Fault Tolerance Approach in Mobile Distributed Systems

Renu
Department of Computer Science & Engineering
Amity University, Haryana

Praveen Kumar
Department of Computer Science & Engineering
Amity University, Haryana

ABSTRACT
Mobile agent become very popular and attracted more importance these days due to the exponential growth of internet applications. The design of fault tolerance system become very challenging due to limited bandwidth of wireless network, mobile host mobility, limited local storage, limited battery power and handoff. A distributed system is a collection of independent entities to solve the problem that cannot be solved individually. A distributed system is susceptible to failure when it does not meet its specifications. Fault tolerant techniques enable systems to perform tasks even in the presence of faults. To deal with failure, a checkpoint is taken at specific place in a program at which standard process is interrupted specifically to preserve the status information. To recover from a failure one may restart computation from the last checkpoints, thereby avoiding repeating computation from the previous consistent global checkpoint. A mobile computing system is a distributed system where some of processes are running on mobile hosts (MHs), whose location in the network changes with time. The number of processes that take checkpoints is minimized to 1) avoid awakening of MHs in doze mode of operation, 2) minimize thrashing of MHs with checkpointing activity, 3) save limited battery life of MHs and low bandwidth of wireless channels. In this paper we provide an overview on Fault Tolerance in Mobile Distributed Systems (MDS).

Keywords
Domino effect, rollback recovery, mobile host, mobile support station, consistent global checkpoint

1. INTRODUCTION
Distributed systems are self-governing computers that appears to the users of the system as a single computer. The term Distributed Systems consists of several computers that do not share memory or a clock, each computer having its own memory and runs its own operating system and communicate with each other by exchanging messages over a communication network [22].

A mobile distributed system (MDS) is a distributed system where some of processes are running on mobile hosts (MHs). A mobile distributed system having fixed and mobile station interconnected through a communication network. The fixed station is located at the fixed location and the mobile station moves from one location to another in the network. Mobile Hosts (MHs) are becoming common in distributed systems due to their accessibility, cost, and mobile connectivity. The term "mobile" means able to move while retaining its network connection. An MH is a computer that may retain its connectivity through a wireless network while on move. Mobile environment is designed for cellular network, which facilitate the mobility management constraints includes the Mobile Host (MHs) and Mobile Support Station (MSS). An MSS has large storage capacity, high computing power, continuous availability and security but MH does not have large storage capacity. An MH communicates with other MH of system with the help of special node called mobile support station (MSS). An MSS provides the services to its local MH. A local MH can directly communicate with an MSS only if the MH is actually located within the cell serviced by MSS. Mobile network maintains the MH in the MSS in a cell. An MSS is connected through both wired and wireless links and acts as interface between the static network and other parts of the mobile network. Static nodes are connected via a high speed wired network. Static network connects all MSSs. A static node that has no support to MH, for this critical applications are required to execute fault-tolerant in the system. The static network provides reliable, sequenced delivery of messages between two MSSs, with arbitrary message latency. The wireless network within a cell also ensures FIFO delivery of messages between an MSS and a local MH i.e. there exist a FIFO channel from an MH to its local MSS and another FIFO channel from the MSS to the MH. If an MH does not leave the cell, then sent message from local MSS to MH would receive in sequence. Message communication from an MH_1 to another MH_2 occurs as follows. MH_1 first sends the message to its local MSS_1 using wireless link. MSS_1 forwards it to MSS_2, the local MSS of MH_2 via a fixed network. MSS_2 then transmit it to MH_2 over its wireless network [1]. However location of MH_2 may not be known to MSS_1 so MSS_1 may require to first determining the location of MH_2.

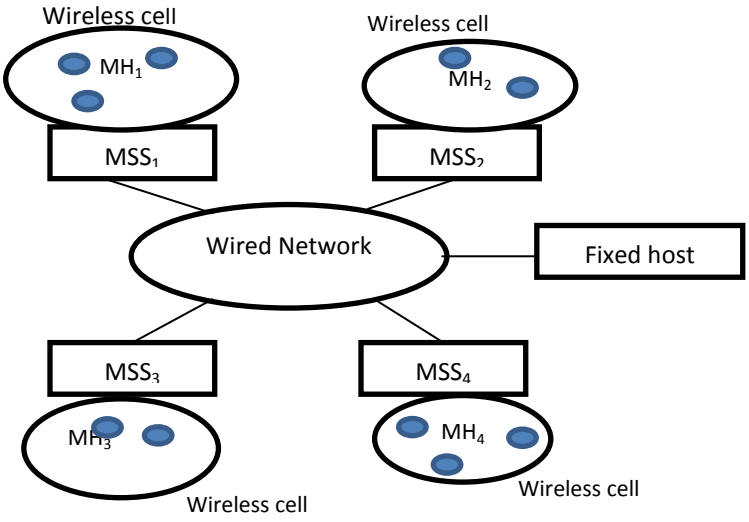

Figure 1: Block diagram of Mobile Distributed System

1.1 Issues or Challenges in designing Algorithms for Mobile Distributed Systems

(1) Mobility: A message sent from MSS to a non-local MH incurs a search cost. Change in location of an MH complicates routing messages. The destination node (MH) disconnected from old MSS and now is connected to new MSS. To handle node mobility the checkpointing algorithm may generate a request for disconnected MH to take its snapshot. Delaying a response to such a request until the MH reconnects with some MSS may significantly increase the completion time [1].

(2) Energy Consumption: The various components like CPU, display, disk drive etc. drain the battery. Message transmission and reception also consumes energy.

(3) Stable Storage Capacity: There is limited storage capacity of MHs system. Each process needs the storage capacity to store the snapshot of the checkpoints. There is Lack of stable storage devices in MH. Rollback recovery uses stable storage to save checkpoints, event logs and other recovery related information.

(4) Bandwidth: Limited communication bandwidth of Network.

(5) Disconnections: Disconnection of one or more MH should not prevent recording the global state of an application executing on MH. The frequent disconnection of MH is an expected feature of the mobile distributed environments.

(6) Synchronization: The energy conservation and low bandwidth constraints require the checkpointing algorithms to minimize the number of synchronization messages and the number of checkpoints [1, 4, 13, 18, 19, 20].

2. SOFTWARE BASED FAULT TOLERANCE APPROACH

In software-based fault, an application is restarted from an earlier checkpoint or recovery point after a failure. This may result in the loss of some processing computation and applications may not be able to meet strict timing targets. Besides providing fault tolerance, check pointing can be used for process migration, debugging distributed applications, job swapping, post-mortem analysis and stable property detection.

Fault Tolerance Approaches

To recovery from failure, a fault tolerance technique is used to bring the system in erroneous state. There are two kinds of recovery approaches: 1) forward recovery 2) backward recovery. Backward recovery approach is used to bring the system back into the previous correct state. Forward recovery approach is used to bring the system in correct new state. There are three steps involved in the forward error recovery. These are:

- Check pointing the error-free state periodically
- Detect that new state is error free
- Restoration in case of failure

The key challenge in the forward recovery is that the number of possible errors should be known in advance. It also requires checking for the redundancy every time while taking new checkpoint. In recovery algorithm each process updates its local state from time to time, so it is difficult to find the recovery line. If the most recently saved state does not form the recovery line, then domino-effect (rollback to initial state of computation losing all the work performed before the failure) can occur. Checkpoint is defined as a chosen place in a program at which a process is interrupted significantly to protect the process status information. So, the solution to this problem is to use the coordinated checkpointing algorithm. Checkpointing is the process of saving the status information by invoking the checkpointing algorithm. The global state (GS) is a collection of the local states of the processes and global checkpoint is a collection of local checkpoints. 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 missing [5]. To recover from a failure one may resume computation from the last checkpoints thereby avoiding repeating computation from the previous consistent global checkpoint. This saves all the computation done up to the last checkpointed state and only the computation done thereafter needs to be redone. The process of resuming computation by rolling back to a saved state is called rollback recovery. The checkpoint-restart is one of the well-known methods to realize dependencies of the processes in distributed systems. When process takes a checkpoint the local state information is need to store in the stable storage. Rolling back a process and again resuming its execution from a prior state involves overhead and delays the overall completion of the process. It is needed to make a process rollback to a most recent possible state. Rollback recovery achieves fault tolerance by periodically saving the state of a process during the failure free execution, and restarting from a saved state on a failure to reduce the amount of lost computation. Rollback recovery can be classified into four categories: uncoordinated or independent checkpointing, coordinated checkpointing, communication induced check-pointing and message logging based checkpointing [23].

2.1 Uncoordinated Checkpointing

In uncoordinated or independent checkpointing, processes do not coordinate their checkpointing activity. There is no coordination required between the processes to take the checkpoint and each process save its local checkpoint independently. Each process is free to decide when to take checkpoint i.e. each process may take a checkpoint when it is most convenient. Thus eliminates synchronization overhead and forms a consistent global state. We can determine the consistent global state by tracking the dependencies among the processes. It may require cascaded rollbacks that may lead to the initial state due to domino-effect. It requires saving multiple checkpoints for each process and periodically invokes garbage collection algorithm to remove the checkpoints that are no longer needed [23]. In this scheme, a process may take a useless checkpoint that will never be a part of global consistent state. Useless checkpoints incur overhead without advancing the recovery line [6].

2.2 Coordinated Checkpointing

In coordinated checkpointing, processes coordinate their checkpointing activities to form a system-wide consistent state. Coordinated checkpointing algorithm is not susceptible to the domino effect. In case of failure, the system state can be restored to such a consistent set of checkpoint (last saved checkpoint), preventing the rollback propagation. This technique has additional overhead at the runtime but it avoids the domino effect at recovery time. This algorithm also requires saving only one checkpoint for each process into the stable storage [23]. The coordinated checkpointing protocols can be classified into two types: blocking and non-blocking. In blocking algorithms**,** blocking of some processes takes place during checkpointing. After taking local checkpoint, to prevent from orphan messages, communication is blocked until the entire checkpointing activity is complete. The disadvantage of this approach is that the no computation can be done during blocking period. So, non-blocking

checkpointing algorithm is preferable. In non-blocking algorithms, no blocking of processes is required for checkpointing. In this scheme processes do not block its computation during checkpointing. The coordinated checkpointing algorithms can also be classified into following two categories: minimum-process and all process algorithms. [7, 9, 11, 13, 14].

2.3 Quasi-Synchronous or Communication Induced Checkpointing

Communication-induced checkpointing avoids the domino-effect without coordination of checkpointing activity. In these protocols, processes take two kinds of checkpoints, local and forced. Local checkpoints can be taken autonomously, while forced checkpoints are taken to promise the eventual progress of the recovery line and to minimize useless checkpoints. As opposed to coordinated checkpointing, these protocols do no exchange any special coordination messages to determine when forced checkpoints should be taken. But, they piggyback protocol specific information on each application message; the receiver uses this information to decide whether it should take a forced checkpoint. This decision is based on the receiver determining if past communication and checkpoint patterns can lead to the creation of useless checkpoints, a forced checkpoint is taken to break these patterns [6, 21].

2.4 Message Logging Based Checkpointing

Message-logging protocols are popular for building systems that can bear process crash failures. Message logging based checkpointing can be used to offer fault tolerance in distributed systems in which all inter-process communication is through messages. Checkpoints are taken such that construction of a consistent checkpoint at recovery is simple, efficient, and fast and domino effect is avoided. Each message received by a process is saved in message log on stable storage. No coordination is required between the checkpointing of different processes or between message logging and checkpointing. The execution of each process is assumed to be deterministic between received messages, and all processes are assumed to execute on fail stop processes. When a process crashes, a new process is created in its place. The new process is given the appropriate recorded local state, and then the logged messages are replayed in the order the process originally received them. All message logging protocols require that once a crashed process recovers, its state needs to be consistent with the states of the other processes [6, 22].

A good checkpointing algorithm for mobile distributed systems should have low memory overheads on MHs, low overheads on wireless channels and should avoid awakening of an MH in doze mode operation. The disconnection of an MH should not lead to infinite wait state. The algorithm should be non-intrusive and should force minimum number of processes to take their local checkpoints [3, 4]

Table 1 Comparison between checkpointing approaches

Checkpointing Approaches	**Advantages**	**Disadvantage**
Uncoordinated Checkpointing	• Eliminate synchronization overhead • Lower run time overhead during execution.	• Domino effect • Recovery from failure is slow. • Regular iteration is required to find the consistent global checkpoint. • Take useless without any coordination. • Need to invoke the garbage collection algorithm every time to eliminate the useless checkpoint.
Coordinated Checkpointing	• Process coordinates the checkpointing activity. • Not susceptible to domino effect. • Maintain only single checkpoint for each process. • Reduce storage overhead • Eliminate the need for garbage collection algorithm.	• Large delay in committing the output • Global checkpoint is needed before sending the message to the outside world process. • Delay and overhead in taking new global checkpoint.
Communication induced checkpointing	• Avoid domino effect, while allowing processes to take their local checkpoint independently. • Eliminate useless checkpoints.	• Processes are forced to take additional checkpoint to advance the global recovery line.
Message Logging based Checkpointing	• Improve efficiency	• Incorrect replay of messages can cause orphan messages.

3. GUIDELINES FOR DESIGNING CHECKPOINTING ALGORITHM FOR MDS

A checkpoint algorithm for mobile distributed systems (MDS) needs to handle many new issues like: mobility, low bandwidth of wireless channels, and lack of stable storage on mobile nodes, disconnections, limited battery power and high failure rate of mobile nodes. These issues make conventional checkpointing techniques unsuitable for such environments. The objective of the research is to design checkpointing schemes for mobile distributed systems with the following features:

- The checkpointing scheme should be able to handle frequent aborts. The loss of checkpointing effort, when any process fails to take its checkpoint in coordination with others, should be low.
- The synchronization message overhead should be low.
- The checkpointing scheme should be applicable to deterministic as well as non-deterministic events.
- If the scheme is blocking, then the blocking time should be negligibly small. Otherwise, if the scheme is non-blocking, the number of useless checkpoints should be very low.
- The checkpointing scheme should be free from domino effect.
- Processes should be able to take checkpoint independently without any domino-effect.

4. REFERENCES

[1] Acharya A. and Badrinath B. R., "Checkpointing Distributed Applications on Mobile Computers," Proceedings of the 3rd International Conference on Parallel and Distributed Information Systems, pp. 73-80, September 1994.

[2] Cao G. and Singhal M., "On coordinated checkpointing in Distributed Systems", IEEE Transactions on Parallel and Distributed Systems, vol. 9, no.12, pp. 1213-1225, Dec 1998.

[3] Cao G. and Singhal M., "On the Impossibility of Min-process Non-blocking Checkpointing and an Efficient Checkpointing Algorithm for Mobile Computing Systems," Proceedings of International Conference on Parallel Processing, pp. 37-44, August 1998.

[4] Cao G. and Singhal M., "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.

[5] Chandy K. M. and Lamport L., "Distributed Snapshots: Determining Global State of Distributed Systems," ACM Transaction on Computing Systems, vol. 3, No. 1, pp. 63-75, February 1985.

[6] Elnozahy E.N., Alvisi L., Wang Y.M. and Johnson D.B., "A Survey of Rollback-Recovery Protocols in Message-Passing Systems," ACM Computing Surveys, vol. 34, no. 3, pp. 375-408, 2002.

[7] Elnozahy E.N., Johnson D.B. and Zwaenepoel W., "The Performance of Consistent Checkpointing," Proceedings of the 11th Symposium on Reliable Distributed Systems, pp. 39-47, October 1992.

[8] Higaki H. and Takizawa M., "Checkpoint-recovery Protocol for Reliable Mobile Systems," Trans. of Information processing Japan, vol. 40, no.1, pp. 236-244, Jan. 1999.

[9] Koo R. and Toueg S., "Checkpointing and Roll-Back Recovery for Distributed Systems," IEEE Trans. on Software Engineering, vol. 13, no. 1, pp. 23-31, January 1987.

[10] Neves N. and Fuchs W. K., "Adaptive Recovery for Mobile Environments," Communications of the ACM, vol. 40, no. 1, pp. 68-74, January 1997.

[11] Parveen Kumar, Lalit Kumar, R K Chauhan, V K Gupta "A Non-Intrusive Minimum Process Synchronous Checkpointing Protocol for Mobile Distributed Systems" Proceedings of IEEE ICPWC-2005, pp 491-95, January 2005.

[12] Pradhan D.K., Krishana P.P. and Vaidya N.H., "Recovery in Mobile Wireless Environment: Design and Trade-off Analysis," Proceedings 26th International Symposium on Fault-Tolerant Computing, pp. 16-25, 1996.

[13] Prakash R. and Singhal M., "Low-Cost Checkpointing and Failure Recovery in Mobile Computing Systems," IEEE Transaction On Parallel and Distributed Systems, vol. 7, no. 10, pp. 1035-1048, October1996.

[14] L. Kumar, M. Misra, R.C. Joshi, "Low overhead optimal checkpointing for mobile distributed systems" Proceedings. 19th IEEE International Conference on Data Engineering, pp 686 – 88, 2003.

[15] Ni, W., S. Vrbsky and S. Ray, "Pitfalls in Distributed Nonblocking Checkpointing", Journal of Interconnection Networks, Vol. 1 No. 5, pp. 47-78, March 2004.

[16] L. Lamport, "Time, clocks and ordering of events in a distributed system" Comm. ACM, vol.21, no.7, pp. 558-565, July 1978.

[17] Parveen Kumar, Lalit Kumar, R K Chauhan, "A Non-intrusive Hybrid Synchronous Checkpointing Protocol for Mobile Systems", IETE Journal of Research, Vol. 52 No. 2&3, 2006.

[18] Parveen Kumar, "A Low-Cost Hybrid Coordinated Checkpointing Protocol for mobile distributed systems", Mobile Information Systems. pp 13-32, Vol. 4, No. 1, 2007.

[19] Lalit Kumar Awasthi, Parveen 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.

[20] Sunil Kumar, R K Chauhan, Parveen 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).

[21] A.Tanenbaum and M. Van Steen, Distributed Systems: Principles and Paradigms, Upper Saddle River, NJ, Prentice-Hall, 2003.

[22] M. Singhal and N. Shivaratri, Advanced Concepts in Operating Systems, New York, McGraw Hill, 1994.

[23] E.N. Elnozahy, L. Alvisi, Y.M. Wang, and D.B. Johnson, Asurvey of reollback-recovery protocols in message-passing system, ACM Computing Surveys, 34(3), 2002, 375-408.