IMCLA: Performance Evaluation of Integrated Multilevel Checkpointing Algorithms using Checkpointing Efficiency

Dilbag Singh¹, Amit chhabra² and Jaswinder Singh³

Dept. of Computer Science and Engineering, Guru Nanak Dev University Amritsar India.
Emails: ¹Dggill2@gmail.com, ²chhabra.amit78@gmail.com, ³jaswindersingh@yahoo.com


Abstract: Main objective of this research work is to improve the checkpoint efficiency for integrated multilevel checkpointing algorithms (IMLCA) and prevent checkpointing from becoming the bottleneck of cloud data centers. In order to find an efficient checkpoint interval, checkpointing overheads has also considered in this paper. Traditional checkpointing methods stores persistently snapshots of the present job state and use them for resuming the execution at a later time. The attention of this research is strategies for deciding when and whether a checkpoint should be taken and evaluating them in regard to minimizing the induced monetary costs. By varying rerun time of checkpoints performance comparisons are which will be used to evaluate optimal checkpoint interval. The purposed fail-over strategy will work on application layer and provide highly availability for Platform as a Service (PaaS) feature of cloud computing.

Keywords: Fail-over, high availability, node-recovery, multilevel checkpointing, checkpointing ratio

I. INTRODUCTION

This research paper is an extension paper of [1], [2], [3]. In this paper checkpointing ratio is calculated in order to evaluate the performance of proposed algorithms in [1], [2], [3].

In this paper, checkpoints are integrated with load balancing algorithms for data centers (cloud computing infrastructure) has been considered, taking into account the several constraints such as handling infrastructure sharing, availability, fail-over and prominence on customer service. These issues are addressed by proposing a smart fail-over strategy which will provide high availability to the requests of the clients. New cloud simulation environment has been proposed in this paper, which has the ability to keep all the nodes busy for achieving load balancing and also executes checkpoints for achieving fail-over successfully.

As checkpointing means loss of computation therefore, the overheads presented due to checkpointing should need to be reduced. Checkpointing should enable a CSP to provide high availability to the requests of the clients in case of failure, which demands frequent checkpointing and therefore significant overheads will be introduced. So it becomes more critical to set checkpointing rerun time. Multilevel checkpoints [9], [10], [11], [12], [13], [14], [15] are used in this research work for decreasing the overheads of checkpoints. In this research work, it is assumed that all nodes have same capability. To check that which node is heavy or lightly loaded depends upon the load on that node. The load on a particular node is calculated based on the total completion time taken by the executing and waiting threads.

For implementation of proposed fail-over strategies, a cloud simulation environment is developed, which has the ability to provide high availability to clients in case of failure/recovery of service nodes. Also in this research work, comparison of proposed is made with existing methods.

II. PROBLEM DEFINITION

Checkpointing is a technique to reduce the loss of computation in the manifestation of failures. Two metrics can be used to illustrate a checkpointing scheme:

(i) Checkpoint overhead (increase in the execution time of the job because of a checkpoint implementation).
(ii) Checkpoint latency (duration of time required to save the checkpoint).

This research work evaluates the expression for “checkpointing ratio (R)” of the checkpointing scheme as a function of checkpoint latency and overhead. Main objective
of this paper is to determine the optimal checkpoint interval "checkpointing rerun time". However, to decrease the checkpointing overheads multilevel checkpointing [9], [10], [11], [12], [13], [14], [15] is also used.

III. SCOPE OF THIS RESEARCH WORK

This paper deals with load distribution and high availability for cloud computing environment. Although the research is on cloud computing based, it does not deal with management and security issue of cloud computing environment. As checkpointing means loss of computation therefore in this paper checkpointing overheads and checkpointing latency is also considered.

Since it is not feasible to run these algorithms (local and global integrated multilevel checkpointing algorithms) on cloud systems, a simulator is developed which will simulate the proposed algorithms. Different type of tests will be implemented using integrated multilevel load balancing algorithm to test various aspects of the cloud environment.

This paper also visualizes the experimental results and draws appropriate performance analysis. Appropriate conclusion will be made based upon performance analysis. For future work suitable future directions will be drawn considering limitations of existing work.

Throughout the dissertation work emphasis has been on the use of either open source tools technologies or licensed software.

IV. RELATED WORK

Cloud computing [4], [5], [6] combines worldwide distributed virtual servers and information systems for generating a worldwide source of computing power and information. A cloud computing can offer a resource balancing effect by scheduling jobs at cloud machines with low utilization.

A proper scheduling and efficient load balancing on cloud computing can lead to improve overall system performance and a lower turn-around time for individual jobs. Load balancing is required to disperse the resource's load evenly so that maximum resource utilization and minimum task execution time could be possible. This is very crucial concern in distributed environment, to fairly assign jobs to resources.

Checkpoint [7], [8], [9], [10], [11], [12], [13] is defined as a designated place in a program at which normal processing is interrupted specifically to preserve the status information necessary to allow resumption of processing at a later time. By periodically invoking the check pointing process, one can save the status of a program at regular intervals. If there is a failure one may restart computation from the last checkpoint thereby avoiding repeating the computation from the beginning.

Failure management in high-performance computing systems and clouds has been widely studied in the literature [15], [16], [17], [18]. Different characterization methods of these failures have been made. For instance, Fu and Xu predict the failure incidences in HPC systems through the spatial and time correlation among past failure events [19]. Gokhale and Tivedi [20] forecast the software reliability representing the system architecture using Markov chains. Zang et al. [21] evaluate the performance analysis of failures in large clusters (data centers).

Cloud computing and virtualization have opened a new window in the failure management. Pausing, resuming, and migration VMs [22], [23] are powerful mechanisms to manage failures in such situations. The checkpointing and rollback recovery technique [24] has been widely used in distributed systems. CSP can offer high availability [25] by using it, while adding tolerable overhead to Ta-Shma et al. [26] present a CDP (Continuous Data Protection) with live-migration-based checkpoint mechanism. They use a central repository approach and intercept migration data flow to create the checkpoint images. Although the authors say that it has good performance, no experimentation is presented.

The architecture presented does not seem to be able to make checkpoints of the VM disk data. Parallax [27] developed by Warfield et al. is a storage subsystem for Xen to be used in cluster Xen Virtual Machines. The solution proposed by the authors makes coupled checkpoints of both memory and disk using a Copy-on-Write mechanism (CoW) to maintain the remote images. There is no real experimentation and no performance results of that prototype.

Availability [25], [28], [29], [30] is a reoccurring and a growing concern in software intensive systems. Cloud systems services can be turned off-line due to conservation, power outages or possible denial of service invasions. Fundamentally, its role is to determine the time that the system is up and running correctly; the length of time between failures and the length of time needed to resume operation after a failure.

Availability needs to be analyzed through the use of presence information, forecasting usage patterns and dynamic resource scaling. Current solutions to achieve high reliability in data centers include VM replication [29], and checkpointing [30]. In particular, several scheduling algorithms for balancing checkpoint workload and reliability have been proposed in [31], [32], [33], with an extension in [29] by considering dynamic VM prices. Nevertheless, previous work has only investigated how to derive optimal checkpoint policies to minimize the execution time of a single task.

Although supercomputing systems use high-quality mechanisms, the schemes become less dependable at larger scales as increased component counts increase overall fault rates.

Requests executing on high-performance computing systems can encounter mean times among failures on the order of hours or days due to hardware breakdowns [34] and soft
errors [35]. For example, the 100,000 node BlueGene/L scheme at Lawrence Livermore National Laboratory (LLNL) practices an L1 cache bit error every 8 hours [36] and a hard failure every 7-10 days. Exascale schemes are expected to fail every 3-26 minutes [37], [38].

Usually, requests allow failures by periodically saving their state to checkpointing files. They write these checkpointing files to dependable storage, typically a parallel file system. When will failure occur, an request can restart from a prior state by reading in a checkpointing file. Checkpointing to a parallel file system will be expensive at large scale, where a single checkpointing implementation can take on the order of tens of minutes [39], [40].

Further, computational abilities of large-scale services have increased more quickly than I/O bandwidths. As computing systems increase in scale, increasing failure rates require more frequent checkpointing, but increased system imbalance makes them more expensive. Checkpointing will become both more critical and less practical [41], [42]. Multi-level checkpointing [43], [44] is a promising approach for addressing this problem, which uses multiple types of checkpointing that have different levels of resiliency and cost in a single application run. The slowest but most resilient level writes to the parallel file system, which can withstand a failure of an entire machine.

Faster but less resilient checkpointing levels utilize node-local storage, such as RAM, Flash or disk, and apply cross-node redundancy schemes. Multi-level checkpointing allows [42], [43], [44] applications to take frequent inexpensive checkpointing and less frequent, more resilient checkpointing, resulting in better efficiency and reduced load on the parallel file system.

There exist many models to describe checkpoint systems implementation. Some of the models use multilevel checkpointing approach [45], [46], [47]. Many researchers have worked to lower the overheads of writing checkpoints. Cooperative checkpoints reduce overheads by only writing checkpoints that are predicted to be useful, e.g., when a failure in the near future is likely [48].

Incremental checkpoints reduce the number of full checkpoints taken by periodically saving changes in the application data [49], [50], [51]. These approaches are orthogonal to multilevel checkpoints and can be used in combination with our work. The checkpoint and rollback technique [51] has been widely used in distributed systems. High availability can be offered by using it and suitable fail-over algorithms. E. Existing methods

The Zeus [52] Company develops software that can let the cloud provider easily and cost-effectively offer every customer a dedicated application delivery solution. The ZXTM [52] software is much more than a shared load balancing service and it offers a low-cost starting point in hardware development, with a smooth and cost-effective upgrade path to scale as your service grows.

The Apache Hadoop [53] software library is a framework that allows for the distributed processing of large data sets across clusters of computers using a simple programming model. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly available service(s) on top of a cluster of computers, each of which may be prone to failures.

JPFF [54] is a general-purpose grid toolkit. Federate computing resources working together and handle large computational applications. JPPF uses divide and conquer algorithms to achieve its work successfully. ZXTM [52], Apache Hadoop [53] and JPPF [54] not provide feature of checkpoints.

Checkpointing overheads [55], [56], [57], [58] have been discussed by many researchers. An integrated checkpointing algorithm implements in parallel with the essential computation. Therefore, the overheads presented due to checkpointing should need to be reduced. Much of the previous work [51] present measurements of checkpoint latency and overhead for a few applications. Several models that define the optimal checkpoint interval have been proposed by different researchers. Young proposed a first-order model that describes the optimal checkpointing interval in terms of checkpoint overhead and mean time to interruption (MTTI). Young’s model does not consider failures during checkpointing and recovery [54], while Daly’s extension lead of Young’s model, a higher-order approximation, does [55].

In addition to considering checkpointing overheads and MTTI, the model discussed in [53] includes sustainable I/O bandwidth as a parameter and uses Markov processes to model the optimal checkpoint interval. The model described in [56] uses useful work, i.e., computation that contributes to job completion, to measure system performance.

V. CHECKPOINT LATENCY AND OVERHEAD

The checkpoint latency [32], [33] era is separated into two types of execution: (1) useful computation, and (2) execution necessary for checkpointing. The two types are usually enclosed in time. However, for modelling purposes, it can be assumed that the two types of executions are divided in time. As shown in the Fig. 1, the first C units of time during the checkpointing latency era is supposed to be used for saving the checkpointing. The lingering (L - C) units of time is supposed to be consumed for useful execution of jobs. Even though the C units of overhead are modelled as being acquired at the commencement of the checkpoint latency era, the checkpoint is considered to have been recognised only at the end of the checkpoint latency era.
D. Singh, et al: IMCLA: Performance Evaluation of Integrated Multilevel ...

Even though the above representation of checkpoint latency and overhead is abridged, now it is required to exhibit that it will lead to perfect exploration. Two discrete conditions may arise when an interval is executed.

A. No failure occur during checkpoint latency

A failure will not arise during the interval is executed. In this case, the accomplishment time from the beginning to the end of an interval is T + C. Of the T + C units, T units are consumed for doing useful execution, while acquiring an overhead of C time units. As shown in Fig. 1, (L - C) units of useful computation is performed during the checkpoint latency period. Similar to Fig. 2, L - C units of useful computation is performed during the latency period. Also, the execution time for the interval is T + C.

B. Failure occur during checkpoint latency

A failure occurs sometime during the interval. When a failure occurs, the task must be rolled back to the previous checkpoint, incurring an overhead of R time units. In Fig. 3, the task is rolled back to checkpoint1 (CP1). After the rollback, L - C units of useful computation performed during the latency period of checkpoint CP1 must be performed again, this is necessary, because the state saved during checkpoint CP1 is the state at the beginning of the latency period for checkpoint CP1. In the absence of a further failure, additional T + C units of execution are required before the completion of the interval. Thus, after a failure,

R + (L - C) + (T + C) = R + T + L units of execution is required before the completion of the interval, provided additional failures do not occur.

VI. CHECKPOINTING OVERHEAD RATIO

The main goal of this research is on understanding the effect of checkpoint latency on performance. The objective is not on offering elaborate prototypes for checkpointing structures, as in many previous works. Consequently, this paper uses a simple prototype that is adequate for purposed work. For instance, it is assumed that C and L are constants for a given scheme. A more elaborate model may undertake C and L to be some function of time.

Let G(t)[27], [28] denote the expected (average) amount of execution time required to perform t units of useful computation. (Useful computation excludes the time spent on checkpointing and migration of jobs.) Then, overhead ratio (r) can be defined as:

\[ r = \lim_{t \to \infty} \frac{G(t) - t}{t} = \lim_{t \to \infty} \frac{G(t)}{t} - 1 \]

Note that r will always remain greater than 0 as it is well known some overheads always present in the computation. Smaller the r states that low overheads are there. As the objective of this research to find optimal interval time overhead ratio is rewrite using the following expression:-

\[ r = \frac{(TET + t1(C) + t2(R))}{t} \]

VII. PROBLEMS IN EXISTING TECHNIQUES

A. Three-node architecture

In Fig. 4 there is a request manager (central cloud), clients send their requests to it. Then request manager first divide the given job into threads and then allocate one of the sub_cloud (service manager) to the threads and global
checkpoint will be updated. Each sub_cloud first selects threads in First Come First Serve (FCFS) fashion and allocate lightly loaded service node (service node) to it. The service node then start execution of that thread or it may add this thread in its waiting queue if it is already doing execution of any other thread. N1 to N12 are service nodes which will provide services to the clients.

B. Sub_cloud failure

Fig. 5 illustrates the sub_cloud failure. In it sub_cloud 2 has been failed, that means any node that belong to failed sub_cloud is no longer available to provide any service to the clients. In existing techniques there exist not such algorithm which migrate the exact load of all node to other nodes except redundant node [4], [5] technique. It means it is required to have protection (secondary) node to take load of primary node in case of failure.

C. Service node failure

Fig. 6 illustrates the service node failure. In it service node (N7) has failed which belong to sub_cloud 2. N7 failed means it is no longer available to provide any service to the demand of clients. In existing techniques there exist not such algorithms which will migrate the load of N7 to other nodes. However checkpointing without load balancing can achieve this task but it is based on random decision means load of failed node may be shifted to some other heavily loaded nodes than lightly loaded nodes.

VIII. SIMULATION RESULTS

Table I give the inputs that are given to the simulator. In Table I various Jobs are given with their serial execution time and also if jobs will execute in parallel then how many numbers of threads can be made from it or how many nodes are required to run given job in parallel fashion.

<table>
<thead>
<tr>
<th>Job Name</th>
<th>Threads</th>
<th>Serial Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>100</td>
<td>3</td>
<td>10</td>
</tr>
</tbody>
</table>

Table I: Inputs for the simulator.

A. Global Checkpoint

Designed simulator first divides job into threads and allocate sub_clouds to them in FIFO fashion and global checkpoint will be updated as shown in Fig. 7. It is giving the detail of the global checkpoint, which is showing that which job is going to be run on which sub_cloud and also other relevant information like entered time of job, number of processors required, serial time, thread time etc.

Fig. 7: Global Checkpoint
B. Local checkpoint

Fig. 8 is showing the local checkpoint in it node has been allocated to threads. For all node whether it belong to sub_cloud1 or sub_cloud2, only one local checkpoint is used in this simulator. Local checkpoint contains information like server status (active or active), job status (executing, waiting or finished), server name and also remaining time of threads(execution time + waiting time) etc.

C. Failure of Nodes

To successfully implement failover strategy, node A and E set to be failed, after 5 seconds local checkpoint detect it and transfer load of failed nodes to other nodes. In Fig. 9 it has shown that node A and E has failed and also the parameters server status and job status has also changed. Note that if any node get failed and recovered before checkpoints will rerun then the execution at those nodes remains continue without any problem.

D. Load rebalancing after Node Failure

GUI will work in such a way that if any node gets failed then CSP detect it with the help of checkpoints. Then CSP share the load of failed nodes among the active nodes. In Fig. 10 it has shown that the load of node A and E has been shared with currently active nodes. Only the threads which are executing or waiting on node A and E will be shared no other thread need not to be restart or to be transfer from one active node to other active node.

E. Node Recovery and Load Rebalancing

If any node get recovered then sub_cloud(s) detect it by checking their flag bits, then CSP share the load of heavy loaded nodes with recovered nodes. In Fig. 11 it has been shown that the node A and E has recovered and they have taken some load from other heavy loaded nodes.

VIII.VI Completed

Each completed job transferred to history table and acknowledgement send to its sender, and it will be deleted from both local and global checkpoints, so that in future if failure occur then checkpoint will not make any changes with completed jobs.

IX. PERFORMANCE ANALYSIS

In order to do performance analysis, two comparisons table has been made in this research work. This section first give the performance comparison of developed simulator with existing methods and later on comparison of different approaches is made using different performance metrics.

A. Comparison with existing methods

Table II is showing the comparison of JPPF/Hadoop, Checkpointing and developed simulator. Table II has shown that developed simulator will give better results than existing methods. As JPPF/Hadoop do not provide feature of checkpointing, therefore node failure result in restartation of entire job, whether some threads of that job has been successfully completed on other.
Table II. Feature’s comparison with existing method

<table>
<thead>
<tr>
<th>Feature</th>
<th>JPPF/ Hadoop</th>
<th>Checkpoint</th>
<th>Integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checkpoints</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Failover</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Load Balancing</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Multilevel Checkpoint</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Job Restartation</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Architecture</td>
<td>2 Tier</td>
<td>2 Tier</td>
<td>3 Tier</td>
</tr>
<tr>
<td>Resources Utilization</td>
<td>Low</td>
<td>Medium</td>
<td>Maximum</td>
</tr>
<tr>
<td>Checkpoint overheads</td>
<td>No</td>
<td>More</td>
<td>Medium</td>
</tr>
<tr>
<td>Checkpoint latency</td>
<td>0</td>
<td>&gt; 0</td>
<td>&gt; 0</td>
</tr>
<tr>
<td>Checkpoint ratio</td>
<td>0</td>
<td>&gt; 0</td>
<td>&gt; 0</td>
</tr>
</tbody>
</table>

**B. Comparison with no checkpoint, checkpoint without load balancing and purposed method**

Table III is showing the performance comparison of different approaches. These approaches are without checkpoints, checkpoints without load balancing algorithms and integration of checkpointing with load balancing algorithms (Purposed technique). It has been clearly shown in Table III that purposed method gives better results than other methods. As in no checkpoint method it not possible to achieve failover without restartation of the jobs, and without integration of checkpointing with load balancing algorithms may cause the problem of random allocation of nodes to the threads, which may migrate load of failed nodes to heavy loaded nodes than lightly loaded nodes.

Table III. Metric’s comparison of different approaches

<table>
<thead>
<tr>
<th>Type of Metric</th>
<th>No chp.</th>
<th>Chp.</th>
<th>Integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Execution time</td>
<td>14.73</td>
<td>13.46</td>
<td>10.94</td>
</tr>
<tr>
<td>Minimum Execution time</td>
<td>15</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Average Waiting Time</td>
<td>14.73</td>
<td>9.46</td>
<td>5.2</td>
</tr>
<tr>
<td>Minimum Waiting Time</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>THP(after 200 seconds)</td>
<td>61</td>
<td>73</td>
<td>102</td>
</tr>
</tbody>
</table>

Fig. 12 illustrates the graph of Average Waiting (AWT). It has been shown that whether time increases, but failure and recovery of nodes do not effect too much as compared to other approaches. Therefore it is clearly shown that the purposed method gives better results than existing methods as AWT of integrated approach always stay lower than the other existing methods lines.

**Fig. 12: Average Waiting Time comparison with existing methods**

Fig. 13 demonstrates the diagram of Average Execution Time (AET) metric. In Fig. 13 it has been shown that whether time intensifications, but disaster and repossession of nodes do not influence too much as equated to other methodologies. Consequently it is undoubtedly revealed that the purposed technique contributes improved fallouts than prevailing approaches as AET in incorporated checkpointing methodologies line continuously vacation subordinate than the supplementary techniques lines.

**Fig. 13: Average Execution Time comparison with existing methods**

In Fig. 14 it has remained publicised that whether time augmentations, but disaster and recouping of nodes do not encouragement too much as associated to other approaches. Accordingly it is unquestionably exposed that the purposed technique donates better-quality fallouts than predominant methodologies as THP in incorporated checkpointing methodology’s line continuously vacation subordinate than the supplementary techniques lines.
D. Singh, et al: IMCLA: Performance Evaluation of Integrated Multilevel...

C. Checkpointing overheads

Primary objective of this paper is to provide high availability and also to decrease checkpointing overheads. Therefore there exists trade-off while selecting appropriate rerun checkpointing interval. As decreasing interval time introduces more and more overheads and also increasing interval will not give effective results.

Table IV is showing the calculation of checkpointing overheads considering different checkpointing intervals. It has been prominently shown in Table iv that the 2 seconds rerun time contains too many overheads than other intervals.

Table IV. Checkpointing overheads.

<table>
<thead>
<tr>
<th>Time in seconds</th>
<th>2 seconds</th>
<th>5 seconds</th>
<th>10 seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>24</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>80</td>
<td>47</td>
<td>21</td>
<td>15</td>
</tr>
<tr>
<td>120</td>
<td>71</td>
<td>31</td>
<td>22</td>
</tr>
<tr>
<td>160</td>
<td>93</td>
<td>44</td>
<td>27</td>
</tr>
<tr>
<td>200</td>
<td>117</td>
<td>66</td>
<td>31</td>
</tr>
</tbody>
</table>

Fig. 15 has shown the values of checkpointing overheads considering several checkpointing intervals graphically. It has been significantly shown in Fig. 15 that the 2 seconds rerun time contains too many overheads than other intervals as its line is quite increasing than other line of respective intervals.

D. Checkpointing ratio

The main goal of this research is on understanding the effect of checkpoint latency on performance. The objective is not on offering elaborate prototypes for checkpointing structures, as in many previous works. Consequently, this dissertation deals in decreasing checkpointing ratio. If checkpointing ratio is 1 then it is best and close to 1 is good.

Table V. Checkpointing ratio.

<table>
<thead>
<tr>
<th>Time in seconds</th>
<th>2 seconds</th>
<th>5 seconds</th>
<th>10 seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>1.12</td>
<td>1.12</td>
<td>1.21</td>
</tr>
<tr>
<td>80</td>
<td>1.23</td>
<td>1.19</td>
<td>1.26</td>
</tr>
<tr>
<td>120</td>
<td>1.28</td>
<td>1.22</td>
<td>1.29</td>
</tr>
<tr>
<td>160</td>
<td>1.39</td>
<td>1.31</td>
<td>1.37</td>
</tr>
<tr>
<td>200</td>
<td>1.55</td>
<td>1.43</td>
<td>1.47</td>
</tr>
</tbody>
</table>

Table V is showing the calculation of checkpointing ratio considering different checkpointing intervals. It has been evidently shown in Table V that the 5 seconds rerun time contains results more close to 1 than other intervals this is because small interval introduces too many overheads whereas large interval results in too many threads restartation. Fig. 16 is demonstrating the checkpointing ratio considering several checkpointing intervals graphically. It has been significantly shown in Fig. 16 that that the 5 seconds rerun time contains results more close to 1 than other intervals.

E. Checkpointing efficiency

The additional objective of this research is on understanding the effect of checkpoint latency on checkpointing efficiency. Therefore it required to calculate checkpoint efficiency (minimum effects on throughput) by using checkpointing ratio to systematically measure the checkpoint efficiency (Ef). Consequently, this paper deals in increasing checkpointing efficiency. If checkpointing efficiency is 1 then it is best and close to 1 is good but less than 0.5 to 0 is bad. Table VI is giving the checkpointing efficiency considering different intervals. Fig. 17 is demonstrating the checkpointing efficiency considering several checkpointing intervals graphically. It has been shown in Fig. 17 that that the 5 seconds rerun time contains results more close to 1 than other intervals therefore it can be say that when interval time is 5 it give better results than others.
X. CONCLUSION AND FUTURE DIRECTIONS

A. Conclusion

This paper has proposed a novel technique to analyse the performance of checkpointing algorithms. The offered technique is based on fail-over algorithms which will provide high availability to cloud's clients, and estimating the required measures by varying the interval time of integrated checkpoint algorithms.

A suitable cloud environment is made with 6 service nodes to analyse the execution time of the parallel jobs and integrated checkpointing algorithms will control the overall execution of the jobs and also provide high availability in case of node failure. Comparisons have been made in this research work by taking different failure time of nodes and checkpointing intervals. Comparisons are made using different well known parameters and metrics. It has been proved that setting of the checkpointing interval is a critical task as if checkpointing rerun time has been decrease too much then it will not give good results.

The proposed method is not limited to the scenario and number of nodes described in this dissertation, or to the failure of nodes used in this research work. It can be used to analyse any checkpointing high availability scheme, with various scenarios. The proposed technique can be also used to provide analytical answers to problems that haven't been dealt with before or were handled by a simulation study. Examples of such problems are deriving the number of checkpoints that minimizes the average completion time and computing the probability of meeting a given deadline.

B. Future directions

In the near future, this research will be extending to the multilevel checkpointing integration for the case where the multilevel checkpointing interval is not fixed. Developed technique which will allow checkpointing rerun time to vary. Therefore checkpointing interval will depend upon the nature of the executing jobs expecting that the extended technique will give less waste time than the proposed one (decreasing checkpointing overheads).

In addition, this research will be extended for improving the way to save and rerun checkpointing. For example, in some requests, there are many communications between nodes. If one performs a checkpointing while there is a large amount of communications going on, the checkpointing overhead will become more expensive. Therefore, the communication or I/O transfer rate may be another factor to consider when performing a checkpoint.

In this research work homogeneous nodes has been considered for simulation environment, in future work heterogeneous nodes will be used for better results.

REFERENCES


[2]. D. Singh, J. Singh, A. Chhabra, Failures in cloud computing data centers in 3-tier cloud architecture, Accepted for publication in International Journal of Information Engineering and Electronic Business (IJEEEB), ISSN: 2074-9023 (Print), ISSN: 2074-9031 (Online).


[10]. S. Bansal, S. Sharma, Identification of Critical Factors in Checkpointing Based Multiple Fault Tolerance for


