An Optimized Fault Recovery Technique in Computational Grid using Checkpointing Approach

Ch. Ramesh Babu\textsuperscript{a},* \hspace{1em} Dr. Ch. D. V. Subba Rao\textsuperscript{b}

\textsuperscript{a}Research Scholar, JNTU Kakinada, AP, India
\textsuperscript{b}Professor, Dept of CSE, SV University Tirupati, AP, India

\begin{abstract}
A Grid is a strewn computational milieu habitually composed of heterogeneous resource which is increasingly growing importance in the present world with advances in the network technology. Large applications executing on Grid or Cluster or parallel processing etc., architectures consisting of computational nodes often create problems with consistency and reliability because of heterogeneity of the resources. The source of the problems is delay in executing jobs and node failures etc. Job checkpointing is one of the most commonly utilized techniques for providing fault tolerance in computational grids. The efficiency of checkpointing depends on the choice of the checkpoint interval. Inappropriate checkpointing interval can delay job execution. In this work, a fault-tolerant job scheduling system based on checkpointing technique is presented and evaluated. When scheduling a job, the system uses both average failure time and last failure rate of grid resources combined with resource response time to generate scheduling decisions. Experiments have shown that the proposed system can considerably improve throughput, turnaround time and failure tendency. The fault-tolerant job scheduling scheme discussed in this paper is based on an optimal number of checkpointing algorithm that deals with problems associated with both job delay and faults based on real-time workload-logs that reduce overhead caused due to checkpointing.

\textit{Keywords:} Checkpoint, Computational grid, Datalogs, Fault-tolerance, Replication.
\end{abstract}

\section{Introduction}

A Grid is a type of distributed computational milieu system which ropes resource sharing, harmonized use of geographically pervasive nodes and multi-owner wherewithal independently from their physical type and location in dynamic virtual organizations that share the same goal of solving large-scale applications. Large applications executing on Grid or cluster architectures consisting of computational nodes create problems with reliability. The source of the problems is node failures and the need for dynamic configuration over extensive runtime [1].

Grid and cluster architectures have gained popularity for computationally intensive parallel applications. However, the complexity of the infrastructure, consisting of computational nodes, mass storage, and interconnection networks, poses great challenges with respect to overall system reliability. Simple tools of reliability analysis show that as the complexity of the system increases, its reliability, and thus, Mean Time to Failure (MTF), decreases. If one models the system as a series reliability block diagram, the reliability of the entire system is computed as the product of the reliabilities of all system components. For applications executing on large clusters or a Grid, the long execution times may exceed the MTF of the infrastructure and, thus, render the execution infeasible. The high failure probabilities are due to the fact that, in the absence of fault-tolerance mechanisms, the failure of a single node will cause the entire execution to fail. Note that this simple example does not even consider network failures, which are typically more likely than computer failure. Fault tolerance is, thus, a necessity to avoid failure in large applications, such as found in scientific computing, executing on a Grid, or large cluster [5].

\footnotesize{
\textsuperscript{*}Ch.Ramesh Babu, Tel: +91-9985677767
E-mail: crb.challagundla@gmail.com
}
2. Related work

2.1. Computational Grids

In computational grids varying resource availability becomes commonplace, often resulting in loss and delay of executing jobs. To ensure good grid performance, fault tolerance should be taken into account. Commonly utilized techniques for providing fault tolerance in distributed systems are periodic job checkpointing and replication. While very robust, both techniques can delay job execution if inappropriate checkpointing intervals and replica numbers are chosen. This paper introduces several heuristics that dynamically adapt the abovementioned parameters based on information on grid status to provide high job throughput in the presence of failure while reducing the system overhead. Simulations (using Perl) are run employing workload and system parameters derived from logs that were collected from several large-scale parallel production systems.

In this context of our problem to have an efficient fault tolerance mechanism this paper comes up with an optimal checkpointing algorithm that reduces overhead caused due to checkpointing. The proposed system uses Job replication to ensure completion of work and Dynamic Load balancing is used to avoid overload in any resources and to achieve maximum resource utilization and maximize throughput.

The main objective of computational grids is to execute the user applications or jobs. Therefore, users submit their jobs to the Grid Scheduler (GS) along with their QoS (Quality of service) requirements. These requirements may include the deadline in which users want jobs to be executed, the type of the resources required to execute the job and the type of the platform needed.

To take in hand this predicament, the average failure time \( AFT \) of the resource is taken into consideration when making scheduling decisions using neural network approach. In this session we have designed an optimal checkpointing algorithm based on the Probability density function. It is observed that both existing algorithms still not able to provide an optimal number of checkpoints intervals [2].

3. Modelling Issues

3.1. Proposed Checkpointing Design

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.

In Figure 1 it shows the different phases of checkpointing 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.

According to the adaptive checkpointing scheme, 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 LF 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 \( t_c - LF \leq E_j \), where \( t_c \) is the current time and \( E_j \) is the execution time of job j on resource r 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 \( t_i \). Each time when checkpoint request is given the following conditions are checked [13].

If \( RE_j \leq MF_i \) and \( I_i < \alpha * E_j \) where \( RE_j \) is the remaining job execution time and \( MF_i \) 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 [12].

\[
I_i^{new} = I_i^{old} + I
\]

else the checkpointing frequency will be increased to \( I_i^{new} = I_i^{old} - I \).
3.2. Checkpointing Based Optimized Technique (CBOT)

The objective of this work is to optimize the performance of the grid in the presence of faults in application. The performance metrics used include throughput, turnaround time and failure tendency. A fault occurs when a grid resource cannot complete its job within the given deadline. The main strategy of the proposed CBOT depends on using the job checkpointing mechanism to minimize the effect of grid faults and to reduce the fault recovery time.

3.2.1. Components of the CBOT

The interaction between different components of the CBOT is shown in Figure 2. The CBOT restarts the execution of the failed job from the last saved checkpoint. Thus, it reduces the response time of the job by reducing the time wasted in re-executing partially completed job from the scratch.

![Figure 2 - Architecture of the CBOT](image)

A grid contains multiple grid resources that provide computing services to users. The main component of the CBOT is the grid scheduler (GS). It receives jobs with their information from users. Job information includes job number, job type, and job size. Also, the user submits QoS requirements of each job such as the deadline to complete its execution, the number of required resources and the type of these resources.

The main function of GS is to find and sort the most suitable resources that can execute the job and satisfy user QoS requirements. In order to perform this function, the GS connects to the grid information server (GIS) to get information of available grid resources that can execute the job. Figure 2 shows the operation of the GS. The GS uses response time, resource failure rate and resource failure time to construct the list of suitable resources that can execute the job.

3.2.2. Grid Scheduling pseudo code

Step 1: Initiate application j jobs to grid.
Step 2: for each job j submitted by the user to the GS;
Step 3: GS gets a list of suitable resources for j from GIS;
Step 4: GS sorts this list according to the response time*FR*AFT of each resource;
Step 5: GS dispatches the job and the list to the CPH;

GIS contains information of all available resources in the grid required by the GS. The information includes resource speed, current load, resource failure rate and total failure time of each resource. The latter is the time the resource spent in the failure case before it is coming again to join grid and work properly [3].

Checkpoint server (CPS) receives and stores partially executed results of a job from the resource in intervals specified by the checkpoint handler (CPH). These intermediate results are called checkpoint status. For each job, there is only one record of checkpoint status. When CPS receives a new checkpoint status it overwrites the old one. If CPS receives a job completion message from the resource it removes the record of such job [9].

CPH is an important component of CBOT. The main functions of CPH are determining the number of checkpoints and determining the checkpoints interval for each job. CPH receives a job with its assigned list of resources from GS. It connects to GIS to get information about the failure history of grid resources assigned to the job. Based on failure rate of the resource, the CPH determines the number of checkpoints and the checkpoint intervals for each job. Then, it submits the job to the first grid resource in the resources list. Figure 2 shows the algorithm used by CPH to calculate the number of checkpoints and the checkpoint interval for each job [6].

3.2.3. CHKPT Calculation pseudo code

Step 1: Initiate Checkpoint intervals.
Step 2: for each job j assigned to a resource r
Step 3: CHN gets FR_r value from the GIS; /* FR_r = F / (S+f)*/
Step 4: CHN calculates the checkpoint interval as CHN = FR_r * T_jre
Step 5: CPN calculates the number of checkpoint interval as CHI = T_jre/CHN;
Where, T_jre: The response time of job j by the resource r, FR_r: Failure rate of the resource r, CHN: The number of checkpoints, CHI: Checkpoint interval.

3.2.4. Grid Information Server pseudo code

Step 1: Start server to initiate jobs.
Step 2: for each job j dispatched by the CPH
Step 3: if (CPH receives a job completion message)
    Sends a message to the GIS to increment S;
Step 4: if (CPH receives a response failure message)
    Sends a message to GIS to increment F;
Step 5: Gets the last checkpoint status from CPS;
Step 7: if (There is a checkpoint stored) CPH dispatches the not completed part of the job along the checkpoint status to the second resource in the resource in the resources list;

Step 8: else CPH dispatches the whole job along to the second resource in the resource list;

If CPH receives a job completion message for a certain job, it will notify the GIS to increase the number of successful times, S, of the resource. Then, it delivers results to the GS which in turn submits it to the user [10] [11]. On the other hand, if the CPH receives a resource failure message, it will notify the GIS to increase the number of failure times, F, of the resource. In this case, the CPH connects to the CPS to get the last checkpoint status of the job and resubmits it along with the job to the second resource in the resources list. Figure 2 shows decisions taken upon resource failure or job completion.

3.2.5. Optimal Checkpoint Generation Algorithm (LF k,MF j)

1) Given that LF,MF and fix k and j values.

2) Find the Probability density function(pdf) for resource x using an exponential distribution is

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

(2)

3) Checkpoint interval I calculated using Markov random walks chains,

\[ I(x) = \Pr(X_n+1 = j \mid X_0 = i) \] ...

(2)

\[ X_n = i, X_{n-1} = i_{n-1}, ..., X_0 = i_0 \] ...

(3)

\[ = \Pr(X_n+1 = j \mid X_n = i) \] ...

(4)

Where, t is the overhead time needed for restarting the job after failure. Take the total useful computation for each case independently as T1 and T2 etc

4) Find the time utilization for job as

\[ T = \frac{\frac{I_f + I_k}{I_f + I_k}}{\frac{I_f + I_k}{I_f + I_k}} \] ...

(5)

5) Then by differentiating times with respect to mean and last failures

\[ \frac{dT_f}{dT} = 0 \] ...

(6)

\[ \frac{dT_k}{dT} = 0 \] ...

(7)

6) Finally optimal checkpointing using the following function as,

\[ \Omega(x) = \frac{kj}{U(x) + kj} \] ...

(6)

Where, O is the optimal checkpointing frequency where U is the inverse function of \( f(x) = xe^x \) called Lambert U function. In this algorithms we have given input as inappropriate number of Checkpoints (Inconsistent), to find optimal Checkpoints (Consistent) and then we have the number of jobs done with optimal checkpoints, time and space complexity. The above algorithm is used to simulate grid jobs with optimal number of checkpoints by using the parameters LastFailure (k) and MeanFailure (j) also to find time and space complexity of the algorithm [7] [8].

Grid is a complex environment and the behavior of the resources in the grid is unpredictable. So, it is difficult to build a grid on a real scale to validate and evaluate scheduling and fault tolerant systems. Therefore, simulation is often used. There are number of well-known grid simulators, such as GridSim, SimGrid and NSGrid. However, none of these simulators support the development of fault-tolerant scheduling algorithms because they have a limited modelling for dynamic grids. So, in order to carry out this study, we have used our implemented grid simulator.

The simulator supports modeling and simulation of grid resources and user applications. It enables the creation of application jobs and mapping of these jobs to resources in the grid. In experiments, we modelled applications with size of 100 jobs. The size of each job is selected to be 20MB. The number of resources in the grid can reach up to 100. The percentage of faults injected is from 10% to 40%. These specifications remain the same in all experiments of measuring performance [2].

We have conducted different simulation experiments by varying the total number of faults injected in the grid and measuring the throughput, turnaround time and the tendency of resources to fail. The proposed system is compared with a recent checkpointing-based scheduling system called Checkpoint based Fault Tolerant Grid System that depends on the response time and resource fault index when scheduling jobs and uses the fault index of each resource when determining the checkpoint intervals and the number of checkpoints for each job.

4. Experimental Results

We have conducted experiments for optimal Checkpointing interval algorithm based on the real-time workload log file analysis.

The Table 1 gives information about the number of jobs submitted to that of number of jobs successfully completed [4]. The simulation is based on the following 18 parameters as given in the Table 2&3and based on the above Table 1 results we have calculated last-failure and mean-failure rate parameters. The mean-failure rate is 650 jobs per 10,000 jobs and last-failure rate is 450 jobs per 10,000 jobs.
Table 1 Computational Workload-logs from NASA grid

<table>
<thead>
<tr>
<th>X (No of Jobs Completed)</th>
<th>Y (No of jobs submitted)</th>
</tr>
</thead>
<tbody>
<tr>
<td>75000</td>
<td>75944</td>
</tr>
<tr>
<td>57000</td>
<td>57930</td>
</tr>
<tr>
<td>79000</td>
<td>79302</td>
</tr>
<tr>
<td>26000</td>
<td>26964</td>
</tr>
<tr>
<td>79000</td>
<td>79302</td>
</tr>
<tr>
<td>74000</td>
<td>76870</td>
</tr>
<tr>
<td>201000</td>
<td>201387</td>
</tr>
<tr>
<td>201000</td>
<td>242264</td>
</tr>
</tbody>
</table>

Table 2 Checkpointing complexity comparison

<table>
<thead>
<tr>
<th>Grid Nodes</th>
<th>Jobs</th>
<th>(CFG) Adaptive Algorithm</th>
<th>(CBOT) Optimal Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CHKPT</td>
<td>Memory(in mbs)</td>
<td>Time</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>500</td>
<td>50</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>1000</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1500</td>
<td>150</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2000</td>
<td>200</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.1 Comparison of Adaptive Checkpointing with CBOT (Optimal Checkpointing)

Here we have taken 18 parameters to analyze the real time workload log file. In this we have find that mean failure rate \( u \) and last failure rate \( k \). The following is the various comparison studies. (Checkpoint based optimal technique)

**Case 1.** Let the jobs and checkpoint intervals are constant in varying grid nodes. Then the results are described in the following table. Here we observed that CBOT algorithm gives optimal results when compared to Adaptive Algorithm [12].

In our experiment, we considered the compilation time, computing cost. We compared our approach CBOT algorithm with Checkpoint based Fault tolerant Grid (Adaptive approach).

**Case 2:** Let the jobs and checkpoint intervals are varying and grid nodes be fixed. Then the results are described in the following table.

We considered the variable number of jobs was executed under uniform number of grid nodes with respective processing time and space parameters. Hence our approach obtains better performance when compared to adaptive approach.

Here we observed that CBOT algorithm gives optimal results when compared to Adaptive check pointing algorithm as shown in Figure 3&4 [13].

Here we considered the variable number of grid nodes was executed under uniform number of jobs with respective processing time and space parameters. Hence our approach obtains better performance when compared to adaptive approach as shown in Figure 5&6 [13].

![Figure 3 Comparison by varying number of jobs](image1)

![Figure 4 Comparisons by varying number of jobs](image2)
5. Conclusions

In this paper, a checkpointing-based scheduling system for grids is proposed and presented. The proposed system depends on average failure time and failure rate of resources combined with response time when taking scheduling decisions. The checkpoint interval is calculated using resource failure rate. The performance of Checkpointing Based Optimized Technique (CBOT) is compared with Checkpointing based Fault-Tolerant Grid (CFG) System scheduling or Adaptive Algorithm system that depends on the response time and the fault index of resources when scheduling resources to execute jobs and CBOT uses the resource fault index when calculating checkpoint interval. The metrics used for evaluation are throughput, turnaround time and failure tendency.

Experimental results show that CBOT effectively schedules jobs in the presence of failures. It is observed that the throughput for the proposed system is better than Adaptive Algorithm. Also, the CBOT improves the turnaround time when compared with the Adaptive Algorithm. Moreover, the failure tendency for the proposed CBOT is far better than the Adaptive Algorithm. Thus, it can be concluded that the proposed scheduling system provides better performance when compared with Adaptive Algorithm. It implies that consideration of the parameters resource failure rate and resource failure time improved the performance of CBOT.

References