International Journal of Computing and Digital Systems

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

Dilbag Singh[1], Amit chhabra[2] and Jaswinder Singh[3]

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

Received 21 Sep. 2012, Revised 20 Oct. 2012, Accepted 2 Nov. 2012

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 determines the optimal checkpoint interval ``checkpointing rerun time". However to decrease 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 draw 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.

JPPF [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.

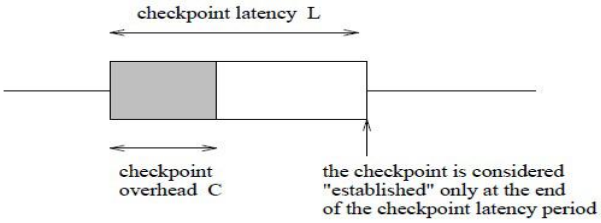


Fig. 1: Modelling latency and overhead (adapted from [27])

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.

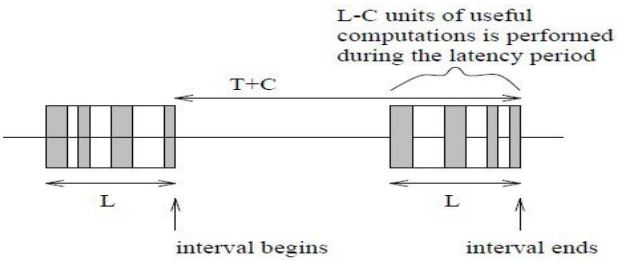


Fig. 2: No failure occur during checkpoint latency (adapted from [27])

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.

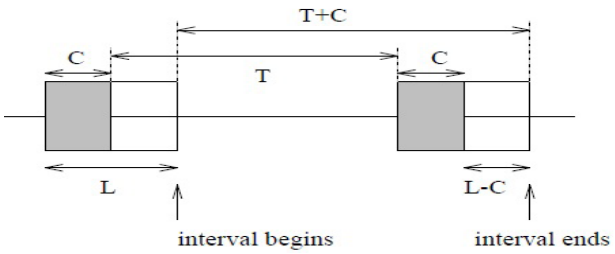


Fig. 3: If failure occurs during checkpoint latency (adapted from [27])

Now consider when the failure occurs, as shown in Fig. 3, the system can be considered to have rolled back to the end of the ``shaded portion'' in the latency period for checkpoint CP1. (Note that no state change occurs during the ``shaded portion".) Now it is apparent that, in the absence of further failure, R + T + L units of execution are required to complete the interval. Thus, this representation of checkpoint latency and overhead yields the same conclusion as the more accurate representation in Fig. 2.

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= (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.

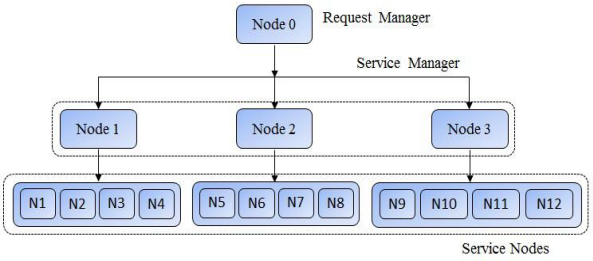


Fig 4: 3-node architecture.

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 is 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.

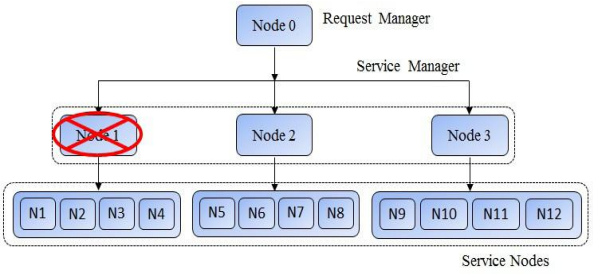


Fig. 5: Sub_cloud failure.

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.

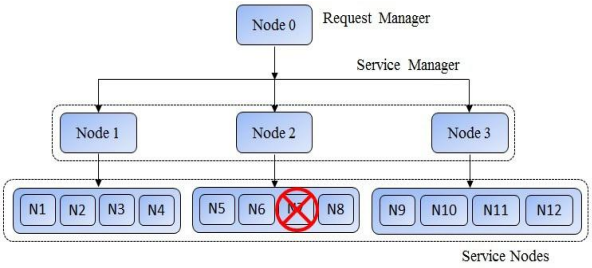


Fig. 6: Service node failure.

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 I: Inputs for the simulator.

Job Name	Threads	Serial Time
1	2	20
2	3	45
3	3	30
4	2	40
...	...	...
100	3	10

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.

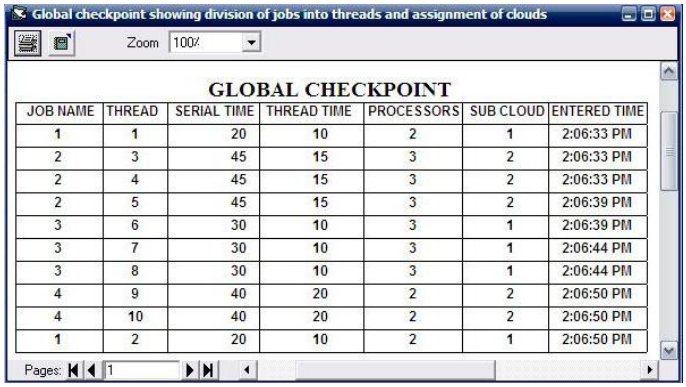
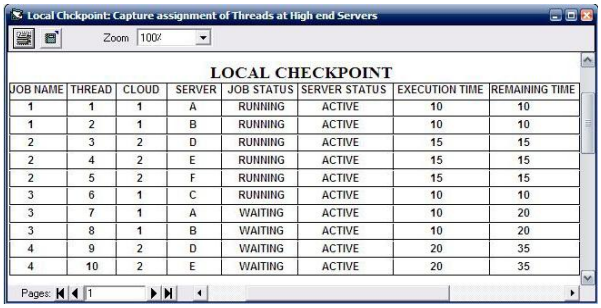


Fig. 7: Global Checkpoint

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.


Local Chckpoint: Capture assignment of Threads at High end Servers

Zoom 100%

LOCAL CHECKPOINT

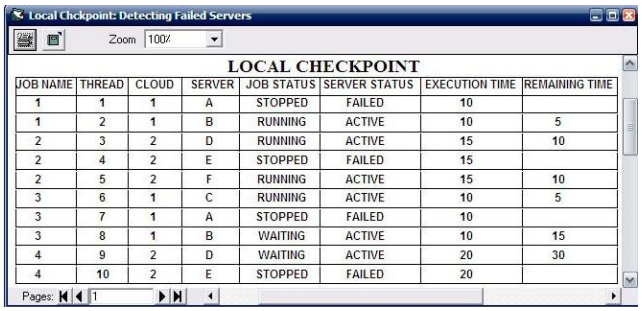
JOB NAME	THREAD	CLOUD	SERVER	JOB STATUS	SERVER STATUS	EXECUTION TIME	REMAINING TIME
1	1	1	A	RUNNING	ACTIVE	10	10
1	2	1	B	RUNNING	ACTIVE	10	10
2	3	2	D	RUNNING	ACTIVE	15	15
2	4	2	E	RUNNING	ACTIVE	15	15
2	5	2	F	RUNNING	ACTIVE	15	15
3	6	1	C	RUNNING	ACTIVE	10	10
3	7	1	A	WAITING	ACTIVE	10	20
3	8	1	B	WAITING	ACTIVE	10	20
4	9	2	D	WAITING	ACTIVE	20	35
4	10	2	E	WAITING	ACTIVE	20	35

Pages: 1

Fig. 8: Local checkpoint

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.


Local Chckpoint: Detecting Failed Servers

Zoom 100%

LOCAL CHECKPOINT

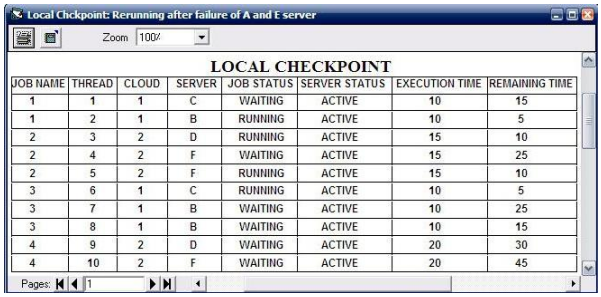
JOB NAME	THREAD	CLOUD	SERVER	JOB STATUS	SERVER STATUS	EXECUTION TIME	REMAINING TIME
1	1	1	A	STOPPED	FAILED	10	
1	2	1	B	RUNNING	ACTIVE	10	5
2	3	2	D	RUNNING	ACTIVE	15	10
2	4	2	E	STOPPED	FAILED	15	
2	5	2	F	RUNNING	ACTIVE	15	10
3	6	1	C	RUNNING	ACTIVE	10	5
3	7	1	A	STOPPED	FAILED	10	
3	8	1	B	WAITING	ACTIVE	10	15
4	9	2	D	WAITING	ACTIVE	20	30
4	10	2	E	STOPPED	FAILED	20	

Pages: 1

Fig. 9. Local checkpoint showing Failed nodes

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.


Local Chckpoint: Rerunning after failure of A and E server

Zoom 100%

LOCAL CHECKPOINT

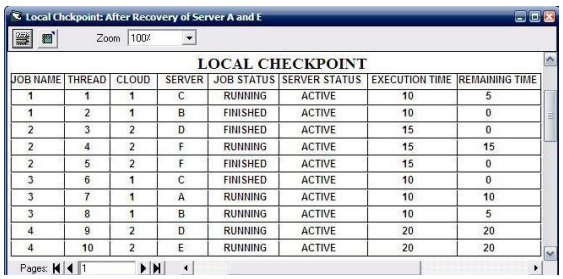
JOB NAME	THREAD	CLOUD	SERVER	JOB STATUS	SERVER STATUS	EXECUTION TIME	REMAINING TIME
1	1	1	C	WAITING	ACTIVE	10	15
1	2	1	B	RUNNING	ACTIVE	10	5
2	3	2	D	RUNNING	ACTIVE	15	10
2	4	2	F	WAITING	ACTIVE	15	25
2	5	2	F	RUNNING	ACTIVE	15	10
3	6	1	C	RUNNING	ACTIVE	10	5
3	7	1	B	WAITING	ACTIVE	10	25
3	8	1	B	WAITING	ACTIVE	10	15
4	9	2	D	WAITING	ACTIVE	20	30
4	10	2	F	WAITING	ACTIVE	20	45

Pages: 1

Fig. 10: Local checkpoint showing rebalancing of load

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.


Local Chckpoint: After Recovery of Server A and E

Zoom 100%

LOCAL CHECKPOINT

JOB NAME	THREAD	CLOUD	SERVER	JOB STATUS	SERVER STATUS	EXECUTION TIME	REMAINING TIME
1	1	1	C	RUNNING	ACTIVE	10	5
1	2	1	B	FINISHED	ACTIVE	10	0
2	3	2	D	FINISHED	ACTIVE	15	0
2	4	2	F	RUNNING	ACTIVE	15	15
2	5	2	F	FINISHED	ACTIVE	15	0
3	6	1	C	FINISHED	ACTIVE	10	0
3	7	1	A	RUNNING	ACTIVE	10	10
3	8	1	B	RUNNING	ACTIVE	10	5
4	9	2	D	RUNNING	ACTIVE	20	20
4	10	2	E	RUNNING	ACTIVE	20	20

Pages: 1

Fig. 11: Rebalancing of load among recovered 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

Feature	JPPF/ Hadoop	Checkpoint	Integrated
Checkpoints	No	Yes	Yes
Failover	No	Yes	Yes
Load Balancing	Yes	No	Yes
Multilevel Checkpoint	No	No	Yes
Job Restartation	Yes	No	No
Architecture	2 Tier	2 Tier	3 Tier
Resources Utilization Checkpoint overheads	Low No	Medium More	Maximum Medium
Checkpoint latency	0	> 0	> 0
Checkpoint ratio	1	> 0	> 0

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

Type of Metric	No chp.	Chp.	Integrated
Average Execution time	14.73	13.46	10.94
Minimum Execution time	15	10	10
Average Waiting Time	14.73	9.46	5.2
Minimum Waiting Time	5	0	0
THP(after 200 seconds)	61	73	102

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.

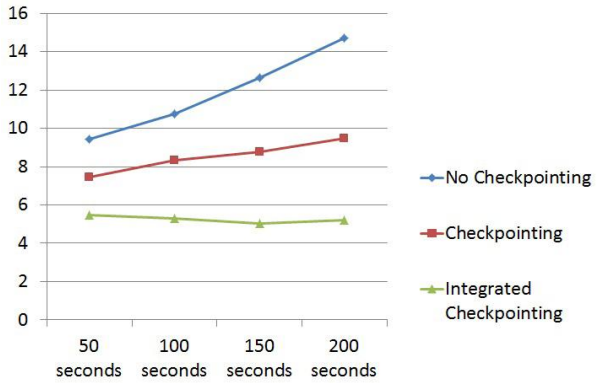


Fig. 12: Average Waiting Time comparison with existing methods

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.

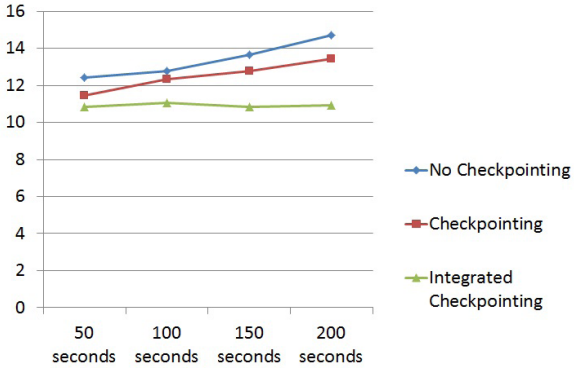


Fig. 13: Average Execution Time comparison with existing methods

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.

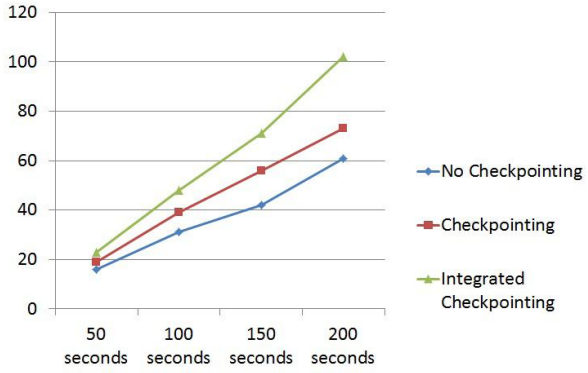


Fig. 14: Throughput comparison

C. Checkpointing overheads

Primary objective of this paper is to provide high availability and also to decrease checkpointing overheads. Therefore there exists trade-of 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.

Time in seconds	2 seconds	5 seconds	10 seconds
40	24	11	7
80	47	21	15
120	71	31	22
160	93	44	27
200	117	56	31

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.

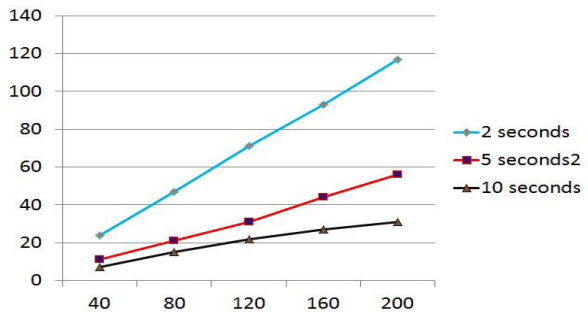


Fig. 15: Checkpointing overheads.

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.

Time in seconds	2 seconds	5 seconds	10 seconds
40	1.12	1.12	1.21
80	1.23	1.19	1.26
120	1.28	1.22	1.29
160	1.39	1.31	1.37
200	1.55	1.43	1.47

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.

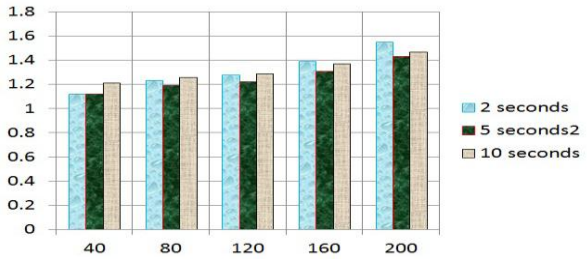


Fig. 16: Checkpointing ratio.

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.

Table VI. Checkpointing efficiency.

Time in seconds	2 seconds	5 seconds	10 seconds
40	0.89	0.89	0.83
80	0.81	0.84	0.79
120	0.82	0.81	0.77
160	0.72	0.76	0.71
200	0.64	0.69	0.68

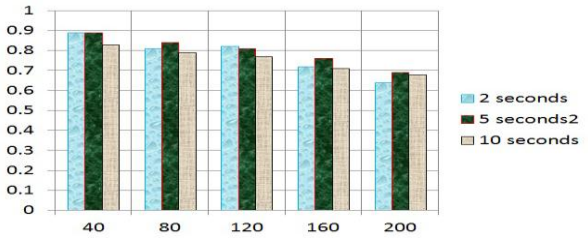


Fig. 17: Checkpointing efficiency.

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 adds too many overheads in the execution time of jobs and if checkpoint rerun time has been increased 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

[1]. D. Singh, J. Singh, A. Chhabra, High Availability of Clouds: Failover Strategies for Cloud Computing Using Integrated Checkpointing Algorithms, csnt, pp.698-703, 2012 International Conference on Communication Systems and Network Technologies, 17 May 2012, ISBN:978-1-4673-1538-8, Rajkot, India. [Online]. Available: http://ieeexplore.ieee.org and http://www.computer.org.

[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 (IJIEEB), ISSN: 2074-9023 (Print), ISSN: 2074-9031 (Online).

[3]. D. Singh, J. Singh, A. Chhabra, Evaluating overheads of integrated multilevel checkpointing algorithms in cloud computing environment," IJCNIS, vol.4, no.5, pp.29-38, 2012, ISSN: 2074-9090, (Print), ISSN: 2074-9104 (Online).

[4]. G. Wang and N. TSE, The impact of virtualization on network performance of amazon EC2 data center, In INFOCOM, 2010 Proceedings IEEE, pp. 1-9, IEEE, 2010.

[5]. B. Wickremasinghe, R.N. Calheiros, R. Buyya, CloudAnalyst: A CloudSimbased visual modeller for analysing cloud computing environments and applications, In 2010 24th IEEE International Conference on Advanced Information Networking and Applications, pp. 446-452, IEEE, 2010.

[6]. A. Kondo, D. Anderson, Exploiting Non-Dedicated Resources for Cloud Computing, In Proceedings of 12th IEEE/IFIP Network Operations & Management Symposium (NOMS 2010), Osaka Japan.

[7]. R. Koo, S. Toueg, Checkpointing and rollback-recovery for distributed systems, IEEE Trans, Softw. Eng., vol. 13, pp. 23-31, Jan. 1997.

[8]. V. G. Kulkarni, V. F. Nicola, K. S. Trivedi, E ects of checkpointing and queueing on program performance, Commun. Statist.-Stochastic Models, vol. 4, no. 6, pp. 615-648, 1999.

[9]. P. L'Ecuyer, J. Malenfant, Computing optimal checkpointing strategies for rollback and recovery systems, IEEE Trans. Computers, vol. 37, pp. 491-496, Apr. 1988.

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

Distributed System, Journal of Emerging Trends in Computing and Information Sciences, vol. 2 no. 1, 2010.
[11]. M. Herlihy, J. Wing, Linearizability: a correctness condition for concurrent objects, ACM Trans. on Progr. Languages and Syst., pp. 463- 492, 1990. International Journal on Internet and Distributed Computing Systems (IJIDCS), vol. 1, no. 1.
[12]. H. Jung, D. Shin, H. Kim, H.Y. Lee, Design and Implementation of Multiple FaultTolerant MPI over Myrinet (M3) , SC—05 Nov 1218,2005, Seattle, Washington, USA Copyright 2005 ACM.
[13]. M. Elnozahy, L. Alvisi, Y. M.Wang, D.B. Johnson, A survey of rollbackrecovery protocols in message passing systems. Technical Report CMUCS- 96-81, USA, Oct. 1996.
[14]. S. Bansal, S. Sharma, I. Trivedi, A Detailed Review of Fault-Tolerance Techniques in Distributed System, International Journal on Internet and Distributed Computing Systems. vol. 1, no. 1, 2011.
[15]. A. Agbaria,W. H Sanders, 'Distributed Snapshots for Mobile Computing Systems, IEEE Intl. Conf. PERCOM04, pp. 1-10, 2004.
[16]. B. Schroeder, G. Gibson, A large-scale study of failures in highperformance computing systems, in International Conference on Dependable Systems and Networks (DSN 2006), Philadelphia, PA, USA, pp. 249-258, Jun. 2006.
[17]. R. Sahoo, et. al, Failure data analysis of a large-scale heterogeneous server environment, International Conference on Dependable Systems and Networks (DSN 2004), Florence, Italy, pp. 772-781, 28 Jul. 2004.
[18]. Y. Liang, Y. Zhang, A. Sivasubramaniam, R. Sahoo, J. Moreira, M. Gupta, Filtering failure logs for a bluegene/l prototype, in International Conference on Dependable Systems and Networks (DSN 2005), Yokohama, Japan, pp. 476-485, Jul. 2005.
[19]. P. Yalagandula, S. Nath, H. Yu, P. Gibbons, S. Seshan, Beyond availability: Towards a deeper understanding of machine failure characteristics in large distributed systems, in USENIX Workshop on Real, Large Distributed Systems (WORLDS'04), San Francisco, CA, USA, Dec. 2004.
[20]. S. Fu, C. Xu, Exploring event correlation for failure prediction in coalitions of clusters, in 2007 ACM/IEEE conference on Supercomputing, Reno, NV, USA, pp. 10-16, Nov. 2007.
[21]. S. Gokhale, K. Trivedi, Analytical Models for Architecture-Based Software Reliability Prediction: A Unification Framework, IEEE Trans. On Reliability, vol. 55, no. 4, pp. 578-590, 2006.
[22]. Y. Zhang, M. Squillante, A. Sivasubramaniam, R. Sahoo, Performance implications of failures in large-scale cluster scheduling, Lecture Notes in Computer Science, vol. 3277, pp. 233-252, 2005.
[23]. M. Rosenblum, T. Garfinkel, V irtual machine monitors: Current technology, future trends, Computer, vol. 38, no. 5, pp. 39-47, 2005.
[24]. A. Nagarajan, F. Mueller, C. Engelmann, S. Scott, Proactive fault tolerance for HPC with Xen virtualization, in 21st annual international conference on Supercomputing, Seattle, WA, USA, pp. 23-32, Jun. 16-20 2007.
[25]. R. Koo, S. Toueg, Checkpointing, rollback-recovery for distributed systems, IEEE Transactions on Software Engineering, vol. 13, no. 1, pp. 23-31, 1987.
[26]. G. Vallee, T. Naughton, H. Ong, S. Scott, Checkpoint/Restart of Virtual Machines Based on Xen, in High Availability, Performace Computing Workshop (HAPCW 2006), Santa Fe, NM, USA, Oct. 17 2006.
[27]. P. Ta-Shma, G. Laden, M. Ben-Yehuda, M. Factor, Virtual machine time travel using continuous data protection, checkpointing, ACM SIGOPS Operating Systems Review, vol. 42, pp. 127-134, 2008.
[28]. A. Warfield, R. Ross, K. Fraser, C. Limpach, S. Hand, Parallax: Managing storage for a million machines, in 10th Workshop on Hot Topics in Operating Systems (HotOS), Santa Fe, NM, USA, pp. 1-11, Jun. 12-15 2005.
[29]. R. Badrinath, R. Krishnakumar, R. Rajan, Virtualization aware job schedulers for checkpoint-restart, in 13th International Conference on Parallel, Distributed Systems (ICPADS'07), vol. 2, pp. 1-7, Dec. 5-7 2007.
[30]. J. D. Sloan, High Performance Linux Clusters With Oscar, Rocks, OpenMosix, Mpi, OReilly, ISBN 10: 0-596- 00570-9 / ISBN 13: 9780596005702, pp. 2-3, Nov.2004.
[31]. R. Badrinath, R. Krishnakumar, R. Rajan, Virtualization aware job schedulers for checkpoint-restart, in Proceedings of 13th International Conference on Parallel, Distributed Systems (ICPADS07), Dec. 2007.
[32]. I. Goiri, F. Juli'a, J. Guitart, J. Torres, Checkpointbased Fault-tolerant Infrastructure for Virtualized Service Providers, in Proceedings of IEEE/IFIP Network Operations, Management Symposium, Aug. 2010.
[33]. M. Zhang, H. Jin, X. Shi, S. Wu, VirtCFT: A Transparent VMLevel Fault-Tolerant System for Virtual Clusters, in Proceedings of Parallel, Dec. 2010.
[34]. Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, and S. L. Scott, An Optimal Checkpoint/Restart Model for a Large Scale High Performance Computing System, in Proceedings of Parallel and Distributed Processing (IPDPS), Apr. 2008.
[35]. B. Schroeder, G. A. Gibson, A Large-Scale Study of Failures in High- Performance Computing Systems, in In Proceedings of the International Conference on Dependable Systems, Networks (DSN), pp. 249-258, Jun. 2006.
[36]. S. E. Michalak, K. W. Harris, N. W. Hengartner, B. E. Takala, and S. A. Wender, Predicting the Number of Fatal Soft Errors in Los Alamos National Laboratorys ASC Q Supercomputer, IEEE Transactions on Device, Materials Reliability, vol. 5, no. 3, pp. 329-335, Sept. 2005.
[37]. J. N. Glosli, K. J. Caspersen, J. A. Gunnels, D. F. Richards, R. E. Rudd, F. H. Streitz, Extending Stability Beyond CPU Millennium: A Micron-Scale Atomistic Simulation of Kelvin-Helmholtz Instability, in Proceedings of the 2007 ACM/IEEE Conference on Supercomputing (SC), pp. 1-11, 2007.
[38]. B. Schroeder, G. Gibson, Understanding Failure in Petascale Computers, Journal of Physics Conference Series: SciDAC, vol. 78, p. 012022, Jun. 2007.
[39]. K. Iskra, J. W. Romein, K. Yoshii, P. Beckman, ZOID: I/OForwarding Infrastructure for Petascale Architectures, in PPoPP '08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 153-162, 2008.
[40]. R. Ross, J. Moreira, K. Cupps, W. Pfeiffer, Parallel I/O on the IBM Blue Gene/L System, Blue Gene/L Consortium Quarterly Newsletter, Tech. Rep., First Quarter, 2006.
[41]. R. Hedges, B. Loewe, T. McLarty, C. Morrone, Parallel File System Testing for the Lunatic Fringe: The Care, Feeding

of Restless I/O Power Users, in Proceedings of the 22nd IEEE / 13th NASA Goddard Conference on Mass Storage Systems, Technologies (MSST), pp. 3-17, 2005.

[42]. R. E. Lyons, W. Vanderkulk, The Use of Triple-Modular Redundancy to Improve Computer Reliability, IBM Journal of Research and Development, vol. 6, no. 2, pp. 200-209, 1962.

[43]. E. Gelenbe, A Model of Roll-back Recovery with Multiple Checkpoints, in Proceedings of the 2nd International Conference on Software Engineering (ICSE '76), pp. 251-255, 1976.

[44]. N. H. Vaidya, A Case for Multi-Level Distributed Recovery Schemes, Texas A&M University, Tech. Rep. 94-043, May 1994.

[45]. J. W. Young, A First Order Approximation to the Optimum Checkpoint Interval, Communications of the ACM, vol. 17, no. 9, pp. 530-531, 1974.

[46]. A. Duda, The Effects of Checkpointing on Program Execution Time, Information Processing Letters, vol. 16, no. 5, pp. 221-229, 1983.

[47]. J. S. Plank, M. G. Thomason, Processor Allocation, Checkpoint Interval Selection in Cluster Computing Systems, Journal of Parallel Distributed Computing, vol. 61, no. 11, pp. 1570-1590, 2001.

[48]. A. J. Oliner, L. Rudolph, R. K. Sahoo, Cooperative Checkpointing: A Robust Approach to Large-Scale Systems Reliability, Proceedings of the 20th Annual International Conference on Supercomputing, pp. 14-23, 2006.

[49]. S. Agarwal, R. Garg, M. S. Gupta, J. E. Moreira, Adaptive Incremental Checkpointing for Massively Parallel Systems, in Proceedings of the 18th Annual International Conference on Supercomputing (ICS), pp. 277-286, 2004.

[50]. S. I. Feldman, C. B. Brown, IGOR: A System for Program Debugging via Reversible Execution, in Proceedings of the 1988 ACM SIGPLAN, SIGOPS Workshop on Parallel, Distributed Debugging (PADD), pp. 112- 123, 1988.

[51]. N. Naksinehaboon, Y. Liu, C. B. Leangsuksun, R. Nassar, M. Paun, S. L. Scott, Reliability-Aware Approach: An Incremental Checkpoint Restart Model in HPC Environments, in Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing, the Grid (CCGRID), pp. 783-788, 2008.

[52]. ZXTM for cloud Hosting Providers, Jan. 2010, [Online]. Available: http://www.zeus.com/cloud-computing/for-cloud-providers.html.

[53]. What Is Apache Hadoop?, [Last Published:] Dec. 28 2011, [Online]. Available: http://hadoop.apache.org.

[54]. JPPF Work distribution, [Last Released] Jan. 31 2012, [Online]. Available: http://www.jppf.org

[55]. P. Kumar, L. Kumar, R. K. Chauhan, A Nonintrusive Hybrid Synchronous Checkpointing Protocol for Mobile Systems, IETE Journal of Research, vol. 52 no. 2 &3, 2006.

[56]. P. Kumar, A Low-Cost Hybrid Coordinated Checkpointing Protocol for mobile distributed systems, Mobile Information Systems. pp. 13-32, vol. 4, no. 1, 2007.

[57]. L. Kumar, P. Kumar, A Synchronous Checkpointing Protocol for Mobile Distributed Systems: Probabilistic Approach, International Journal of Information, Computer Security, vol.1, no.3 pp. 298-314.

[58]. S. Kumar, R. K. Chauhan, P. Kumar, A Minimum-process Coordinated Checkpointing Protocol for Mobile Computing Systems, International Journal of Foundations of Computer science, vol 19, no. 4, pp. 1015- 1038, 2008.

[59]. G. Cao , M. Singhal , On coordinated checkpointing in Distributed Systems, IEEE Transactions on Parallel, Distributed Systems, vol. 9, no.12, pp. 1213-1225, Dec 1998.

[60]. G. Cao , M. Singhal, On the Impossibility of Minprocess Non-blocking Checkpointing, an Efficient Checkpointing Algorithm for Mobile Computing Systems, Proceedings of International Conference on Parallel Processing, pp. 37-44, Aug. 1998.