International Journal of Technical Research and Applications e-ISSN: 2320-8163,
www.ijtra.com Volume-2, Special Issue 1 (July-Aug 2014), PP. 51-56

APPLICATION LEVEL CHECKPOINT-BASED APPROACH FOR CRUSH FAILURE IN DISTRIBUTED SYSTEM

Moh Moh Khaing, Phyu Phyu Tar
Department of Information and Communication Technology
University of Technology (Yatanarpon Cyber City)
Mandalay Division, Myanmar

***Abstract*— Fault-tolerance is an important and critical issue in distributed and parallel processing system. Distributed system consists of a collection of interconnected stand-alone computers working together as a single, to produce complete result. If the numbers of computing nodes are increased concurrently and dynamically in distributed computing, it may have the many changes to become crush failures. In this paper, we propose application level checkpoint-based fault tolerance approach for distributed computing. The proposed system uses Coordinated Checkpointing techniques and Systematic Process Logging (SPL) as global monitoring mechanism. The proposed system implements on distributed multiple sequences alignment (MSA) application using genetic algorithm (GA).**

***Index Terms*— Coordinated Checkpointing, Genetic Algorithm (GA), Systematic Process Logging (SPL), Multiple Sequences Alignment (MSA)**

I. INTRODUCTION

A distributed system is a collection of autonomous computers linked by a network and equipped with software that makes it appear to its users as a single coherent system [12]. Distributed system facilitates sharing of resources among geographically separate users, improve performance and speed, and the participating systems, being heterogeneous can each use the best tools for the assigned tasks. Many computational nodes create problems with respect to reliability, availability and usability. The sources of the problems are node or crush failure for dynamic configuration over extensive runtime.

A crash or node failure occurs when a computing node prematurely halts, but was working correctly until it stopped. An important aspect with node failures is that once the computing node has halted, nothing is heard from it anymore. An omission failure occurs when a computing node fails to respond to a request. Several things might go wrong.

As the result of node failure, it may become more chances to appear receive and send omission failure. In the case of a receive omission failure, the computing node perhaps never got the request in the first place. It may be the case that the connection between a client and a server has been correctly established, but that there was no thread listening to incoming requests. Also, receive omission failure will generally not affect the current state of the node, as the node is unaware of any message sent to it. Likewise, send omission failure happens when the server has done its work, but somehow fails in sending a response [1].

Replication-based recovery approach, Fusion-based approach, Checkpoint–based library compiler and System level checkpointing approach can be used to tolerate crush failures. But these approaches are based on operating system level and place many replicas in distributed computing. Moreover, these approaches are high overhead, and less dynamic in distributed computing environments and that are rarely used in [2], [3], [4]. In this system, the proposed fault tolerance approach is implemented on the application layer without using any operating system supports. In this approach, all worker nodes (WNs) take local checkpoints at the state of process and store all content of that checkpoint status as the log file. Global checkpoint is taken upon the worker's connection by head node (HN). This approach can get the portable checkpoint facilities and equally load balancing mechanism. The proposed fault tolerance approach will be implemented on distributed multiple sequences alignment (MSA) application using genetic algorithm (GA).

This paper is organized as follows. Section II presents the previous related works of fault-tolerance in distributed system and parallel computing. Section III presents background theory of proposed system and Section IV describes overview of proposed system. Section V and section VI explains the two processing phase of proposed system. Section VII shows the implementation of proposed system in MSA application and section VIII concludes the paper and advantages of this proposed system.

II. RELATED WORKS

J.P. Walters [2] proposed replication-based fault tolerance for MPI application. However, related issues are consistency among such replicas and need to be addressed the overhead carefully. Major problem of this system is number of backups so that fusion based approach is appeared in. That is emerging as a popular technique to handle multiple faults but requires fewer backup machines than replication based approaches.

In fusion based fault tolerance technique [3], [4], backup machines are used which is cross products of original computing machines. But there is high overhead during failure recovery. Hence this technique is suitable if the percentage of fault is low.

S. Bansai [5] projected Dynamic Rank-Based fault tolerance algorithm for load redistribution works as a sequential restoration algorithm and reassignment algorithm for distribution of failure nodes to least loaded computing nodes works as a concurrent recovery reassignment algorithm.

M.Tripathy [6] planned Hierarchical Checkpointing that includes three types of checkpoint: Local disk, Mirrored, Storage and three types of recovery for transient failure, permanent failure and storage failure.

S.Kumar [7] anticipated that two phase coordinated checkpointing algorithm which has first phase is each process takes a tentative checkpoint and second phase is tentative checkpoint was replaced by the permanent one. It takes the waiting time both tentative checkpoint and permanent checkpoint.

III. BACKGROUND THEORY

A. Checkpointing Approach

Checkpointing technique has been used in distributed system as fault tolerance mechanism. A checkpoint is a snapshot of the current state of a process and saves enough information in non-volatile stable storage. If the volatile storage contents are lost due to process failure, one can reconstruct the process state from the information saved in the non-volatile stable storage.

Checkpoint-based technique can be divided into three subcategories: (1) Asynchronous or Uncoordinated checkpointing, (2) Synchronous or Coordinated checkpointing and (3) Quasi-synchronous or Communication induced checkpointing [8], [9], [10].

1) Uncoordinated Checkpointing: uncoordinated Checkpointing allows each process the maximum autonomy in deciding when to take checkpoints. The main advantage is that each process may take a checkpoint when it is most convenient. But there are several disadvantages. First, there is the possibility of the domino effect, which may cause the loss of a large amount of useful work, possibly all the way back to the beginning of the computation. Second, a process may take useless checkpoint that will never be part of a global consistent state. Useless checkpoints incur overhead and do not contribute to advancing the recovery line. Third, it forces each process to maintain multiple checkpoints, and to invoke periodically a garbage collection algorithm to regain the checkpoints that are no longer useful. Fourth, it is not suitable for applications with frequent output commits because it requires global coordination to compute the recovery line. In order to determine a consistent recovery line, all processes record their dependencies among checkpoints during failure-free operation.

2) Coordinated Checkpointing:

coordinated Checkpointing requires processes to orchestrate their checkpoints in order to form a consistent global state. Coordinated checkpointing simplifies recovery and is not susceptible to the domino effect, since every process always restarts from its most recent checkpoint. Also, it requires each process to maintain only one permanent checkpoint on stable storage, reducing storage overhead and eliminating the need for garbage collection. However, there is the large latency involved in committing output, since a global checkpoint is needed before messages can be sent to outside world. A straightforward approach to coordinated checkpointing is to block communications while the checkpointing protocol executes. A coordinator takes a checkpoint and broadcasts a request message to all processes, asking them to take a checkpoint. When a process receives this message, it stops its execution, flushes all the communication channels, takes a tentative checkpoint, and sends an acknowledgment message back to the coordinator. After the coordinator receives acknowledgments from all processes, it broadcasts a commit message that completes the two-phase checkpointing protocol. After receiving the commit message, each process removes the old permanent checkpoint and atomically makes the tentative checkpoint permanent.

3) Communication-Induced Checkpointing:

communication-induced checkpointing avoids the domino affect while allowing processes to take some of their checkpoints independently [13]. However, process independence is constrained to guarantee the eventual progress of the recovery line, and therefore processes may be forced to take additional checkpoints. The checkpoints that a process takes independently are called local checkpoints, while those that a process is forced to take are called forced checkpoints. Communication-induced checkpointing piggybacks protocol-related information on each application message. The receiver of each application message uses the piggybacked information to determine if it has to take a forced checkpoint to advance the global recovery line. The forced checkpoint must be taken before the application may process the contents of the message, possibly incurring high latency and overhead. It is therefore desirable in these systems to reduce the number of forced checkpoints to the extent possible. In contrast with coordinated checkpointing, no special coordination messages are exchanged in [11].

B. Systematic Process Logging

Systematic Process Logging (SPL) which was derived from a log-based method. The motivation for SPL is to reduce the amount of computation that can be lost, which is bound by the execution time of a single failed task. In case of a fault, task duplication needs to be avoided. Depending on the timing of the fault, this could result in a significant number of duplicated nodes, since each duplicated task itself may be the initiator of a significant portion of computation. In implementation of SPL, duplication avoidance is achieved using a unique and reproducible identification method of all vertices in the graph.

IV. OVERVIEW OF PROPOSED SYSTEM

A. Design

The proposed system architecture includes only one head node (HN) and (*1*....*3*) worker nodes (WNs). All nodes are connected on local network and they compute their jobs without sharing storage spaces and input/output streams. They execute their jobs independently. Fig.1. shows the distributed architecture of checkpoint based fault tolerance system.

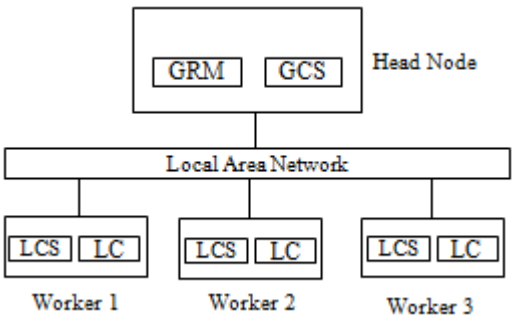

Fig.1. Overview Design for the proposed Fault-Tolerance System

In this proposed architecture, global resources monitor (GRM), and global checkpoint storage (GCS) are coded in head node (HN) and local checkpoint (LC) and local checkpoint storage (LCS) are coded in worker node (WN). HN creates the global checkpoint by taking the local checkpoint of each worker node. Global resource monitor (GRM) uses to manage load balancing. Thus, it always checks and stores the state and content of each worker node. GRM takes the responsibility for failure detection and load balancing from failed node to other running nodes. Workers' conditions are stored in log files as global checkpoint storage (GCS). In each

worker node, local checkpoint (LC) is placed and takes the checkpoint of every alignment process and its contents independently and periodically. Local checkpoint storage (LCS) uses to store each worker node's processing conditions.

B. System Flow

The proposed Checkpoint-Based Fault Tolerance System has two Phases: (1) Checkpointing Phase and (2) Load Balancing Phase. The system flow of the proposed system is shown in Fig. 2.

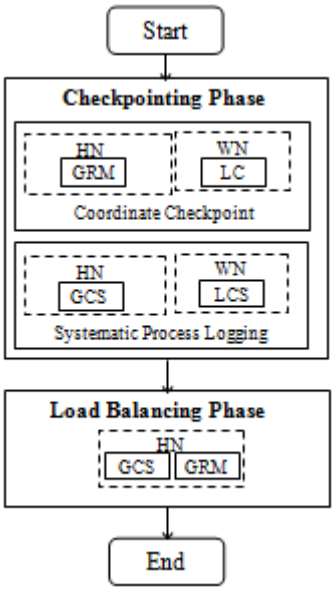

Fig. 2. System Flow for the proposed Fault-Tolerance System

V. CHECKPOINTING PHASE

The proposed application level checkpoint-based approach is applied at the application layer and does not use any other operating system supports. In Checkpointing phase, there are two process stages: Coordinated Checkpointing (CC) and Systematic Process Logging (SPL). In CC stage, head node (HN) performs global resource monitor (GRM) and worker node (WN) makes local checkpoint (LC). In SPL stage, HN takes global checkpoint storage and worker node acts local checkpoint storage.

In this system, in coordinated checkpointing (CC), initial, head node (HN) accepts the multiple input sequences. HN selects one sequence from multiple input sequences and the selected sequence is sent to each worker node. The selecting and sending processes are doing until all input sequences are done. Global resource monitor (GRM) from head node records all workers' connection and their processes. In systematic process logging (SPL), HN stores workers' conditions in log file. Moreover, according to store all monitoring information of workers in log file, global checkpoint storage (GCS) provides fault tolerance behavior when one worker is crushed. Thus, in GCS, log file stores the client number, IP address and port number, current status, current time and duration time of between processes for every worker.

Therefore, in CC, when worker node (WN) accepts the input sequence, local checkpoint (LC) records worker's contents. Then worker calculates to form the aligned sequence using genetic algorithm (GA). Every worker nodes (WNs) applies the GA to produce aligned sequence. The multiple sequences alignment (MSA) application using GA has four main steps and the input sequence must pass these four steps completely. The four steps are (i) Initial Population, (ii) Generating and Selection the best population by calculating the fitness of population (iii) Crossover and (iv) Mutation.

The WN first selects the number of initial population, number of generation, number of crossover and mutation and minimum number of gaps to be aligned sequence before starting sequence alignment process. Then, the WN creates the initial population using the given number of population size upon the input sequence. The next step is selecting the best population with best fitness and generates the new population using the generation number and minimum gaps. The third step is making the crossover operation of generated sequences using the input crossover number and makes the mutation operation using input mutation rate after crossover operation. After the input sequence had passed through the four genetic operators of MSA application, the aligned sequence is come out as result. This aligned sequence is sent to the HN. In SPL, each worker takes the local checkpoint (LC) and stores the contents of processing aligned sequence as log file. In local checkpoint storage (LCS), log file of WN stores the client number, IP address and port number, process state, and current time.

VI. LOAD BALANCING PHASE

Load Balancing Phase has the responsibilities for detecting node failure, making decision and transferring jobs to another available node. This phase proposes to solve above conditions using global monitoring information of all workers from head node (HN).

Global resource monitor (GRM) from head node records all workers' connection and their processes. Then, GCS stores all monitoring information of workers as records in log file. HN sends input sequence to each worker node at each time. When worker node is crushed, head node gets crush message and receives node failure report. The head node notices node failure condition. It finds available node and makes decision which node to transfer the job from crushed node using the global monitoring information.

When all aligned sequences are received from every worker nodes (WNs), HN combines the result and shows the aligned sequences result to user.

VII. IMPLEMENTATION

A. Input Sequences for MSA

Multiple sequences alignment (MSA) application uses DNA (Deoxyribonucleic Acid) sequences in medical field. The DNA sequences are presented as FASTA format which organizes the A, C, G, T nucleotides. These four kinds of nucleotides is the basic unit of an organism and stands for adenosine (A), cytosine(C), guanine (G) and thymine (T).

B. Multiple Sequences Alignment

Multiple sequences alignment (MSA) application helps to design new protein or genes and it belongs to a class of optimization problems with exponential time complexity, called combinatorial problems. MSA can align different kinds of genetic sequences such as DNA Sequences, Protein Sequences.

MSA application refers to the problem of optimally aligning two or more sequences with inserting gaps between the genes. The objective is to get number of matching gene symbols between the sequences according to use only minimum gap insertions. Identical sub sequences are

eventually aligned in a one-to-one correspondence naturally; gaps are not inserted in both sequences at the same position. In the end, the sequences end up with the same size.

In this system, using distributed MSA, multiple sequences (*1…n*) are accepted by head node (HN) as input and then each sequence is sends to each worker node (WN). This process is repeated until all input sequences are finished. All worker nodes (WNs) concurrently align the input sequence using genetic operators; generating population, selection, crossover and mutation with inserting minimum gaps. The aligned sequence result is sent by WN to HN and HN combines all aligned sequences to become the new completed aligned sequences and displays to the user. The following sequences show the input DNA sequences for MSA and its output aligned DNA sequences:

Input multiple DNA Sequences:

>seq1: AAGGAAGGAAGGAAGGAAGGAAGG
>seq2: AAGGAAGGAATGGAAGGAAGGAAGG
>*seq3: AAGGAACGGAATGGTAGGAAGGAAGG*

Output for aligned DNA Sequences:

>seq1:A-AGGA-AGGA-AGGAA-------GG-----AA-GAAGG
>seq2:------------AAGGAAGGAATGGAAGGAAGGAAGG
>seq3:----------AAGGAACGGAATGGTAGGAAGGAAGG

C. Experiments

The proposed system implemented on four personal computers and one of them is used as head node (HN) and remaining nodes are made as worker nodes (WNs). All WNs use window operating system and these nodes are connected with local area network.Java socket framework is used to transfer the messages on local network.

Initially, HN works runnning on local network at global checkpoint. Then, HN accepts the input DNA sequences and selects one sequence for each WN in Fig 3. The global resource monitor (GRM) takes the checkpoint of clients' connection states such as running, accept, receive and reject. It shows three worker nodes' running state as shown in Fig. 4. In running state, there are four conditions: Client type to show worker number, IP address to show WN, Status to show worker node's conditions, Current Time to show system time, Time Duration to show time within each client's running state to accept and receive state or running state to reject state.

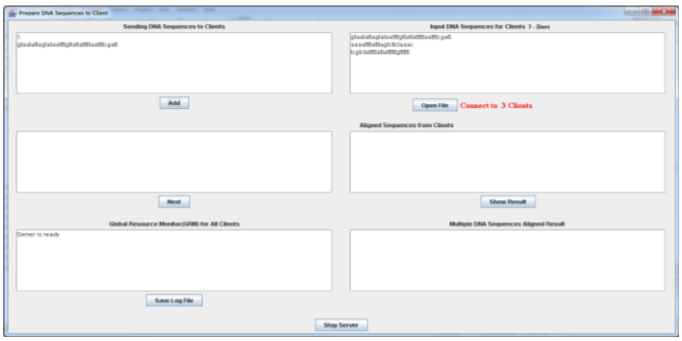

Fig. 3. Preparing Input Sequence for Worker Node by Head Node

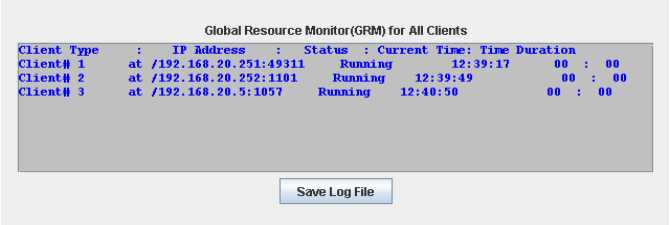

Fig. 4. Monitoring information by GRM

When WN works running state at local checkpoint, the WN accepts input sequence from HN and calculates that sequence to form aligned sequence using genetic operators: population, generation, and crossover and mutation rate. The worker process is shown in Fig. 5. After calculation of genetic algorithm (GA), the aligned sequence is sent to HN. The local checkpoint (LC) records worker node's conditions such as accept and send. Local checkpoint storage (LCS) stores these records in log file according to use "Save Log File" button.

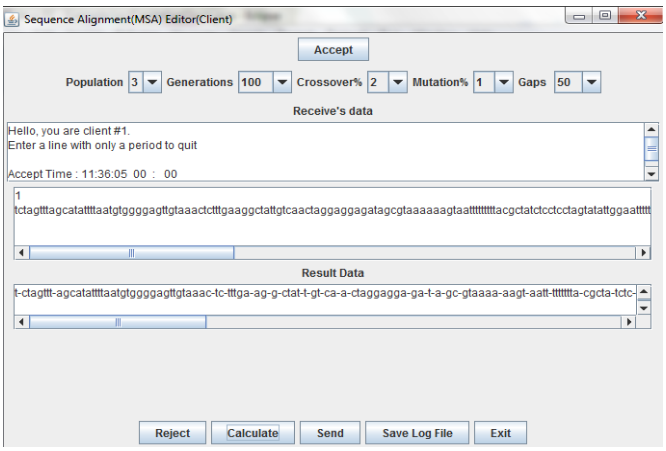

Fig. 5. Sequence Alignment Process by WN

When the aligned sequence is received by HN, the system shows receiving information results from WNs as shown in Fig. 6. In this system, therefore, global resource monitor (GRM) demonstrates worker nodes' conditions as shown in Fig. 7.

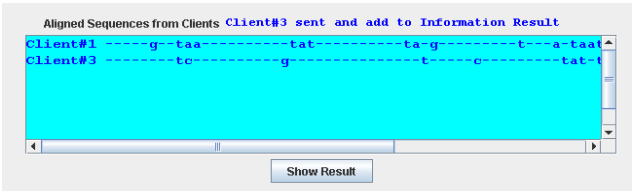

Fig. 6. Receiving Aligned Sequence from WN by HN

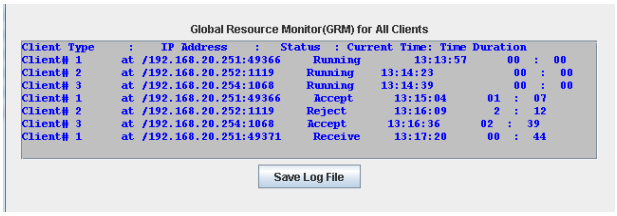

Fig. 7. Monitoring Information of WN by GRM

If WN does not accept input sequence, HN receives crush message "Client is crushed" as illustrate in Fig. 8. The global

checkpoint storage (GCS) detects failure condition and GRM records this condition as reject state. The head node searches available worker node by observing global resource monitor at global checkpoint storage and makes decision which node to transfer the job from crushed node in Fig. 9. After that, global checkpoint storage stores all records from global resource monitor as log file by using "Save Log File" Button.

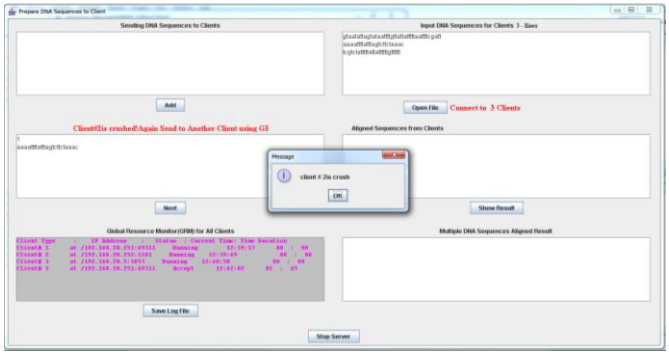

Fig. 8. HN receives crush message from WN

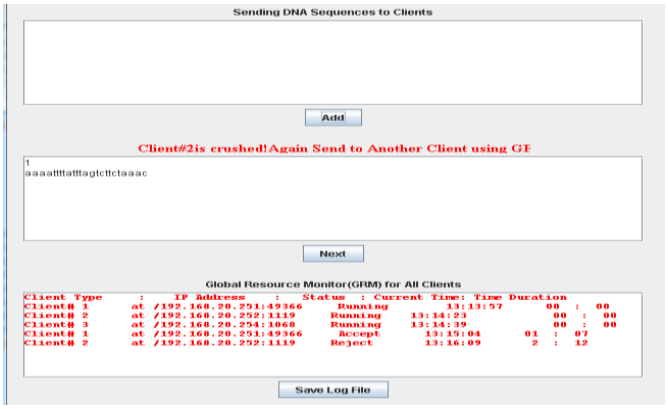

Fig. 9. GRM detects failure and decides available WN for load balancing

GRM finds the available non-failure node to compute the sequence from failed node according to the WNs current condition.WNs'current condition are described in global checkpoint monitoring information: Running, Accept and Receive.When HN receives the crush report from one WN , this failed WN condition is recorded as Reject state by GRM in global checkpoint storage. But,non-failure WNs are continuously processing their input sequence to produce aligned sequence. After non-failure WNs had computed the sequence alignment , they sent the result aligned sequence to HN. This condition is recorded as Receive state by GRM .

Therefore , WNs that has the Receive state is said to be transfer the job from failed node. GRM chooses available node that has been displaying Receive state firstly among other non-failure node and decides that node to transfer job from failure node in Fig. 10.

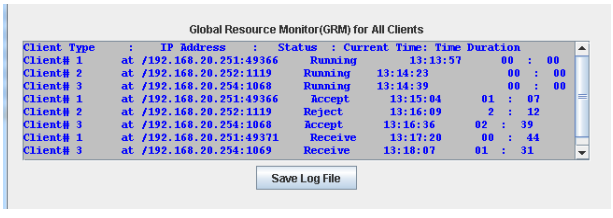

Fig. 10. GRM chooses worker which get firstly Receive State

After that the worker node again processing the alignment process for failure node's job and sends the result to HN.The system receive the result from WN in Fig. 11.

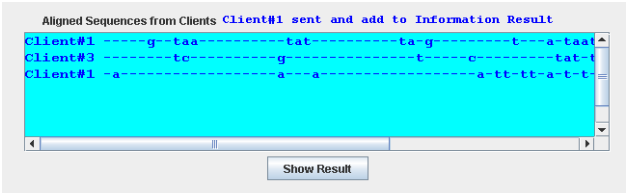

Fig. 11. WN again processes MSA and sends the result to HN.

The whole system is finished all input sequences are computed and display to the all aligned sequences by using "Show Result" Button in Fig. 12.

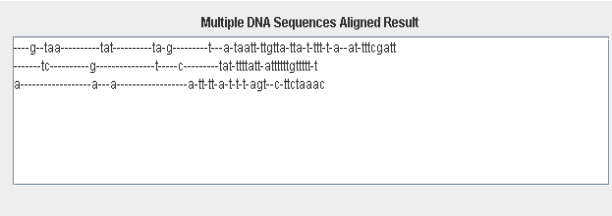

Fig. 12. Show all aligned sequences

All system process are finished and the worker nodes are stopped by clicking the "Exit" Button .And then, the whole system is stopped when head node uses the "Stop Server" Button.This system can operate within one hour period.

VIII. CONCLUSION

The proposed checkpoint-based fault tolerance approach offers some advantages. While this system implements the checkpoint on the application level, the checkpoint is taken in alignment processing and does not need to support any operating system. Using Coordinated Checkpointing and Systematic Process Logging, it can avoid domino effect and can reduce useless checkpoint. There is no need large storage space because stored checkpoint records are automatically deleted with the ability of checkpoint monitoring information. This system improves user availability and system reliability for MSA using global resource monitor (GRM) as load balancing mechanism. This system can improve performance and accuracy of distributed computing and it can reduce the time and overhead cost.

ACKNOWLEDGEMENT

We are very grateful to our rector from University of Technology (Yatanarpon Cyber City) and professors from Department of Information and Communication Technology (Technology University) for fruitful discussions during the preparation of this paper.

REFERENCES

[1] G. Cao, "Introduction to Distributed Systems", Department of Computer & Engineering ,2009.

[2] J. P. Walters and V.Chaudhary, "Replication-Based fault Tolerance for MPI Applications", IEEE Transactions On Parallel and Distributed Systems, volume 20, 2009.

[3] V. Ogale, B. Balasubramanian and V. K. Garg, "Fusion-based Approach for Tolerating Faults in Finite State Machines", IEEE

International Symposium on Parallel & Distributed Processing, 2009.

[4] V.K Garg, "Implementing fault-tolerant services using fused state machines" ,Technical Report ECE-PDS-2010-001, Parallel and Distributed Systems Laboratory,ECE Dept, University of Texas at Austin ,2010 .

[5] S.Bansai, S.Sharma, "An Improved Multiple Faults Reassignment based Recovery in Cluster Computing", Journal of Computing, Volume2, November 2010, ISSN 2151-9617.

[6] M.Tripathy, C.R.Tripathy, "A Hierarchical Shared Memory Cluster Architecture with Load Balancing and Fault Tolerance", International Journal of Computer Applications ,Volume 25, June 2011.

[7] S. Kumar, P. Kumar, "Hierarchical Non-blocking Coordinated Checkpointing Algorithms for Mobile Distributed Computing", International Journal of Computer Science and Security (IJCSS), Volume (3),2002.

[8] A. Khunteta , "Analysis of Checkpointing Algorithms" , (IJCSE) International Journal on Computer Science and Engineering Vol. 02, No. 04, 2010.

[9] P.Kumar, R.Setiya, and P.Gahlan,"Checkpointing Algorithms for Distributed Systems", International Journal of Computing Science and Communication Technologies, Volume 2, July 2009.

[10] L. Guoliang, C. Shuyu, Z. Xiaoqin, "A Non-blocking Checkpointing Algorithm for Distributed Systems", International Journal of Digital Content Technology and its Applications, Volume 5, July 2011.

[11] V. Shah, V. Kapadia , "Load Balancing by Process Migration in Distributed Operating System" , International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-2, Issue-1, March ,2012.

[12] S. Prusty , "Checkpointing and Rollback Recovery in Distributed Systems" , Department of Computer Science and Engineering Indian Institute of Technology, Guwahati , April 2009

[13] http://en.wikipedia.org/wiki/Smith Waterman_algorithm