Fault Tolerance Improvement techniques in Grid Computing

Jyoti chaudhary
Assistant Professor, Deptt. Of Computer Engineering, The Technological Institute of Textile & Sciences, Bhiwani (Haryana), India
Jyotisепtember20@rediffmail.com

Abstract: Grid computing is a distributed computing model that provides access to the geographically distributed heterogeneous resources. These computational grid systems are highly unreliable in nature because of which they need fault tolerance to be an integral part of the system to increase reliability. Commonly utilized techniques for providing fault tolerance are discussed. This paper provides state of the art of two basic fault tolerance techniques and their comparative analysis.

KEYWORDS: Grid Computing, Fault Tolerance, Check Point.

1. INTRODUCTION

A grid computing infrastructure provides access to high level computational capabilities in a reliable, consistent, pervasive and inexpensive manner. Geographically distributed resources cooperate with each other to solve big problems. It enables users to use its resources for large-scale computing applications in science, engineering and commerce [1].

Since grid environments are extremely heterogeneous and dynamic, with components joining and leaving the system all the time, more faults are likely to occur in grid environments [2]. Also, the likelihood of error occurrence is exacerbated by the fact that many grid applications will perform long tasks that may require several days of computation. This will lead to a number of new conceptual and technical challenges to fault-tolerance researchers. So major challenge in dynamic grid with thousands of nodes connected to each other is fault tolerance. The more resources and components involved the more complicated and error-prone becomes the system.

2. FAULT TOLERANCE

Fault tolerance is the ability of a system to perform its function correctly even in the presence of faults. Fault tolerance makes the system more dependable. Fault tolerance is also defined as preserving the delivery of expected services despite the presence of fault-caused errors within the system itself. Errors are detected and corrected but permanent faults are located and removed while the system continues to deliver acceptable services [3].

Different types of faults, classified based upon several factors, are as follows:

1. Hardware Faults: These faults occur due to faulty hardware components such as CPU, memory and other peripherals. These faults are very difficult to recover, they may either be trouble-shooted or replaced.

2. Application Faults: These faults may occur due to exceptions or errors on the node which may be due to Denial of Service attacks, viruses etc.

3. Middleware Faults: Middleware is the interface between the user and the resources on Grid and performs all the major tasks like resource management, resource allocation, job scheduling and fault tolerance etc. Any exceptions in the working of the middleware may lead to faults in Grid.

4. Configuration or Network faults: The network faults may lead to a node failure due to loss of connection, packet loss or the corruption of the data during the transmission from one node to the other and it may lead to the faulty or inconsistent results so it can affect the quality of service requirement of the user.

According to the survey conducted among Grid users that what kind and extent of faults they are encountering in the usage of Grid [5]. The results are shown in figure 1. These kinds of faults make Grid less reliable for executing the jobs with high QOS requirements. Dealing with [1] these complex failure scenarios is challenging. Detecting that something is wrong is not so difficult (in general, symptoms are quickly identified), but difficulties arise to identify the root cause of the problem. In case of any failure there are chances that the transparency provided by the middleware will be compromised and the user will have to dig deep into the middleware, operating system of the network this is not an easy task for a human being. This is a major limitation of Grid. i.e. why fault tolerance is the major area of research in the field of Grid computing now days.

In computational grids, fault tolerance is important as the dependability of grid resources may not be guaranteed. It is needed to enable the grid to continue its work when one or more resources fail. In this sense, a fault-tolerant service must be included to detect errors and recover from them and thus avoiding the failure of the grid.

The main aim of this paper is to review two basic fault-tolerance techniques and make a comparative study of them.
3. REVIEW OF EXISTING FAULT TOLERANCE TECHNIQUES:

This section provides a brief appraise of the two basic fault
tolerance techniques used in the grid environment. Checkpoint-
recovery and job replication are the two basic fault tolerances

3.1 Checkpoint – Recovery Technique:

This is one of the most popular techniques used to provide fault
tolerance in unreliable systems. This technique provides the
recovery and resuming mechanisms which can resume the
execution of the job from the failed stage. So that the time and
resources used to execute the job before the point of failure will not
be useless. The execution of the job can be resumed using the
checkpoints which are computed and stored while the job is being
executed. The checkpointing approach is very effective while
executing the jobs with a long execution time e.g. scientific
experiments and simulations which may last for days, weeks or
even months. So to recover the processed part of such jobs is very
beneficial from the resource utilization and execution time point of
view. The widely used checkpointing techniques in Grid are:

3.1.1 Application Level Checkpointing

In this technique the checkpointing functionality is inserted in the
application code so that application should be able to store and
manage check points itself. CPPC (Com Piler for Portable
Checkpointing) [7] is an application level checkpointing tool
focused on inserting checkpointing code into long running message
passing applications. It consists of a runtime library containing
checkpoint support routines, together with a compiler that
automates the use of the library. CPPC provides all the features
which are key issues for fault tolerance support on large scale
heterogeneous systems such as Grid. It uses portable code and
protocols, and generates portable checkpoint files while avoiding
traditional solutions which have some scalability overhead.

3.1.2 System Level Checkpointing

This checkpointing is done at the level of the system executing the
job. The system level checkpoints save the entire state of the system
as well as the job. It requires more data for saving the state than
application checkpoints; this means system can checkpoint any
application at an arbitrary point in its execution and allows
programmers to be more productive.

3.1.3 Full Checkpoint or Incremental Checkpointing

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 [9, 10, and 11]. 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.

3.1.4 Uncoordinated or Coordinated Checkpointing

In uncoordinated checkpointing each process takes its checkpoint
independently of the other processes though it may lead to domino
effect (processes may be forced to rollback up to the execution
beginning). Since there is a chance for losing the whole computation, these protocols are not popular in practice.

Coordinated checkpoint protocols produce consistent checkpoints;
however, the recovery process is simple to implement. Communication
Induced Checkpointing (CIC) tries to take advantage of
uncoordinated and coordinated checkpoint techniques. Based on the
uncoordinated approach, it piggy backs causality dependencies in
all messages and detects risk of inconsistent state. When such a risk
is detected, some processes are forced to checkpoint. A detailed
survey of checkpointing protocols may be found in [12].

3.1.5 Kernel or Low Level Checkpointing

Here checkpointing procedures are included in the kernel,
checkpointing is transparent to the user and generally no changes
are required to the programs to make them checkpointable. When
the system restart after failure, the kernel is responsible for
managing the recovery operation. To date, there have been a few
low-level checkpointing packages [13]. Each checkpointing package
offers a different functionality and interface. Because of
technical issues the checkpointing packages impose some
limitations on applications that are to be checkpointed. So the
integration of low-level checkpointing packages with the Grids is a
difficult task. AltixC/R [14]is kernel-level checkpointing package.
The required kernel-level code is provided in a form of a
dynamically loaded kernel module so it is easy to use and install.
The package is able to checkpoint multi-process programs.

3.1.6 User level Checkpointing

In this approach, a user level library is provided to do the
checkpointing. To checkpoint, application programs are linked to
this library. This approach generally requires no changes in the
application code; however explicit linking is required with user
level library, which is also responsible for recovery from failure.

The efficiency and performance of any checkpointing FT technique
will be based on the following factors:

- **Frequency** of checkpoints effect the performance by either
  increasing or decreasing the checkpointing overhead because
  resources and time is consumed to calculate the checkpoints and to
  store on the storage media.

- **Availability** of the checkpoints effect the resuming time of a
  failed process because it takes time to locate and transfer the
  checkpoint data from the storage (which can be a server or any
  node) to the node which is going to execute the job after
  rescheduling the job. Another issue with the storage of the
  checkpoints is that whether to keep all the checkpoint data on a
  single node or more than one as suggested by [8]. Considering the
  possibility of the failure of the single node on which checkpoint
  data is residing the checkpoints can be replicated on more than
  one node to increase the availability and to avoid any kind faults or
  errors which may be caused while transmitting the data from one
  node to another.

- **Checkpoint Size** if a central site is being used to store the
  checkpoints then the size of a single checkpoint can be a problem
  because the storage space requirement will also increase with that
  [15]. To deal with this issue the checkpoint data can be compressed
  before storing or transferring over the network and the older
  checkpoints should be removed from the system when a latest
  checkpoint has been generated.

The frequency and number of replicas of checkpoints can be
decided, according to QoS requirements or according to the
reliability of the resources which are going to be used for the
execution of the job [8], at the scheduling time of the job. But the
frequency of checkpoints is difficult to decide at the scheduling
level because different jobs may have different stages of
checkpoints so we cannot decide a fix frequency and time of
checkpoints for all the jobs. Every FT technique which is designed
or existing at present have a particular area of application but there
exists no technique which have answer to all type of faults and problems of Grid.
This is one of the few existing fault tolerance techniques which are being used at present to deal with the faults:

RFOH: A New Fault Tolerant Job Scheduler in Grid Computing [16]

This technique is based on managing the resources to keep a track of their failures. The resources with high rate of failure are avoided while selecting the resources for executing the job. They have also used the check pointing to recover a job after failure.

3.2 Replication Technique:

It is a technique based on an inherent assumption that any single resource is much susceptible to failure as compared to simultaneous failure of multiple resources. Unlike check pointing, the replication avoids task re-computation by executing several copies of the same task on more than one compute stations. The job replication and determination of the optimal number of replicas involves many technical considerations. The task replication in grids has been studied in [17]. A number of approaches have been used to implement replication in grid computing environment.

3.2.1 Static vs. Dynamic replication

The static replication [18] means that, when some replica fails, it is not replaced by a new one. The number of replicas of the original task is decided before execution. While in case of dynamic replication, new replicas can be generated during run time. Gallop [19] used an initial set of replicas based on user preferences which we call static replicas (e.g. the user wants x replicas). The user may also indicate that they do not want the number of active replicas falling below y replicas. This may require that new replicas are started if enough sites fail and the number of active replicas falls below y. We call this capability dynamic replication.

3.2.2 Active vs. passive replication

In the former, the state of replicas is kept closely synchronized; replicas service the same requests in parallel and undergo the same state transitions. This algorithm is referred to as the active replication [20]. In the latter, a primary replica services requests on behalf of clients. Other replicas are kept as standby and can take over in the case of a primary failure [21]. This is sometimes referred to as passive replication. Further, two approaches of passive replication, using the concept of overloading, have been used in the literature; Primary Backup vs. Backup overloading.

3.2.3 Primary Backup vs. Backup overloading Replication

Overloading techniques are used to deal with timing faults and to improve the schedulability. In PB-overloading primary of a task is scheduled onto the same or overlapping time slot with the backup of another task on a processor. While in BB-overloading backups of multiple tasks are scheduled onto the same or overlapping time slot on a processor [22]. Since PB-overloading can assign an earlier start time than that of the BB-overloading, thus increasing the schedulability In [23], R. Al-Omari et al. concluded that the PB-overloading is able to achieve better performance than BB-overloading, and BB-overloading algorithm is better than no-overloading.[24]. In short, hybrid overloading is a new technique which combines the advantages of both PB and BB overloading. All three overloading strategies are compared through a stochastic analysis, as well as by simulating them under diverse system conditions.

Fault Tolerance within a Grid Environment [7]

This technique is based on the replication. It assumes that Grid is a collection of huge number of resources so to achieve the required QoS more than required number of resources can be used. They have proposed the method of voting to select the correct output out of at least more than half outputs and reduce the execution time of the job.

4. Comparitive Appraises:

The above mentioned techniques have been compared on the basis of resource utilization; check pointing overhead caused by the particular technique, processing cost which will be based on the number and type of resources being used, replication and the total execution time. The results are shown in table 1. The existence or value of one parameter will affect the other dependent parameter values e.g. replication may decrease the execution time but it will increase the execution cost and decrease the resource utilization due to the redundancy.

<table>
<thead>
<tr>
<th>Techniques Parameters</th>
<th>RFOH [16]</th>
<th>FT in Grid [17]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource Utilization</td>
<td>HIGH</td>
<td>LOW</td>
</tr>
<tr>
<td>Replication</td>
<td>NA</td>
<td>HIGH</td>
</tr>
<tr>
<td>Checkpointing Overhead</td>
<td>LOW</td>
<td>NA</td>
</tr>
<tr>
<td>Execution Time</td>
<td>AVG</td>
<td>HIGH</td>
</tr>
<tr>
<td>Processing Cost</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 1: Comparison of Fault Tolerance Techniques

Together check pointing and job replication techniques [8] can be used to achieve a more fault tolerant environment than any of these individually. Replication will run duplicate jobs and check pointing will be used to store the intermediate results. It will be an overhead to run duplicate jobs and to manage their checkpoints but this cost can be bear in case of high QoS requirements of the user. Yulan and Yanhang [8] have implemented the same technique and found out that it provides better results.

5. Conclusion:

The existing techniques of fault-tolerance in grid environment with their importance, combinations and variations have been discussed. Replication of job/data is necessary in order to increase the resource availability to the computing nodes. An application of check pointing is important to formulate organized policies to recover from a system under errors or faults. It effectively prevents system from being led to a failure state. Security aware scheduling of grid jobs migrated through agents improves grid performance significantly. A newer concept, service level behavior or global behavior on understanding the grid services representing one unified service system has changed the whole understanding perspective of fault scenario in grid computing. However, accepting the importance of all the aforesaid areas, to put forward a future direction of work, this research would next focus on check pointing technique for better performance of applications running in grid.

6. References


[27] Ran Zheng, Hai Jin,”An integrated management and scheduling scheme for computational grids”


