I. J. Computer Network and Information Security, 2012, 7, 1-11
Published Online July 2012 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2012.07.01

UML Based Integrated Multilevel Checkpointing Algorithms for Cloud Computing Environment

Dilbag Singh, Jaswinder Singh, Amit Chhabra
Dept. of Computer Science & Engineering, Guru Nanak Dev University Amritsar, Punjab,143001, India
Dggill2@gmail.com, chhabra.amit78@gmail.com,jaswindersingh@yahoo.com

Abstract — Main objective of this research work is to improve the checkpoint efficiency for integrated multilevel checkpointing algorithms 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.

Index Terms — 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 Unified Modeling Language (UML) has been verified using StarUML. This paper also extended for more systematic performance comparison using checkpoint latency and checkpoint ratio.

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 purposed 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.

In order to verify UML diagram of cloud computing environment StarUML software is used. For implementation of purposed 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 purposed is made with existing methods.

II. PROBLEM DEFINITION AND THE SCOPE

2.1 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 [12], [13], [14], [15] is also used.

2.2 The scope of this research work

i. This research work deals with load distribution and high availability for cloud computing environment.

ii. It does not deal with management and security issue of cloud computing environment.
iii. As checkpointing means loss of computation therefore in this research work checkpointing overheads and checkpointing latency is also considered.
iv. 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.
v. Different type of tests will be implemented using integrated multilevel load balancing algorithm to test various aspects of the cloud environment.
vi. Visualization of the experimental results and drawing appropriate performance analysis.
vii. Appropriate conclusion will be made based upon performance analysis.
viii. For future work suitable future directions will be drawn considering limitations of existing work.

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

III. RELATED WORK

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. 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.

Checkpoint [4], [5] 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 [8] Company develops software that can let the cloud provider easily and cost-effectively offer every customer a dedicated application delivery solution. The ZXTM [7],[8] 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 [10] 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 [11] 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 [7], [8], Apache Hadoop [10] and JPPF [11] not provide feature of checkpoints.

Checkpointing overheads [20], [21], [22], [23] 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 [20], [21], [22], [23], [24], [25], [26] 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 [29], while Daly's extension lead of Young's model, a higher-order approximation, does [30]. In addition to considering 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. CHECKPOINT LATENCY AND OVERHEAD

The checkpoint latency [27], [28] 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

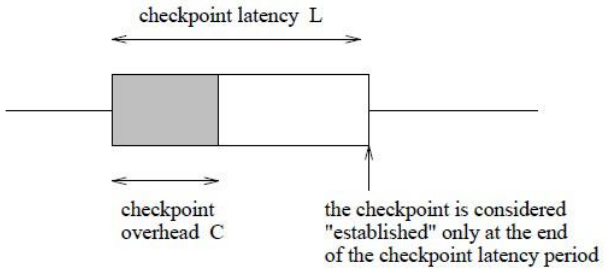

Fig. 1: Modelling checkpoint latency and overhead (adapted from [27])

Fig. 1. 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.

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.

4.1 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.

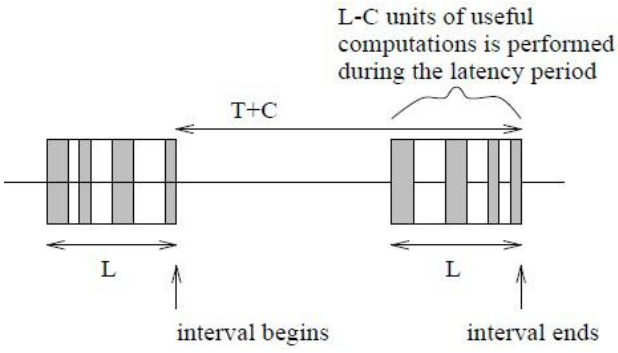

Fig. 2: If no failure will occur during checkpoint latency (adapted from [27])

4.2 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.

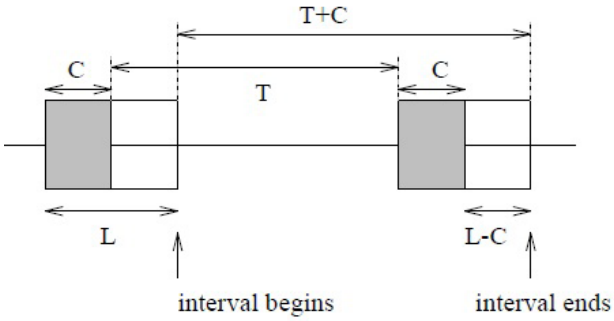

Fig. 3:If failure occur during checkpoint latency (adapted from [27])

Now consider Fig. 3, 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.

V. 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 (mean) 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:

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

Fig. 4: Checkpoint overhead ratio (adapted from [27])

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:-

$$r= (TET + t1(C) + t2(R))/t$$

VI. PROBLEMS IN EXISTING TECHNIQUES

6.1 Three-node architecture

Figure Fig. 6demonstrate the 3-tier architecture of cloud computing environment. In Fig. 5 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.

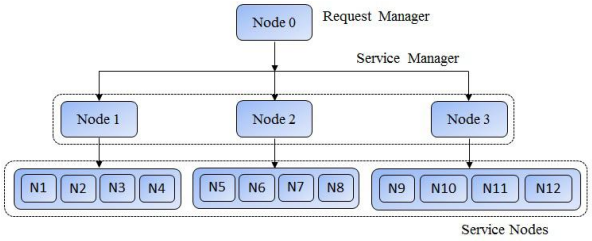

Fig 5: 3-node architecture.

6.2 Sub cloud failure

Fig. 6 illustrates the sub cloud failure. In Figure Fig. 6 sub cloud 2 has been failed. Sub cloud 2 failed 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.

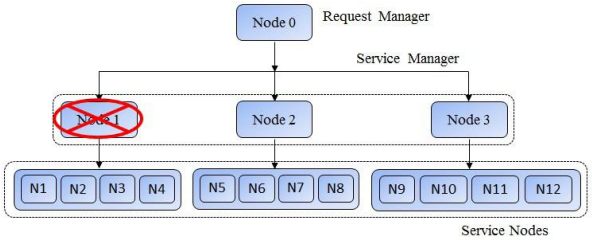

Fig. 6: Sub cloud failure.

6.3 Service node failure

Fig. 7 illustrates the service node failure. In Fig. 7 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.

6.4 Job restartation due to node failure

Fig. 8 is illustrating the parallel job execution in distributed environment. Initially clients submits their jobs for execution, if no sub cloud 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 Fork and Join concept, 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 sub cloud conquer the results of threads using Join concept and the output will be passed to the client.

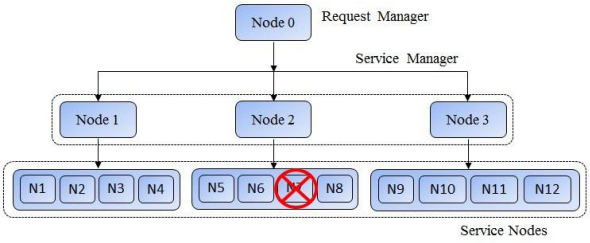

Fig. 8: Service node failure.

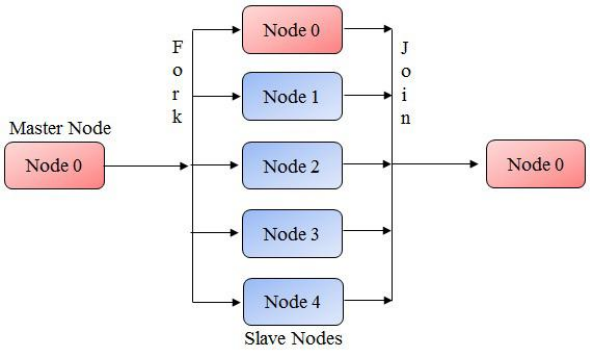

Fig.9: Parallel job execution in distributed environment.

6.5 Job restartation

In Fig. 10, node 2 has been failed. As existing methods do not use checkpointing so it is not feasible to migrate the load of failed node (node 2) to other active nodes. The major problem is that if other nodes successfully accomplish the execution of the threads belong to same job, will be wasted as no log files are used. Therefore job restartation will be there. In Fig. 10, failure of node 2 will waste the successful execution of the other nodes. Therefore a single node failure results in too much loss of computation. So this is a major draw back of existing methods.

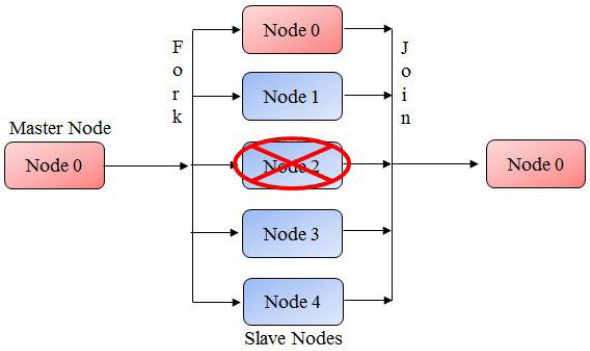

Fig.10: Job restartation.

VII. PROPOSED LOAD BALANCING ALGORITHMS TO ACHIEVE MULTILEVEL CHECKPOINTING

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. 11 and Fig. 12 two different flowcharts named as global and local flowchart are shown. To decrease checkpointing overheads by using multilevel checkpointing two different algorithms are used in this research work.

The procedure for global checkpoint algorithm is shown in Fig. 11 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 submit 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 will 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.

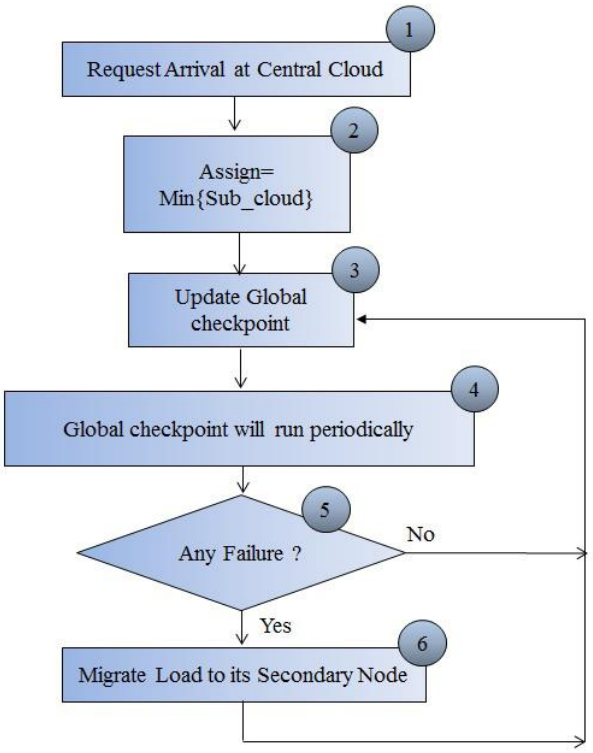

Fig. 11: Global Checkpointing Algorithm's Flowchart

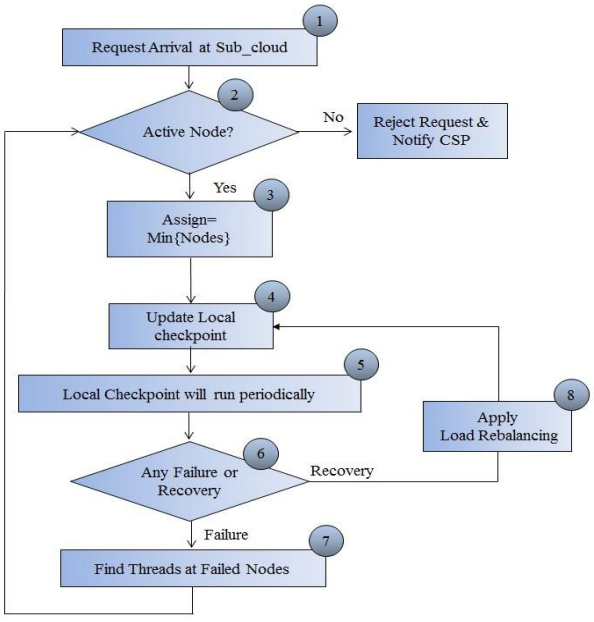

Fig. 12: Local Checkpointing Algorithm's Flowchart

Fig. 12 is showing the phases 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.

VIII. UML VERIFICATION USING STARUML

In order to verify UML diagrams StarUML software is used some of its outputs are:

8.1 Class diagram

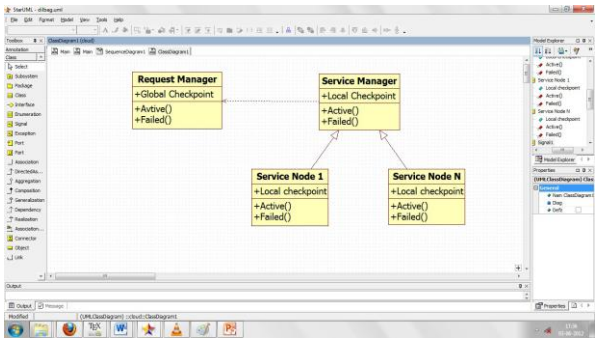

Fig. 13: Class diagram reflecting relationship among CSP, sub clouds and service nodes.

Describe the attributes and operations of a class (node) and also the constraints. It represents an abstract model of purposed method, consists of classes and their relationships. As shown in Fig. 13. In Fig. 13 class diagram is showing that sub cloud inherits the properties of nodes and also CSP inherits the properties of sub cloud. The main use of this inheritance is to check whether the giving node is active or inactive.

8.2 Sequence diagram

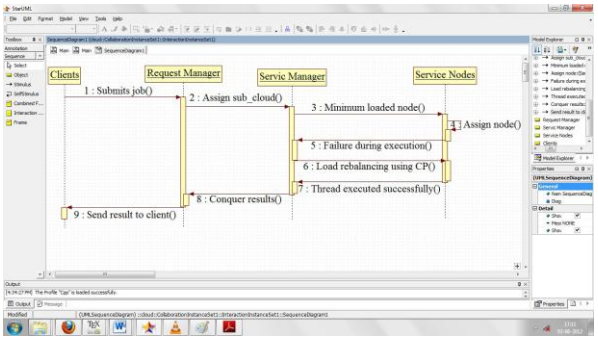

Fig. 14: Sequence diagram showing sequence of messages between objects.

It deals with some sequences, which are the sequence of messages flowing from one object to another. Fig. 14 has demonstrated that the message sequence for CSP object to sub cloud via nodes. Now understand the time sequence of message flows. Message flow is nothing but a method call of an object. The first call is to transfer workload which is a method of CSP and sub cloud at global and local level respectively. The next call can be Failed () or Recovered () and the last call is Finish () which is a method of confirming that the job has been executed successfully at nodes.

8.3 Statechart diagram

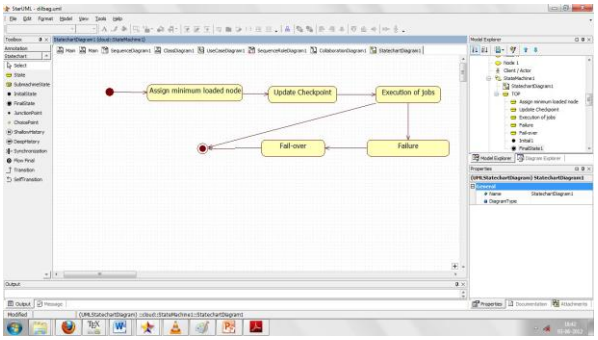

Fig. 15: Statechart diagram showing the flow of control between states in the simulator.

A statechart diagram describes a state machine. State machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events.

It describes the flow of control from one state to another state. Consider the statechart diagram of purposed method (Fig. 16) in which states are arrived for events like send request, confirm minimum_sub_cloud, and then assign minimum_server and start executing the job.

By setting up certain constraints in StarUML it has been proved that Fig. 13, Fig. 14, Fig. 15 is verified successfully.

IX. 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. Threads are generated on random bases

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

Table I: Inputs for the simulator.

9.1 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. 16. Fig. 16 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.

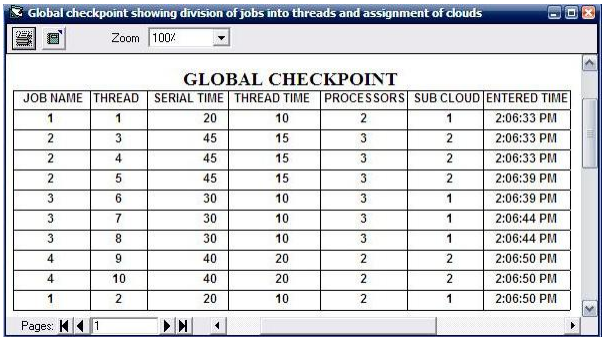

Fig. 13: Global Checkpoint

9.2 Local checkpoint

Fig. 14 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.

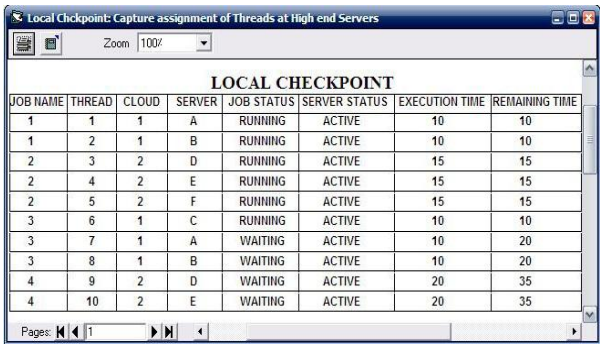

Fig. 14: Local checkpoint

9.3 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. 15 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.

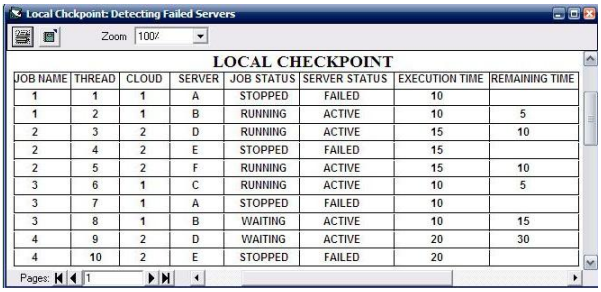

Fig.16. Local checkpoint showing Failed nodes

9.4 Load rebalancing after node failure

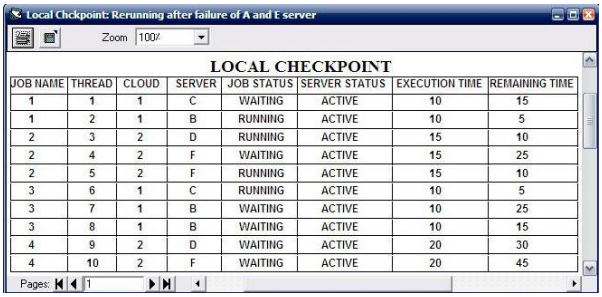

Fig. 18: Local checkpoint showing rebalancing of load

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. 18 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.

9.5 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. 19 it has been shown that the node A and E has recovered and they have taken some load from other heavy loaded nodes.

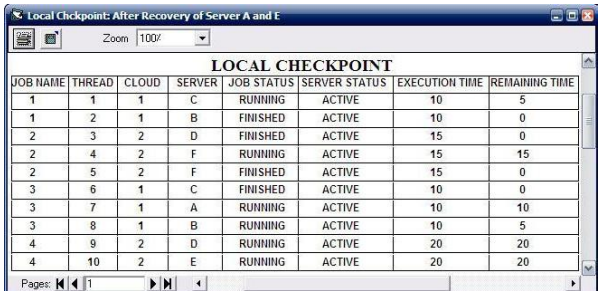

Fig. 19: Rebalancing of load among recovered nodes

9.6 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.

X. PERFORMANCE ANALYSIS

In order to do performance analysis, comparisons has been made in this research work based upon certain metrics. 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.

10.1 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.

Feature	Existing methods	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
Checkpoint overheads	No	More	Medium
Checkpoint latency	0	> 0	> 0
Checkpoint ratio	1	> 0	> 0

Table II. Feature's comparison with existing method

10.2 Comparison with no checkpoint, checkpoint without load balancing and purposed method

Type of Metric	No chp.	Chp.	Integrated
Mean Execution time	14.73	13.46	10.94
Minimum Execution time	15	10	10
Mean 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 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.

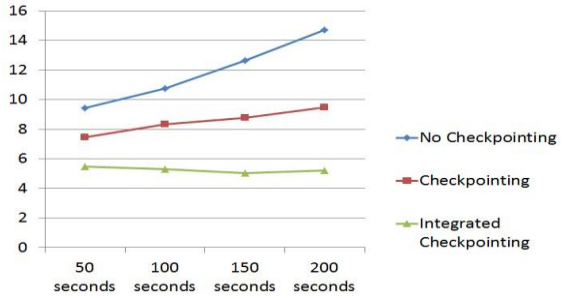

Fig. 20: Mean Waiting Time comparison with existing methods

Fig. 20 illustrates the graph of Mean Waiting Time (MWT). In Fig. 20 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 MWT of integrated approach always stay lower than the other existing methods lines.

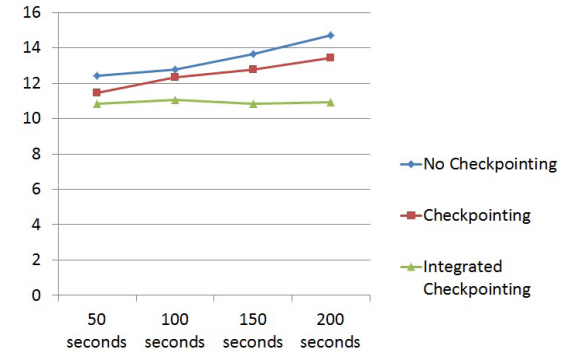

Fig. 21: Mean Execution Time comparison with existing methods

Fig. 21 demonstrates the diagram of Mean Execution Time (MET) metric. In Fig. 21 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 MET in incorporated checkpointing methodologies line continuously vacation subordinate than the supplementary techniques lines.

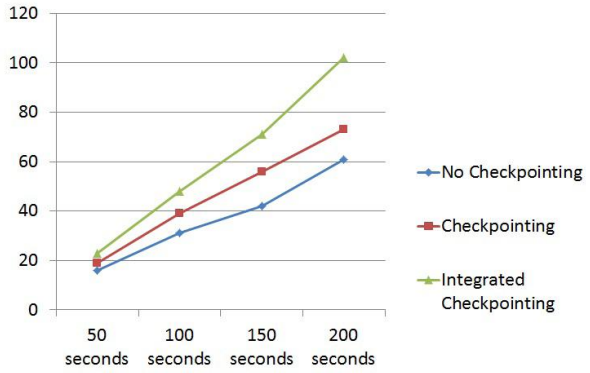

Fig. 22: Throughput comparison

Fig. 22 exhibits the diagram of Throughput (THP) metric. In Fig. 22 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.

10.3 Checkpointing overheads

Primary objective of this research work 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.

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

Table IV. Checkpointing overheads.

Fig. 23 has shown the values of checkpointing overheads considering several checkpointing intervals graphically. It has been significantly shown in Fig. 23 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.

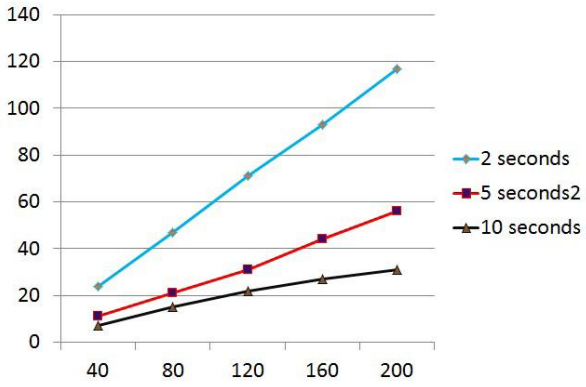

Fig. 23: Checkpointing overheads.

XI. CONCLUSION AND FUTURE DIRECTIONS

11.1 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 research work, 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 Mean completion time and computing the probability of meeting a given deadline.

11.2 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 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: Fail-over 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," Accepted for publication in International Journal of Computer Network and Information Security (IJCNIS, ISSN: 2074-9090, (Print), ISSN: 2074-9104 (Online).

[4] 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.

[5] 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

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

[7] 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.

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

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

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

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

[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.

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

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

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

[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 Mobile 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 Computer Science, Texas A& M University College Station, TX 77843-3112, Technical Report 95-015, March 1995, [Online]. Available: citeseerx. 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 checkpointing 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 Guru Nanak Dev University, Amritsar Punjab India in Department of Computer Science and Engineering. He has completed his master degrees in computer science in 2010 at Guru Nanak Dev University, Amritsar Punjab. Now he is going to complete his M. tech in June 2012. His research interests include Parallel computing, software structure, embedded system, object detection and identification, and location sensing and tracking.

Amit Chhabra is an 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 an 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.

