I.J. Information Engineering and Electronic Business, **2012, 3, 1-8**
Published Online July 2012 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijieeb.2012.03.01

Failures in Cloud Computing Data Centers in 3-tier Cloud Architecture

Dilbag Singh, Jaswinder Singh, Amit Chhabra
Department of Computer Science & Engineering, Guru Nanak Dev University Amritsar, Punjab，143001, India
dggill2@gmail.com, chhabra.amit78@gmail.com, jaswindersingh@yahoo.com

***Abstract*—** This paper presents an methodology for providing high availability to the demands of cloud's clients. To succeed this objective, failover approaches for cloud computing using combined checkpointing procedures with load balancing algorithms are purposed in this paper. Purposed methodology assimilate checkpointing feature with load balancing algorithms and also make multilevel barrier to diminution checkpointing overheads. For execution of purposed failover approaches, a cloud simulation environment is established, which the ability to provide high availability to clients in case of disaster/recovery of service nodes. Also in this paper comparison of developed simulator is made with existing approaches. The purposed failover strategy will work on application layer and provide highly availability for Platform as a Service (PaaS) feature of cloud computing.

***Index Terms*—** Failover, Load balancing, Node-recovery, Multilevel checkpointing, Restartation

I. INTRODUCTION

Cloud computing [1] 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. Service node failure can happen in the following situations:

1) Node loses connectivity: This may happen due to interconnectivity problems of nodes. The possible causes are communication media failure, port crashes etc.

2) Node is ceased: Hardware failure or any unexpected reason makes node be inactive.

To achieve failover one solution is to implement the concept of redundancy [2]. 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.

A checkpoint is a local state of a job saved on stable storage. By periodically executing the checkpointing, one can save the status of a process at consistent intervals [3], [4]. If there is a failure, one may resume computation from the earlier checkpoints, thereby, avoiding restating execution from the beginning. The process of restarting computation by rolling back to a consistent state is called rollback recovery. In cloud computing environment, since the nodes in the data centers do not share memory [5], [6] therefore it is required to migrate the load of failed node to other node in case of failure, 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. An integrated checkpointing algorithm implements in parallel with the essential 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 become 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.

II. PROBLEM DEFINITION

Checkpointing is a technique to reduce the loss of computation in the manifestation of failures. Main emphasis of this research is to use the approaches which attain high availability by using amalgamation of checkpointing and load balancing algorithms. However to decrease checkpointing overheads multilevel checkpointing [6], [7], [8], [9], [10], [11], [12] is also used. Checkpointing has been used by many researchers but it may result, delay in execution 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 assimilated with load balancing algorithms.

III. LITERATURE REVIEW

Availability [13] 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. Fundamentally, its role is to deter-mine 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.

Checkpoint [14], [15] 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 check-pointing approach [6], [7], [8]. 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 [9]. Incremental checkpoints reduce the number of full checkpoints taken by periodically saving changes in the application data [10], [11], [12]. These approaches are orthog-onal to multilevel checkpoints and can be used in combination with our work. The checkpoint and rollback technique [16] has been widely used in distributed systems. High availability can be offered by using it and suitable failover algorithms.

The ZEUS [17] Company develops software that can let the cloud provider easily and cost-effectively offer every customer a dedicated application delivery solution. The ZXTM [16], [17] 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 [18] 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 [19] 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 [16], [17], Apache Hadoop [18] and JPPF [19] not provide feature of checkpoints.

A Checkpointing overhead [20], [21], [22], [23] has 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 [20], [21], [22], [23], [24], [25], [26] present measurements of checkpoint latency and overhead for a few applications.

Several models [27], [28], [29], [30], [31] 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). Youngs model does not consider failures during checkpointing and recovery [29], while Dalys extension lead of Youngs model, a higher-order approximation, does [30]. In addition to consider-ing checkpointing overheads and MTTI, the model discussed in [28] includes sustainable I/O bandwidth as a parameter and uses Markov processes to model the optimal checkpoint interval. The model described in [31] uses useful work, i.e., computation that contributes to job completion, to measure system performance.

IV. LIFE CYCLE OF PARALLEL JOBS.

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.

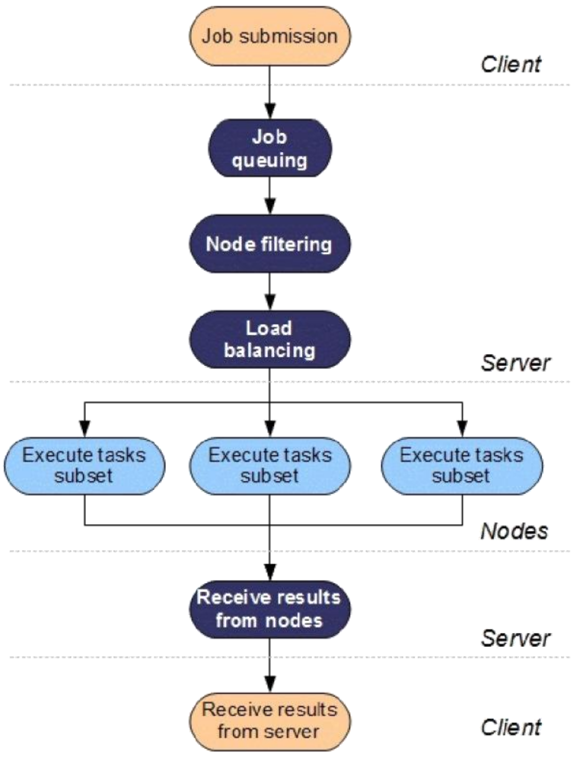


Fig. 1. Life cycle of Parallel jobs (adapted from [19])

V. PURPOSED FAILOVER STRATEGIES

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 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

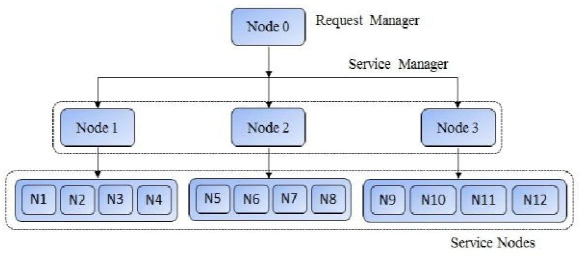


Fig. 2. 3 Tier Architecture

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.

Proposed load balancing algorithms

Proposed load balancing algorithms are developed considering main characteristics like reliability, high availability, performance, throughput, and resource utilization. However to fulfill 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 checkpointing [6], [7], [8], [9], [10], [11], [12], 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 work? 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 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 subcloud.

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 allocates minimum loaded nodes to the threads in such a way that load remains 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.

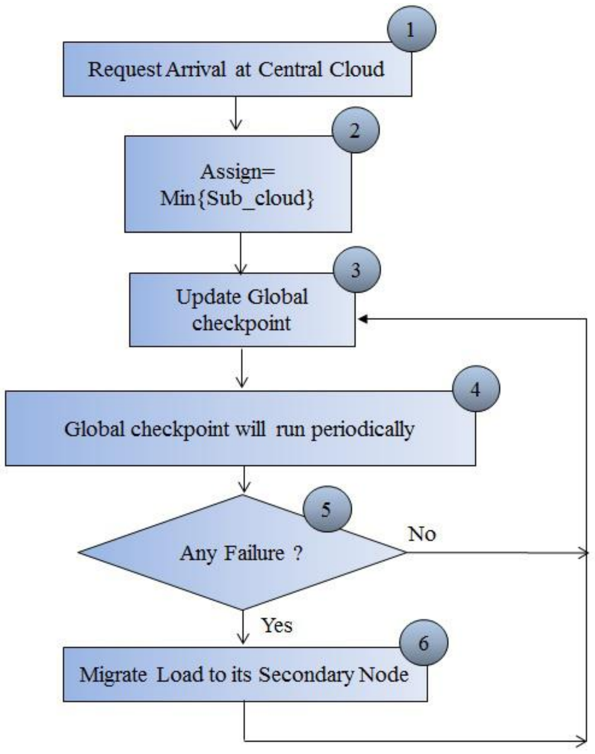

Request Arrival at Central Cloud
1
Assign= Min{Sub_cloud}
2
Update Global checkpoint
3
Global checkpoint will run periodically
4
Any Failure ?
5
No
Yes
Migrate Load to its Secondary Node
6

Fig. 3. Global Checkpointing Algorithm’s Flowchart

VI. EXPERIMENTAL SETUP

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

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

Step 2: CSP then allocate one of the subclouds to the Step 3: After Step 2 local algorithm come in action. Each subcloud paramount chooses threads in FIFO fashion and allocate lightly loaded node to it.

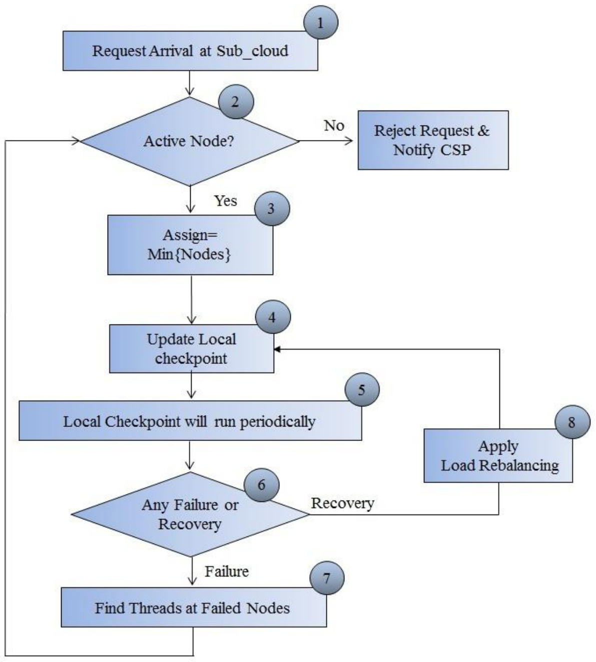


Fig. 4. Local Checkpointing Algorithm’s Flowchart

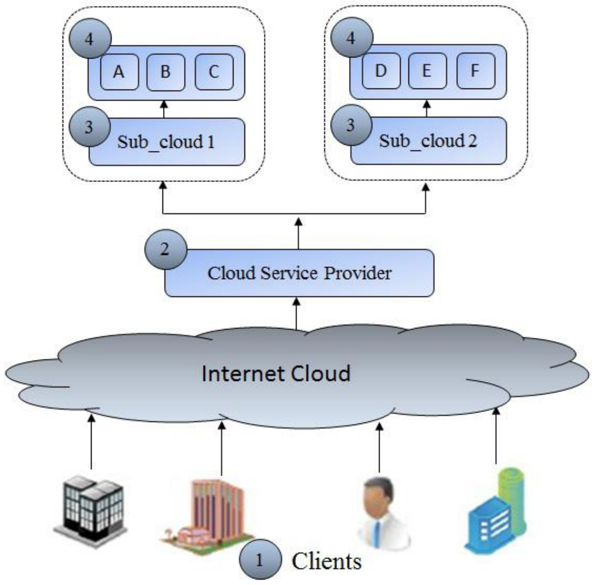


Fig. 5. Simulator Environment

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.

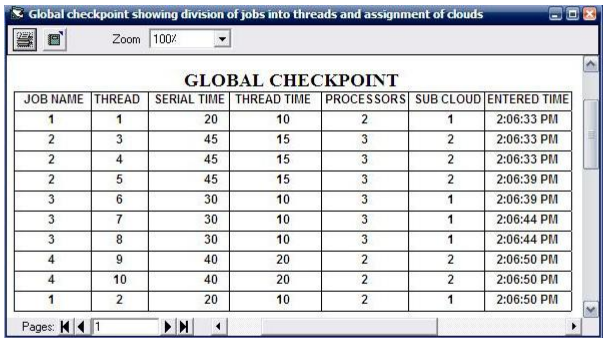
VII. SIMULATION RESULTS

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.

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

7.1 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.

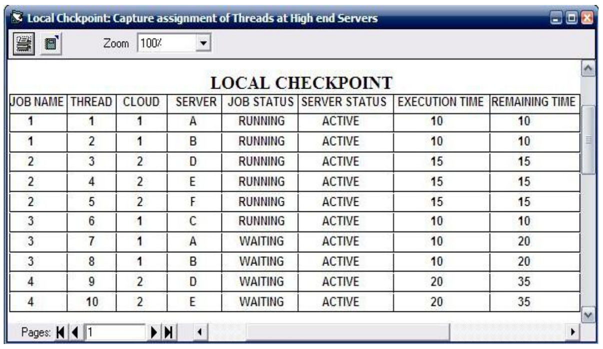

Global checkpoint showing division of jobs into threads and assignment of clouds
Zoom 100%
Pages: 1

GLOBAL CHECKPOINT

JOB NAME	THREAD	SERIAL TIME	THREAD TIME	PROCESSORS	SUB CLOUD	ENTERED TIME
1	1	20	10	2	1	2:06:33 PM
2	3	45	15	3	2	2:06:33 PM
2	4	45	15	3	2	2:06:33 PM
2	5	45	15	3	2	2:06:39 PM
3	6	30	10	3	1	2:06:39 PM
3	7	30	10	3	1	2:06:44 PM
3	8	30	10	3	1	2:06:44 PM
4	9	40	20	2	2	2:06:50 PM
4	10	40	20	2	2	2:06:50 PM
1	2	20	10	2	1	2:06:50 PM

Fig. 6. Global Checkpoint

7.2 Local checkpoint


Local Chckpoint: Capture assignment of Threads at High end Servers
Zoom 100%
Pages: 1

LOCAL CHECKPOINT

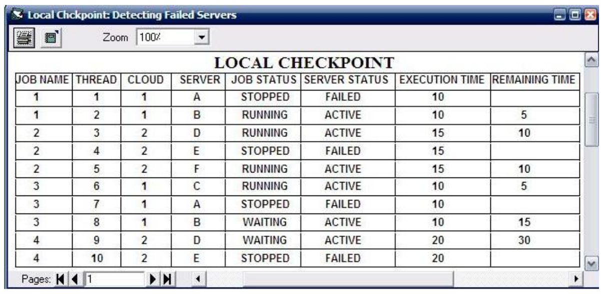
JOB NAME	THREAD	CLOUD	SERVER	JOB STATUS	SERVER STATUS	EXECUTION TIME	REMAINING TIME
1	1	1	A	RUNNING	ACTIVE	10	10
1	2	1	B	RUNNING	ACTIVE	10	10
2	3	2	D	RUNNING	ACTIVE	15	15
2	4	2	E	RUNNING	ACTIVE	15	15
2	5	2	F	RUNNING	ACTIVE	15	15
3	6	1	C	RUNNING	ACTIVE	10	10
3	7	1	A	WAITING	ACTIVE	10	20
3	8	1	B	WAITING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

Fig. 7. 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, wait-ing or finished), server name and also remaining time of threads(execution time + waiting time) etc.

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



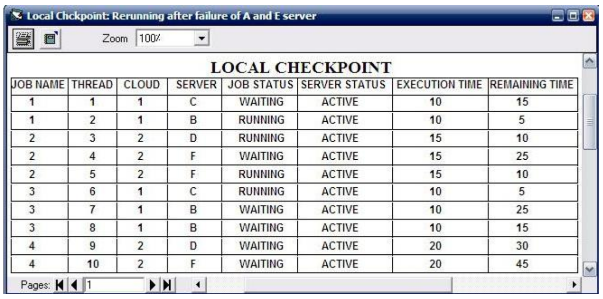
LOCAL CHECKPOINT

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	

Fig. 8. Local checkpoint showing Failed nodes

7.4 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.



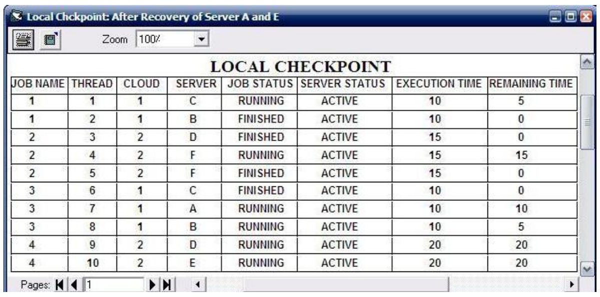
LOCAL CHECKPOINT

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

Fig. 9. Local checkpoint showing rebalancing of load

7.5 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.



Local Chckpoint: After Recovery of Server A and E

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

Fig. 10. Rebalancing of load among recovered nodes

7.6 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.

8.1 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 nodes. The

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	Low	Medium	Maximum

Table II. Feature's comparison with existing method

problem of checkpointing technique [15], [16] 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.

8.2 Comparison with no checkpoint, checkpoint without load balancing and purposed method Chp. Means Checkpointing

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

Table III. Metric's comparison of different approaches

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 checkpoint-ing 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 illustrates the graph of Average Waiting (AWT). In Fig. 11 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.

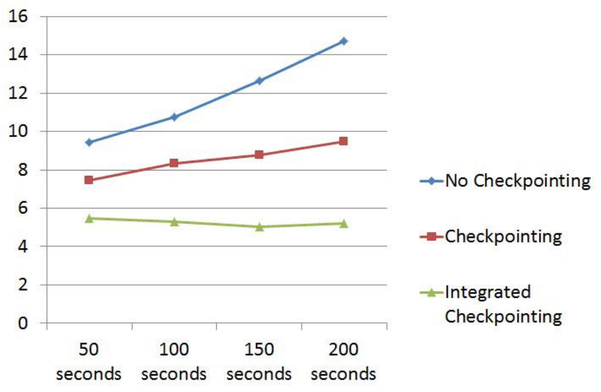


Fig. 11. Average Waiting Time comparison with existing methods

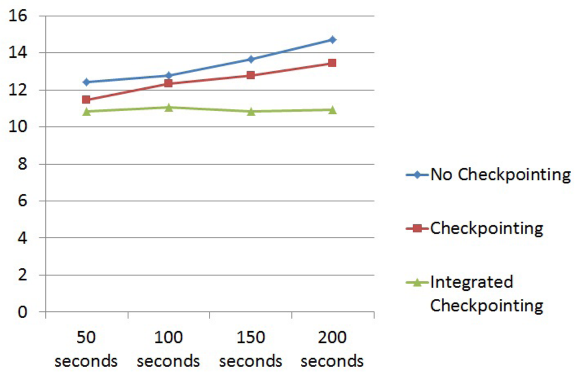


Fig. 12. Average Execution Time comparison with existing methods

Fig. 12 demonstrates the diagram of Average Execution Time (AET) metric. In Fig. 12 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 exhibits the diagram of Throughput (THP) metric. In Fig. 13 it has remained publicised that whether time augmentations, but disaster and recouping of nodes

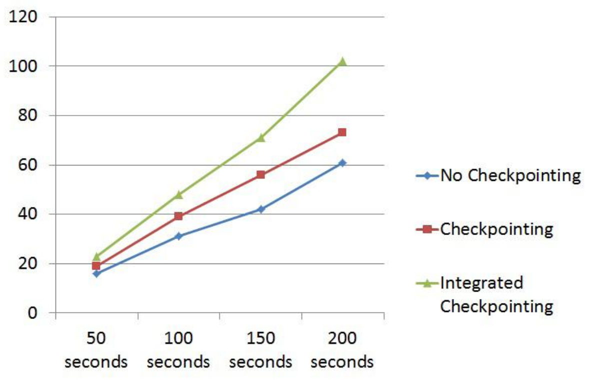


Fig. 13. Throughput comparison with existing methods

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 methodologys line continuously vacation subordinate than the supplementary techniques lines.

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

[2] 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.

[3] Alvisi, Lorenzo and Marzullo, Keith," Message Logging: Pessimistic, Optimistic, Causal, and Optimal," IEEE Transactions on Software En-gineering, Vol. 24, No. 2, February 1998, pp. 149-159.

[4] L. Alvisi, B. Hoppe, K. Marzullo, "Nonblocking and Orphan-Free mes-sage Logging Protocol," Proc. of 23rd Fault Tolerant Computing Symp., pp. 145-154, June 1993.

[5] A. Agbaria, W. H Sanders, "Distributed Snapshots for Mobile Computing Systems," IEEE Intl. Conf. PERCOM04, pp. 1-10, 2004.

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

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

[8] 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.

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

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

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.
[16] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for dis-tributed systems," IEEE Transactions on Software Engineering, vol. 13, no. 1, pp. 23-31, 1987.
[17] "ZXTM for cloud Hosting Providers," Jan. 2010, [Online]. Available: http://www.zeus.com/cloud-computing/for-cloud-providers.html.
[18] "What Is Apache Hadoop?,"[Last Published:] 12/28/2011 02:56:30, [Online]. Available: http://hadoop.apache.org.
[19] "JPPF Work distribution,"[Last Released] 1/31/2012, [Online]. Avail-able: http://www.jppf.org.
[20] 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.
[21] P. Kumar, "A Low-Cost Hybrid Coordinated Checkpointing Protocol for mobile distributed systems," Mobile Information Systems. pp 13-32, Vol. 4, No. 1, 2007.
[22] L. Kumar, P. Kumar, "A Synchronous Checkpointing Protocol for Mo-bile Distributed Systems: Probabilistic Approach," International Journal of Information and Computer Security, Vol.1, No.3 pp 298-314.
[23] 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).
[24] G. Cao , M. Singhal , "On coordinated checkpointing in Distributed Systems," IEEE Transactions on Parallel and Distributed Systems, vol. 9, no.12, pp. 1213-1225, Dec 1998.
[25] G. Cao , M. Singhal, "On the Impossibility of Minprocess Non-blocking Checkpointing and an Efficient Checkpointing Algorithm for Mobile Computing Systems," Proceedings of International Conference on Parallel Processing, pp. 37-44, August 1998.
[26] G. Cao , M. Singhal, "Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing systems," IEEE Transaction On Parallel and Distributed Systems, vol. 12, no. 2, pp. 157-172, February 2001.
[27] Nitin H. Vaidya, "On Checkpoint Latency," Department of Com-puter Science, Texas A& M University College Station, TX 77843-3112, Technical Report 95-015, March 1995, [Online]. Available: cite-seerx.ist.psu.edu.
[28] R. Subramaniyan, R. Scott Studham, and E. Grobelny, "Optimization of checkpointingrelated I/O for high-performance parallel and distributed computing," In Proceedings of The International Conference on Parallel and Distributed Processing Techniques and Applications, pp 937943, 2006.
[29] John W. Young, "A first order approximation to the optimum checkpoint interval," Communications of the ACM, 17(9):530531, 1974.
[30] J. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," Future Generation Computer Systems, pp 303312, 2006.
[31] K. Pattabiraman, C. Vick, and AlanWood, "Modeling coordinated check-pointing for large-scale supercomputers," In Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN05), pp 812821, Washington, DC, 2005. IEEE Computer Society.

Dilbag Singh is a student of Department in Computer Science and Engineering, Guru Nanak Dev University, Amritsar Punjab India. He completed his master degrees in computerl science in 2010 at Guru Nanak Dev University, Amritsar Punjab. Now he is M.tech student and going to complete his M. tech in June 2012. His research interests include Parallel computing, software structure, embedded system, object detection, identification, location sensing and tracking.

Amit Chhabra is a associate professor in the Department of Computer Science and Engineering, Guru Nanak Dev University, Amritsar Punjab India. He has done M.tech in IT and now perusing PHD in Cloud Computing from Guru Nanak Dev University, Amritsar Punjab. His research interests include Parallel and Distributed computing.

Jaswinder Singh is a associate professor in the Department of Computer Science and Engineering, Guru Nanak Dev University, Amritsar Punjab India. He has done MCA and now perusing PHD from Guru Nanak Dev University, Amritsar Punjab. His research interests include Theory of Computer Science and Software engineering.