International Journal of Distributed and Parallel Systems (IJDPS) Vol.2, No.1, January 2011

A NEW CO-ORDINATED CHECKPOINTING AND ROLLBACK RECOVERY SCHEME FOR DISTRIBUTED SHARED MEMORY CLUSTERS

Minakshi Tripathy[1] and C.R. Tripathy[2].

Department of Computer Science and Engineering
V.S.S. University of Technology, Burla,Sambalpur,Orissa,
E-Mail: minakshiom@gmail.com, minakshiom@yahoo.co.in

ABSTRACT

In this paper, an unified lightweight error recovery scheme based on coordinated checkpointing and rollback for distributed shared memory clusters is proposed. The new scheme maintains multiple globally consistent checkpoints of the state of a distributed shared memory cluster and recovers to a pre-fault checkpoint of the system. It also describes and evaluates the coordinated checkpointing. The coordinated checkpoint neither needs to exchange coordination messages nor adds information to the process messages. It only accesses stable storage when checkpoints are saved. Each of the processes saves its state independently from the other processes. The checkpoint timers are set at different processes. Based on the results of performance evaluation the proposed scheme is shown to outperform the previously proposed checkpoint and recovery schemes for distributed shared memory clusters.

KEYWORDS

Fault tolerance, global states, checkpoint timers, clock drift rate, consistency, and recoverability.

1. INTRODUCTION

The performance and fault tolerance are in general competing and often conflicting goals in high performance computing. Clusters in high performance cluster computing environmen, when a system is in operation, failures can happen anytime resulting in service unavailability. The first step in the design of a fault tolerant system is to provide a reliable environment. The system must detect and diagnose errors rapidly and apply corrective procedure intelligently. In recent days, the distributed shared memory clusters have received much attention due to their several attractive features. The fault tolerance is a major concern for large distributed shared memory clusters. As the number of interconnected nodes in the system increases, tolerating node failures become essential for parallel applications with large execution times [1-5].

The checkpointing is to restore the last non faulty state of the failing task to recover from faults. The checkpoint is saved in advance into a stable storage and is restored with event of failures of a task. The rollback recovery is the most widely used means for system recovery in the occurrence of errors where the system executes as a succession of system states. In the event of occurrence of an error, the system rolls back to a previously reached state and resume execution from that state. The saved states are called checkpoints and the action is called *checkpointing* or *taking a checkpoint*. This kind of system recovery from a legitimate system state based on checkpoints is known as checkpoint based rollback recovery [6-10].

The checkpointing has been a popular method of providing fault tolerance in high end systems. The checkpoint techniques are broadly classified into three categories based on the operating system, middleware and application layer [11-12]. Taking checkpoints at the operating system layer provides transparency of the checkpointing mechanism to the middleware and application layer and is called as transparent checkpoints. The checkpoint mechanism on application software is application specific and is called the explicit checkpoint. Taking checkpoints at the middleware level is called as implicit checkpoints which also include compiler assisted checkpoints. The explicit checkpoints are further classified into coordinated checkpointing, induced checkpointing and communication induced checkpointing. The coordination of all the constituent system components when taking checkpoints with timing to form a consistent global state of the system is known as coordinated checkpointing. The other category of checkpointing timing techniques where constituent system components take their own checkpoints without synchronizing with each other is referred to as independent checkpoints. Finally a hybrid checkpoint technique that combines both the aforementioned techniques is called the communication induced checkpoints.

Among all the types of checkpointing, the coordinated checkpointing typically can be the approach of choice for recoverable distributed shared memory because it is simpler to implement when the non failed nodes are assumed to be always accessible. In an un-coordinated checkpoints, each process determines independently from the others the instant when its state to save, while in the coordinated checkpointing, the processes coordinate among themselves to determine which process states to include in the application checkpoints. The coordination is quite essential to guarantee that the application checkpoint is consistent and recoverable. The coordinated checkpointing have shown better performance than the uncoordinated checkpoint, when used with parallel application. It can also tolerate failures that affect the multiple processes simultaneously [13-15]. The time based coordinated checkpointing in [13] does not support rollback recovery where as the coordinated checkpointing in [14] has no timers to support time based coordinated checkpointing. The fault tolerant embedded system in [15] designs an optimization strategy that meets time and cost constraints with simple checkpointing and replication for error recovery.

In this paper a new coordinated checkpointing and rollback recovery scheme for distributed shared memory clusters is proposed. We assume that the processor clocks are approximately synchronized. We use coordinated checkpointing to coordinate time while creating checkpoints. Each process saves its state whenever the local clock reaches checkpoint time. We set checkpoint timers at the different processes to save new checkpoints before the timer expires. Contrary to other time based checkpoints; it avoids overheads by preventing processes from sending messages that might become in-transit. The coordinated checkpointing defines a time interval before checkpoint creation during which processes are not allowed to send messages.

The rest of the paper is organized as follows. The proposed system structure is described in the Section2. The Section3 discusses the proposed coordinated checkpointing and rollback recovery, examples and the proposed algorithm. In the Section4, we consider the performance analysis. The results and discussions are carried out in the Section5. Finally, the concluding remarks are presented in the Section6.

2. SYSTEM STRUCTURE

This section presents a brief discussion on the distributed shared memory cluster system environment followed by the proposed coordinated checkpointing and rollback recovery structure to be applied on distributed shared memory clusters system. A system is an entity with a well-defined behaviour in terms of output it produces. The output is a function of the input it receives and the system logic with the passage of time as observed by the system clock. The system can be decomposed into constituent subsystems called system components, each component being a system of its own, which further can be decomposed and so on.

2.1. Distributed Shared Memory Cluster System Environment

This subsection describes the distributed shared memory cluster environment. In a distributed shared memory cluster system where processes interact with each other, the states of two processes are consistent if they agree on what messages have been exchanged between them. The coordinated checkpointing and rollback recovery schemes require that the most recent checkpointed state of every process is consistent with the most recent checkpointed state of every other process. Hence the set of all the committed checkpoints in stable storage is always a consistent snapshot of the entire system. Coordinated checkpointing with rollback recovery do not require all system processes to be checkpointed and rolled back together. Instead, communication between processes is tracked and the states of interacting sets of processes are checkpointed or rolled back in coordination.

The system is composed of a set of nodes interconnected by a network. A node contains a processor, a local distributed shared memory and a local hardware clock. The clocks do not need to be synchronized among nodes. However it is assumed that local clocks drift from real time with a maximum drift rate (ρ). The drift rates are in order of 10^{-5} or 10^{-6} for most quartz clocks that are available in commodity computers. The messages are delivered to processes with a bounded delivery time t_{dmax}. The total time to send a message, transmits the message through the network, and than receive the message is smaller than t_{dmax}.

2.2. The Proposed System Structure

In a distributed shared memory cluster system, a failure is said to occur in a system when the system environment observes an output from the system that does not confirm to its specification. An error is the part of the system i.e. any system component is liable to lead to failure. A fault is the adjudged cause of an error and may itself be the result of a failure. Hence, a fault causes a failure, which subsequently may result to another fault and so on. The term unit of failure is used to denote a part of a system that fails independently from other parts of the system, which is monitored by an error detection mechanism. In order to take corrective actions and prevent system failures, the errors must be detected first and than a fault repair or system recovery mechanism can be employed to prevent the system from experiencing a failure. The Fig.1 provides an illustration of the system structure for the proposed coordinated checkpointing rollback recovery through a symbolic flow diagram. The system structure is based on feed-back mechanism.

The structure consists of process coordinator, error manager, error detector, recovery manager, stable storage and checkpoints as described below.

i. Process Coordinator- The coordinator is one of the processes that start the application. The process coordinator initiates the timers for other processes.

ii. Error Manager- Each process deterministically executes the assigned task and conveys its result to the error manager. The error manager processes it and than forwards to the error detector.

iii. Error Detector- The Error detector is responsible for detecting errors and notifying to the recovery manager.

iv. Recovery Manager- The Recovery manager maintains recovery information on the stable storage. Upon detection of a fault, it suspends execution, loads the latest checkpoint from stable storage and than resumes execution from the last checkpoint.

v. Stable Storage- Stable storage is the part of the system where checkpoints are saved and which is not subject to errors.

vi. Checkpoint- It is the component state to be checkpointed. It is responsible for checkpoint activity of recoverable process. The activity includes the decision of when to take checkpoint or transferring the recovery data.

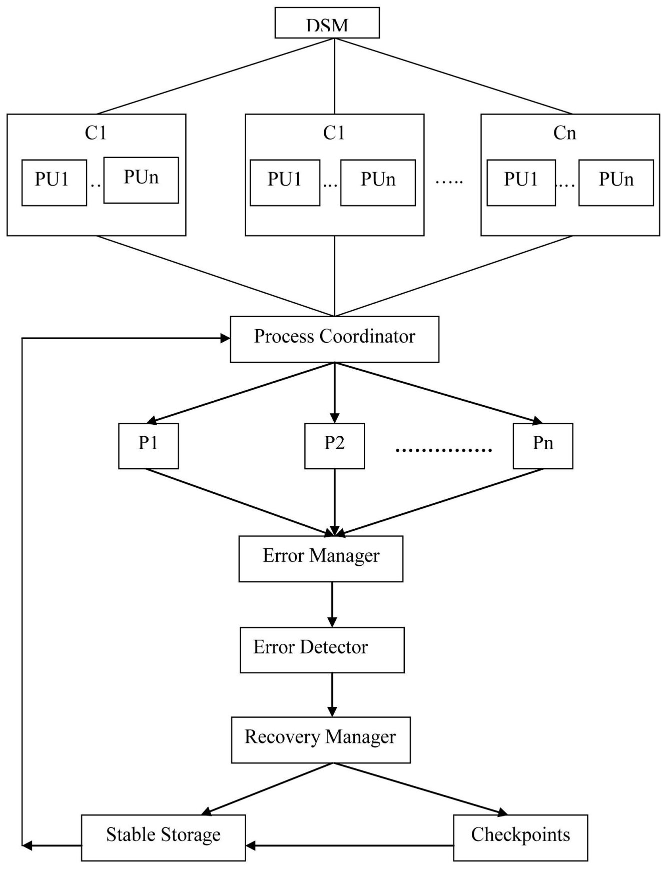

Figure1: The Proposed System Structure of coordinated checkpointing and rollback recovery scheme

3. PROPOSED COORDINATED CHECKPOINTING

This section discusses the details of the proposed coordinated checkpointing technique and the rollback recovery for distributed shared memory clusters. A checkpointing session is initiated by a checkpoint timer associated with each process. The main responsibility of coordinated checkpointing is to save global states of the application. A global state includes the state of each process belonging to the application with some messages. A generic coordinated checkpointing should record recoverable consistent global states, which satisfy the following two properties.

i. Consistency- If the global state includes a process state containing a receive event for a message as recv(mi) than, another process state must contain the corresponding send event for that message as send(mi).

ii. Recoverability- If the global state includes a process state containing the send event send(mi), but no other process state contains the corresponding receive event recv(mi) then the checkpoint must save message mi.

The global states saved by coordinated checkpointing are used to recover the application from failures that have affected one or more of its processes. Than, the application rolls back to the last stored state and than starts to re-execute. This could only be accomplished if the application restarts from a global state that could have occurred during one execution with no failures.

In the next subsection to follow, we propose a time based coordinated checkpointing technique for distributed shared memory clusters. To arrive at the analytical model with time based coordinated checkpointing and rollback recovery scheme, let us consider the following notations.

NOTATIONS:

ρ	:	Clock drift rate
D	:	Checkpointing timer interval in seconds.
MD	:	Maximum deviation i.e. maximum time interval
t_{dmin}	:	Minimum time interval
t_{dmax}	:	Bounded message delivery time
t	:	Time interval between two checkpoints while resuming execution.
s	:	Checkpoint size
T	:	Checkpoint period
Tc	:	Time to perform a checkpoint to storage
CN	:	Optimum number of checkpoints
B	:	Bandwidth for a processor in bytes/second to access storage.
p	:	Total number of processes.
P	:	Total number of processors in the system.
E	:	Total execution time including required time to perform all checkpoints.

3.1. Time Based Coordinated Checkpointing

In the proposed scheme, the time based coordinated checkpointing synchronizes checkpoint timers to record recoverable consistent states of the application. Each process executes the checkpoint creation procedure periodically whenever the local checkpoint timer expires. The timers are synchronized when the interval becomes higher than the time taken to store a checkpoint. This interval is proportional to the maximum message delivery time. The time based coordinated checkpointing ensures consistency and recoverability property. The coordinated checkpointing procedure initiates process's checkpoint timers in such a way that the timers will expire within an interval of D seconds. Ideally, D should be made as small as possible, because that reduces the periods in which processes are not allowed to send messages. The coordinated checkpointing procedure selects one of the processes to be the process coordinator, which initiates the timers of other processes. The coordinator adds to its local time the checkpoint period T to obtain the first checkpoint time. Than, it sets two timers *createnewchkpt* and *stopsendingmess*. Since different messages can experience distinct network delays, the coordinator loop sends the interval until it receives all answers within a time period smaller then $D+2*t_{dmin}$ where t_{dmin} is the minimum time interval.

If the timers are not properly synchronized, e.g. process P1 sends a message m1 after saving its state and P2 receives m1 before storing its state. The consistency property is violated because the global state contains recv(m1), but does not include send(m1). To avoid this problem, the proposed time based coordinated checkpointing disallows message sending during an interval after the checkpoint timer has expired. The nth checkpoint satisfies the consistency property if no process is allowed to send messages $MD-t_{dmin}$ seconds after saving its nth checkpoint. The maximum deviation is the maximum time interval that can separate the expiration of any two timers i.e. $MD=D+2nt\rho$. (1)

It depends on the initial deviation between timers (D) and the clock drift rate since last initialization ($2nt\rho$). The MD increases slowly as drift rates are small. For instance, if the coordinated checkpointing procedure starts timer with D=10ms and if we assume $\rho=10^{-6}$, than the MD will be 100ms after 12.5 hours.

3.2. Proposed Rollback Recovery

In case of detection of an error in a system, the system recovery techniques based on checkpointing from the last checkpoint state is called as rollback recovery. The failure of a node causes the loss of all states in that node. Under such circumstances the checkpoint contains the private process states and corresponding shared memory state from which correct execution may be restarted. With coordinated recovery, the rollback of processes that were active on the failed node requires the rollback of processes on other nodes that communicated with the failed processes. All these processes are rolled back to their previous checkpoint in coordination. The first step in recovery is to rollback to the most recent checkpoint in the private states of all processes that include processes on the failed node. Next, all modifications to the task's shared address space performed by all these processes since their most recent checkpoints are undone.

3.2.1. Time Based Rollback Recovery

Here, we propose a time based rollback recovery technique. The easiest way to guarantee the recoverability property is to avoid the creation of message that might become in-transit. The nth application checkpoint satisfies the recoverability property if no process is allowed to send messages $MD+t_{dmax}$ seconds before its timer expires. A message can become in-transit if it is sent before a process creates its checkpoint and received after the other process has saved its state. If e.g. a process P2 sends two messages m1 and m2, message m1 does not have to be stored but message m2 would have to be stored if process P2 was allowed to send it.

The time based rollback recovery uses two timers: one that expires $MD+t_{dmax}$ seconds before checkpoint time and another that expires at checkpoint time. Whenever the first timer terminates, it calls *stopsendingmess* procedure to set a flag. The procedure *createnewchkpt* saves the process state, increments checkpoint time by checkpoint period T and than resets the timer. The variable CN counts the number of checkpoints that have been created since last resynchronization. The algorithm (CCRR) in the subsection illustrates the time based rollback recovery mechanisms along with the coordinated checkpointing.

3.2.2. Proposed Algorithm (CCRR)

Next, we present the proposed coordinated checkpointing and rollback recovery (CCRR) algorithm for distributed shared memory clusters. The proposed algorithm has the time complexity of O(pt) for the p number of processes with the checkpoint time interval t.

```
Coordinated checkpointing {
        Save process states;
        Store checkpoint data;
        Time=getTime();
        For Each (pi $\varepsilon$ processes)
                interval=getTime()-time;
                if (interval<(D+2*tdmin))
                        send=TRUE;
                        send(mi,Pi);
                        recv(mi,Pj)
                End if
        End for
chkpt_time=getTime()+interval-tdmin;
setTimer(createnewchkpt,chkpt_time);
Propagate checkpoint messages;
Commit checkpoint;
}
rollback recovery {
        restore process states;
        restore data from last checkpoint;
setTimer(stopsendingmess,chkpt_time-(D+2T$\rho$+tdmax));
Propagate rollback commit messages;
}
createnewchkpt{
        CN=CN+1;
Chkpt_time=chkpt_time+T;
setTimer(createnewchkpt,chkpt_time);
if(getTime()-(chkpt_time-T)<=D+2(CN-1)T$\rho$-tdmin))
        send=TRUE;
}

stopsendingmess{
        send=FALSE;
setTimer(stopsendingmess,chkpt_time+T-(D+2(CN+1)T$\rho$ + tdmax));
}
```

4. PERFORMANCE ANALYSIS

The consideration of runtime on large systems like a distributed shared memory cluster is very important. The users always wish to use the system in the most optimal manner. The knowledge about checkpoints with interval helps the user to make more informed decisions and tradeoffs between their resource usage and application resiliency. We attempt to provide this information through the performance analysis discussed in the subsequent paragraphs.Here we propose an analytical model for the performance evaluation of the distributed shared memory clusters.

Based on the specified parameters, the time required to save the state of all processes during a checkpoint can be given by

$$T_c = \frac{P \times s}{B} \qquad (2)$$

If Ta is the application execution time without checkpointing enabled, the total application runtime T with checkpointing enabled is given by

$$T = T_a + CN \times T_c \qquad (3)$$

Given the total application execution time and checkpoint time interval, the total number of optimum checkpoints can be derived by

$$CN = \frac{T_a}{t} \qquad (4)$$

Based on the above formulas, the total application execution time can be derived to

$$T = T_a + \left(\frac{T_a}{t}\right)\left(\frac{Pxs}{B}\right) = T_a\left(1 + \frac{sP}{tB}\right) \qquad (5)$$

5. Results and Discussions

The proposed algorithm (CCRR) is implemented under matlab test bed. This section provides the experimental results. We have used synthetic application of 20,40,60,80 and 100 processes implemented on our distributed shared memory clusters architecture [4] with the proposed coordinated checkpointing rollback recovery structure consisting of 1 to 10 processors. For the experiments we assume bandwidth as 500 MB/sec and execution times are assigned randomly using both uniform and exponential distribution. Checkpoint size and checkpoint interval was set in between 100-1000MB and 100-1000ms respectively. The results were obtained using the average of 5-10 experiments and than compared with the existing method RRC [2]. The Table1 reports the performance of CCRR scheme with assigned P, t and s. The results in Table1 indicate that CCRR scheme performs better than the RRC [2]. The Figure2 shows the performance of checkpoint time and checkpoints with respect to checkpoint size. It presents that a large number of checkpoints reduce checkpoint time. The Figure3 shows the results of comparision of execution time vs. processors in proposed CCRR scheme and RRC [2] scheme. The results are significantly better with fast improvement in execution time in case of the proposed CCRR scheme.

Table 1: Performance of coordinated checkpointing and rollback recovery scheme

Proc (P)	Chk. Int. (t)	Chk. Size (s)	Chk. Time(Tc)	No. of Chk.(CN)	Exe. Time(T)
1	100	100	0.2	100	25050
2	200	200	0.8	100	24096
3	300	300	1.8	77	23138.6
4	400	400	3.2	38	15121.6
5	500	500	5	100	13130
6	600	600	7.2	17	10122.4
7	700	700	9.8	13	9127.4
8	800	800	12.8	100	8128
9	900	900	16.2	8	7129.6
10	1000	1000	20	100	6120

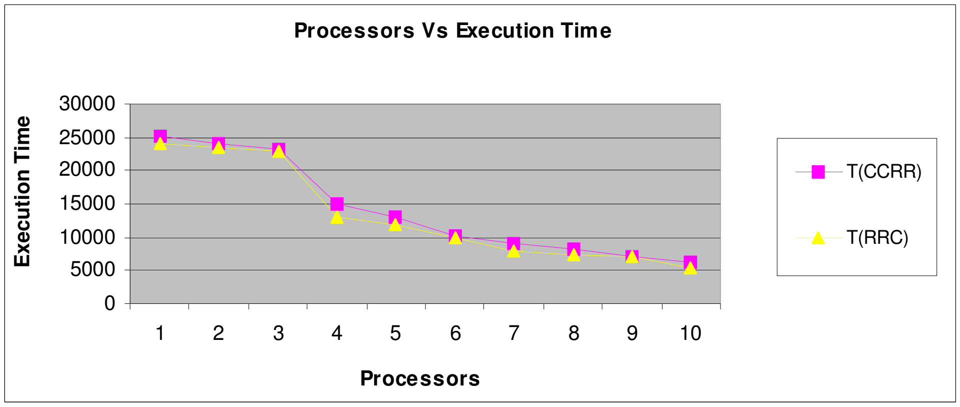

Figure2: Comparison of Number of Processors Vs Execution Time

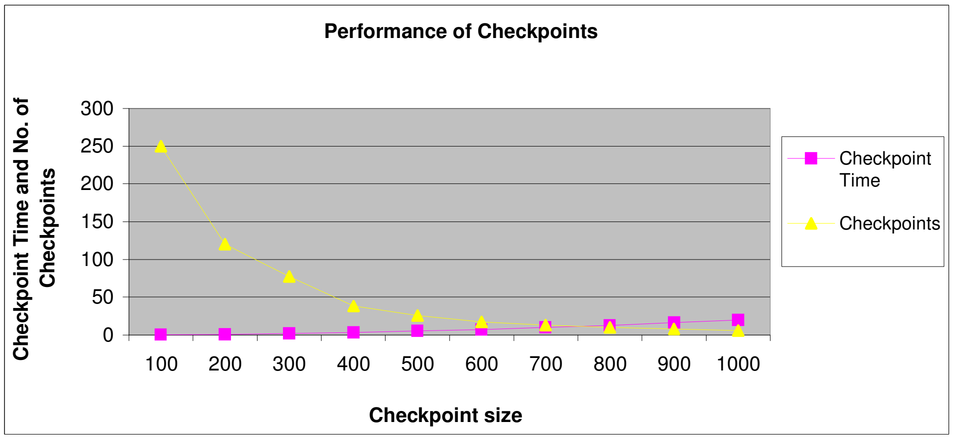

Figure3: Comparison of Checkpoint Time and Checkpoints Vs Checkpoint size

6. CONCLUSIONS

This paper presented a new error recovery scheme for distributed shared memory clusters based on coordinated checkpointing and rollback. Based on the results of comparision with existing methods, we conclude that the proposed coordinated checkpointing scheme leads to a better performance, requires less space in stable storage, is easier to implement and obtains a predictable rollback distance. The major contribution to the overhead in this approach is due to the checkpoint saving, while the cost of synchronization is actually insignificant. This time based CCRR uses a simple initialization procedure to start the checkpoint timers. Contrary to previous time based checkpointing [2], the new scheme also eliminates the overheads of in-transit message storage and addition of information to message. This is accomplished by preventing processes from sending messages during an interval before the checkpoint time and the CCRR technique leads to faster execution time with better performance.

REFERENCES

[1] Florin Sultan, Thu Nguyen and Liviu Iftode, Scalable Fault- Tolerant Distributed Shared Memory, In the Proceedings of the ACM/IEEE conference on Supercomputing, 4-10 Nov 2000, pp 1-12.

[2] Mikael Väyrynen, Virendra Singh and Erik Larsson, Fault-Tolerant Average Execution Time Optimization for General-Purpose Multi-Processor System-On-Chips, Design, In the Proceedings of Automation & Test in Europe Conference & Exhibition, 20-24 April 2009, pp 484-489.

[3] Zizhong Chen, Ming Yang, Guillermo Francia and Jack Dongarra, Self Adaptive Application Level Fault Tolerance for Parallel and Distributed Computing,, IEEE International Parallel and Distributed Processing Symposium, 26-30 Mar 2007, pp 414-421.

[4] Minakshi Tripathy and C.R. Tripathy, Design and Analysis of a Distributed Shared Memory Cluster Architecture based on "Dynamic Data Structure Task Scheduling", Proceedings of the International Conference on Electronic Systems, 07-09 Jan 2011, pp 395-398.

[5] Mohammed H. Mahafzah, On Evaluating the Reliability of Multiprocessor Systems, European Journal of Scientific Research, Vol. 41, No. 4, April 2010, pp 490-496.

[6] G.Cabillic, G. Muller and I. Puaut, The Performance of Consistent Checkpointing in Distributed Shared Memory Systems, In the Proceedings of the 14th Symposium on Reliable Distributed Systems, 13 – 15 Sept 1995,pp 96-105.

[7] Gaurav Suri, Bob Janssens and W. Kent Fuchs, Reduced Overhead Logging for Rollback Recovery in Distributed Shared Memory, In the Proceedings of 25th International. Symposium on Fault Tolerant Computing, 27-30 June 1995, pp 279-288.

[8] Harish Gapnati Naik, Rinku Gupta and Pete Beckman, Analyzing Checkpointing Trends for Applications on the IBM Blue Gene/P System, In the Proceedings of the International Conference on Parallel Processing, 22-25 Sept 2009**,** pp 81-88.

[9] Robert E. Storm and Shaula Yemini, Optimistic Recovery in Distributed Systems, ACM Transactions on Computing Systems, Vol. 3, No. 3, August 1985, pp 204-226.

[10] Frank Liberato, Rami Melhem, Daniel Mosse, Tolerance to Multiple Transient Faults for Aperiodic Tasks in Hard Real-Time Systems, IEEE Transactions on Computers, Vol. 49, No..9, Sept 2000, pp: 1089-1095.

[11] Richard Koo and Sam Toueg, Checkpointing and Rollback Recovery for Distributed Systems, IEEE Transaction on Software Engineering, Vol-13, No.1, Jan 1987, pp 23-31.

[12] G. Cao, M. Singhal, On Co-ordinated Checkpointing in Distributed Systems, IEEE Transactions on Parallel and Distributed Systema, Vol.-9, No.-12, Dec 1998, pp 12313-12325.

[13] Nuno Neves and W. Kent Fuchs, Using time to improve the performance of coordinated checkpointing, In the Proceedings of the International Computer Performance & Dependability Symposium, 04-06 Sept 1996, pp 282—291.

[14] G. Janakiraman and Yuval Tamir, Coordinated Checkpointing-Rollback Recovery Error Recovery for Distributed Shared Memory Multicomputers, In the Proceedings of the 13th Symposium on Reliable Distributed Systems, 25-27 Oct 1994, pp 42-51.

[15] Paul Pop, Viacheslav Izosimov, Petru Eles, Zebo Peng, Design Optimization of Time-and Cost-Constrained Fault-Tolerant Embedded Systems With Checkpointing and Replication, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol-17, No.3, March 2009, pp: 389-402.

Authors

Ms. Minakshi Tripathy received the degree of B.Sc. (PCM), M.Sc. (Statistics) and MCA from Sambalpur University. She has done 'A' level course from DOEACC, New Delhi. She is currently a Ph.D. (Computer Science) student at Sambalpur University, Burla, Orissa. She has publications in four different international conferences and two international journals. Her research interest includes shared memory, cluster computing, load balancing and fault tolerance.

Prof. (Dr.) C.R. Tripathy received the B.Sc. (Engg.) in Electrical Engineering from Sambalpur University and M. Tech. degree in Instrumentation Engineering from I.I.T., Kharagpur respectively. He got his Ph.D. in the field of Computer Science and Engineering from I.I.T., Kharagpur. He has more than 50 publications in different national and international Journals and Conferences. His research interest includes Dependability, Reliability and Fault–tolerance of Parallel and Distributed system. He was recipient of "Sir Thomas Ward Gold Medal" for research in Parallel Processing. He is a fellow of Institution of Engineers, India. He has been listed as leading scientist of World 2010 by International Biographical Centre, Cambridge, England, Great Britain.