Volume 5, Issue 5, May 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Three Phase Non-blocking Checkpointing Algorithm for Mobile Distributed System

Deepika, Gaurav Garg, Parveen Kumar
Department of CSE, Amity University,
Haryana, India

Abstract: ***Mobile computing raises many new issues, such as lack of stable storage, low bandwidth of wireless channels, high mobility and limited battery life. These issues make traditional checkpointing algorithms unsuitable for checkpointing mobile distributed systems. Checkpointing can be coordinated, uncoordinated, or communication-induced. Coordinated checkpointing is good approach to introduce fault tolerance in a distributed system transparently. most of the algorithm are blocking and two phase algorithm. To remove the problem of blocking, we propose non-blocking three phase commit algorithm, wherein minimum process will take checkpoint and process will not be blocked in the process of checkpointing.***

Keywords: Fault tolerance, Mobile hosts (MHs), Mobile Support Station (MSS), consistent global state, domino effect.

I. INTRODUCTION

A Distributed system is a collection of nodes or processes that communicate with each other by exchanging messages. A mobile computing system is a distributed system where some processes are running on mobile hosts (MHs) that can move. To communicate with MHs, mobile support stations (MSSs) are added. An MSS communicates with other MSSs by wired or wireless network, but it communicates with MHs by wireless networks. A "cell" is a geographical coverage area of an MSS in which it can support an MH. An MSS acts as an interface between the static network and a part of the mobile network. Static nodes are connected by a high speed wired network [1].

Due to the mobility of MHs and the constraints of wireless networks, there are some new issues that complicate the design of checkpointing algorithms [2]:

Changes in the location of an MH complicate the routing of messages. Messages sent by an MH to another MH may have to be rerouted since the destination MH may have changed. Although different routing protocols [2] apply different techniques to address the mobility, generally speaking, locating an MH increases the communication delay and message complexity.

Due to the vulnerability of mobile computers 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 [1] is to utilize the stable storage at the 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. Since the wireless network has low bandwidth and the MHs have relatively low computation power, the checkpointing algorithm should only force a minimum number of processes to take checkpoints.

- The battery at the MH has limited life. To save energy, the MH can power down individual components during periods of low activity [5]. This strategy is referred to as the doze mode operation.
- The MH in sleeping mode is awakened on receiving a message. Therefore, energy conservation and low bandwidth constraints require the checkpointing algorithm to minimize the number of synchronization messages.
- MHs may disconnect from the network temporarily or permanently. The disconnection of MHs should not prevent the checkpointing process.

A. Checkpointing

A checkpoint is a local state of a process saved on the stable storage. In a distributed system, 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. The state of channels corresponding to a global state is the set of messages sent but not yet received. 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 [5]. To recover from a failure, the system restarts its execution from the previous consistent global state saved on the stable storage during fault-free execution. This saves all the computation done up to the last checkpointed state and only the computation done thereafter needs to be redone.

A-I. Uncoordinated Checkpointing

In uncoordinated or independent checkpointing, processes do not send checkpoint message to each other and each process records its local checkpoint independently [11]. In this technique each process has its own right to decide when

to take checkpoint, i.e., each process may take a checkpoint when it is most convenient. It eliminates coordination overhead all together and forms a consistent global state on recovery after a fault [11]. After a failure, a consistent global checkpoint is established by tracking the dependencies. It may require cascaded rollbacks that may lead to the initial state due to domino-effect . It requires multiple checkpoints to be saved for each process and Periodically invokes garbage collection algorithm to reclaim the checkpoints that are no longer needed. In this scheme, a process may take a useless checkpoint that will never be a part of global consistent state.

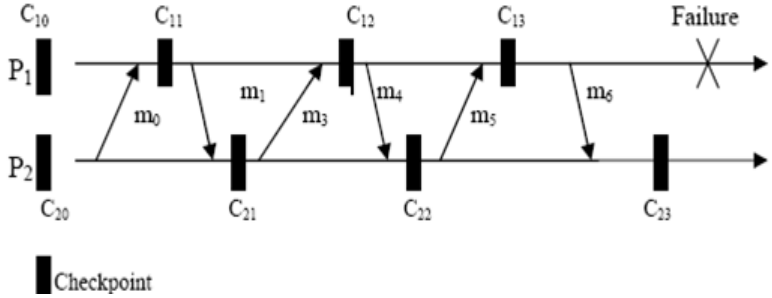

Figure 1: Domino Effect

Useless checkpoints incur overhead without advancing the recovery line [3]. The main disadvantage of this approach is the domino-effect [Figure 1]. In this example, processes P1 and P2 have independently taken a sequence of checkpoints. The interleaving of messages and checkpoints leave no consistent set of checkpoints for P1 and P2, except the initial one at {C10, C20). Consequently, after P1 fails, both P1 and P2 must roll back to the beginning of the computation [32]. 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.

A-II. Coordinated Checkpointing

In coordinated or synchronous checkpointing, one process initiate checkpointing and send message to other process to take checkpoint. In this, processes take checkpoints in such a manner that the resulting global state is consistent. Mostly it follows two-phase commit structure [11], [8]. 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. It guarantees that the computation needed to reach the checkpointed state will not be repeated. A tentative checkpoint, however, can be undone or changed to be a permanent checkpoint. The coordinated checkpointing protocols can be classified into two types: blocking and non-blocking. In blocking algorithms, process get blocked after taking the checkpoint [4]. In non-blocking algorithms, no processes get blocked for taking checkpoint [11]. 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 [11]. In minimum-process algorithms, it is not required to take checkpoint by all process, only minimum interacting processes are required to take their checkpoints in an initiation [4].

A-III. Message Logging Based Checkpointing Protocols

Message-logging protocols (for example [4], are popular for building systems that can tolerate process crash failures. Message logging and checkpointing can be used to provide fault tolerance in distributed systems in which all inter-process communication is through messages. 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.
Coordinated checkpointing is a commonly used technique to prevent complete loss of computation upon a failure. In this approach, the state of each process in the system is periodically saved on the stable storage, which is called a checkpoint of the process. To recover from a failure, the system restarts its execution from a previous consistent global checkpoint saved on the stable storage. 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 [12]. This saves all the computation done up to the last checkpointed state and only the computation done thereafter needs to be redone.
In the present study, we design a three phase commit checkpointing algorithm for mobile distributed systems, where minimum process will take the checkpoint. In this way, we try to balance the checkpointing overhead and the loss of computation on recovery. We also reduce the piggybacked information onto each computation message. For minimum-process checkpointing, we propose a non- blocking algorithm, where processes are allowed to take forced checkpoint before processing the message and process will not be blocked during checkpointing process..

The rest of the paper is organized as follows. We in Section 2, we describe some related work. The Proposed algorithm is provided in Section 3.Section 4 presents conclusions.

II. RELATED WORK

Guohong C. and Singhal M.,[4] had introduced the concept of "Mutable Checkpoint" which is neither a tentative checkpoint nor a permanent checkpoint to design efficient checkpointing algorithms for mobile computing system. Mutable Checkpoint can be saved anywhere e.g. the main memory or local disk of MHs. Taking a mutable checkpoint avoids the overhead of transferring large amount of data to stable storage at MSSs over the wireless network. This algorithm presents techniques to minimize the number of mutable checkpoints Based on Mutable Checkpoints. The Simulation results show that the overhead of taking mutable checkpoints is negligible. Their proposed algorithm is based on Mutable Checkpoint which is non blocking algorithm avoids the avalanche effect and forces only a minimum number of processes to take their checkpoints on the stable storage.

KHATRI Y., [3] proposed a message- induced checkpointing scheme with the exception that processes will not take any basic checkpoints. A process will take a checkpoint only when it receives a message from the other process. Here apart from this message-induced checkpoint, a new kind of forced checkpoint is incorporated to ensure a small re-execution of the processes after recovery from a failure.

The proposed algorithm runs periodically to find out the set of globally consistent checkpoints (GCC). In case of failure, participating processes just need to rollback to their previous checkpoints. Recovery is as simple as in case of synchronous approach and further no coordination is required among processes.

TANTIKUL T. and MANIVANNAN D., [4] We present a communication-induced checkpointing and recovery algorithm with selective message logging that is suitable for mobile computing system. The recovery algorithm is fully asynchronous. A failed process can roll back independently to its latest checkpoint and continue its computation without waiting for other processes to roll back to a consistent checkpoint. Since the local disks of mobile hosts are not reliable, checkpoints are stored at the mobile support stations.

KUMAR P.[11], conclude that In minimum-process checkpointing, some processes, having low communication activity, may not be included in the minimum set for several checkpoint initiations and thus may not advance their recovery line for a long time. In the case of a recovery after a fault, this may lead to their rollback to far earlier checkpointed state and the loss of computation at such processes may be exceedingly high. Furthermore, due to scarce resources of MHs, this loss of computation may be undesirable. In all-process checkpointing, recovery line is advanced for each process after every global checkpoint but the checkpointing overhead may be exceedingly high, especially in mobile environments due to frequent checkpoints. MHs utilize the stable storage at the MSSs to store checkpoints of the MHs [1]. Thus, to balance the checkpointing overhead and the loss of computation on recovery, we design a hybrid checkpointing algorithm for mobile distributed systems, where an all-process checkpoint is taken after certain number of minimum-process checkpoints. The number of times, the minimum-process checkpointing algorithm is executed, depends on the particular application and environment and can be fine-tuned.

KUMAR P., AND GARGR., [8] Proposed the concept of Soft Checkpoint based Coordinated Checkpoint protocol. In this algorithm Suppose, during the execution of the checkpointing algorithm, Pi takes its checkpoint and sends m to Pj. Pj receives m such that it has not taken its checkpoint for the current initiation and it does not know whether it will get the checkpoint request or not. If Pj takes its checkpoint after processing m, m will become orphan. In order to avoid such orphan messages, we use the following technique as mentioned in .

If Pj has sent at least one message to a process, say Pk, and Pk is in the tentative minimum set, there is a good probability that Pj will get the checkpoint request. Therefore, Pj takes its mutable checkpoint before processing m [4]. In this case, most probably, Pj will get the checkpoint request and its mutable checkpoint will be converted into permanent one. Alternatively, this message is buffered Pj. Pj will process m only after taking its tentative checkpoint or after getting commit as in.

BISWAS S. and NEOGY S., [10] Proposed an algorithm for mobile distributed system by considering movement pattern of mobile host. Movement pattern of MHs may be of three types: i) intercell ii) intracell iii) combination of the two.

It is their observation that hand-off is not only dependent on mobility rate but also on movement pattern. If an MH moves in intercell movement pattern hand-off rate will be proportional with mobility rate hence checkpointing should be hand-off based. If an MH moves in intracell movement pattern no hand-off will occur hence hand-off based checkpointing will not serve the purpose.

Periodic checkpointing will work in such cases where interval of checkpoints will be chosen based on average failure rate of mobile hosts. Checkpoint interval is inversely proportional to failure rate .

III. PROPOSED WORK

A. System Model

Our system model is similar to [13]. It consist of *n* spatially separated sequential processes P_0, P_1,.., P_{n-1}, running on MHs or MSSs, constituting a mobile distributed computing system. Each MH/MSS has one process running on it. The processes do not share memory or clock. Message passing is the only way for processes to communicate with each other. Each process progresses at its own speed and messages are exchanged through reliable channels, whose transmission delays are finite but arbitrary. A process in the cell of MSS means the process is either running on the MSS or on an MH supported by it. It also includes the processes of MHs, which have been disconnected from the MSS but their checkpoint related information is still with this MSS. We also assume that the processes are non-deterministic. The i^{th} CI (checkpointing interval) of a process denotes all the computation performed between its i^{th} and $(i+1)^{th}$ checkpoint, including the i^{th} checkpoint but not the $(i+1)^{th}$ checkpoint.

When an MH sends an application message, it is first sent to its local MSS over the wireless cell. The MSS piggybacks appropriate information with the application message, and then routes it to the destination MSS or MH. When the MSS receives an application message to be forwarded to a local MH, it first updates the data structures that it maintains for the MH, strips all the piggybacked information, and then forwards the message to the MH. Thus, an MH sends and receives application messages that do not contain any additional information; it is only responsible for checkpointing its local state appropriately and transferring it to the local MSS.

B. Basic Idea

The proposed Scheme is based on keeping track of direct dependencies of processes. Initiator MSS collects the direct dependency vectors of all processes, computes the tentative minimum set [subset of the minimum], and sends the checkpoint request along with the minimum set to all MSS's. This will reduce the time to collect the coordinated checkpoint. It will also reduce the number of useless checkpoints and the blocking of the processes.

Suppose, during the execution of the checkpointing algorithm, Pi takes its checkpoint and sends m to Pj. Pj receives m such that it has not taken its checkpoint for the current initiation and it does not know whether it will get the checkpoint request. If Pj takes its checkpoint after processing m, m will become orphan. In order to avoid such orphan messages, we present the following technique. If Pj has received a message before receiving the minimum set then it will take forceful checkpoint before processing the message. when Pj will receive the minimum set then it will check whether it is minimum set or not, if yes then it will convert forceful checkpoint to volatile checkpoint otherwise discard it. If a process receive the message after receiving the minimum set then it will check whether it is in minimum set or not, if yes then it will take chekpoint before processing the message otherwise it will process the message. In this way, we have tried to minimize the number of useless checkpoints and the blocking of the processes. Exact dependencies among processes are maintained. It abolishes useless checkpoint requests and reduces duplicate checkpoint requests.

C. Algorithm

First phase of the algorithm

When a process, say Pi, running on an MH, say MHi, initiates a checkpointing, it sends a checkpoint initiation request to its local MSS. The initiator MSS maintains the dependency vector of Pi. On the basis of dependency vector, the set of dependent processes of Pi form the minimum set. The initiator MSS broadcasts volatile checkpoint request to all MSSs. When an MSS receive checkpoint request along with minimum set, it checks, if any processes in minimum set are in its cell. If so, the MSS broadcast the volatile checkpoint request message to all MH. The processes which are included in the minimum set will take volatile checkpoint and sends a response to its local MSS. After receiving response messages from the processes the local MSS sends a response to the initiator MSS. In the first phase, all process takes the volatile checkpoints instead of tentative checkpoint. Volatile checkpoint is equivalent to temporary checkpoint for a process running on a static host, but different for processes running on MHs. To take a tentative checkpoint an MH has to record its local state and need to transfer it to its local MSS, but the volatile checkpoint is stored on the local disk of the MH and the cost of taking volatile checkpoint is very small as compared to the tentative one. For a disconnected MH that is a member of minimum set, the MSS that has its disconnected checkpoint, considers its disconnected checkpoint as the required come.

Second Phase of the Algorithm

After receiving response at the initiator MSS from every MSS algorithm enters the second phase. If the initiator MSS comes to know that all relevant processes have taken their volatile checkpoints successfully, it asks them to convert their volatile checkpoints into tentative ones. Alternatively, if initiator MSS comes to know that some process has failed to take its volatile checkpoint in the first phase, it issues abort request to all MSS. In this approach the MHs need to abort only the volatile checkpoints not the tentative ones. In this algorithm we try to reduce the cost of checkpointing effort in case of abort of checkpointing algorithm in first phase.

Third Phase of the Algorithm

Finally, when the initiator MSS comes to know that all processes in the minimum set have taken their tentative checkpoints successfully, it send commit request to all MSSs. When a process in the minimum set gets the commit request, it converts its tentative checkpoint into permanent one and discards its earlier permanent checkpoint, if any.

D. An Example

Suppose we have 6 Process. P2 is dependent on P1 and P3 and P3 is dependent on P1. At time t1 P2 initiate checkpointing process. It broadcast the message for dependency vector. All process send their dependency vector. P2 calculate the minimum set and broadcast the volatile checkpoint request along with minimum set(not shown in Figure).In the mean time P4 send the message to P3. P3 has not received minimum set. so P3 does not know whether it will be i the minimum set or not. P3 will take forceful checkpoint before processing the message and when it will receive minimum set it will convert forceful checkpoint to volatile one. At time t2, P2 will send the volatile checkpoint request. now P1 has send the message m4 to p6 after taking volatile checkpoint. p6 receive the message after receiving the minimum set. P6 does not find itself in minimum set, so it will not take the checkpoint and process the message. At time t3, P2 will send the commit message to all the process. The process which has taken volatile checkpoint will convert into tentative one. After receiving the positive response from all the process P2 will send message for converting tentative checkpoint to permanent one.

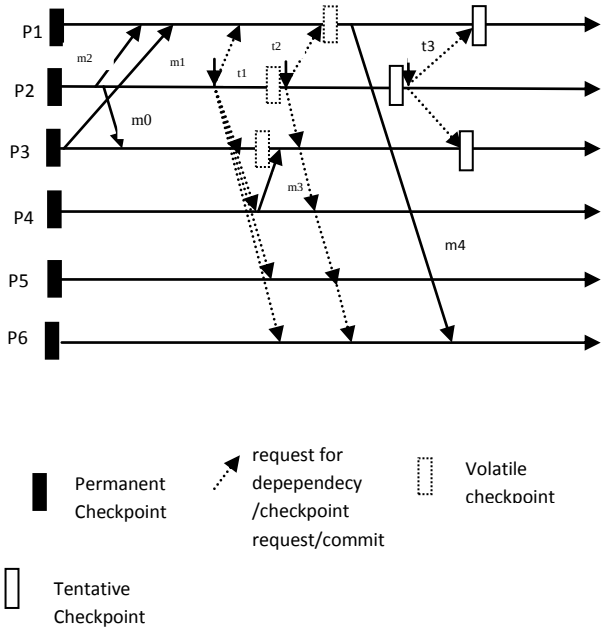

Figure 2: Example of Three phase

IV. CONCLUSION

Mobile computing is a rapidly emerging trend in distributed computing. A mobile computing system consists of mobile hosts (MHs) and mobile support stations (MSSs), connected by a communication network. The mobility of MHs in mobile computing systems generates many new constraints, such as handoffs, lack of stable storage, low communication bandwidth of a wireless channel, and energy conservation, which make the traditional checkpointing algorithm unsuitable. In this paper, we propose a non-blocking three phase commit checkpoint algorithm for mobile distributed system. Volatile Checkpoint need not to be store on MSS, it can be saved on the local disk of MH. By this technique we have reduce the overhead of MSS.

REFRENCES

[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., "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.

[3] Khatri Yogita, "An Optimized Approach to Checkpointing used for Recovery in Mobile Environment", IEEE 978-1-4673-1989-8/12, 2012.

[4] Tantikul T. and Manivannan D., "A Communication-Induced Checkpointing and Asynchronous Recovery Protocol for Mobile Computing Systems", Proceedings of the Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'05) IEEE 0-7695-2405-2/05, 2005.

[5] G.H. Forman and J. Zahorjan, "The Challenges of Mobile Computing", Computer, pp. 38-47, Apr. 1994.

[6] G. Cao and M. Singhal, "Low-Cost Checkpointing with Mutable Checkpoints in Mobile Computing Systems", Proc. 18th Int'l Conf. Distributed Computing Systems, pp. 464-471, May 1998.

[7] G. Cao and M. Singhal, "On Coordinated Checkpointing in Distributed Systems", IEEE Trans. Parallel and Distributed System pp. 1213-1225, Dec. 1998.

[8] P. Kumar and R. Garg "Soft Checkpoint based Coordinated Checkpoint protocol for Mobile Distributed System" in IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 3, No 5, May 2010.

[9] 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*, January 2005.

[10] S.Biswas and S. Neogy "A Mobility based Checkpoint protocol for Mobile Computing System" in IJCSIT, Vol 2, No.1, Feburary 2010.

[11] P. Kumar , "A Low-Cost Hybrid Coordinated Checkpointing Protocol for Mobile Distributed Systems" *Mobile Information Systems* [An International Journal from IOS Press, Netherlands] pp 13-32, Vol. 4, No. 1, 2007.

[12] Adnan Agbaria, William H. Sanders, "Distributed Snapshots for Mobile Computing Systems", Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications (Percom'04), pp. 1-10, 2004.

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

