AN EFFICIENT FAULT-TOLERANCE TECHNIQUE USING CHECK-POINTING AND REPLICATION IN GRIDS USING DATA LOGS
S Dilli Babu, Ch.Ramesh Babu, Ch.D.V Subba Rao
M.Tech Scholar ,Dept of CSE,SVUCE, SVU Tirupathi, AP, INDIA.
Research Scholar, JNTUK, AP, INDIA.
Professor, Dept of CSE,SVUCE, SVU Tirupathi, AP, INDIA.
sdilli@rocketmail.com, chramesh522@gmail.com, Subbarao_chdv@hotmail.com

ABSTRACT
Grid computing systems are increasingly growing importance in the present world with advances in the network technology. Grids are composed of many geographically disturbed resources, each having its own administration domain. Grid computing involves decentralized heterogeneous, geographically distributed resources that can work on a job together. Since the resource availability is dynamic in nature, the grid infrastructure is prone to failure of job lose or delay. So in order to adapt to the failure, fault tolerant mechanism must be implemented. Commonly used techniques for fault tolerance are checkpointing and load replication or job migration mechanism. To have an efficient fault tolerance mechanism this paper introduces a novel checkpointing algorithm based on real-time work load logs that reduces overhead caused due to checkpointing. The proposed system uses Job replication to ensure completion of work, optimal replication and dynamic Load balancing is used to avoid overload in any resources and to achieve maximum resource utilization and maximize output of the grid.

Keywords— Grid Computing, Fault Tolerance, Work load logs

I. INTRODUCTION
The Grid is rapidly emerging as the means of coordinated resource sharing and problem solving in multi-institutional virtual-organizations while providing dependable, consistent, pervasive access to global resources. The emergence of computational Grids and the potential for seamless aggregation and interactions between distributed services and resources, has led to the start of new era of computing. Tremendously large number and the heterogeneous nature of grid computing resource make the resource management a significantly challenging job. Resource management scenarios often include resource discovery, resource monitoring, resource inventories, resource provisioning, fault isolation, variety of autonomic capabilities and service level management activities. Out of this fault tolerance has become the main topic of research as till date there is no single system that can be called as the complete system that will handle all the faults in grids.

Checkpointing is one of the fault-tolerant techniques to restore faults and to restart job fast. The algorithms for checkpointing on distributed systems have been under study for years. These algorithms can be classified into three classes: coordinated, uncoordinated and communication-induced algorithms [1]. But we cannot place checkpointing blindly as it can result either performance degradation or expensive recovery cost. So this paper comes up with an optimal checkpointing placement algorithm so that it can avoid any overhead produced from it. This system implements job replication and dynamic load replication mechanism to improve the efficiency of fault tolerance in the grid environment. The grid computing environment is described in Section II. Section III presents detail about checkpointing. Section IV describes the job optimal replication mechanism, Section V describes experimental results, various tools and techniques used to compare the results are discussed.

II. GRID COMPUTING
The rapid and impressive growth of the internet, there has been a rising interest in web based parallel computing.
In fact, many projects have been incepted to exploit the Web as an infrastructure for running coarse-grained distributed parallel applications. In this context, the web has the capability to become a suitable and potentially infinite scalable meta computer for parallel and collaborative work as well as a key technology to create a pervasive and ubiquitous grid infrastructure [5]. This section will discuss the grid architecture and the details of Virtual Organization.

2.1 Grid architecture

The Grid architecture shown in Figure 1 consists of Computational resources (CRs) that contain number of general services. User submits the job through the resource broker and resource broker finds whether the requested resource is available by collecting information from the Grid Scheduler. Grid Scheduler (GS) is responsible for job resource matchmaking. Information Service (IS) collects the resource status information required by the Grid Scheduler. Checkpointing Server (CS) collects the information taken during checkpointing. Whenever the failure occurs the job will restart from the checkpointing information stored in Checkpoint Server. Load Balancing Server (LS) is used for balancing the load in the grid site. Here each site contain Information Service, Checkpoint Server and Load Balancing server. So it reduces the latency taken for recovering from failure. Since the checkpoint information will be collected at each site’s Checkpoint Server the resources can recollect the information very quickly. Since distance between the resource and CS is less the latency also will decrease. The checkpoint interval will be decided by the CS by considering the information about the mean failure of the resources from the Information Service. So inorder to minimize the overhead produced from doing the checkpoint we need to find an optimal checkpointing interval.

Load balancing Server considers the information available in the IS and monitors when the load balancing need to be applied. Whenever any of the CRs becomes overloaded it will invoke load balancing. At that time the load balancing server collects the information from the IS and decide which of the CRs is under loaded and migrate those jobs waiting for execution in overloaded CRs to the under loaded CRs.

III. CHECKPOINTING

Checkpointing is the commonly used technique for fault tolerance and improving system availability. It stores the status of current application state, and then it can be used for restarting the execution in case of failure avoiding the job to start from scratch.

![Figure 2: Life Time of Grid Application](image)

The figure 2 shows different phases of application that undergoes during its execution. Once the application has been started, it does some of its work and then pauses to write the checkpoint to stable storage. After completion of checkpoint the application restarts its work.

When the application initiates checkpointing at time t, the running job will pause for a time tckpt, the checkpoint latency or interval is topt, if job fails in between the time t and t+topt then it will force the job to restart from the previous checkpoint. This application takes trst time to restart after the Sometimes the interrupt may occur in between the work thus it has to redo all its work done after last successful checkpoint. So it collects the information from the last successful checkpoint and restarts its application. Then redo all its lost work since last checkpoint. Making progress on the work continues until it is time again to write another checkpoint or a new interruption occurs. During checkpointing, the images of processes and the state of communication channels are collected and stored in a special dedicated and reliable machine named Checkpoint Server. In order to reduce
overhead caused during checkpointing different approaches have been developed. One of the techniques is known as incremental checkpointing [13]. It stores only modified data during checkpointing. At the initial checkpointing operation all pages of the program address space are saved. After each checkpointing operation all the modified pages will be updated in checkpoint server.

Another technique is called min-max checkpoint placement [4] that determines the uncertain circumstances in terms of the system failure. Here checkpoint interval is considered without the complete knowledge on system failure distribution. Even if optimal checkpoint interval is found before, the parameters used for calculating the interval may change over time. So another approach were found known as cooperative checkpointing [7] is a set of semantic policies that allow the application, compiler and system to jointly decide when the checkpoints should be performed. According to the adaptive checkpointing scheme[1], it uses the information about the remaining job execution time, time left before the deadline and the expected remaining needed, to decide whether checkpointing is to be done or not.

It also gives information on when the next checkpoint request need to be given. There are two methods in adaptive checkpointing scheme Last Failure Dependent Checkpointing (LastFailureCP) and Mean Failure Dependent Checkpointing (MeanFailureCP). In LastFailureCP algorithm it omits unnecessary checkpoint placement with reference to the total execution time and failure frequency of the resource. This algorithm keeps a time stamp LFr that gives the time when the last failure had occurred. Initially checkpointing request will be given at time interval I and then request will be executed by Grid Scheduler by comparing whether __ , where tc is the current time and __ is the execution time of job j on resource . If the condition is true then checkpointing is allowed otherwise checkpointing is omitted.

In case of MeanFailureCP [1], the checkpointing interval changes according to the remaining execution time and mean failure interval. Initially the algorithm gives checkpoint request within fixed and preferably short time period ti. Each time when checkpoint request is given the following conditions are checked.

If \( MF < MFr \) and, where \( F \) is the remaining job execution time and \( MFr \) is average failure interval of the resource \( r \) where job \( j \) assigned. If the condition is true then the frequency of the checkpointing will be reduced by increasing the checkpoint interval \( Ir \) new = \( Ir \) old + I else the checkpointing frequency will be increased to \( Ir \) new = \( Ir \) old - I

A. OPTIMAL CHECKPOINTING ALGORITHM

The optimal checkpointing interval by its nature can be defined as a function of failure rate that is time varying optimal checkpointing in order to meet the criteria of certain optimality. We try to maximize the utilization by selecting an optimal checkpoint rate. The utilization \( U \) can be defined as the fraction of time that is spent doing useful computation out of the total runtime between two job restarts. The remaining time is regarded as total overhead introduced by the checkpointing and the cost of restarting. Consider the mean time to failure as \( \mu \) then it is possible to predict failure for certain interval of time. The failure probability can be calculated by using Exponential distribution.

1) Probability density function:

Probability density function (pdf) of an exponential distribution is

\[
\begin{align*}
    f(x;j) &= \begin{cases} 
        ke^{-kx}, & x \geq 0 \\
        0, & x < 0
    \end{cases} 
\end{align*}
\]

Consider \( \mu \) as the checkpointing frequency. The utilization can be calculated as \( U(t) = R(t) / t \) Where \( R(t) \) is the useful computation time when job fails at time \( t \). Consider two cases of failure, one is during checkpointing and another between two checkpointing.

Algorithm:

1. For particular interval \( i \) find the useful computation.

\[
\begin{align*}
    R(i;t) = \begin{cases} 
        \frac{t-1}{u} + \frac{1}{u} - \frac{1}{u} - v & \text{for } \frac{1}{u} \leq t < \frac{i + 1}{u} + v \\
        \frac{i}{u} + v & \text{for } \frac{i}{u} + v \leq t \leq \frac{i + 1}{u} 
    \end{cases}
\end{align*}
\]

Where \( v \) is the overhead time needed for restarting the job after failure

2. Take the total useful computation for each case independently as \( R1 \) and \( R2 \)

3. Find the utilization as \( U = \frac{kJ}{t} \) …….(3)

4. Then by calculating u
Where $\mu$ is the optimal checkpointing frequency where $W(w)$ is the inverse function of $f(w) = \text{J}_\infty$ called Lambert W function.

**IV JOB REPLICATION**

Job replication (JR) creates multiple copies of each job, and sends the original and copies to different processors to execute and waits until first replication is finished. In an R-JR run, each job creates R-1 replication and all the replications are executed. So each job will contain exact R samples. R copies of all P jobs are distributed to P processors; therefore each processor receives each iteration R different jobs. As soon as a processor finishes execution of one of the copies, it notifies other processors by sending the message that they can kill the job and start the next job in the sequences.

Figure 3: Job Replication

Figure 3 shows the situation for a 2-JR run on four processors. Here each job and its copy are distributed to R = 2 processors, and during one iteration, each processor receives R = 2 jobs. Processor one finished as first job A and sends a “finalize” message to processor two. Sending the message over the Internet takes some time, and therefore, it takes a while before the other processors start the next job. Each job-type time, which is the duration of a specific job type (the original and its copies), is equal to the minimum of all its job times plus a possible send time. An individual processor time of one iteration is equal to the sum of the job-type times, which were sent to that processor and the send times of the “kill” messages. Finally, the Iteration Time (IT) of all the processors corresponds to the sum of the ST and the maximum of all processor times. Consequently, the same groups of processors execute in each iteration the same job. Thus, divide the P processors in K” execution groups, which all consist of R processors.

**V. EXPERIMENTAL RESULTS**

<table>
<thead>
<tr>
<th>Grids</th>
<th>Jobs</th>
<th>Checkpoints(CHKPT)</th>
<th>Memory(in mbs)</th>
<th>Time(in Sec)</th>
<th>Algorithm CHKPT</th>
<th>Mem</th>
<th>TTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>100</td>
<td>20</td>
<td>.8</td>
<td>.8</td>
<td>15</td>
<td>.9</td>
<td>.4</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>20</td>
<td>1</td>
<td>.8</td>
<td>15</td>
<td>.8</td>
<td>.7</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>20</td>
<td>1.5</td>
<td>1</td>
<td>15</td>
<td>.9</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3: These are the various results in Checkpointing

**VI CONCLUSION**

This paper proposes an optimal checkpoint placement algorithm, job replication and dynamic load balancing mechanism. The checkpoint algorithm removes the overhead caused by other checkpointing mechanism like last failure checkpoint dependent checkpointing and mean failure checkpointing. The checkpointing rate is computed using exponential distribution that increases the utilization time. The dynamic load balancing keeps the system stable.

**REFERENCES**


