Protocol for Coordinated Checkpointing using Smart Interval with Dual Coordinator

Manoj Kumar Niranjan
Rustamji Institute of Technology,
BSF Academy, Tekanpur

Mahesh Motwani
UIT-RGPV, Bhopal

ABSTRACT
Checkpointing is a very popular technique for fault tolerance in distributed systems. The proposed protocol tolerates the transient faults. In the protocol, all processes take checkpoints to form a global consistent checkpoint. The protocol handles the failures of initiator and non-initiator.

Keywords
Distributed Systems, Checkpointing, Fault Tolerance, Smart Interval.

1. INTRODUCTION
A distributed system is an application that executes a collection of protocols to coordinate the actions of multiple processes on a network, such that all components cooperate together to perform a single or small set of related tasks. A Fault Tolerant Distributed System can recover from failures without performing incorrect actions. The failure may be a network failure, network partition failure, timing failure, byzantine failure, omission failure, fail-stop failure or halting failure.[1] A good distributed system must overcome to these failure which can be achieved by fault tolerance. The fault tolerance can be achieved by using Checkpointing which is a popular fault tolerance technique. Our paper presents a new algorithm for checkpointing which can tolerate the failure of any process (node) as well as Coordinator Process (Node). Our algorithm tolerates the temporary failures which generally occurs due to software problems and can be removed by restarting the process.

2. CHECKPOINTING
Checkpointing is the method of periodically recording the states of the system onto the stable storage. Any such periodically saved state is called the checkpoint of the process [2]. A global state [3] of a distributed system is a set of individual process state per process [2]. Checkpointing may be one of two types, i.e., independent and coordinated checkpointing. In Independent checkpointing, each process takes checkpoint independently without requiring any synchronization when a checkpoint is taken [4]. In coordinated checkpointing, the processes coordinate their checkpointing action in such a way that the set of local checkpoints taken is consistent [5,6,7].

3. EXISTING WORK
In the existing work, the initiator communicates with other processes to create a checkpoint. In these old checkpointing protocols, if message communication takes place after checkpoint request of initiator, the global checkpoint may be inconsistent. This is shown in fig. 1 in which message m is sent by P0 after receiving a checkpoint request from the initiator. If m reaches P1 before the checkpoint request, the checkpoint will become inconsistent because checkpoint c1,x confirms that message m is received from P0, while checkpoint c0,x says that it is not sent from P0. [8]

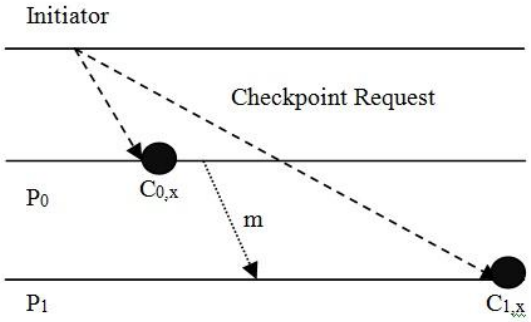

Fig. 1. Message communication between P0 and P1 causing inconsistent checkpoint

In another protocol, the message communication is allowed within a fixed time interval only. This concept reduces message communication [9] which is beneficial in decreasing the communication overhead. The main drawback of this protocol is the fixation of a particular process as initiator process. Since a fixed process will act as initiator in entire system execution, thus the probability of failure will be high.

In another checkpointing protocol, the process initiator is not fixed which reduces the probability of failure of initiator. The drawback of this protocol is that the message communication could be accomplished at any time i.e., there is no concept of fixed time interval for message communication. Hence it increases communication overhead and output commit latency [10].

If we discuss the existing protocols, we found that there is no protocol that takes care of initiator process. The existing protocols assume that initiator process never fails. Our algorithm removes this assumption.

4. PROPOSED WORK
The proposed protocol overcomes to these shortfalls. The proposed protocol uses a fixed time interval for message communications which controls the message communication. This fixed time interval is called smart interval. This concept reduces the communication overhead. The protocol also gives chance to every process to act as initiator process which reduces the probability of failure of initiator.

The present work suggests a new coordinated checkpointing algorithm in which each process is given chance to act as checkpoint initiator. The checkpoint initiator sends messages to other processes to be prepare for checkpoint and then to take checkpoint. However, a process has to maintain a log of received, sent and unacknowledged messages of the current checkpointing interval. After receiving take checkpoint message from initiator, all the processes change the status of

checkpoint from tentative to permanent and send it to initiator. The set of these checkpoints form global checkpoint. If initiator does not receive all the local checkpoints, it issues abort message to all other processes for not making the tentative checkpoint permanent.

At any instance, initiator and co-initiator work together. After creating global checkpoint, the initiator maintains another copy on co-initiator that can be used in case of failure of initiator.

5. SYSTEM MODEL

Let us consider a distributed system of 'n' processes, P0, P1, ……, Pn-1. The no. of processes 'n' is fixed for the duration of execution. Let the checkpoints be denoted as CPki, i.e., initial checkpoint CPk0 (i=0), first checkpoint CPk1 (i=1), second checkpoint CPk2 (i=2) and so on (here k is the process no.). The initial checkpoint is taken when the system is being initialized. Each process maintains its own independent data structures, states and computations. Processes have no shared memory and no global clock. All communications among processes are through message passing only. We are assuming followings:

The network is secure, reliable and homogeneous with infinite bandwidth and zero latency. The topology doesn't change and the transport cost is zero.

The network guarantees reliable FIFO (First In First Out) delivery of messages between any pair of processes. The assumption of FIFO delivery assures the message synchronization.

There is one initiator process and one co-initiator process. In case of failure of initiator process, the co-initiator process will act as initiator and the next process will act as co-initiator.// Is This assumption

Here, latency is the time between initiating a request for data and the beginning of the actual data transfer. Bandwidth is a measure of the capacity of a communications channel. The higher a channel's bandwidth, the more information it can carry. The topology is the different configuration that can be adopted in building networks, such as ring, bus, star or mesh. The network will be homogeneous if it is running a single network protocol.

The message communication will took place only in specified time interval which is elapsed between the control messages for prepare checkpoint and take checkpoint. If any process sends a message within this time interval, it has to be logged and the process execution is continued. This enables handling of lost messages. [10] The initiator process sends the control messages for prepare checkpoint and take checkpoint to other processes.

6. PROTOCOL DESCRIPTION

The checkpoint initiator process sends checkpoint-prepare-request-message to other processes to start checkpointing. The other processes send their responses to the initiator process. If initiator process received replies from all processes within specified time-interval then it sends take-checkpoint-request-message and if initiator process does not receive replies from any process within specified time-interval then it will send abort-checkpoint-request-message. The set of checkpoint of all processes received by initiator process is called global checkpoint. A local checkpoint is denoted by CPki where k is the process id and i is the checkpoint number. The ith global checkpoint is the set CPi={CP0i, CP1i,……, CPn-1i} in a system of n processes. CPi is said to be consistent if and only if $\forall j,k \in [0,n-1]: j \neq k \Rightarrow (CPji \rightarrow CPki)$ where $\rightarrow$ denotes the happened-before relation described by Lamport in [12].

The maximum transmission delay to reach a message to destination is t. The T is the checkpointing interval. Here T>3t, since checkpoint interval (T) is obviously greater than specified time-interval and the length of specified time-interval is bound to be at least 3t to survive the transmission delay of control messages (checkpoint-prepare-request-message, response of checkpoint-prepare-request-message and take-checkpoint-request-message and each transmission will take at least t) and to enable logging of computational messages. Fig.2 shows the message communication using smart-interval [11]. The P1, P2, P3 are processes which are communicating during interval. Here, K and (K+1) are two consecutive checkpoints. The S.I. is smart interval.

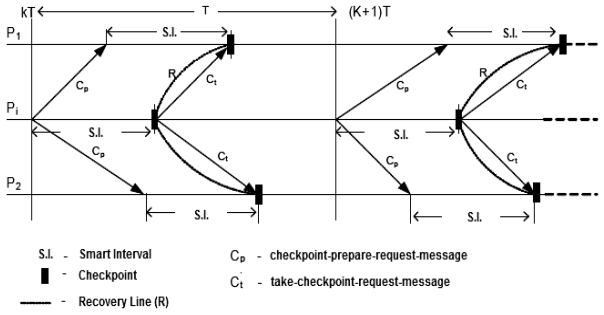

Fig. 2. Diagram showing message communication during specified time-interval

Now, if the initiator process fails, a new initiator process has to be selected. The protocol should also save the global checkpoint which is stored at the initiator. Our protocol creates a backup copy of global checkpoint which can be used at the failure of initiator process. The backup copy will be stored at the process which will act as initiator, if initiator process fails. The process next to initiator process will act as co-initiator process.

7. CHECKPOINTING PROCESS

The checkpoint process starts at the time of system initialization. After T time interval (which is decided by the programmer) of previous checkpoint, the initiator process starts the process of checkpointing. The first process will act as initiator process in the beginning and will be denoted by Pinit. The process next to initiator will act as co-initiator and will be denoted by Pbinit.

The initiator process Pinit sends checkpoint-prepare-request-message to all other processes at tprep. On receiving checkpoint-prepare-request-message, each process write tentative checkpoint after sending response to the initiator.

1) Now, if initiator receives response from all processes, within (tprep+2*Ttrns), the initiator process sends take-checkpoint-request-message to all processes. When receiver receives take-checkpoint-request-message from initiator process, the tentative checkpoint is made permanent. This will save the states of all processes which are responsible for preparing a global checkpoint.

2) Now, suppose if one or more process fails after responding to checkpoint-prepare-request-message, then the tentative checkpoint (which is prepared in

response to checkpoint-prepare-request-message) is used to recover the failed process.

3) Now suppose if one or more process fails to respond to checkpoint-prepare-request-message, the initiator process sends abort-checkpoint-request-message to all processes. On receiving this, the tentative checkpoint is deleted. The copy of unacknowledged message keeps in a log in this case.

4) If the global checkpoint created successfully, then it has to be saved on backup initiator P_{binit}. The Pinit sends the global checkpoint data to P_{binit}. After receiving the global checkpoint P_{binit} sends acknowledgement message to P_{init}. After receiving the acknowledgement message from P_{binit}, P_{init} starts the process of next checkpoint.

5) If the P_{init} fails, then there may be three states:

- P_{init} may fail before starting the checkpoint process
- After starting the checkpoint process but before completion of checkpoint process
- After completion of checkpoint process, but before sending the global checkpoint to backup initiator.

6) If P_{init} fails before starting the checkpoint process, then P_{binit} and other process will not get checkpoint-prepare-request-message from P_{init}. If P_{binit} does not receive the checkpoint-prepare-request-message within the specific time interval, then it first sends a test message to P_{init} to confirm the status of initiator. If P_{init} replies positively, then P_{binit} takes no action, otherwise P_{binit} starts the process of next checkpoint. It also resets its role, now, it acts as initiator and next process will become co-initiator. After finding the next initiator (which will be act as backup initiator), the checkpointing process continues as above.

7) If P_{init} fails after starting the checkpoint process but before completion of checkpoint process, then P_{binit} will not get global checkpoint data. If P_{binit} does not get the global checkpoint data which should be received within (tprep+2*Ttrns), then it sends a test message to P_{init} to confirm the status of initiator. If P_{init} replies positively, then P_{binit} takes no action, otherwise P_{binit} starts the process of next checkpoint. It also resets its role, now, it acts as initiator and next process will become co-initiator. After finding the next initiator (which will be act as backup initiator), the checkpointing process continues as above.

8) If P_{init} fails after creating the global checkpoint but before sending it to backup initiator, then also like previous step, backup initiator P_{binit} will not receive the global checkpoint data within (tprep+2*Ttrns). Now, it will send a test message to Pinit to confirm the status of initiator. If P_{init} replies positively, then P_{binit} takes no action, otherwise P_{binit} starts the process of next checkpoint. It also resets its role, now, it acts as initiator and next process will become co-initiator. After finding the next initiator (which will be act as backup initiator), the checkpointing process continues as above.

9) In step (7) and (8), if P_{binit} gets positive reply from P_{init}, but does not receive the global checkpoint data, then it sends request message to send the global checkpoint data, i.e., send-global-checkpoint-message. It waits for t time to receive the global checkpoint data. If it does not receive the global checkpoint within t, then it again sends test message to P_{init} and if it gets positive reply then it repeat then it repeat the step (9) until it get the global checkpoint data. If it does not get positive reply, then it start acting as initiator like step (7) and (8).

8. ALGORITHM

Step-I:
This step is executed at initiator process P_{init}

i. Send *checkpoint-prepare-request-message* to remaining processes at t^{prep} for $(k+1)^{th}$ checkpoint
ii. Remove $(k-1)^{th}$ checkpoint, if exist.
iii. Receive response from other processes within $(t^{prep}+2*T^{trns})$
iv. If all processes respond positively then
    Send *take-checkpoint-request-message* to all processes
    Create *global-checkpoint* and send it to P_{binit}.
    Else (if even a single process does not respond positively or response does not arrive to initiator process)
    a. Send *abort-checkpoint-request-message* to all processes
    b. Retain copies of unacknowledged messages in a log

Step-II:
This step is executed at other process P_{oth}

i. Receive *checkpoint-prepare-request-message* from initiator at t^{rec}
ii. Send own response to initiator
iii. If response is positive then Call save_state(P_{oth}) to write *tentative-checkpoint* asynchronously
iv. Wait for decision of P_{init} till $(t^{rec}+T^{trns}+T^{trns})$
v. If received decision is *take-checkpoint-request-message* then Change status of *tentative-checkpoint* to permanent
    Else
    Delete *tentative-checkpoint*
vi. Delete messages whose acknowledgements have received. Log unacknowledged messages.

Step-III:
This step is executed at any process P_{any} for receiving message

i. If ((checkpoint number in message)=(checkpoint number in P_{any}))
    a. Send (tag1,s_id)
    b. Receive(message)
ii. else if ((checkpoint number in message)>(checkpoint number in P_{any}))
    a. save_state(P_{any})
    b. send(tag1,s_id)
    c. receive(message)
iii. else if ((checkpoint number in message)<(checkpoint number in P_{any}))
    a. send (tag2,s_id)
    b. receive(message)

Step-IV:
This steps is executed at any process P_{any} for writing unacknowledged messages

i. for all k
if (ack[k]=0) then write k^{th} message in buffer

Step-V:
This steps is executed when initiator process fails
i. if P_{init} fails
a. Reset the status of P_{binit} to P_{init}
b. Reset the status of process next to P_{binit} to P_{binit}

9. PERFORMANCE RESULTS

The presented algorithm is simulated using parallel virtual machine java libraries. The environment used for simulation is Ubuntu 13.10 with Open JDK 7. Since, we assumed consistent network bandwidth, we created all the process on a single computer with intel i3 processor and 2GB DDR3 RAM. The results of simulation are as under:

No. of Process	Total Exec. Time (in milli-seconds)	Checkpoint Interval (in milli-seconds)	No. of Faults		Total Execution Time with Faults (in milli-seconds)	% time increase
			Initiator	Others		
10	1000	100	1	4	1413	41.30
11	2000	110	2	6	2637	31.85
12	3000	120	3	8	3819	27.30
13	4000	130	4	10	4945	23.63
14	5000	140	5	12	6264	25.28
15	6000	150	6	14	7645	27.42
16	7000	160	7	16	8843	26.33
17	8000	170	8	18	10161	27.01
18	9000	180	9	20	11514	27.93
19	10000	190	10	22	12806	28.06
20	11000	200	11	24	14275	29.77
21	12000	210	12	26	15662	30.52
22	13000	220	13	28	17063	31.25
23	14000	230	14	30	18304	30.74
24	15000	240	15	32	19921	32.81
25	16000	250	16	34	21462	34.14
26	17000	260	17	36	22917	34.81
27	18000	270	18	38	24121	34.01
28	19000	280	19	40	25683	35.17
29	20000	290	20	42	27164	35.82
30	21000	300	21	44	28908	37.66
31	22000	310	22	46	30166	37.12
32	23000	320	23	48	31559	37.21
33	24000	330	24	50	32676	36.15
34	25000	340	25	52	33622	34.49
35	26000	350	26	54	34607	33.10
36	27000	360	27	56	35703	32.23
37	28000	370	28	58	34543	23.37
38	29000	380	29	60	35484	22.36
39	30000	390	30	62	35698	18.99
40	31000	400	31	64	36576	17.99
41	32000	410	32	66	37495	17.17

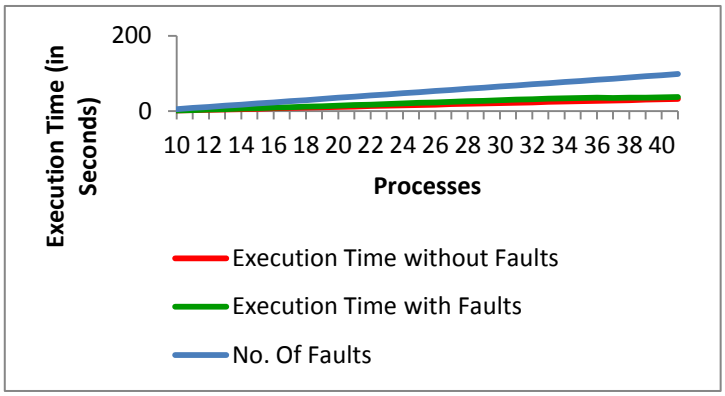

Fig. 3: Graphical representation of results

10. CONCLUSION

The checkpointing protocol of this paper reduces the communication overhead because messages are transmitted in Smart Interval only. A global checkpoint includes each and every checkpoint taken by the processes of the system, so it has to be retained. In the proposed protocol, whenever initiator process Pi sends checkpoint-prepare-request-message for (k+1)th checkpoint, the protocol will automatically delete the (k-1)th global checkpoint which results simplified garbage collection. There may be two types of failures, transient and permanent. The protocol is useful in tolerating transient failures occurred in initiator and non-initiator processes.

11. REFERENCES

[1] Introduction to Distributed System Design, Google Code University, http://code.google.com/edu/parallel/dsd-tutorial.html#Basics

[2] D. Manivannan, R.H.B. Netzer & M. Singhal, "Finding Consistent Global Checkpoints in a Distributed Computation", IEEE Trans. On Parallel & Distributed Systems, Vol.8, No.6, pp. 623-627 (June 1997)

[3] J. Tsai & S. Kuo, "Theoretical Analysis for Communication-Induced Checkpointing Protocols with Rollback-Dependency Trackability"; IEEE Trans. On Parallel & Distributed Systems, Vol.9, No. 10, pp. 963-971 (October 1998)

[4] B. Bhargava and S.R. Lian, "Independent Checkpointing and Concurrent Rollback for Recovery in Distributed Systems-An Optimistic Approach", Proceeding of IEEE Symposium on Reliable Distributed Systems, pp. 3-12 (1988)

[5] Guohong Cao, and Mukesh Singhal, "On Coordinated Checkpointing in Distributed Systems," IEEE Transactions On Parallel And Distributed Systems," Vol. 9, No. 12, pp.1213-122 (Dec.1998)

[6] Sharma D. D. and Pradhan D. K., "An Efficient Coordinated Checkpointing Scheme for Multicomputers," Proc. IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems, pp 36-42 (June 1994)

[7] E.N. Elnozahy, D.B. Johnson, and W. Zwaenepoel, "The Performance of Consistent Checkpointing," Proc. 11th Symp. Reliable Distributed Systems, pp. 39–47 (Oct. 1992)

[8] E.N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi-Min Wang and David B. Johnson, "A Survey of Rollback-Recovery Protocols in Message-Passing Systems", ACM Computing Surveys (CSUR), Volume 34, Issue 3 (September 2002) Page(s):375-408 (2002)

[9] Ch. D.V. Subba Rao and M.M. Naidu, "A New, Efficient Coordinated Checkpointing Protocol Combined with Selective Sender-Based Message Logging", IEEE/ACS International Conference on Computer Systems and Applications, AICCSA 2008, pp. 444-447 (2008)

[10] Sarmistha Neogy, Anupam Sinha, Pradip K Das, "CCUML: A Checkpointing Protocol for Distributed System Processes", IEEE Transactions on TENCON 2004, IEEE Region 10 Conference, Volume B, 21-24 Nov. 2004, Page(s):553 – 556 (2004)

[11] J. Makhijani, M.K. Niranjan, M.Motwani, A.K. Sachan, A. Rajput, "An efficient protocol using smart interval for coordinated checkpointing", International Conference on Advances in Information Technology and Mobile Communication – AIM 2011

[12] K.M. Chandy & L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems", ACM Trans. On Computer Systems, Vol. 3, no. , Feb 1985, pp 63-75 (1985)