2012 International Conference on Communication Systems and Network Technologies

High Availability of Clouds: Failover Strategies for Cloud Computing using Integrated Checkpointing Algorithms

Dilbag Singh
Dept. Computer Science & Engineering
Guru Nanak Dev University
Amritsar (Punjab) India.
Email:dggill2@gmail.com

Jaswinder Singh
Dept. Computer Science & Engineering
Guru Nanak Dev University
Amritsar (Punjab) India.
Email:jaswindersingh@yahoo.com

Amit Chhabra
Dept. Computer Science & Engineering
Guru Nanak Dev University
Amritsar (Punjab) India.
Email:chhabra.amit78@gmail.com

***Abstract*—This paper presents an approach for providing high availability to the requests of cloud's clients. To achieve this objective, failover strategies for cloud computing using integrated checkpointing algorithms are purposed in this paper. Purposed strategy integrate checkpointing feature with load balancing algorithms and also make multilevel checkpoint to decrease checkpointing overheads. For implementation of purposed failover 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 paper comparison of developed simulator is made with existing methods. The purposed failover strategy will work on application layer and provide highly availability for Platform as a Service (PaaS) feature of cloud computing.**

***Keywords*-Failover; Load balancing; Node-recovery; Multi-level checkpointing; Restartation;**

I. Introduction

Cloud computing [3] is currently emerging as a powerful way to transform the IT industry to build and deploy custom applications. In cloud environment jobs keep on arriving to the data centers for execution and nodes[1]will be allocated to the jobs for their execution as per their requirements and successfully executed jobs will leave the nodes. In this scenario it may possible that some nodes will become inactive while executing threads due to some failure. So there is need of efficient failover strategy for handling failures as it may cause restartation of entire work, whether some threads of the job has been successfully done on other nodes. In case of node failure, that means, the node is no longer accessible to service any demand of clients, the cloud must migrate jobs to the other node.

Hardware failure or any unexpected reason makes node be inactive. To achieve failover one solution is to implement the concept of redundancy [16]. High availability is achieved by having multiple secondary nodes that are exact replicas of a primary node. Constantly, they monitor the work of the primary node waiting to take over if it fails. In this basic form, only a single primary node is in active use while the remaining secondary nodes are in stand-by mode.But this solution is only feasible for servers or if the nodes are few. As this research work focus on providing the high availability for service nodes, having replica of all service nodes will not be feasible as it will increase complexity, cost etc, thus to have stand-by secondary nodes solution proved to be inefficient. 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, failover and prominence on customer service. These issues are addressed by proposing a smart failover strategy which will provide high availability to the requests of the clients. New cloud simulation environment has been purposed in this paper, which has the ability to keep all the nodes busy for achieving load balancing and also execute checkpoints for achieving failover successfully. 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 depend 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. Maximum Execution Time(MaxET), Minimum Execution time(MinET), Maximum Waiting Time(MaxWT) and Minimum Waiting Time(MinWT) are different metrics used in this research paper for performance comparisons. However some other parameters are also considered for comparing developed simulator and existing methods.

II. Problem Definition

This research work focus on providing high availability in cloud environment by using failover strategies for cloud data centers. Main emphasis is to use the methods which achieve high availability by using integration of checkpointing and load balancing algorithms. However to decrease checkpointing overheads multilevel checkpointing [9], [10], [11], [12], [13], [14], [15] is also used. Checkpointing has been used by many researchers but it may result, delay in execution

[1]Nodes and Service nodes are considered as synonym in this paper.

978-0-7695-4692-6/12 $26.00 © 2012 IEEE
DOI 10.1109/CSNT.2012.155

time as node sharing is achieved by using random decisions. In random decisions, the load balancing is not taken into consideration which might result into transfer of the load of crashed node to already heavy loaded nodes than lightly loaded nodes. To overcome this problem, in this research work checkpointing has been integrated with load balancing algorithms.

III. Literature Review

Availability [6] is a reoccurring and a growing concern in software intensive systems. Cloud systems services can be turned offline due to conservation, power outages or possible denial of service invasions. Checkpoint [1], [2] 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.

There exist many models to describe checkpoint systems implementation. Some of the models use multilevel checkpointing approach [9], [10], [11]. 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 [12]. Incremental checkpoints reduce the number of full checkpoints taken by periodically saving changes in the application data [13], [14], [15]. These approaches are orthogonal to multilevel checkpoints and can be used in combination with our work. The checkpoint and rollback technique [4] has been widely used in distributed systems. High availability can be offered by using it and suitable failover algorithms.

The ZEUS [5] Company develops software that can let the cloud provider easily and cost-effectively offer every customer a dedicated application delivery solution. The ZXTM [4],[5] 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 [7] 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 [8] 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 [4], [5], Apache Hadoop [7] and JPPF [8] not provide feature of checkpoints.

IV. Life cycle of parallel jobs

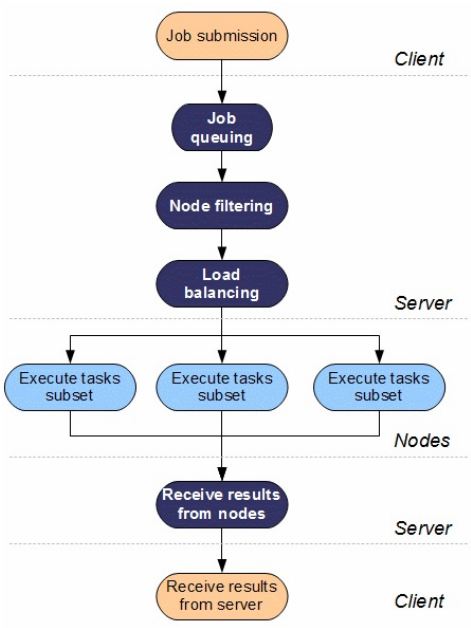


Figure 1. Life cycle of Parallel jobs (adapted from [8])

Fig. 1 is showing the life cycle of parallel jobs in distributed environment. Firstly clients submits their jobs for execution, if no subcloud is free then jobs are queued after this phase node filtering will be done that will detect the currently active nodes by checking the status of all nodes. After node filtering load balancing algorithm will come into action to balance the load of the given jobs among active nodes. It also shows that the main source of parallelism is provided by the load balancer, whose role is to split each job into multiple subsets that can be executed on multiple nodes in parallel. After successful execution of threads, each node send its final results back to the sub_cloud, then subcloud conquer the results of threads and the output will be passed to the client.

V. Proposed Failover Strategy

In order to achieve high availability for cloud computing using checkpoints based load balancing algorithms, two algorithms has purposed in this research work. Checkpoints based load balancing is defined as the feasible allocation or distribution of the work to highly suitable nodes so that execution time of the job could be minimized. This section discusses the procedure that how checkpoints based load balancing algorithms works and later on how proposed integrated checkpointing algorithms will provide high availability to the requests of the clients. Fig. 2 is showing the

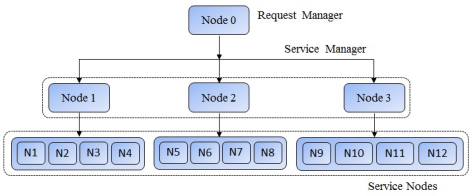


Figure 2. 3 Tier Architecture

three tier architecture for cloud environment. Fig. 2 has shown that there is a request manager (central cloud), clients send their requests to it all other nodes and their connectivity not deal directly with the clients. Thus request manager allow clients to submit their jobs. Then request manager first divide the given job into threads and also allocate one of the subcloud (service manager) to the threads and global checkpoint is also updated. Each subcloud first selects threads in First in First Out (FIFO) fashion and allocate lightly loaded service node to it. The service nodes 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.

A. *Proposed load balancing algorithms*

Proposed load balancing algorithms are developed considering main characteristics like reliability, high availability, performance, throughput, and resource utilization. However to fullfill these requirements of failover strategies, in Fig. 3 and Fig. 4 two different flowcharts named as global flowchart and local flowchart are shown. To decrease checkpointing overheads by using multilevel chekpointing [9], [10], [11], [12], [13], [14], [15], two different algorithms are used in this research work.

The flowchart of global checkpointing algorithm is shown in Fig. 3 that shows how global algorithm will works? It will take the following steps to assign the subcloud to the requests of the clients:

Step 1: Firstly clients submits their jobs to the CSP that is at central cloud

Step 2: CSP divide the jobs into threads and then allocate a minimum loaded subcloud to the jobs.

Step 3: After allocation of the sub_cloud, global checkpoint will be updated.

Step 4: Global checkpoint will run periodically.

Step 5: By reading checkpoint CSP will check whether any subcloud has failed or no failure occur. If no failure occur then a new save-point will be created and global checkpoint will be updated.

Step 6: If failure is found then work will be migrated from failed node to failed node's secondary node and global checkpoint will be updated.

Fig. 4 is showing the flowchart of local checkpointing

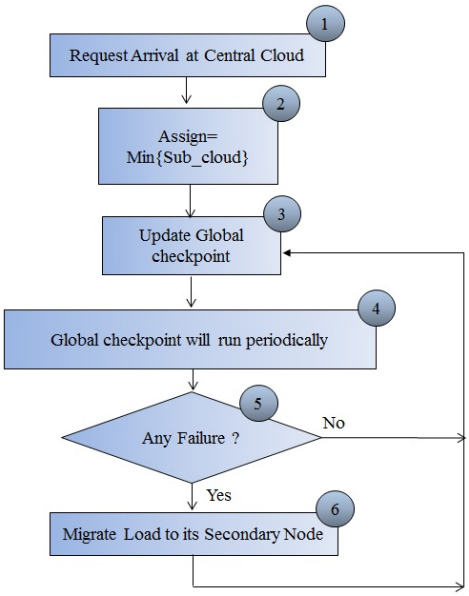


Figure 3. Global Checkpointing Algorithm's Flowchart

algorithm, which will work on sub_cloud. This algorithm will applied on sub_clouds and also nodes attached to it. It will take the following steps to allocate the nodes to the threads:

Step 1: Firstly threads will arrive on the sub_cloud.

Step 2: Then subcloud will check that whether any node is active or not? If no node is active then CSP will be notified by a message that "Subcloud is not responding"

Step 3: Then subcloud allocate minimum loaded nodes to the threads in such a way that load remain balance on the nodes.

Step 4: Local checkpoint will be updated.

Step 5: Global checkpoint will run periodically and a new save-point will be created every time .

Step 6: By reading checkpoint CSP will check whether any node has found to be failed or any node has recovered from failure.

Step 7: If any node found to be failed then subcloud will shift that node's load to the currently active nodes in such a way that load remain balance on active nodes and local checkpoint will be updated.

Step 8: If any node has been recovered then it will take load of some of other nodes which are heavy loaded and local checkpoint will be updated.

VI. Experimental Set-up

In order to implement the purposed failover strategy a suitable experimental set-up has been made as shown in Fig. 5. It take following steps to execute the jobs of the clients:

Step 1: Firstly clients submits their requests to the CSP via internet.

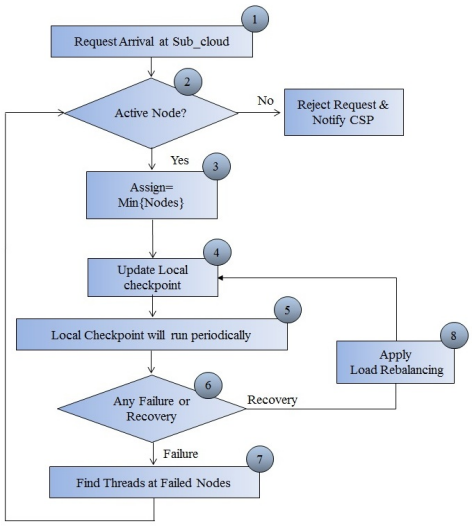


Figure 4. Local Checkpointing Algorithm's Flowchart

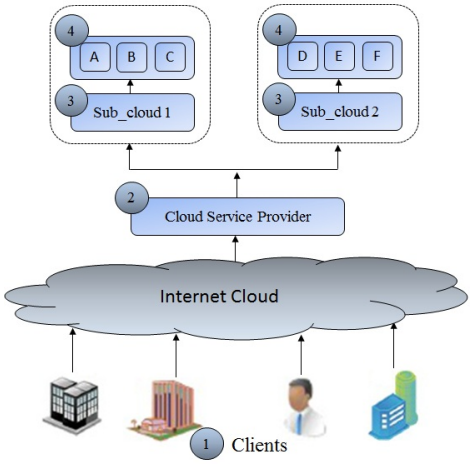


Figure 5. Simulator Environment

Step 2: CSP then allocate one of the subcloud to the given job and also divide the jobs into the threads and global checkpoint will be updated.

Step 3: After Step 2 local algorithm come in action. Each subcloud first selects threads in FIFO fashion and allocate lightly loaded node to it.

Step 4: Then node start execution of the inputed thread or it may add this thread into its waiting queue, if it is already doing execution of any other thread and local checkpoint will be updated.

VII. 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 TO THE SIMULATOR

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

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. 6. Fig. 6 giving detail of the global checkpoint, which is showing that which job is going to be run on which subcloud and also other relevant information like entered time of job, number of processors required, serial time, thread time etc.

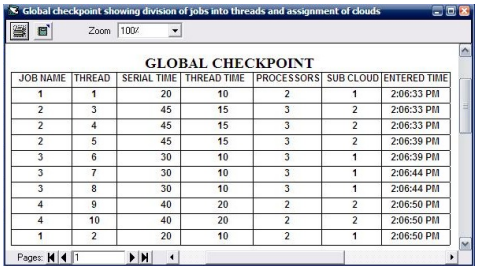


Figure 6. Global Checkpoint

B. Local checkpoint

Fig. 7 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 deactive), job_status(executing, waiting or finished), server name and also remaining time of threads(execution time + waiting time) etc.

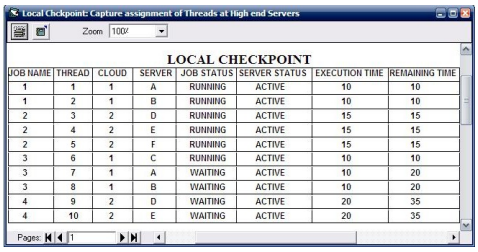
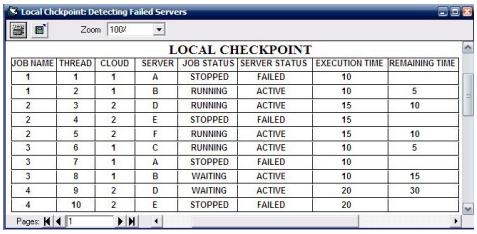


Figure 7. Local checkpoint

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. 8 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



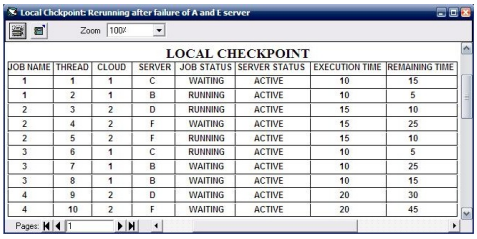
JOB NAME	THREAD	CLOUD	SERVER	JOB STATUS	SERVER STATUS	EXECUTION TIME	REMAINING TIME
1	1	1	A	STOPPED	FAILED	10	
1	2	1	B	RUNNING	ACTIVE	10	5
2	3	2	D	RUNNING	ACTIVE	15	10
2	4	2	E	STOPPED	FAILED	15	
2	5	2	F	RUNNING	ACTIVE	15	10
3	6	1	C	RUNNING	ACTIVE	10	5
3	7	1	A	STOPPED	FAILED	10	
3	8	1	B	WAITING	ACTIVE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

Figure 8. Local checkpoint showing Failed nodes

any node get failed and recovered before checkpoints will rerun then the execution at that nodes remains continue without any problem.

D. Load rebalancing after Node Failure

GUI will work in such a way that if any node get failed then CSP detect it with the help of checkpoints. Then CSP share the load of failed nodes among the active nodes. In Fig. 9 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.

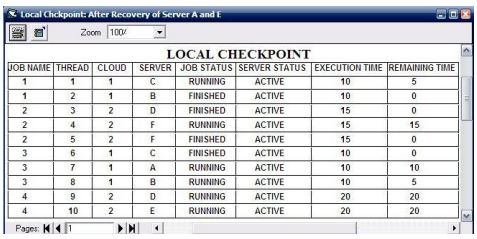


JOB NAME	THREAD	CLOUD	SERVER	JOB STATUS	SERVER STATUS	EXECUTION TIME	REMAINING TIME
1	1	1	C	WAITING	ACTIVE	10	15
1	2	1	B	RUNNING	ACTIVE	10	5
2	3	2	D	RUNNING	ACTIVE	15	10
2	4	2	F	WAITING	ACTIVE	15	25
2	5	2	F	RUNNING	ACTIVE	15	10
3	6	1	C	RUNNING	ACTIVE	10	5
3	7	1	B	WAITING	ACTIVE	10	25
3	8	1	B	WAITING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

Figure 9. 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. 10 it has been shown that the node A and E has recovered and they have taken some load from other heavy loaded nodes.



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

Figure 10. Rebalancing of load among recovered nodes

F. 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.

VIII. 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, Chekcpointing and developed simulator. Table II has shown

Table II
Feature's comparison with existing method

Feature	JPPF/Hadoop	Checkpoint	Purposed Method
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
Utilization of resources	Low	Medium	Maximum

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 nodes. The problem of checkpointing technique [2], [4] without load balancing algorithms also proved to be inefficient as migration of threads is done using random decisions and also not use multilevel checkpointing which may result in overheads.

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. Fig. 11 is showing the graph of four different metrics, which are shown in Table

Table III
METRIC'S COMPARISON WITH DIFFERENT APPROACHES

Type of Metric	No checkpoint	Checkpoint	Integrated
Maximum Execution time	55	45	35
Minimum Execution time	15	10	10
Maximum Waiting Time	35	25	15
Minimum Waiting Time	5	0	0

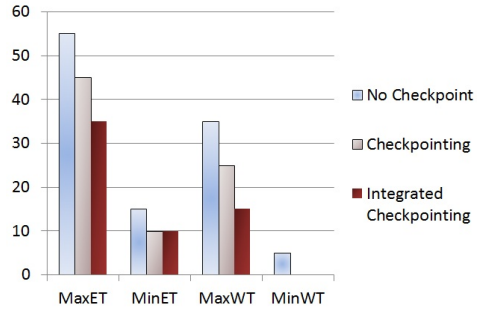


Figure 11. Metric's comparison with existing method graphically

III. Fig. 11 shows the difference between different methods graphically and it is clearly shown that the purposed method gives better results than existing methods.

IX. CONCLUSION AND FUTURE DIRECTIONS

This paper proposes a smart failover strategy for cloud computing using integrated checkpointing algorithms, which include the support of load balancing algorithms and multi-level checkpointing. A simulator environment has been developed that implement the purposed method. Performance comparison of existing methods has been made with the purposed method. It has been concluded with the help of performance metric's comparison that the proposed failover strategy gives good results than existing methods.
In this paper homogeneous nodes has been considered for simulation environment, in future work heterogeneous nodes will be used for better results.

REFERENCES

[1] Y. J. Wen, S. D. Wang, "Minimizing Migration on Grid Environments: An Experience on Sun Grid Engine," National Taiwan University, Taipei, Taiwan Journal of Information Technology and Applications, March, 2007, pp. 297-230.

[2] S. Kalaiselvi, "A Survey of Check-Pointing Algorithms for Parallel and Distributed Computers," Supercomputer Education and Research Centre (SERC), Indian Institute of Science, Bangalore V Rajaraman Jawaharlal Nehru Centre for Advanced Scientific Research, Indian Institute of Science Campus, Bangalore Oct. 2000,pp. 489-510, [Online]. Available: www.ias.ac.in/sadhana/Pdf2000Oct/Pe838.pdf.

[3] Reese, G., "Cloud Application Architectures: Building Applications and Infrastructure in the cloud (Theory in Practice)", O'Reilly Media, 1st Ed., 2009 pp 30-46.

[4] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," IEEE Transactions on Software Engineering, vol. 13, no. 1, pp. 23-31, 1987.

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

[6] K. Stanoevska-Slabeva, T. W. S. Ristol, "Grid and cloud Computing and Applications, A Business Perspective on Technology," 1st Ed., pp. 23-97, 2004.

[7] "What Is Apache Hadoop?,"[Last Published:] 12/28/2011 02:56:30, [Online]. Available: http://hadoop.apache.org.

[8] "JPPF Work distribution,"[Last Released] 1/31/2012, [Online]. Available: http://www.jppf.org.

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

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

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

[12] A. J. Oliner, L. Rudolph, and R. K. Sahoo, "Cooperative Checkpointing: A Robust Approach to Large-Scale Systems Reliability," in ICS 06: Proceedings of the 20th Annual International Conference on Supercomputing, 2006, pp. 14-23.

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

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

[15] N. Naksinehaboon, Y. Liu, C. B. Leangsuksun, R. Nassar, M. Paun, and 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 and the Grid (CCGRID), 2008, pp. 783-788.

[16] J. D. Sloan, High Performance Linux Clusters With Oscar, Rocks, OpenMosix and Mpi, O'Reilly, Nov.2004, ISBN 10: 0-596-00570-9 / ISBN 13: 9780596005702, pp. 2-3, [Online]. Available: gec.di.uminho.pt/discip/minf/cpd0910/PAC/livro-hpl-cluster.pdf.