ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 2, Issue 1, January 2013

On Fault Tolerance of Resources in Grid Environment

Minakshi Memoria, Mukesh Yadav

Abstract—**Grid computing, most simply stated, is distributed computing taken to the next evolutionary level. The goal is to create the illusion of a simple yet large and powerful self managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources. However, in the grid computing environment there are certain aspects which reduce efficiency of the system, job scheduling of the resources and fault tolerance are the key aspect to improve the efficiency and exploit the capabilities of emergent computational systems. Because of dynamic and distributed nature of grid, the traditional methodologies of scheduling are inefficient for the effective utilization of the resource available. The fault tolerance strategy proposed will improve the performance of the overall computational grid environment. In this paper we propose an efficient job scheduling, replication and check pointing to improve the efficiency of the grid environment. The simulation results illustrate that the proposed strategy effectively schedules the grid jobs and reduce the execution time.**

KEYWORDS : Distributed System, Middleware

I. INTRODUCTION

Grid computing is a term referring to the combination of computer resources from multiple administrative domains to reach a common goal. The **grid** can be thought of as a distributed system with non-interactive workloads that involve a large number of files. What distinguishes grid computing from conventional high performance computing systems such as cluster computing is that grids tend to be more loosely coupled, heterogeneous, and geographically dispersed. Although a grid can be dedicated to a specialized application, it is more common that a single grid will be used for a variety of different purposes. Grids are often constructed with the aid of general-purpose grid software libraries known as middleware.

The standardization of communications between heterogeneous systems created the Internet explosion. The emerging standardization for sharing resources, along with the availability of higher bandwidth, are driving a possibly equally large evolutionary step in grid computing.

A. What grid computing can do

Minakshi Memoria*, CSE Department., Dronachary College of Engg. Gurgaon, India, 09953679907*

Mukesh Yadav*, CSE Departmente, Gurgaon Institute of Technology & Management, Gurgaon, India, 09313051807,*

When you deploy a grid, it will be to meet a set of customer requirements. To better match grid computing capabilities to those requirements, it is useful to keep in mind the reasons for using grid computing.

a. Exploiting underutilized resources

The easiest use of grid computing is to run an existing application on a different machine. The machine on which the application is normally run might be unusually busy due to an unusual peak in activity. The job in question could be run on an idle machine elsewhere on the grid. There are at least two prerequisites for this scenario. First, the application must be executable remotely and without undue overhead. Second, the remote machine must meet any special hardware, software, or resource requirements imposed by the application. For example, a batch job that spends a significant amount of time processing a set of input data to produce an output set is perhaps the most ideal and simple use for a grid. If the quantities of input and output are large, more thought and planning might be required to efficiently use the grid for such a job. It would usually not make sense to use a word processor remotely on a grid because there would probably be greater delays and more potential points of failure. In most organizations, there are large amounts of underutilized computing resources. Most desktop machines are busy less than 5 percent of the time. In some organizations, even the server machines can often be relatively idle. Grid computing provides a framework for exploiting these underutilized resources and thus has the possibility of substantially increasing the efficiency of resource usage. The processing resources are not the only ones that may be underutilized. Often, machines may have enormous unused disk drive capacity. Grid computing, more specifically, a “data grid”, can be used to aggregate this unused storage into a much larger virtual data store, possibly configured to achieve improved performance and reliability over that of any single machine. If a batch job needs to read a large amount of data, this data could be automatically replicated at various strategic points in the grid. Thus, if the job must be executed on a remote machine in the grid, the data is already there and does not need to be moved to that remote point. This offers clear performance benefits. Also, such copies of data can be used as backups when the primary copies are damaged or unavailable.

Another function of the grid is to better balance resource utilization. An organization may have occasional unexpected peaks of activity that demand more resources. If the applications are grid-enabled, they can be moved to underutilized machines during such peaks. In fact, some grid implementations can migrate partially completed jobs. In general, a grid can provide a consistent way to balance the loads on a wider federation of resources. This applies to CPU, storage, and many other kinds of resources that may be available on a grid. Management can use a grid to better view the usage patterns in the larger organization, permitting better planning when upgrading systems, increasing capacity, or retiring computing resources no longer needed.

b. Parallel CPU capacity

The potential for massive parallel CPU capacity is one of the most attractive features of a grid. In addition to pure scientific needs, such computing power is driving a new evolution in industries such as the bio-medical field, financial modeling, oil exploration, motion picture animation, and many others. The common attribute among such uses is that the applications have been written to use algorithms that can be partitioned into independently running parts. A CPU intensive grid application can be thought of as many smaller "sub jobs," each executing on a different machine in the grid. To the extent that these sub jobs do not need to communicate with each other, the more "scalable" the application becomes. A perfectly scalable application will, for example, finish 10 times faster if it uses 10 times the number of processors. Barriers often exist to perfect scalability. The first barrier depends on the algorithms used for splitting the application among many CPUs. If the algorithm can only be split into a limited number of independently running parts, then that forms a scalability barrier. The second barrier appears if the parts are not completely independent; this can cause contention, which can limit scalability. For example, if all of the sub jobs need to read and write from one common file or database, the access limits of that file or database will become the limiting factor in the application's scalability. Other sources of inter-job contention in a parallel grid application include message communications latencies among the jobs, network communication capacities, synchronization protocols, input-output bandwidth to devices and storage devices, and latencies interfering with real-time requirements.

c. Applications

There are many factors to consider in grid-enabling an application. One must understand that not all applications can be transformed to run in parallel on a grid and achieve scalability. Furthermore, there are no practical tools for transforming arbitrary applications to exploit the parallel capabilities of a grid. There are some practical tools that skilled application designers can use to write a parallel grid application. However, automatic transformation of applications is a science in its infancy. This can be a difficult job and often requires top mathematics and programming talents, if it is even possible in a given situation. New computation intensive applications written today are being designed for parallel execution and these will be easily grid-enabled, if they do not already follow emerging grid protocols and standards.

d. Virtual resources and virtual organizations for collaboration

Another important grid computing contribution is to enable and simplify collaboration among a wider audience. In the past, distributed computing promised this collaboration and achieved it to some extent. Grid computing takes these capabilities to an even wider audience, while offering important standards that enable very heterogeneous systems to work together to form the image of a large virtual computing system offering a variety of virtual resources. The users of the grid can be organized dynamically into a number of virtual organizations, each with different policy requirements. These virtual organizations can share their resources collectively as a larger grid. Sharing starts with data in the form of files or databases. A "data grid" can expand data capabilities in several ways. First, files or databases can seamlessly span many systems and thus have larger capacities than on any single system. Such spanning can improve data transfer rates through the use of striping techniques. Data can be duplicated throughout the grid to serve as a backup and can be hosted on or near the machines most likely to need the data, in conjunction with advanced scheduling techniques. Sharing is not limited to files, but also includes many other resources, such as equipment, software, services, licenses, and others. These resources are "virtualized" to give them a more uniform interoperability among heterogeneous grid participants. The participants and users of the grid can be members of several real and virtual organizations. The grid can help in enforcing security rules among them and implement policies, which can resolve priorities for both resources and users.

e. Access to additional resources

In addition to CPU and storage resources, a grid can provide access to increased quantities of other resources and to special equipment, software, licenses, and other services. The additional resources can be provided in additional numbers and/or capacity. For example, if a user needs to increase his total bandwidth to the Internet to implement a data mining search engine, the work can be split among grid machines that have independent connections to the Internet. In this way, the total searching capability is multiplied, since each machine has a separate connection to the Internet. If the machines had shared the connection to the Internet, there would not have been an effective increase in bandwidth. Some machines may have expensive licensed software installed that the user requires. His jobs can be sent to such machines more fully exploiting the software licenses. Some machines on the grid may have special devices. Most of us have used remote printers, perhaps with advanced color capabilities or faster speeds. Similarly, a grid can be used to make use of other special equipment. For example, a machine may have a high speed, self feeding, DVD writer that could be used to publish a quantity of data faster. Some machines on the grid may be connected to scanning electron microscopes that can be operated remotely. In this case, scheduling and reservation are important. A specimen could be sent in advance to the facility hosting the microscope. Then the user can remotely operate the machine, changing perspective views until the desired image is captured. The grid can enable more elaborate access, potentially to remote medical diagnostic and robotic surgery tools with two-way interaction from a distance. The variations are limited only by one's imagination. Today, we have remote device drivers for printers. Eventually, we will see standards for grid-enabled device drivers to many unusual devices and resources. All of these will make the grid look like a large virtual machine with a collection of virtual resources beyond what would be available on just one conventional machine.

f. Resource balancing

A grid federates a large number of resources contributed by individual machines into a greater total virtual resource. For applications that are grid-enabled, the grid can offer a resource balancing effect by scheduling grid jobs on machines with low utilization. This feature can prove invaluable for handling occasional peak loads of activity in parts of an larger organization. This can happen in two ways:

- ❖ An unexpected peak can be routed to relatively idle machines in the grid.
- ❖ If the grid is already fully utilized, the lowest priority work being performed on

the grid can be temporarily suspended or even cancelled and performed again later to make room for the higher priority work. Without a grid infrastructure, such balancing decisions are difficult to prioritize and execute. Occasionally, a project may suddenly rise in importance with a specific deadline. A grid cannot perform a miracle and achieve a deadline when it is already too close. However, if the size of the job is known, if it is a kind of job that can be sufficiently split into sub jobs, and if enough resources are available after preempting lower priority work, a grid can bring a very large amount of processing power to solve the problem. In such situations, a grid can, with some planning, succeed in meeting surprise deadline. Other more subtle benefits can occur using a grid for load balancing. When jobs communicate with each other, the Internet, or with storage resources, an advanced scheduler could schedule them to minimize communications traffic or minimize the distance of the communications. This can potentially reduce communication and other forms of contention in the grid.

Finally, a grid provides excellent infrastructure for brokering resources. Individual resources can be profiled to determine their availability and their capacity, and this can be factored into scheduling on the grid. Different organizations participating in the grid can build up grid credits and use them at times when they need additional resources. This can form the basis for grid accounting and the ability to more fairly distribute work on the grid.

g. Reliability

High-end conventional computing systems use expensive hardware to increase reliability. They are built using chips with redundant circuits that vote on results, and contain much logic to achieve graceful recovery from an assortment of hardware failures. The machines also use duplicate processors with hot pluggability so that when they fail, one can be replaced without turning the other off. Power supplies and cooling systems are duplicated. The systems are operated on special power sources that can start generators if utility power is interrupted. All of this builds a reliable system, but at a great cost, due to the duplication of high-reliability components.

In the future, we will see a complementary approach to reliability that relies on software and hardware. A grid is just the beginning of such technology. The systems in a grid can be relatively inexpensive and geographically dispersed. Thus, if there is a power or other kind of failure at one location, the other parts of the grid are not likely to be affected. Grid management software can automatically resubmit jobs to other machines on the grid when a failure is detected. In critical, real-time situations, multiple copies of the important jobs can be run on different machines throughout the grid. Their results can be checked for any kind of inconsistency, such as computer failures, data corruption, or tampering. Such grid systems will utilize "autonomic computing." This is a type of software that automatically heals problems in the grid, perhaps even before an operator or manager is aware of them. In principle, most of the reliability attributes achieved using hardware in today's high availability systems can be achieved using software in a grid setting in the future.

h. Management

The goal to virtualize the resources on the grid and more uniformly handle heterogeneous systems will create new opportunities to better manage a larger, more disperse IT infrastructure. It will be easier to visualize capacity and utilization, making it easier for IT departments to control expenditures for computing resources over a larger organization. The grid offers management of priorities among different projects. In the past, each project may have been responsible for its own IT resource hardware and the expenses associated with it. Often this hardware might be underutilized while another project finds itself in trouble, needing more resources due to unexpected events. With the larger view a grid can offer, it becomes easier to control and manage such situations. Administrators can change any number of policies that affect how the different organizations might share or compete for resources. Aggregating utilization data over a larger set of projects can enhance an organization’s ability to project future upgrade needs. When maintenance is required, grid work can be rerouted to other machines without crippling the projects involved.

Autonomic computing can come into play here too. Various tools may be able to identify important trends throughout the grid, informing management of those that require attention.

II. NEED FOR FAULT TOLERANCE IN GRID COMPUTING

Computational grid consists of large sets of diverse, geographically distributed resources that are grouped into virtual computers for executing specific applications. As the number of grid system components increases, the probability of failures in the grid computing environment becomes higher than that in a traditional parallel computing. Compute intensive grid applications often require much longer execution time in order to solve a single problem. Thus, the huge computing potential of grids, usually, remains unexploited due to their susceptibility to failures like, process failures, machine crashes, and network failures etc. This may lead to job failures, violating timing deadlines and service level agreements, denials of service, degraded user expected quality of service. Thus fault management is a very important and challenging for grid application developers. It has been observed that *interaction*, *timing*, and *omission* faults are more prevalent in grid. Fault tolerance is the ability of a system to perform its function correctly even in the presence of faults. The fault tolerance makes the system more dependable. A complementary but separate approach to increase dependability is fault prevention. This consists of techniques, such as inspection, whose intent is to eliminate the circumstances by which faults arise. A failure occurs when an actual running system deviates from this specified behavior. The cause of a failure is called an error. An error represents an invalid system state that does not comply the system specification. The error itself is the result of a defect in the system or fault. In other words, a fault is the root cause of a failure. However, a fault may not necessarily result in an error; nevertheless, the same fault may result in multiple errors. Similarly, a single error may lead to multiple failures. The level of fault tolerance is reflected by quantifying the system dependability. Dependability means that our system can be trusted to deliver the service(s) for which it has been designed. It can be measured by two of the metrics like reliability and availability. *Reliability* characterizes the ability of a system to perform, on demand, its service correctly. Availability means that the system is up to perform this service when it is asked to do so. Technically, reliability is defined as the probability that a system will perform correctly up to a given point in time. Closely related to reliability are the mean time to failure(MTTF) and mean time between failures (MTBF).The first is the average time the system operates until a failure occurs, whereas the second is the average time between two consecutive failures.

MTBF = MTTF + MTTR, where MTTR is the mean time to repair *Availability* is defined as the probability that a system is operational at desired time. For a given system, this characteristic is strongly dependent on the time it takes to restore it to service after some failure.

Availability = (MTTF) / (MTTR + MTTF)

III. GRID FAULT MANAGEMENT

Various approaches are used for tolerating faults in grid, so grid fault management can be classified as

A. Pro-active vs. Post-active management

In literature, the work on grid fault tolerance can be divided into pro-active and 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 handles the job failures after it has occurred. However, in the dynamic systems only post-active mechanism is relevant.

B. Push Model vs. Pull Model

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*, 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* the failure detector sends live-ness requests (*“Are you alive?”* messages) periodically to grid components.

C. Grid Middleware Based

Like any middleware, a grid middleware is also responsible to hide, from the application developer, the technical details related to different syntax and access methods and to provide a consistent and homogeneous access to resources managed locally. It presented a failure detection service (FDS) and a flexible failure handling framework (Grid-WFS) as a fault tolerance mechanism on the grid. The FDS enables the detection of both task crashes and user defined exceptions.

D. Agent based

Here autonomous, light-weight, intelligent agents monitor with individual faults. Agents maintain log of various information related to hardware conditions, memory utilization, resource constraints, network status and component failure. Based on this information and critical states, agent can enhance the reliability and efficiency of grid services

E. Application level

The performance of system level fault tolerance mechanisms is restricted due to the large process state, low I/O bandwidth, and the high frequency of failures. Either an application would spend more time in taking checkpoints or it does not get sufficient time to save its core to disk before the next failure occurs. Therefore, low overhead application level fault tolerance schemes are more suitable alternative in long-lived computational tasks. However, most application level fault tolerance schemes proposed in literature are non-adaptive due to fast changing computational scenario. However, in order to achieve high reliability and survivability, the fault tolerance schemes in such applications need to be adaptable to dynamic system environments. The CPPC (Controller/Precompiler for Portable Checkpointing) framework implements checkpointing module into the application code. The CPPC-G service is responsible for

(a) the submission and monitoring of the application execution

(b) the management of checkpoint files generated by CPPC-enabled applications, and

(c) the detection

and automatic restart of failed executions.

F. Fault tolerance at job site

The failure at job site has cascading effect on the grid performance. They reduce the resource availability, which tend to make the clusters unusable that result in the loss of user submitted jobs. It eventually slows down the overall speed of computation. Hence, in order to ensure the high system availability, the job site failure handling is inevitable. A cluster head is responsible to coordinate the computation related activities and to provide other necessary services, such as job scheduling. If job sites are made up of clusters, then the failure of the cluster head causes the services to be unavailable till the cluster head recovers or replaced by some back up head. The transition from failed head to back up head is uninterrupted. Hence, it provides an excellent solution for stateless services. While some approaches are self healing, provides job level fault resilience in order to guarantee the high availability of the site.

IV. Existing Fault Tolerance Techniques

Many fault tolerance techniques such as retry, replication, message logging and check pointing are available in traditional distributed paradigms.

A. Retry

Retry is the simplest failure recovery technique in which we hope that whatever is the cause of failures, the effect will not be encountered in subsequent retries .

B. Replication

In replication based technique we have replicas of a task running on different machines and as long as not all replicated tasks crash (i.e. host crash etc), chances are that the task execution would succeed .

C. Message Logging

In message logging all participating nodes log incoming messages to stable storage and when a failure is encountered than these message logs are used to compute a consistent global state. Algorithms that take this approach can be further classified into those that use pessimistic and those that use optimistic message logging

D. Check-pointing

Check-pointing is relatively more popular fault tolerant approach used in distributed systems, where the state of the application is stored periodically on reliable and stable storage, normally a hard disk etc. In case of problem during execution, i.e. after crash etc, the application is restarted from the last checkpoint rather than from the beginning .

V. Related work

A. Fault Tolerant Based Job Scheduling with CRS

As the proposed strategy considers fault tolerance in a computational grid environment, the aim is to optimize user-centric metrics in the presence of faults. These metrics include TTR, execution time and the number of jobs completed within deadline even in the presence of faults. For simplicity, it is assumed that a fault occurs when a grid resource is unable to complete its job in the given deadline. When such a fault is detected the fault occurrence information about the grid resource is updated. This fault occurrence information is used while making a job check pointing before allocating job to the grid resource. The fault index of all available resources of the grid are maintained and updated. The fault index of the grid resource will suggest its vulnerability to faults (i.e., higher the fault index is higher the failure rate). The fault index of a grid resource is incremented every time the resource does not complete the assigned job within the deadline and also on resource failure. The fault index of a resource is decremented whenever the resource completes the assigned job within deadline. The availability of created checkpoints is increased by providing CRS. A key feature of this service is the ability to replicate and monitor checkpoints in the grid environment. Hence the current application state can be taken at any time and will not depend on the checkpoint server.

B. Components of Proposed Strategy

The interaction between different components of a computational grid in the proposed scheduling and fault tolerant strategy is shown in figure. In the proposed approach, in addition to scheduler which reduces response time of the job, the fault tolerant mechanism reduces the wastage of time in redo of partially completed job from the scratch. Instead, it restarts the failed job from the last saved checkpoint to complete. A grid resource is a member of a grid and it offers computing services to grid users. Grid users register themselves to the Grid Information Server (GIS) of a grid by specifying QoS requirements such as the deadline to complete the execution, the number of processors, type of operating system and so on. The components used in the proposed architecture are described below:

a. Scheduler

Scheduler is an important entity of a grid. Scheduler receives jobs from grid users. It selects feasible resources for those jobs according to acquired information from GIS. Then it generates job-to-resource mappings. The entities of scheduler are Schedule Manager, Match Maker, Response Time Estimator, Resource Selector and Job Dispatcher. When the schedule manager receives a grid job from a user, it gets the details of available grid resources from GIS. It then passes the available resource list to the entities in MTTR scheduling strategy. The Match Maker entity performs match making of the resources and job requirements. Response Time Estimator entity estimates the response time for the job on each matched resource based on Transfer time, Queue Wait time and Service time of the job. Resource selector selects the resource with minimum response time. A job dispatcher dispatches the jobs one by one to the checkpoint manager.

b. GIS

GIS contains information about all available grid resources. It maintains details of the resource such as processor speed, memory available, load and so on. All grid resources that join and leave the grid are monitored by GIS. Whenever a scheduler has jobs to execute, it consults GIS to get information about available grid resources.

c. Checkpoint Manager

It receives the scheduled job from the scheduler and sets checkpoint based on the failure rate of the resource on which it is scheduled. Then it submits the job to the resource. Checkpoint manager receives job completion message or job failure message from the grid resource and responds to that accordingly. During execution, if job failure occurs, the job is rescheduled from the last checkpoint instead of running from the scratch. Checkpoint manager implements checkpoint setter algorithm to set job checkpoints.

d. Checkpoint Server

On each checkpoint set by the checkpoint manager, job status is reported to the checkpoint server. Checkpoint server save the job status and return it on demand i.e., during job/resource failure. For a particular job, the checkpoint server discards the result of the previous checkpoint when a new value of checkpoint result is received.

e. Fault Index Manager

Fault Index Manager maintains the fault index value of each resource which indicates the failure rate of the resource. The fault index of a grid resource is incremented every time the resource does not complete the assigned job within the deadline and also on resource failure. The fault index of a resource is decremented whenever the resource completes the assigned job within the deadline. Fault index manager updates the fault index of a grid resource using fault index update algorithm.

a. Checkpoint Replication Server

When new checkpoint is created, Checkpoint Replication Server initiates CRS which will replicate the created checkpoints into remote resources by applying RRSA. Once replicated, details are stored in Checkpoint Server. To obtain information about all checkpoint files, Replication Server queries the Checkpoint Server. During the entire application runtime , CRS monitors the

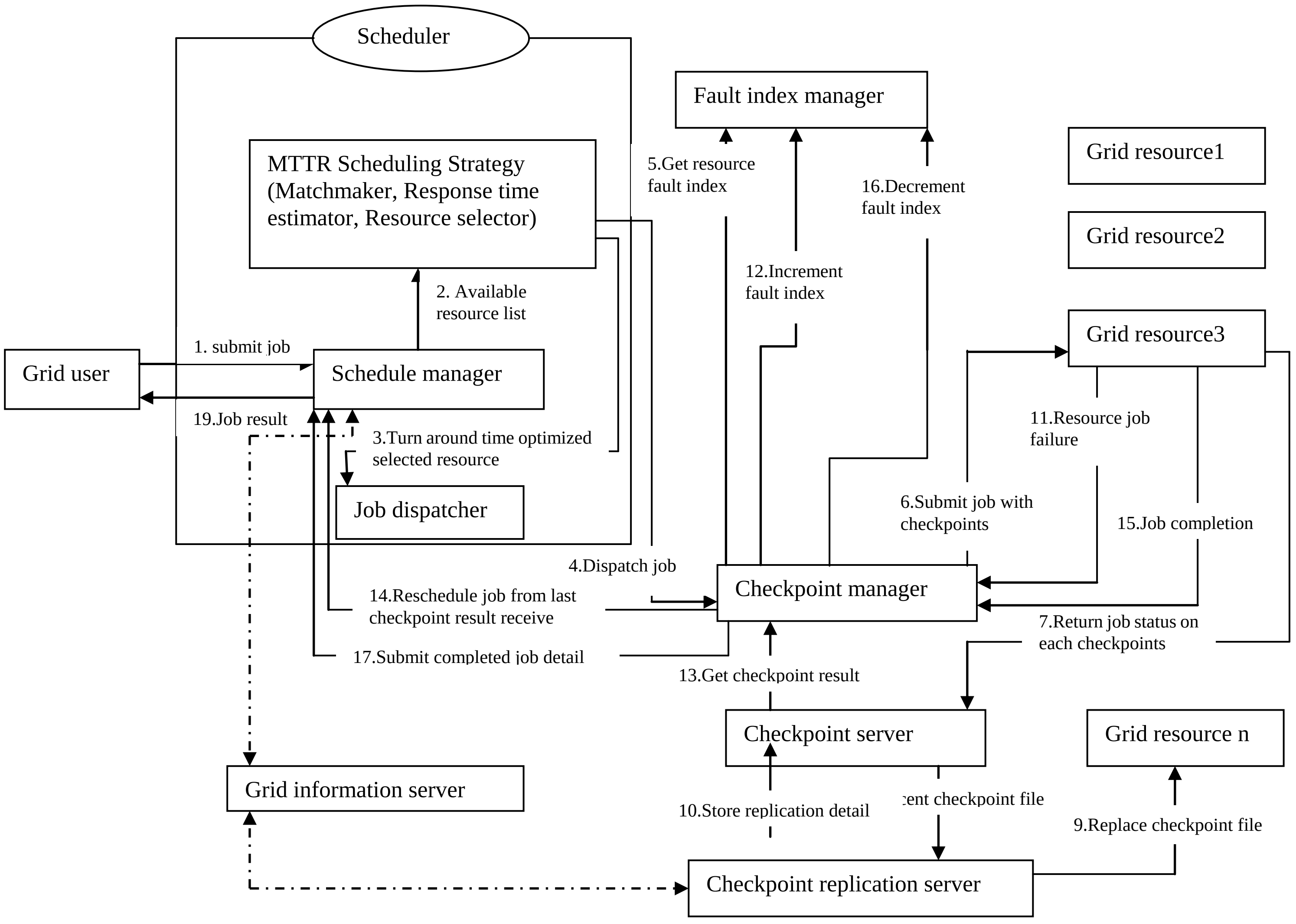
Scheduler
Fault index manager
Grid resource1
Grid resource2
MTTR Scheduling Strategy (Matchmaker, Response time estimator, Resource selector)
5.Get resource fault index
16.Decrement fault index
12.Increment fault index
2. Available resource list
Grid resource3
1. submit job
Grid user
Schedule manager
19.Job result
3.Turn around time optimized selected resource
11.Resource job failure
Job dispatcher
6.Submit job with checkpoints
15.Job completion
4.Dispatch job
Checkpoint manager
14.Reschedule job from last checkpoint result receive
7.Return job status on each checkpoints
17.Submit completed job detail
13.Get checkpoint result
Checkpoint server
Grid resource n
Grid information server
10.Store replication detail
ent checkpoint file
9.Replace checkpoint file
Checkpoint replication server

Figure 1: computational grid environment

Checkpoint Server to detect newer checkpoint versions. Information about available resources, hardware, memory and bandwidth details are obtained from GIS. NWS and Ganglia tools are used to determine these details. The required details are periodically propagated by these tools to the GIS. Depending on transfer sizes, available storage of the resources and current bandwidths, CRS selects a suitable resource using RRSA to replicate the checkpoint file.

VI. PERSPECTIVE & OBJECTIVES

The main aim is to resolve all the problems occurred due to fault occurrence. There are so many algorithms made for these like **fault index update algorithm** (To do indexing for checkpoint manager), **RRSA algorithm(**for replication), **Checkpoint setter algorithm(** to set the interval at which checkpoint is to be set).
But I have to solve these problems either by algorithm or any tool.

So objectives of my research are:

To apply Checkpointing + Replication + Scheduling simultaneously?

VII. PROPOSED WORK

The aims and objectives can be reached out by imparting extra constraints in scheduling in dependent task and independent task and fault prevention by replication and check pointing.

The proposed work will be completed in 4 phases:

PHASE-1	Check pointing
PHASE-2	Replicating
PHASE-3	Re-Scheduling for dependent task
PHASE-4	Re- Scheduling for independent task

VIII. EXPECTED OUTCOME OF THE PROJECT

A HIGHLY SECURE OR RELIABLE VIRTUAL GRID IN WHICH YOU CAN SHARE ANY RESOURCE FROM ANY CLUSTER EVEN WITH EXISTENCE OF FAULT IN SYSTEM.

REFERENCES

[1] www.redbooks.ibm.com/redbooks/pdfs/sg246895.pdf

[2] Dev S. Gokul, Sujith R., Amirutha V.S., Divyalakshmi S. “International Conference on Computing and Control Engineering (ICCCE 2012)” ISBN 978-1-4675-2248-9 © 2012 Published by Coimbatore Institute of Information Technology

[3] Garg Ritu and SinghAwadhesh Kumar “International Journal of Computer Science & Engineering Survey (IJCSES)” Vol.2, No.1, Feb 2011

[4] D´ıaz, Xo´an C Pardo Daniel., J. Mart´ın Mar´ıa, Gonz´alez Patricia “International Conference onComputer Communication and Management” CSIT vol.5 (2011) © (2011) IACSIT Press, Singapore

[5] Nandagopal Malarvizhi “International Journal of Engineering Science and Technology “Vol. 2(9), 2010, 4361-4372

[6] Haider Sajjad, Naveed, Riaz Ansari, Akbar Muhammad, Raza Perwez Mohammad, MoyeezUllah Ghori Khawaja ” International Conference on Computer Communication and Management” Proc .of CSIT vol.5 (2011) © (2011) IACSIT Press, Singapore

[7] Thenmozhi, S., A. Tamilarasi and P.T. Vanathi “Journal of Computer Science” 8 (6): 978-982, 2012 ISSN 1549-3636

[8] Das Arindam and Sarkar Ajanta De “ International Journal of Grid Computing & Applications” (IJGCA) Vol.3, No.3, September 2012

Minakshi Memoria received her M. Tech degree from MD University, Rohtak and B.Tech in computer science from MD University, Rohtak. She is pursuing Phd in the field of grid computing. She is currently working as an Associate professor in Dronacharya college of engineering, Gurgaon,

Mukesh Yadav is currently working as a Professor of Computer Sc. & Engineering at Gurgaon Institute of Technology & Management, Gurgaon (HR) India. He received his B.E in Computer Science & Engineering from Maharishsi Dayanand University, Rohtak, Haryana, India in 2000, M.Tech in Computer Sc. & Engg. From Guru Jambheswar University, Hisar, Haryana in 2002 and Ph.D (Computer Sc.) from Bundhelkhand University, Jhansi (UP) in 2010. He has more than 10 years of rich experience of teaching, research & development in various institutes of repute in Haryana. He has authored or co-authored over 25 technical papers in national, international journals and conferences. He served as session chair for many national and international conferences and reviewer of many national /international journals. Mukesh’s current research interests include mobile computing, operating systems and software reliability, wireless networking systems and grid computing. He is Professional/Life member of technical society-ACM/IEEE and Computer Society of India.