An Efficient Synchronous Checkpointing Protocol for Mobile Distributed Systems

Parveen Kumar¹ Rachit Garg²

Abstract—Recent years have witnessed rapid development of mobile communications and become part of everyday life for most people. In order to transparently adding fault tolerance in mobile distributed systems, Minimum-process coordinated checkpointing is preferable but it may require blocking of processes, extra synchronization messages or taking some useless checkpoints. All-process checkpointing may lead to exceedingly high checkpointing overhead. In order to balance the checkpointing overhead and the loss of computation on recovery, we propose a hybrid checkpointing algorithm, wherein an all-process coordinated checkpoint is taken after the execution of minimum-process coordinated checkpointing algorithm for a fixed number of times. In the minimum-process coordinated checkpointing algorithm; an effort has been made to optimize the number of useless checkpoints and blocking of processes using probabilistic approach and by computing an interacting set of processes at beginning. We try to reduce the loss of checkpointing effort when any process fails to take its checkpoint in coordination with others. We reduce the size of checkpoint sequence number piggybacked on each computation message.

I. BACKGROUND

Recent years have witnessed rapid development of mobile communications and become part of everyday life for most people. In the future, we will expect more and more people will use some portable units such as notebooks or personal data assistants. With increasing use small portable computers, wireless networks and satellites, a trend to support “Computing of the move” has emerged. This trend is known as mobile computing or “anytime” or “anywhere” computing. This enables the user to access and exchange information while they travel, roam in their home environments, or work at their desktop computers. Mobile Hosts (MHs) are increasingly becoming common in distributed systems due to their availability, cost, and mobile connectivity. An MH is a computer that may retain its connectivity with the rest of the distributed system through a wireless network while on move. An MH communicates with the other nodes of the distributed system via a special node called mobile support station (MSS). A “cell” is a geographical area around an MSS in which it can support an MH. An MSS has both wired and wireless links and it 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].

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.

In independent checkpointing, processes do not synchronize their checkpointing activity and processes are allowed to records their local checkpoints in an independent way. After a failure, system will search a consistent global state 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. It may require cascaded rollbacks that may lead to the initial state due to domino effect [6]. Acharya and Badrinath[1] were the first who present a uncoordinated checkpointing algorithm for mobile computing systems. In their algorithm, an MH takes a local checkpoint whenever a message reception is preceded by a message sent at that MH. If the send and receive of messages are interleaved, the number of local checkpoints will be equal to half of the number of computation messages, which may degrade the system performance.

In coordinated or synchronous checkpointing, processes take checkpoints in such a manner that the resulting global state is consistent. Mostly it follows the two-phase commit structure [2], [5], [6], [7], [10], [15]. 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 the case of a fault, processes rollback to the last checkpointed state [6]. The Chandy-Lamport [5] algorithm is the earliest non-blocking all-process coordinated checkpointing algorithm.

The existence of mobile nodes in a distributed system introduces new issues that need proper handling while designing a checkpointing algorithm for such systems [1], [4], [14], [16]. These issues are mobility, disconnections,
finite power source, vulnerable to physical damage, lack of stable storage etc. Prakash and Singhal [14] proposed a nonblocking minimum-process coordinated checkpointing protocol for mobile distributed systems. They proposed that a good checkpointing protocol for mobile distributed systems should have low overheads on MHs and wireless channels; and it 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 it should force minimum number of processes to take their local checkpoints. In minimum-process coordinated checkpointing algorithms, some blocking of the processes takes place [3], [10], [11], or some useless checkpoints are taken [4], [15].

In minimum-process coordinated checkpointing algorithms, a process Pi takes its checkpoint only if it a member of the minimum set (a subset of interacting process). A process Pi in the minimum set only if the checkpoint initiator process is transitively dependent upon it. Pj is directly dependent upon Pk only if there exists m such that Pj receives m from Pk in the current checkpointing interval [CI] and Pk has not taken its permanent checkpoint after sending m. The ith CI of a process denotes all the computation performed between its ith and (i+1)th checkpoint, including the ith checkpoint but not the (i+1)th checkpoint.

The koo-Toueg[10] proposed a minimum process coordinated checkpointing algorithm for distributed systems with the cost of blocking of processes during checkpointing. However, this algorithm requires minimum number of synchronization message and number of checkpoints but each process uses monotonically increasing labels in its outgoing messages. The initiator process sends the checkpoint request to Pi only if it has received m from Pi in the current CI. Similarly, Pi sends the checkpoint request to other processes. In this way, a checkpointing tree is formed and at last the leaf node processes take checkpoints. The time taken to collect coordinated checkpoint in mobile systems may be too large due to mobility, disconnections and unreliable wireless channels. The extensive blocking of processes may degrade the system performance. Cao and Singhal [4] achieved non-intrusiveness in the minimum-process algorithm by introducing the concept of mutable checkpoints. Kumar and Kumar [21] proposed a minimum-process coordinated checkpointing algorithm for mobile distributed systems, where the number of useless checkpoints and the blocking of processes are reduced using a probabilistic approach. Singh and Cabilllic [20] proposed a minimum-process non-intrusive coordinated checkpointing protocol for deterministic mobile systems, where anti-messages of selective messages are logged during checkpointing. Higaki and Takizawa [8], and Kumar et al [17] proposed hybrid checkpointing protocols where MHs checkpoint independently and MSSs checkpoint synchronously. Neves et al. [13] gave a time based loosely synchronized coordinated checkpointing protocol that removes the overhead of synchronization and piggybacks integer csn (checkpoint sequence number). Pradhan et al [19] had shown that asynchronous checkpointing with message logging is quite effective for checkpointing mobile systems.

Most of the proposed checkpointing algorithms do not addressing the multiple concurrent initiations in their algorithms, as it may exhaust the limited battery and congest the wireless channels. The authors claim in that their algorithm supports concurrent initiations [4]. But in[15] authors proves that the algorithm in[4] is designed to only handle the situation where the system has only one checkpoint initiator at a time and can cause inconsistency when there are multiple forced checkpoints or multiple concurrent checkpoint initiations. In[22], the author point out following problems in allowing concurrent initiations in minimum-process checkpointing protocols, particularly in case of mobile distributed systems:

i) If Pi and Pj concurrently initiate checkpointing and Pj belongs to the minimum set of Pi, then Pj’s initiation will be redundant one. Some processes, in Pj’s minimum set, will unnecessarily take multiple checkpoints by hardly advancing their recovery line. In other words, an MH may be asked to store multiple checkpoints in its local disk. It may also transfer multiple checkpoints to its local MSS.

ii) Sometimes, multiple triggers need to be piggybacked onto normal messages. Trigger contains the initiator process identification and its csn. Even if a process takes a checkpoint and no concurrent initiation is going on, it will piggyback its trigger, unnecessarily. If we do not allow concurrent initiation, no trigger is required to be piggybacked onto normal messages. Hence, concurrent initiations increase message size.

Authors [23] have proposed a minimum process coordinated checkpointing algorithm for mobile distributed system, where no useless checkpoints are taken and an effort is made to minimize the blocking of processes. They captured the transitive dependencies during the normal execution. The Z-dependencies are well taken care of in this protocol. They also avoided collecting dependency vectors of all processes to compute the minimum set.

In this paper [24], authors propose a nonblocking coordinated checkpointing algorithm for mobile computing systems, which requires only a minimum number of processes to take permanent checkpoints. They reduce the message complexity as compared to the Cao-Singhal algorithm [4], while keeping the number of useless checkpoints unchanged.

II. INTRODUCTION

The system model is similar to [3], [4]. A mobile computing system consists of a large number of MH’s and relatively fewer MSS’s. The distributed computation we consider consists of n spatially separated sequential processes denoted by P0, P1, ..., Pn-1, running on fail-stop MH’s or on MSS’s. Each MH or MSS has one process running on it. The processes do no share common memory or common 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. We assume the processes to be non-deterministic. Similar to [3], [21], [22] initiator process collects the dependency vectors of all processes and computes the tentative minimum set. 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 use the following technique as mentioned in [21].

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 [22].

In minimum-process checkpointing, some processes may not be included in the minimum set for several checkpoint initiations due to typical dependency pattern; and they may starve for checkpointing. In the case of a recovery after a fault, the loss of computation at such processes may be unreasonably high [22]. In Mobile Systems, the checkpointing overhead is quite high in all-process checkpointing [14]. 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.

In coordinated checkpointing, if a single process fails to take its checkpoint; all the checkpointing effort goes waste, because, each process has to abort its tentative checkpoint. In order to take the tentative checkpoint, an MH needs to transfer large checkpoint data to its local MSS over wireless channels. Hence, the loss of checkpointing effort may be exceedingly high. Therefore, we propose that in the first phase, all concerned MHs will take soft checkpoint only. Soft checkpoint is similar to mutable checkpoint [4], which is stored on the memory of MH only. In this case, if some process fails to take checkpoint in the first phase, then MHs need to abort their soft checkpoints only. The effort of taking a soft checkpoint is negligible as compared to the tentative one. When the initiator comes to know that all relevant processes have taken their soft checkpoints, it asks all relevant processes to come into the second phase, in which, a process converts its soft checkpoint into tentative one. Finally, the initiator issues the commit request.

In the present study, we present a hybrid scheme, where an all process checkpoint is enforced after executing minimum-process algorithm for a fixed number of times as in [22]. In the first phase, the MHs in the minimum set are required to take soft checkpoint only. In the minimum-process algorithm, a process takes its forced checkpoint only if it is having a good probability of getting the checkpoint request as in [21].

III. THE PROPOSED CHECKPOINtING SCHEME

A. An Example

We explain the minimum-process checkpointing algorithm with the help of an example. In Figure 1, at time t1, P1 initiates checkpointing process and sends request to all processes for their dependency vectors. At time t2, P1 receives the dependency vectors from all processes and computes the tentative minimum (mset) set as in [21], which in case of Figure 1 is \{P0, P1, P2\}. P1 sends this tentative minimum set to all processes. A process takes its soft checkpoint if it is a member of the tentative minimum set. When P0 and P2 get the mset, they find themselves in the mset; therefore, they take their soft checkpoints. When P3, P4 and P5 get the mset, they find that they are not its members; therefore, they do not take their checkpoints. P1 sends m8 after taking its checkpoint and P0 receives m8 before getting the mset. In this case, P0 buffers m8 and processes it only after taking its soft checkpoint. After taking its soft checkpoint, P1 sends m11 to P3. At the time of receiving m11, P4 has received the mset and it has not taken its checkpoint, therefore, P4 takes bitwise logical AND of sendv4 and mset and finds the resultant vector is not all zeroes [sendv3[1]=1 due to m3; mset[2]=1]. P3 concludes that most probably, it will get the checkpoint request in the current initiation; therefore, it takes its mutable checkpoint before processing m11. When P2 takes its soft checkpoint, it finds that it is dependent upon P3 and P3 is not in the minimum set [known locally]; therefore, P2 sends checkpoint request to P3. On receiving the checkpoint request, P3 converts its mutable checkpoint into soft one.

After taking its checkpoint, P2 sends m13 to P4. P4 takes the bitwise logical AND of sendv4 and mset and finds the resultant vector to be all zeroes (sendv4=[000001]; mset=[111000]). P4 concludes that most probably, it will not get the checkpoint request in the current initiation; therefore, P4 does not take mutable checkpoint but buffers m13. P4 processes m13 only after getting commit request. P5 processes m14, because, it has not sent any message since last permanent checkpoint. After taking its checkpoint, P1 sends m12 to P2. P2 processes m12, because, it has already taken its checkpoint in the current initiation. At time t3, P1 receives responses from all relevant processes and issues tentative checkpoint request along with the exact minimum set [P0, P1, P2, P3] to all processes. On receiving tentative checkpoint request, all relevant processes convert their soft checkpoints into tentative ones and inform the initiator. Finally, at time t4, initiator P2 issues commit. On receiving commit following actions are taken. A process, in the minimum set, converts its tentative checkpoint into permanent one and discards its earlier permanent checkpoint, if any. A process, not in the minimum set, discards its mutable checkpoint, if any, or processes the buffered messages, if any.
B. Handling Node Mobility and Disconnections

Suppose, an MH, say MHi, disconnects from an MSS, say MSSk, it stores its own checkpoint, say disconnect_ckpti, and other support information, e.g. ddv, at MSSk. During disconnection period, MSSk acts on behalf of MHi as follows. If checkpointing process is initiated and MHi is in cell of MSSj, it is connected to the MSSj, if g_chkpt is reset. Otherwise, it waits for the g chkpt to be reset. Before connection, MSSj collects its ddv, buffered messages from MSSk; and MSSk discards MHi’s support information and disconnect_ckpti. The buffered messages are processed by MHi, in the order of their receipt at the MSSk. MHi’s ddv is updated on the processing of buffered messages.

Comparison with existing non-blocking algorithm

In Cao-Singhal algorithm [20], suppose, Pi receives m from Pj before taking its checkpoint and Pi is in the minimum set. In this case, after taking its checkpoint, Pi sends checkpoint request to Pj due to m. If Pj has taken some permanent checkpoint request after sending m, the checkpoint request to Pj is useless. To enable Pj to decide whether the checkpoint request is useful, Pj also piggybacks csn[i] and a huge data structure MR along with the checkpoint request to Pj. These useless checkpoint requests and piggybacked data structures increase the message complexity of the algorithm.

Whereas, in our algorithm, no such useless checkpoint requests are sent and no such information is piggybacked onto checkpoint requests. The csn[i] is integer; its size is 4 bytes. In worst case the size of MR is (4n +n/8) bytes (n is the number of processes in the distributed system). Intuitively, we can say that the number of useless checkpoints in the proposed algorithm will be negligibly small as compared to the algorithm [20].

the minimum set, MSSk converts its disconnected checkpoint into permanent one. On global checkpoint commit, MSSk also updates MHi’s ddv, as if, it is a normal process. On the receipt of messages for MHi, MSSk stores them in a queue without updating ddv. When MHi, enters in the.

The proposed protocol suffers from the following limitations with respect to the existing algorithm [20]. Initiator MSS collects dependencies of all processes, computes the tentative minimum set, and broadcasts the tentative minimum set along with the checkpoint request to all MSS’s. Initiator MSS broadcasts exact minimum set along with the commit request on the static network. Blocking of processes also takes place. Concurrent executions of the algorithm are avoided.

IV. Conclusions

We propose a hybrid checkpointing algorithm, wherein an all-process coordinated checkpoint is taken after the execution of minimum-process coordinated checkpointing algorithm for a fixed number of times. In minimum-process checkpointing, we try to reduce number of useless checkpoints and blocking of processes. We have proposed a probabilistic approach to reduce the number of useless checkpoints. Thus, the proposed protocol is simultaneously able to reduce the useless checkpoints and blocking of processes at very less cost of maintaining and collecting dependencies and piggybacking checkpoint sequence numbers onto normal messages. Concurrent initiations of the proposed protocol do not cause its concurrent executions. We try to reduce the loss of checkpointing effort when any process fails to take its checkpoint in coordination with others.
V. REFERENCES


