium.

[23] Satish Tadepalli, Calvin Ribbens and Srinidhi Varadarajan, "GEMS: A Job Management System for Fault Tolerant Grid Computing", High-Performance Computing Symposium, ISBN: 1-56555-278-4.

[24] Eric roman, (2002) "A Survey of checkpoint/restart Implementations", Lawrence Berkley National Laboratory, CA.

[25] Paul Townend, jie Xu, (2003) "Fault tolerance within a grid Environment", As part of the e-Demand project at the University of Durham, DHI 3LE, United Kingdom.

[26] A. Nguyen-Tuong, (2000) "Integrating fault-tolerance techniques in Grid applications", Ph.D, Dissertation, university of Virginia.

[27] J. Daly, (2003) "A model for predicting the optimum checkpoint interval for restart dumps", in Proceedings of the ICCS2003, Lecture Notes in Computer Science 2660, Vol. 4, pp:3-12.

[28] H. Lee, K. Chung, S. Chin, J. Lee, D. Lee, S. Park and H.Yu, (2005) " A resource management and fault tolerance services in grid computing", J Parallel Distributed Computing, vol. 65(11), pp.1305-1317.

[29] Antonios Litke, Dimitrios Halkos, Konstantinos Tserpes, Dimosthenis Kyriazis and Theodora Varvarigou, (2009) " Fault tolerant and prioritized scheduling in OGSA-based mobile Grids", Concurrency and computation practice and experience, 21:533-556.

[30] Foster I , Kesselman C, Lee C, Lindell B, Nahrstedt K and Roy A,(1999) "A distributed resource management architecture that supports advance reservation and co-a allocation", Proceedings of the International Workshop on QoS, London, U.K.,27-36.

[31] Eduardo Huedo, Ruben S. Montero, Ignacio M. Llorente, (2006) ”Evaluating the reliability of computational grids from the end user’s point of view”, Journal of Systems Architecture, 727-736.

[32] S. Hwang and C. Kesselman, (2003) “Grid Workflow: A Flexible Filure handling Framework for the Grid”, In Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, Seattle, Washington, USA.

[33] Israel Koren and C.Mani Krishna, (2007) “Fault Tolerant Systems”, Elsevier Inc., ISBN: 978-81-312-1530-2.

AUTHORS

P. Latchoumy received B.Tech degree in Computer Science and Engineering from Pondicherry Engineering College, Pondicherry University, Puducherry, India in 1997. She received M.E in the area of Computer Science and Engineering from Crescent Engineering College, Anna University, Chennai, Tamil Nadu, India in 2005. Currently, she is an Assistant Professor in the Department of Information Technology and pursuing her Ph.D in the area of Fault Tolerance in Grid Computing from BSA University, Chennai, Tamilnadu, India. Her research interests include Fault Tolerance, Grid Computing, Scheduling and Distributed Computing.

Dr. P. Sheik Abdul Khader has 26 years of Teaching Profession and 15 years of Research Experience. He obtained his Ph.D degree from School of Computer Science and Engineering, Anna University, Chennai, Tamil Nadu, India. Currently, he is a Professor and Head in the Department of Computer Applications, BSA University, Chennai, Tamilnadu, India. He has presented papers in 34 Conferences organized by IEEE and Springer and also published papers in 12 Refereed International Journals. He is currently guiding 15 Ph.D Research Scholars in different thrust areas. His research interests include Routing Algorithms, Fuzzy, MANET, Distributed Computing, Genetic Algorithms, Grid Computing and Cloud Computing.