Challenges of Error Recovery Techniques for Mobile Distributed Systems

Anil Kumar
Department of Computer Science & Engineering
HCTM, KAITHAL, HARYANA (INDIA)

ABSTRACT:
The emergence of mobility in a distributed system, has led to the start of new era of computing. Recent technological advances in mobile or hand-held devices and wireless technology have made the mobile computing affordable. Due to new emerging characteristics of mobile node, mobile computing environment is more error prone as compared to fixed infrastructure. In this paper we present failure recovery techniques, issues and challenges with respect to mobile distributed systems.

KEYWORDS: Mobile, checkpoint, MSS, MH.

I. INTRODUCTION:

A Mobile Computing System is a distributed system where some of processes are running on Mobile Hosts (MHs) [5]. The term "Mobile" means able to move while retaining its network connection. To communicate with MHs, mobile support stations (MSSs) are added. An MSS communicates with other MSSs by wired networks, but it communicates with MHs by wireless networks refer to fig.1.

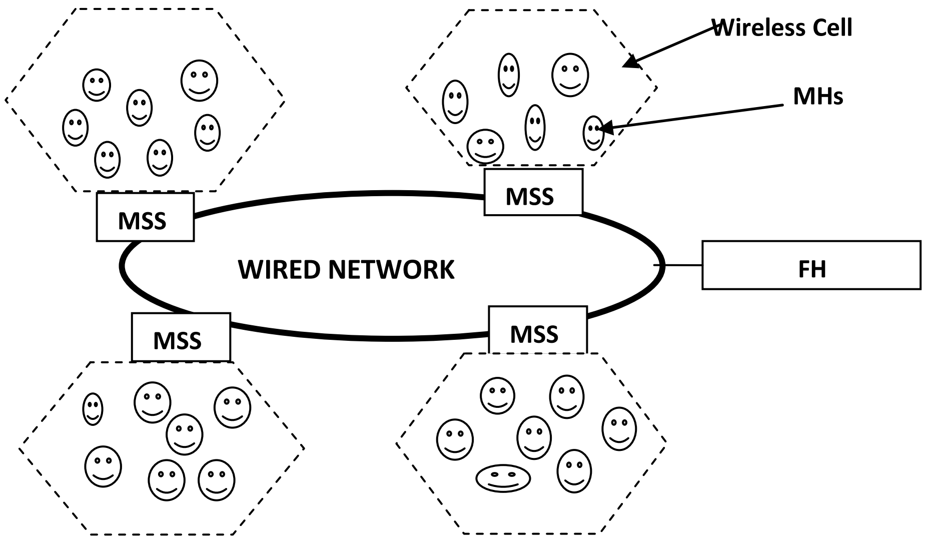

Fig. 1 Working Block Diagram Mobile Distributed System (MDS)

A cell is a geographical area around an MSS in which it can support an MH. An MH can change its geographical position freely from one cell to another cell or even area covered by cell. At any given instant of time an MH may logically belong to only one cell ; its current cell defines the MH's location and the MH is considered local to MSS providing wireless coverage in the cell. An MSS has both wired and wireless links and acts as an interface between static network and a part of mobile network. Static network connects all MSSs. A static node that has no support to MH can be considered as an MSS with no MH. Critical applications are required to execute fault-tolerance on such system [17]. The static network provides reliable, sequenced delivery of messages between any two MSSs, with arbitrary message latency.

1.1 MOBILE COMMUNICATION CHARACTERISTICS AND THEIR IMPLICATIONS:

Mobile data communication has several characteristics that must be taken into consideration when developing any failure recovery method. These characteristics are as following:

1. Peer-to-peer client/server communication: The communication pattern between an application in fixed network and its peer application on MH is peer-to-peer; which means that the communication can be initiated by either side (MH or MSS).
2. Real time communication: Real time issue arises when there are actions that must be completed within a specified amount of time otherwise they become useless or even harmful after that. In this context, the entity that initiates requests should receive replies within a specified period of time otherwise timeouts occur. The real time aspects of mobile communication originate from both the end user application and the physical system.
3. High message rate: The number of messages received and sent per unit time is high Therefore; any recovery technique that uses message logging has to deal with two particular problems i.e. overhead and storage.
4. Distributed service architecture: The fixed network is distributed over a large geographic area to provide mobility. It is normally that several applications running on different nodes cooperate together to complete a single service for an MS. This distributed architecture will affect the selection of fault tolerance approach
5. Scarce radio resources: The limited bandwidth of the air interface underlines the need of efficient communication between mobile stations and fixed network.

2. FAULT TOLERANCE SYSTEM AND ERROR RECOVERY TECHNIQUES:

A fault is anomalous physical condition which could lead to system failure. Failure can be classified in following two categories:

- **HARD FAILURE**:

Hard failure implying permanently failure or complete loss of connectivity of node These types of failures are non-voluntary in nature and processes stops any further actions forever such as falls, breaks, lost or stolen.

- **SOFT FAILURE**:

Soft failures do not permanently damage the node. In such case, MH informs to MSS prior to its occurrence such as battery discharge, disconnections or operating crashes. [15]

Fault tolerance is survival attribute of system and fault tolerant techniques enable a system to perform tasks in the presence of faults which involves fault detection, fault location, fault containment and fault recovery.

Failure recovery is a process that involves restoring an erroneous state to an error-free state. Recovery from errors in fault tolerant systems can be characterized as either rollback or roll forward refer to fig. 2].

2.1 **FORWARD ERROR RECOVERY:**

When system detect the error, forward error recovery technique takes the system state at that time and correct it and to be able to move forward. Hence, in this technique the nature of error and damaged caused by faults must be completely and accurately assessed, which make it possible to remove those errors in the process state and enable the process to move forward [26]. This approach is not used in distributed and mobile systems as accurate assessment of all the faults may not be possible.

Replication implements roll-forward mechanism where the entity (mainly a server application) is replicated to establish a group of replicas and in the event of the failure of one entity, the other replicas can take over and continue processing requests. Active replication where all server replicas run concurrently and passive replication in which one member of the server group is designated as the primary, are two best known replication approaches

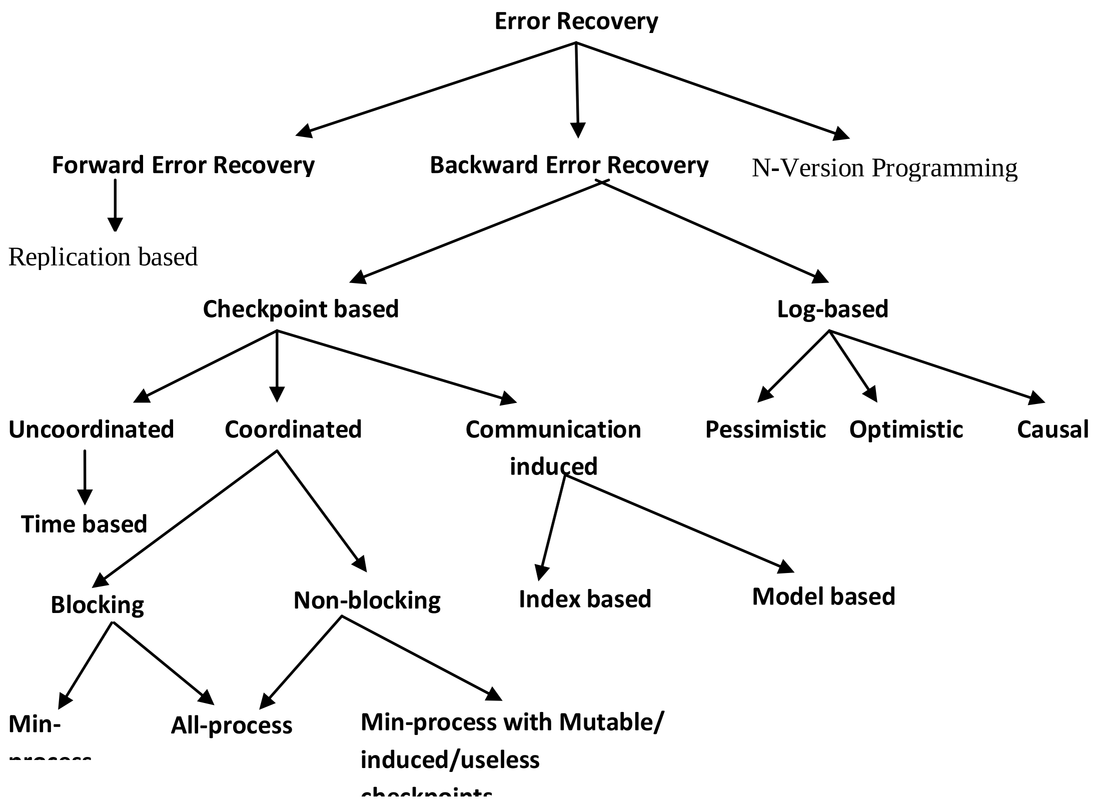

Fig. 2. A partial view of Error Recovery Techniques

2.2 BACKWARD ERROR RECOVERY:

Backward error recovery or checkpoint restart has been largely employed as a fault tolerant mechanism for DSs. In this technique, system rollback to some earlier state, correct it and roll forward from here. Thus, backward error recovery is more general recovery mechanism [25]. Backward error recovery can be log based or checkpoint based which is explained in the next sections.

Log-based Rollback Recovery Mechanism

In log-based recovery, sending message history of processes since last checkpoint, are kept in main memory [16]. In case of a failure, a process can ask fault-free processes the needed messages. "Spooling" can be performing if volatile message logging takes too much memory space. In message logging protocols, each process periodically records its LS and logs the messages that it receives after having recorded that state on stable storage. When a process crashes, a new process is created in its place. The new process is given the appropriate recorded LS, and then the logged messages are replayed in same order as the process originally received them. All message-logging protocols require that once a crashed process recovers, its recovered state is consistent with the states of the other processes [20]. There are three types of logging protocols.

- **PESSIMISTIC LOGGING**:

The pessimistic logging approach does not require any synchronization between processes but received messages are logged synchronously. During logging, it blocks the receiver until the message is logged to a stable storage. A process P_i never sends a message until it knows that all messages received and processed so far are logged. In such way it guaranteed that orphan is never created in pessimistic logging approach. During recovery all messages received in the time between the latest checkpoint and the fault are replayed to it from the stable storage in the same order as they were received before the fault [14].

- **OPTIMISTIC LOGGING**:

In optimistic message logging approach, message may not be logged immediately. The receiver continues its normal actions. The messages are logged at some point of time during idle time of the system [12]. The application does not block, and the determinants are spooled to stable storage asynchronously. This approach has less average cost of logging a message in the comparison of pessimistic approach ($Cost_{opti\_log} < Cost_{pessi\_log}$) [12]. It reduces failure free overhead, but complicates the recovery process.

- **CAUSAL LOGGING**:

Causal logging approach is a mix of the optimistic (orphan free) and pessimistic logging (non-blocking) approach which avoid the orphan and blocking. In this approach, dependency information is piggybacked on application messages and this dependency information including with message contents are logged in the volatile memory of sender [12]. Hence, this approach is non-blocking, orphan free and has only one overhead of storing a message in volatile memory.

Checkpointing and Recovery Mechanism

Checkpointing and rollback recovery is an efficient error recovery mechanism used in DSs [1]. It enable a system to tolerate failures by periodically saving the entire state during failure free execution and rolling back to the saved state if a failure occur. It works on fail-stop model and mainly has two phases: (a) saving a checkpoint in stable storage. (b) Checkpoint recovery following the failure.

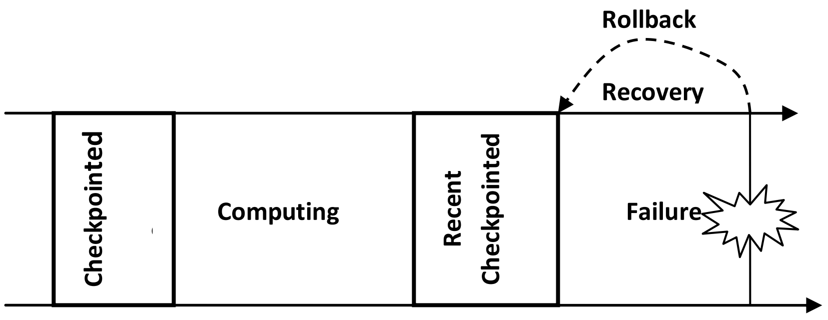

Fig. 3 Checkpoint and Recovery

During first phase of checkpointing approach, the state of each process in the system is periodically saved on stable storage, which is called a checkpoint of a process. To recover from a failure refer to fig. 3, the system restarts its execution from a previous error-free, CGS [4]. In a DS, since the processes in the system do not share memory, a global state of the system is defined as a set of local states, one from each process. A global state is said to be "consistent" if it contains no orphan message; i.e., a message whose receive event is recorded, but its send event is lost [4]. Three flavors of checkpointing based recovery protocols are coordinated checkpointing, uncoordinated checkpointing and communication induced checkpointing (CIC).

- **COORDINATED CHECKPOINTING**:

Coordinated checkpointing is a commonly used technique for fault tolerant in mobile DSs. In coordinated approach it is assumes that a single process which is know as initiator, invokes the checkpointing algorithms to determining the CGC. In this approach processes communicate and synchronize through system messages before taking checkpoint and coordinate their checkpointing actions in such a way that checkpointing approach yields a CGS. Mostly it follows two-phase commit structure [2], [19], [21], [30]. In the first phase, processes take tentative checkpoints and in the second phase, these are made permanent. The main advantage is that only one permanent checkpoint and at most one tentative checkpoint is required to be stored. In case of a fault, processes rollback to last checkpointed state. A permanent checkpoint cannot be undone. In some approaches initiator of the checkpointing process forces the dependent processes (minimum processes). The coordinated checkpointing protocols can be classified into two types: blocking and non-blocking. In blocking algorithms, as mentioned above, some blocking of processes takes place during checkpointing [2]. In non-blocking algorithms, no blocking of processes is required for checkpointing [19], [21]. The coordinated checkpointing algorithms can also be classified into following two categories: minimum-process and all process algorithms. In all-process coordinated checkpointing algorithms, every process is required to take its checkpoint in an initiation [19], [21]. In minimum-process algorithms, minimum interacting processes are required to take their checkpoints in an initiation [2]. In coordinated approach CGS is achieved during run-time, while in the independent approach the determination of a consistent recovery line was left to the recovery phase, which could result in some rollback propagation [28]. It does not suffer from rollback propagations.

- **UNCOORDINATED CHECKPOINTING**:

In independent checkpointing, processes do not synchronize their checkpointing activity and processes are allowed to records their local checkpoints in an independent way [18], [20], [28], [31]. After a failure, system will search a CGS by tracking the dependencies from the stable storage. The main advantage of this approach is that there is no need to exchange any control messages during checkpointing. But this requires each process to keep several checkpoints in stable storage and there is no certainty that a global consistent state can be built. The main disadvantage of uncoordinated approach is the domino-effect [20]. In [Fig. 4], processes P_1 and P_2 have independently taken a sequence of checkpoints. The interleaving of messages and checkpoints leave no consistent set of checkpoints for P_1 and P_2, except the initial one at {C_{10}, C_{20}). Consequently, after P_1 fails, both P_1 and P_2 must roll back to the beginning of the computation.

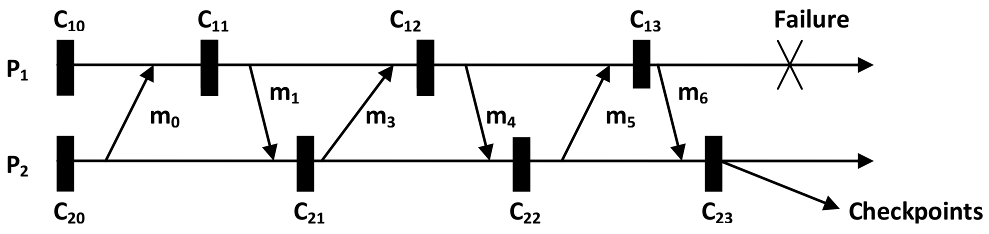

Fig. 4 Domino-effect

It should be noted that global state {C11, C21} is inconsistent due to orphan message m1. Similarly, global state {C12, C22} is inconsistent due to orphan message m4. The possibility of the Domino effect may cause the loss of large amount of useful work and also increases the checkpointing overheads. Rollback propagations also make it necessary for each processor to store multiple checkpoints, potentially leading to a large storage overhead.

- **COMMUNICATION-INDUCED CHECKPOINTING:**

In the CIC approach, a GC is similar to the approach of coordinated checkpointing while rollback propagation can be avoided by forcing additional un-coordinated local checkpoint in processes [11], [26]. Protocols piggyback control information on application messages, thereby, avoids addition of explicit control message, to the computation, during checkpoint creation [10]. Quasi-synchronous checkpointing algorithms can be classified into two categories [20]. First is Model based checkpointing in which checkpointing protocol tries to avoid the domino effect by relying on preventing patterns of communications and checkpoints that could result in inconsistent states among the existing checkpoints and second is Index based checkpointing where a sequence number is assigned to local checkpoint local checkpoint. These assigned sequence number monotonically increasing after every checkpoint, such that the checkpoints having the same index at different processes form a consistent state. These index numbers are piggybacked on application an message which helps the receiver in deciding when to take snapshot. Index based checkpointing protocol, can be used with time coordination [6] to reduce the number of total checkpoints.

2.3 N-VERSION PROGRAMMING:

N-version programming [9] uses design diversity approach and it is defined as the independent generation of N>=2 functionally equivalent programs from the same initial specification. Independent generation of programs means that the programming efforts are carried out by N development teams that do not interact with respect to the programming process. The initial specification is a formal specification in a specification language. The goal of the initial specification is to state the functional requirements completely and unambiguously, while leaving the choice of implementations to the N programming efforts. N-version programming assumes that all programs contain faults, but it relies on the fact that the number of hidden faults will be small and that they will be in different locations in each of the versions. Wherever possible, different algorithms, programming languages and compilers are used in each separate effort.

3. LIMITATIONS OF EXISTING ERROR RECOVERY TECHNIQUES :

In this section, we explain the general limitations of these techniques and leave the specific ones concerning the mobile environment to the next chapter.

3.1 BACKWARD ERROR RECOVERY HAS TWO ASSUMPTIONS:

- Transient faults: Without assuming that faults are transient, the faulty process will certainly fail again at exactly the same place. The faulty process will roll back to the latest saved state and then continues its execution (exactly the same program instructions are repeated) to restore the pre-failure state before it hits the error again. Note that the faulty entity may or may not reactivate the permanent fault depending on the latest checkpoint time, but it will certainly hit the error.
- Good checkpoints: Rollback assumes that only good data is saved to a stable storage and this implies that the fail-stop property must be upheld. In other words, the saved states must not contain the error that is caused by the transient fault.

3.2 FORWARD ERROR RECOVERY HAS ALSO TWO ASSUMPTIONS:

- Transient fault: Replication approach depends on the assumption that most of the software faults are transient. If this assumption is not applied, then all members of the replica group will fail at the same time, for example because of a permanent software bug.
- Fail-stop: Most of the replication techniques assume fail-stop property, i.e. an entity works correctly or stops functioning completely. This assumption can be relaxed at the cost of more complex voting algorithm and an increase in the number of replicas.

3.3 N-VERSION ERROR RECOVERY TECHNIQUES ASSUMPTIONS:

N-version or the use of diversity has no technical limitation in general, but its main limitation is its high cost both with respect to implementation and maintenance. There is a big discussion whether it is better to concentrate on developing one reliable version rather than less reliable multi-versions. The two assumptions about the nature of fault fail-stop and transient are dated back to the early 1980's and they can be probably true for some relatively simple applications. But, these assumptions will simply not hold for modern distributed communication applications. Everyday experience with communication applications has shown that many (if not most) of the software faults are permanent and they are reproducible, but they require rare sequence of events to be activated. This can be explained with the fact that it is almost impossible and not realistic to test every path and combination in these large and complex applications [3],[7].

4. CHALLENGES FOR IN DESIGNING ERROR RECOVERY TECHNIQUES FOR MDSS :

The existence of mobile nodes in Distributed Systems introduces new challenges that need proper handling while designing a checkpointing algorithm for such systems. MHs are integral part of mobile computing environment which frequently changes its locations. The portable computers can get arbitrarily small, down to the size of; say a walkman, a pocketbook, a watch, or a ring. The implications of portability are small size and weight and dependent on battery. Also wireless communication is susceptible to high failure rate and transmission interference or interception. This is a fixed network consisting of base stations, routers, gateways, resource management, mobility management units, etc. that exist to support the operation of the wireless mobile stations. The fixed network takes the overall coordination and control of the communication with the MSs and it uses peer-to-peer based protocols to achieve that. Due to the unique characteristics of mobile devices and wireless connectivity communication there are following issues that complicate the design checkpointing algorithms for MDS and need to handle more carefully.

- **MOBILITY**:

Changes in location of MH complicates routing of messages. Messages sent by a node to another node may have to be rerouted because the destination node (MH) disconnected from old MSS and now connected to new MSS. Checkpointing schemes that send control messages to MHs, will first need to locate the MH within the network, and thereby incur a search overhead [13], [14].

- **LIMITED BANDWIDTH**:

There is a wireless communication between MHs and their local MHH. In terms of data rate, the data rates of infrared networks range from 19.2 kbps to 1 Mbps and that for radio networks is 19.2 kbps. Wireless LANs have a data rate of 1 to 2 Mbps and that can be extended to 10 Mbps. Adaptive communication protocols have been proposed to compensate for the slow speed of some existing mobile communication links and to save the communication cost by reducing link usage. Low bandwidth constraints are satisfied by reducing the number of system messages required to collect a consistent snapshot [27].

- **FREQUENT DISCONNECTION**:

in mobile computing all the MHs are connected to their local MSS through wireless link and this connection is temporary with periods of disconnection. MHs may disconnect from the network temporarily or permanently [27].

- **LACK OF STABLE STORAGE**:

Due to vulnerability of mobile node to catastrophic failures e.g. loss, theft or physical damage, the disk storage on an MH cannot be considered as the stable storage. A reasonable solution is to utilize the stable storage at MSSs to store checkpoints of the MHs. Thus, to take a checkpoint, an MH has to transfer a large amount of data to its local MSS over the wireless network [1].

- **SMALL STORAGE CAPACITY**:

Small size and weight of a mobile computer means restricted memory size, small storage capacity and small user interface. So large amount of checkpointed data are not stored on local MHs memory

- **LIMITED BATTERY LIFE**:

The battery at the MH has limited life and there is not any permanent source of charging during moving from one location to other locations. Therefore energy conservation checkpointing techniques are required for MDS.

5. REQUIREMENTS FOR FAILURE RECOVERY IN MOBILE INFRASTRUCTURE :

There are following important requirements for failure recovery in mobile environment [3],[7].

1. High availability
2. Low overhead without real-time drawbacks
3. No assumptions on faults
4. Cost effective
5. Low Coordination Overhead
6. Low Context-Saving overhead

6. CONCLUSION :

In this paper we presents different issues and challenges for MDSs which provides high availability of services as a user can access the information from “anywhere” or “anytime” but it is less reliable compares to distributed systems. A system is said to be reliable if it can continue to provide the correct services, in the even of failure also. In mobile system, a MH is more error prone compares to fixed host (FH) as it frequently changes its location. A single failure in mobile distributed systems (MDSs) can affects a large number of users and computation. As a result, the mobile systems need to be able to tolerate faults to increase its reliability. Due to the mobility of nodes and wireless connectivity, MDSs have different characteristics, for example, week wireless connectivity, frequently disconnection, lack of stable storage on mobile nodes, finite power source, and vulnerable to physical damages that makes the already existing distributed fault tolerance algorithms unsuitable. Hence there is a great need to design an efficient checkpoint and faults tolerance protocols for MDS that specifically focuses on lessening power consumption, effective using the limited available memory and utilizing the bandwidth effectively.

REFERENCES:

1. Acharya A. and Badrinath B.R., "Checkpointing Distributed Application on Mobile Computers", in the Proc. of the 3rd Int'l Conf. on Parallel and Distributed Information Systems, pp. 73-80, Sept. 1994.
2. Koo R. and Toueg S., "Checkpointing and Roll-Back Recovery for Distributed Systems", IEEE Trans. on Software Engg., Vol.13, No.1, pp.23-31, Jan. 1987.
3. M. Zib Beiroumi, High Available Mobile Infrastructure Applications, proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE 2005), pp. 181-190, Chicago, USA, Nov, 2005.
4. Cao G. and Singhal M., "On Coordinated Checkpointing in Distributed Systems", IEEE Trans. on Parallel and Distributed Systems, Vol. 9, No.12, pp. 1213-1225, Dec.1998.
5. Cao G. and Singhal M., "Mutable Checkpoints: A New Checkpointing Approach for Mobile Computing Systems", IEEE Trans. on Parallel and Distributed Systems, Vol. 12, No.2, pp. 157-172, Feb. 2001.
6. Singh A.K., "On Mobile Checkpointing using Index and Time Together", World Acdemy of Science, Engineering and Technology, Vol 32, pp. 144-151, 2007.
7. M. Zib Beiroumi, Recovery of Infrastructure Software in the Mobile Network, NTS-17, 17th Nordic Teletraffic Seminar, pp. 137-148, August 25, 2004, Fornebu, Norway.
8. Changheng Shao, Fengjing Shao, Xiaoning Song, and Rencheng Sun, "A Dynamic Checkpointing and Rollback Recovery Solution Based on Task Switching", in the Proc. of the Int'l Symp. on Intelligent Information Systems and Applications (IISA'09) Qomgdap. P.R. China, pp. 354-358, Oct. 28-30, 2009.
9. A. Avizienis and L. Chen, On the implementation of N-version programming for software fault tolerance during execution, Proceedings of the IEEE COMPSAC 77, pages 149–155, Nov 1977.
10. Tsai J., "An Efficient Index-Based Checkpointing Protocol with Constant size Control Information on Messages," IEEE Trans. on Dependable and Secure Computing, Vol. 2, No. 4, pp. 278-296, Oct-Dec 2005.
11. Najib A. Kafahi, Said AI-Bokhitan and Ahmed AI-Nazer, "On Disk-based and Diskless Checkpointing for Parallel and Distributed Systems", An Empirical Analysis, Information Technology Journal, Vol. 4(4), pp. 367-376, 2005.
12. Mandal P.S. and Mukhopadhyaya K., "Performance Analysis of Different Checkpointing and Recovery Schemes using Stochastic Model" Journal of Parallel and Distributed Computing, No.66, pp. 99-107, 2006.
13. Awerbuch B. and Peleg D., "Concurrent Online Tracking of Mobile Users", in the Proc. of the ACM Symp. on Comm., Arch. and Protocols SIGCOMM, 1991.
14. Acharya A., "Structuring Distributed Algorithms and Service for networks with Mobile Hosts", Ph. D. Thesis, Rutgers University, 1995.
15. Singhal M. and Shivaratri Niranjan G., "Advance Concept in Operating System" Tata Mcgraw-Hill, 2005.
16. Alvisi, Lorenzo and Marzullo, Keith, "Message Logging: Pessimistic, Optimistic, Causal, and Optimal", IEEE Trans. on Software Engineering, Vol.24, No.2, pp.149-159, Feb.1998.
17. Adnan Agbaria, William H. Sanders, "Distributed Snapshots for Mobile Computing Systems", in the Proc. of the Second IEEE Annual Conf. on Pervasive Computing and Communications (Percon '04), pp. 1-10, 2004.
18. Bhargava B. and Lian S.R., "Independent Checkpointing and Concurrent Rollback for Recovery in Distributed Systems-An Optimistic Approach", in the Proc. of the 17th IEEE Symp. on Reliable Distributed Systems, pp. 3-12, 1998.
19. Candy K.M. and Lamport L., "Distributed Snapshots: Determining Global State of Distributed Systems", ACM Trans. on Computing Systems, Vol. 3, No. 1,pp. 63-75, Feb.1985.
20. Elnozahy E.N., Alvisi L., Wang Y.M. and Johson D.B., "A Survey of Rollback- Recovery Protocols in Message-Passing Systems", ACM Computing Surveys, Vol.34, No.3, pp. 375-408, 2002.
21. Elnozahy E.N., Johson D.B. and Zwaenepoel W., "The Performance of Consistent Checkpointing", in the Proc. of the 11th Symp. on Reliable Distributed Systems, pp. 39-47, Oct. 1992.
22. Elnozahy and Zwaenepoel W, "Manetho: Transparent Roll-back Recovery with Low-overhead, Limited Rollback and Fast Output Commit", IEEE Trans. on Computers, Vol. 41, No. 5, pp. 526-531, May 1992.
23. Elnozahy and Zwaenepoel W, "On the Use the Implementation of Message Logging", in the 24th Int'1 Symp. on Fault Tolerant Computing, IEEE Computer Society, pp. 298-307, June 1994.
24. Johnson D., "Distributed Systems Fault Tolerance Using Message Logging and Checkpointing", Ph. D. Thesis, Rice Univ., Dec.1989.
25. Manivannan D., Netzer R.H. and Singhal M., "Finding Consistent Global Checkpoints in a distributed computation", IEEE Trans. on Parallel & Distributed Systems, Vol.8, No.6, pp.623-627, June 1997.
26. Pardhan D.K., and Vaidya N., "Roll-forward Checkpointing Scheme: Concurrent Retry with Non-dedicated Spares", in the Proc. of the IEEE Workshop on Fault-Tolerant Parallel and Distributed System, pp. 166-174, July 1992.
27. Prakash R. and Singhal M., "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems", IEEE Trans. on Parallel and Distributed Systems, Vol. 7, No.10, pp1035-1048, Oct. 1996.
28. Storm R. and Temini S., "Optimistic Recovery in distributed Systems", ACM Trans. on Computer Systems, pp. 204-226, Aug. 1985.
29. Sistla A.P. and Welch J.L., "Efficient Distributed Recovery Using Message Logging", in the Proc. of the 18th Symp. on Principles of Distributed Computing", pp. 223-238, Aug. 1989.
30. Gupta B., Rashimi S., Rishad A. Rias, and Guru, "A low-Overhead Non-blocking Checkpointing Algorithm for Mobile Computing Environment", LNCS 3947, pp. 597-608,