Checkpointing Techniques for Distributed Mobile Systems

Dr Deepak Uprety[1], Naheeda zaib[2], Saiba Jan[3]
[1,2,3]*Computer science and engineering, Nims University*

Abstract- Checkpoint and rollback recovery are well-known techniques for handling failures in distributed systems. As the number of processors increases, so does the failure rate. Therefore, it is important to develop efficient checkpoint and recovery algorithms to handle such large-scale system failures so that these systems can be fully utilized. We presented a new communication-induced checkpoint algorithm that helps reduce contention in accessing stable memory to store checkpoints.

Keywords: routing, ad-hoc network, communication guidance, checkpoint, distribution system.

1.INTRODUCTION

In our algorithm, processes involved in distributed computation can independently initiate consistent global checkpoints by saving their current state, called temporary checkpoints. Other processes involved in the computation learn to initiate consistent global checkpoints through information piggybacked in application messages or limited control messages as needed [4]. When the process sees the start of a new consistent global checkpoint, it takes a temporary checkpoint after processing the message. Temporary checkpoints taken may be flushed to stable storage if there is no contention for access to stable storage. Preliminary checkpoints, along with message logs stored in stable storage, form a consistent global checkpoint.

2.REVIEW OF LITERATURE

In our algorithm, processes involved in distributed computation can independently initiate consistent global checkpoints by saving their current state, called temporary checkpoints. Other processes involved in the computation learn to initiate consistent global checkpoints through information piggybacked in application messages or limited control messages as needed [4]. When the process sees the start of a new consistent global checkpoint, it takes a temporary checkpoint after processing the message. Temporary checkpoints taken may be flushed to stable storage if there is no contention for access to stable storage. Preliminary checkpoints, along with message logs stored in stable storage, form a consistent global checkpoint.

3.SCOPE OF THE STUDY

The scope of this research is defined as checkpoint-based rollback recovery [10]. It is one of the widely used techniques in various fields such as scientific computing, databases, telecommunications, and critical applications in distributed systems.

4.OBJECTIVES OF THE STUDY

1.System Model

A disbursed computation includes N sequential processes, denoted through P0, P1, P2, ..., and PN-1, jogging concurrently on many computer systems in a network. Processes do now no longer share worldwide reminiscence or the worldwide bodily clock. Message passing is the simplest way processes talk with every other. Computation is asynchronous. Each technique evolves at its personal rate, and messages are transmitted over communique channels with finite however arbitrary transmission delays. The channel is believed to be FIFO and the computation is piecewise deterministic. Our set of rules generates a restricted variety of manipulate messages [22] and collects constant worldwide manipulate factors simplest while needed.

2. Consistent Global Checkpoints

Process execution is modelled by three types of events: message-sending events, message-receiving events,

and internal events. Process states depend on each other through inter communication. The global control points for distributed computation are a set of control points, including control points from each process involved in distributed computation [26]. An orphan message M that impacts the global checkpoint is a message for which a received event (M) has been recorded in the global checkpoint, but a corresponding sent event (M) has not been recorded. A global checkpoint is said to be consistent if there are no orphaned messages associated with it. Figure 5.2 shows two global checkpoints S1 and S2. S1 is a consistent global checkpoint, but M5 is an orphan message relative to S2, so S2 is not a consistent global checkpoint. Next, we introduce the algorithm.

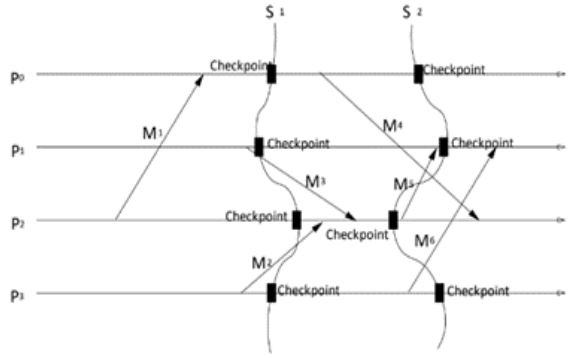

Fig 1 Global Checkpoints

3. Notations for algorithms

Below is the notation used to describe the algorithm and its correctness proof. • Ci, k indicates the (permanent) local control point occupied by Pi. It consists of his two parts, a preliminary checkpoint CTi that records the state of the process, and a set of log messages log Seti associated with the checkpoint. - CTi, k indicates a preliminary control point obtained from Pi with control point sequence number k. It is usually first stored in memory and flushed to stable storage after logging the relevant logs (log Seti, k). - log Seti, k indicates a set containing all messages sent and received by Pi after the preliminary control point CTi, k was acquired and before the control point Ci, kis completed. Therefore, Ci, k= CTi, k∪logSeti, k.• CFEi, k indicates the event representing the last operation of checkpoint Ci, k. Therefore, all messages send/receive events in logSeti, k precede CFEi, k. For every event e of Pi, e−hb→Ci, k⇐⇒e−hb→CFEi, k.A Lamport event, −hb→, that occurred before relation [6] is defined as the transitive closure of the union of the other two relations: −hb→ = (−xo→ ∪ −m→) +. The −xo→ relationship captures the order in which process-local events are executed. The i event (denoted by ep, i) of each process Pp is always executed before the (i + 1) event: ep, i−xo → ep, i+1. The relationship −→m indicates the relationship between send and receive events of the same message. If a is the sending event of a message and b is the corresponding receiving event of the same message, then a−→m is b [23]. • Sk denotes a global checkpoint consisting of checkpoints with sequence number k from each process. Therefore Sk= {Ci, k|i∈ {0, 1, N − 1}}.

4. Basic Idea behind this algorithm

To illustrate the basic idea behind this algorithm, we use the Spatio-temporal diagram of distributed computation [20], which consists of four processes, shown in Figure 5.4. P0, P1, P2, and P3 are the four processes involved in the computation. Initially, the status of each method is normal, and the initial checkpoint with sequence number 0 is indicated by the filled rectangular box in the diagram. Suppose P0 initiates a consistent global checkpoint by taking a preliminary checkpoint CT0,1. After taking checkpoint CT0,1, it changes the status from regular to tentative and starts logging in memory all the messages it sends and receives until that checkpoint exits. P0 then sends message M2 to her P1. Upon receiving M2, P1 signals that P0 has acquired CT0,1. So after processing M2, P1 gets preliminary checkpoint CT1,1, and P1's status changes from normal to introductory.

Similarly, P2 and P3 occupy preliminary checkpoints CT2,1 and CT3,1 after receiving messages M4 and M3, respectively. P1 knows that the status of P0 and P1 is tentative before message M3 is sent. P1 piggybacked this information onto M3.

P3, therefore, knows that the status of P0, P1, and P3 is tentative before message M5 is sent. Upon receiving M5, P2 knows that all processes are in uncertain status. At this point, P2 resumes checkpoint sequence number 1 by deleting preliminary checkpoints CT2,1 (if not already run) and the set of logged messages {M5, M6} to stable storage. Done. And C2,1= CT2,1 ∪ {M5, M6}. The "F" mark in the diagram indicates the event that finalizes the current pre-checkpoint. When a process completes an intermediate checkpoint, its status becomes healthy (after a process has taken an

intermediate checkpoint, it can only take another intermediate checkpoint after it has completed and already taken intermediate checkpoint). Increase). A consistent global checkpoint S1 = {C0,1, C1,1, C2,1, C3,1} was recorded.

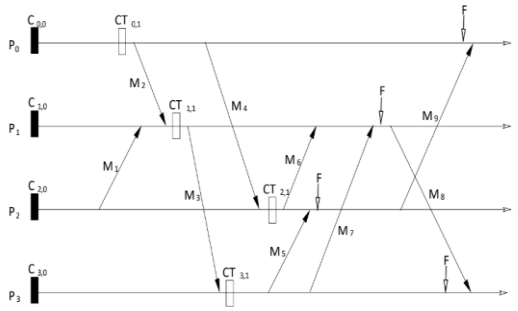

Fig 2 An example illustrating the basic idea behind our algorithm

5 Data Structures

Each process Pi manages the following data structures:

1. csn i: an integer variable containing the sequence number of the current checkpoint of process Pi. The sequence number of the probe representing the initial state of Pi is 0. The Pi first configures csnito0. csni is incremented when a new intermediate checkpoint is taken.

2. stat i: A variable representing the current status of process Pi. Process status can be either provisional or normal. The status of process Pi is updated as follows: The Pi's status is initially set to normal. As soon as the Pi acquires a provisional control point, the Pi's status is changed to provisional. After the Pi realizes that the status of all processes is temporary (through information piggybacked in the application's message), the Pi returns its state to normal after the current temporary checkpoint is completed. return.

3. logSet i: The set of messages logged after the Pi takes a temporary checkpoint. Once the statiis is provisionally set, the Pi will set logSetito to empty and start logging messages sent and received by the Pi to logSeti. The logSet therefore contains messages sent and received by the Pi after the preliminary checkpoint was taken and before the checkpoint was completed. When the process status changes from temporary to normal, the temporary checkpoint and corresponding LogSetia are flushed to stable storage.

4. tentSet i: A tentative set of processes held on the Pi. If statii is set to normal, tentSetii is set to empty. PitentSetito sets {Pi} when Pi captures a tentative control point. When Pi receives the message, it determines that tentSetito is the union of tentSeti and preprocessing is piggybacked on the message. So, this set contains a set of processes that have taken interim checkpoints, to the best of the Pi's knowledge.

5 allPSet: This is the set of all processes i.e. {P0, P1, PN-1}.

5.6 The Checkpointing Algorithm

Assume that every process takes an initial checkpoint that represents the initial state of the process. The initial checkpoint sequence number is set to 0. Also, if the status is transient, the process cannot take a new checkpoint [7].

5.CONSISTENT GLOBAL CHECKPOINTING INITIATION

Any process with a healthy status can take a new temporary checkpoint, thus starting a consistent global checkpoint. When process Pi takes a provisional checkpoint, it changes its state from normal to provisional, increments the checkpoint sequence number csni and assigns it as the provisional checkpoint sequence number, clears logSetito, and initializes tentSetito {Pi}. At any given time, tentSeti is the set of all processes that, to Pi's knowledge, have taken a tentative checkpoint that matches Pi's current tentative checkpoint. After the Pi takes a preliminary checkpoint, it will start logging all sent and received messages to logSet i until the state returns to normal. Csni and tentSetia are piggybacked on all application messages.

Sending Messages

Each process Pi piggybacks the current values of csni, stati and tentSeti into each application message. The csni value piggybacked into the message helps the receiver determine if the sender has taken a new temporary checkpoint. This will start a concurrent or new consistent global checkpoint. These values piggybacked on message M are called M.csn, M. stat, and M. tent Set respectively.

Receiving Messages

In our algorithm, each process can independently occupy preliminary control points at the same time. A process completes an interim checkpoint when it

learns (via a message received from the process) that all other processes have taken interim checkpoints that match the most recent interim checkpoint. After completing the latest preliminary checkpoint Ci, k, process Pi may take the next preliminary checkpoint Ci, k+1 before any other process completes the preliminary checkpoint corresponding to Ci, k. I can do it.

Case (1) M. stat=stati=normal. In this case, neither Pi nor Pji knows about the start of a new consistent global checkpoint, so no additional action needs to be taken beyond processing M.

Case (2) M. stat= stati= tentative. In this case, both Pi and Pj have taken a new tentative checkpoint concurrently. The following four subcases arise:

In subcase (a) M.csn xss=removed xss=removed>csni+ 1. In this case, Pj completed the checkpoint with sequence number csni+ 1. This is impossible because csni is Pi's last temporary checkpoint sequence number. So, this case does not occur. Therefore, this case is not shown in the formal description of the algorithm.

Subcase (b) M.csn= csni. In this case, Pi and Pj have acquired control points that belong to the same global control point Scsni. In this case, to know how many processes M was processed first and then took a temporary checkpoint belonging to the global checkpoint Scsni, Pi is the tent Update set. Once the updated tent set matches all her PSets, the Pi will log a message and complete the preliminary checkpoint.

Subcase (c) M.csn = csni+1. In this case, before sending M, Pj completed a checkpoint with sequence number csni and also took an intermediate checkpoint with sequence number M.csn. So, Pi knows that every process already has a temporary control point that belongs to the global control point Scsni.

Subcase (d) M.csn>csni+ 1. In this case, Pj has completed the control point with sequence number csni+ 1. This is impossible because csni is Pi's last temporary checkpoint sequence number. Thus, this scenario does not materialize. Therefore, this case is not shown in the formal description of the algorithm.

Commits a temporary checkpoint associated with a consistent global checkpoint with the specified sequence number. Preliminary checkpoints, along with the saved message log, are called process checkpoints and are assigned the same sequence number as the saved preliminary checkpoints. A formal description of the basic checkpoint algorithm is shown in Figure 5.6.

When P_i starts

/ Initialization */*

csn_i=0; $stat_i$= *normal*.

Procedure: takeTentativeCheckpoint (i: integer)

csn_i= csn_i+1; $stat_i$= *tentative*;

$tentSet_i$= {P_i}; */* Include the process id in the set */*

logSeti=Ø; */* Initialize the message log to empty set */*

Take tentative checkpoint $CT_{i, csni}$.

When P_i starts to take a checkpoint

takeTentativeCheckpoint (i).

When P_i sends a message *M* to P_j

M.csn= csn_i; */* Piggy-back current value of csn_i, $stat_i$, and $tentSet_i$with the message */*

M.stat= $stat_i$; *M. tentSet*= $tentSet_i$; if $stat_i$== *tentative* then $logSet_i$= $logSet_i$∪ {*M*}; *Send* (*M*).

When P_i receives a message *M* from P_j

if $stat_i$== *normal* then

Process *M*.

if *M. stat*== *tentative* then

if *M.csn*== csn_i+1 then */* $P_j$has initiated a new consistent global checkpoint */*

takeTentativeCheckpoint(i);

$logSet_i$= $logSet_i$∪{*M*}; *tentSeti*= *M.* */* Log the received message */*

tentSet∪$tentSet_i$.

else /*$stat_i$== *tentative* */

$logSet_i$= $logSet_i$∪ {*M*}; if *M. stat*== *normal* /* *Log the received message* */
then
if *M.csn*== csn_ithen /* P_j*has finalized the checkpoint* $C_{j,\ csni}$*/
Flush $logSet_i$− {*M*} and $CT_{i,\ csnito}$ the stable storage;/* P_i *finalizes its checkpoint* $C_{i,\ csni}$*/$stat_i$= *normal*.
Process *M*.
else/* *M. stat*== *tentative* */
if*M.csn*== csn_ithen/* P_j*has taken the checkpoint* $CT_{j,\ csnibefore}$ *sending the message* */
Process *M*.
$tentSet_i$= *M. tentSet*∪$tentSet_i$.
if$tentSet_i$== *allPSet*then /* *Each process* P_k*has already taken* $CT_{k,\ csni}$*/$stat_i$= *normal*.
Flush *logSeti* and *CTi, csni* to the stable storage.
else if*M.csn* == *csni* + 1 then /* *Pj has finalized Cj, csni and took a new tentative checkpoint after that* */
stati = normal; /**So, Pi finalizes Ci, csni, excludes M from the log and takes a new tentative checkpoint* */
Flush *logSeti − {M}* and *CTi, csni* to the stable storage.
Process M.
takeTentativeCheckpoint(i).
logSeti = logSeti∪{M}.
tentSeti = M. tentSet∪tentSeti.

Figure 5.6: The Basic Checkpointing Algorithm

6.CONCLUSION

This article introduced a new communication-induced checkpointing algorithm that makes all checkpoints a consistent global checkpoint. In this algorithm, each process first saves a preliminary checkpoint to memory and removes it to stable memory after there is no contention for access to regular memory or after the initial checkpoint completes. Messages sent or received after the process has taken a preliminary checkpoint are logged in memory until the primary checkpoint is conducted. Previous checkpoints can be flushed to stable storage at any time before they are finished, thus reducing/eliminating network steady storage contention caused by multiple processes saving checkpoints simultaneously. Furthermore, unlike existing communication-directed checkpoint algorithms, this algorithm generally does not force processes to checkpoints before processing received messages to prevent wasted checkpoints. So, a process can process the first received message and then take a checkpoint. This improves message response time. It's also useful for methods to get essential regularly scheduled checkpoints at these times.

REFERENCE

[1] Wikipedia Ipv6

[2] Andrew D. Birreland Bruce J. Nelson. Implementing remote procedure calls. ACM Transactionson ComputerSystems,2(1):39–59, February 1984.

[3] Suparna Biswasand Sarmisthat Neogy". Snapshot and Recovery Using Node Mobility among Clusters in Mobile Ad Hoc Network" springer "Volume 176 of the series Advances in Intelligent Systems and Computing pp 447-456, 2010

[4] Yogita Khatri "Distance-based Asynchronous recovery approach in mobile computing environment" International Journal of Distributed and Parallel Systems (IJDPS) Vol.3, No.3, May 2012

[5] P.K. Suri and Menu Satya "An efficient snapshot protocol for mobile distributed systems", cilistol 1, issue 2: page no 109-114 (2012)

[6] K. Mani Chand and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. ACMTransactions on Computer Systems, 3(1):63–75, February1985.

[7] B. Gupta et.al," A Low-Overhead Non- Block Checkpointing Algorithm for Mobile Computing Environment", GPC 2006, LNCS 3947, pp. 597-608, 2006

[8] Cao G. and Singhal M., "Mutable Snapshots: A New Snapshot Approach for Mobile Computing systems," IEEE Transaction on Parallel and Distributed Systems, vol. 12, no. 2, pp. 157 -172, February 2001.

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

[10] P. Ramanathan and K.G. Shin, "Use of Common Time Base for Checkpointing and Rollback Recovery in a Distributed System," IEEE Trans. Software Eng., vol. 19, no. 6, pp. 571–583, June 1993

[11] L.M. Silva and J.G. Silva, "Global Checkpointing for Distributed Programs," Proc. 11th Symp. Reliable Distributed Systems, pp. 155–162, Oct. 1992.

[12] Suparna Biswas &SarmisthaNeogy "A mobility-based checkpointing protocol for mobile computing system", IJCSIT vol 2, no. 1, Feb 2010.

[13] J.L. Kim and T. Park, "An Efficient Protocol for Checkpointing Recovery in Distributed Systems," IEEE Trans. Parallel and Distributed Systems, vol. 5, no. 8, pp. 955–960, Aug. 1993

[14] Wood., W.G., "A Decentralized recovery control protocol", IEEE Symposium on Fault-Tolerant Computing, 1981.

[15] Storm R., and Termini, S., "Optimistic recovery in distributed systems", ACM Trans. Computer Systems, Aug 1985, pp. 204-226.

[16] Rachit Garg &Parveen Kumar" Low overhead checkpointing protocols for mobile distributed systems: A Comparative STUDY" International Journal of Engineering Science and Technology, Vol. 2(7), 2010, 3267 -3276

[17] Bidyut Gupta, Shahram Rahimi, and ZipingLiu., "A New high performance checkpointing approach for mobile computing system", CSNS International Journal of Computer Science and Network Security, VOL 6, N05B May 2006

[18] Cao G. and Singhal M., "On the Impossibility of min-process nonblocking checkpointing and an efficient checkpointing algorithm for mobile computing systems", Proceedings of International Conference on Parallel Processing, August 1998, pp. 37-44.

[19] R.C. Gass, B. Gupta, "An Efficient checkpointing scheme for mobile computing systems", European Simulation Symposium, Oct 2001 (18-20), pp.1-6

[20] Carroll Morgan. Global and logical time in distributed algorithms. Information Processing Letters, 20:189–194, May 1985

[21] SuparnaBiswas&SarmisthaNeogy "A mobility-based checkpointing protocol for mobile computing system", IJCSIT vol 2, no. 1, Feb 2010. Kapoor et al., International Journal of Advanced Research in Computer Science and Software Engineering 6(5), May- 2016, pp. 65-73© 2016, IJARCSSE All Rights Reserved Page | 72

[22] M. Raynal. About logical clocks for distributed systems. Operating Systems Review,26(1):41–48, January 1992

[23] K.M. Chandy, L. Lamport, Distributed snapshots: determining global states of distributed systems, ACMTrans. Compute. Systems 3 (1) (1985) 63–75

[24] Lalit Kumar, and R K Chauhan., "Pitfalls in a minimum process coordinated checkpointing protocols for mobile distributed", ACCST Journal of Research, Volume III, No. 1, 2005 pp. 51-56.

[25] Prof. S. M. Tidke, RuchaRavindraGalgali "Predicting resource allocation in a distributed environment by using online predictive approach a review" International Journal of Advanced Research in Computer and Communication Engineering, Vol. 2, Issue 12, December 2013

[26] Parveen Kumar "A Minimum process global state detection scheme for mobile distributed systems "IJEST, vol. 2(7),2010, 2853-2858

[27] J.L.kim& T. park "An Efficient Protocol for Checkpointing Recovery in Distributed Systems", IEEE Transactions on Parallel and Distributed Systems, Vol -4, Aug.1993, Page 955-960

[28] Richard Koo, Sam Toueg, "Checkpointing and rollback-recovery for distributed systems (1987) Published in, Software Engineering, IEEE Transactions on (Volume: SE-13, Issue: 1)

[29] Acharya A., "Structuring distributed algorithms and services for networks with mobile hosts", Ph.D. Thesis, Rutgers University, 1995.

[30] 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.

[31] RuchiTuli, Parveen Kumar, "The Design and performance of a checkpointing scheme for mobile

ad hoc networks", Springer-Verlag CCIS 203, pp 204-212, 2011.

[32] NunoNeves and W. Kent Fuchs. “Adaptive recovery for mobile environments”, in Proc.IEEE High-Assurance Systems Engineering Workshop, October 21-22, 1996, pp.134

[33] 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.

[34] Taesoon Park, Namyoon Woo and Heon Y. Yeom, "An Efficient Recovery Scheme for Fault-Tolerant Mobile Computing Systems", FGCS- 19, 2003 Kumar, P.," A Low-cost hybrid coordinated checkpointing protocol for mobile distributed systems", Mobile Information Systems pp 13-32, Vol. 4, No. 1.,2007.

[35] Pushpendra Singh, Gilbert Cabillic, “A Checkpointing algorithm for mobile computing environment”, LNCS, No. 2775, pp 65-74, 2003.