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](image)

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.[1] 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 take 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 receives replies from all processes within specified time-interval then it sends abort-checkpoint-request-message and if initiator process does not receive replies from any process within specified time-interval then it will send tentative checkpoint-request-message. The set of checkpoint of all processes received by initiator process is called global checkpoint. A local checkpoint is denoted by CPk 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 ∀j,k∈{0,n-1};j≠k⇒(CPji→CPkj) where → 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.

![Diagram showing message communication during specified time-interval](image)

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 process 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*Truns), 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 processes fail 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\textsubscript{binit}. The P\textsubscript{init} sends the global checkpoint data to P\textsubscript{binit}. After receiving the global checkpoint P\textsubscript{binit} sends acknowledgement message to P\textsubscript{init}. After receiving the acknowledgement message from P\textsubscript{binit}, P\textsubscript{init} starts the process of next checkpoint.

5) If the P\textsubscript{init} fails, then there may be three states:
   • P\textsubscript{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\textsubscript{init} fails before starting the checkpoint process, then P\textsubscript{binit} and other process will not get checkpoint-prepare-request-message from P\textsubscript{init}. If P\textsubscript{binit} does not receive the checkpoint-prepare-request-message within the specific time interval, then it first sends a test message to P\textsubscript{init} to confirm the status of initiator. If P\textsubscript{init} replies positively, then P\textsubscript{binit} takes no action, otherwise P\textsubscript{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\textsubscript{init} fails after starting the checkpoint process but before completion of checkpoint process, then P\textsubscript{binit} will not get global checkpoint data. If P\textsubscript{binit} does not get the global checkpoint data which should be received within (t\textsubscript{prep}+2*T\textsubscript{trns}), then it sends a test message to P\textsubscript{init} to confirm the status of initiator. If P\textsubscript{init} replies positively, then P\textsubscript{binit} takes no action, otherwise P\textsubscript{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\textsubscript{init} fails after creating the global checkpoint but before sending it to backup initiator, then also like previous step, backup initiator P\textsubscript{binit} will not receive the global checkpoint data within (t\textsubscript{prep}+2*T\textsubscript{trns}). Now, it will send a test message to P\textsubscript{init} to confirm the status of initiator. If P\textsubscript{init} replies positively, then 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\textsubscript{binit} gets positive reply from P\textsubscript{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\textsubscript{init} and if it gets positive reply then it repeat then it repeat 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\textsubscript{init}

i. Send checkpoint-prepare-request-message to remaining processes at t\textsuperscript{prep} for (k+1)\textsuperscript{th} checkpoint

ii. Remove (k-1)\textsuperscript{th} checkpoint, if exist.

iii. Receive response from other processes within (t\textsuperscript{prep}+2*T\textsubscript{trns})

iv. If all processes respond positively then Send take-checkpoint-request-message to all processes

Create global-checkpoint and send it to P\textsubscript{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\textsubscript{oh}

i. Receive checkpoint-prepare-request-message from initiator at t\textsuperscript{prep}

ii. Send own response to initiator

iii. If response is positive then Call save_state(P\textsubscript{oh}) to write tentative-checkpoint asynchronously

iv. Wait for decision of P\textsubscript{init} till (t\textsuperscript{prep}+T\textsubscript{trns}+T\textsubscript{unack})

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\textsubscript{any} for receiving message

i. If (checkpoint number in message)=(checkpoint number in P\textsubscript{any})

   a. Send (tag1,s_id)

   b. Receive(message)

ii. else if (checkpoint number in message)> (checkpoint number in P\textsubscript{any})

   a. save_state(P\textsubscript{any})

   b. send(tag1,s_id)

   c. receive(message)

iii. else if (checkpoint number in message)< (checkpoint number in P\textsubscript{any})

   a. send (tag2,s_id)

   b. receive(message)

Step-IV:
This steps is executed at any process P\textsubscript{any} for writing unacknowledged messages
for all \( k \)
if \( \text{ack}[k]=0 \) then write \( k \)th message in buffer

**Step-V:**
*This step is executed when initiator process fails*

i. if \( P_{\text{init}} \) fails
a. Reset the status of \( P_{\text{init}} \) to \( P_{\text{out}} \)

b. Reset the status of process next to \( P_{\text{init}} \) to \( P_{\text{out}} \)

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:

<table>
<thead>
<tr>
<th>No. of Process</th>
<th>Total Exec. Time (in milliseconds)</th>
<th>Checkp. Initiation Interval (in milliseconds)</th>
<th>No. of Faults</th>
<th>Total Execution Time with Faults (in milliseconds)</th>
<th>% time increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1000</td>
<td>100</td>
<td>1</td>
<td>1413</td>
<td>41.30</td>
</tr>
<tr>
<td>11</td>
<td>2000</td>
<td>110</td>
<td>2</td>
<td>2637</td>
<td>31.85</td>
</tr>
<tr>
<td>12</td>
<td>3000</td>
<td>120</td>
<td>3</td>
<td>3819</td>
<td>27.30</td>
</tr>
<tr>
<td>13</td>
<td>4000</td>
<td>130</td>
<td>4</td>
<td>4945</td>
<td>23.63</td>
</tr>
<tr>
<td>14</td>
<td>5000</td>
<td>140</td>
<td>5</td>
<td>6264</td>
<td>25.28</td>
</tr>
<tr>
<td>15</td>
<td>6000</td>
<td>150</td>
<td>6</td>
<td>7645</td>
<td>27.42</td>
</tr>
<tr>
<td>16</td>
<td>7000</td>
<td>160</td>
<td>7</td>
<td>8843</td>
<td>26.33</td>
</tr>
<tr>
<td>17</td>
<td>8000</td>
<td>170</td>
<td>8</td>
<td>10161</td>
<td>27.01</td>
</tr>
<tr>
<td>18</td>
<td>9000</td>
<td>180</td>
<td>9</td>
<td>11514</td>
<td>27.93</td>
</tr>
<tr>
<td>19</td>
<td>1000</td>
<td>190</td>
<td>10</td>
<td>12806</td>
<td>28.06</td>
</tr>
<tr>
<td>20</td>
<td>1100</td>
<td>200</td>
<td>11</td>
<td>14275</td>
<td>29.77</td>
</tr>
<tr>
<td>21</td>
<td>1200</td>
<td>210</td>
<td>12</td>
<td>15662</td>
<td>30.52</td>
</tr>
<tr>
<td>22</td>
<td>1300</td>
<td>220</td>
<td>13</td>
<td>17063</td>
<td>31.25</td>
</tr>
<tr>
<td>23</td>
<td>1400</td>
<td>230</td>
<td>14</td>
<td>18304</td>
<td>30.74</td>
</tr>
<tr>
<td>24</td>
<td>1500</td>
<td>240</td>
<td>15</td>
<td>19921</td>
<td>32.81</td>
</tr>
<tr>
<td>25</td>
<td>1600</td>
<td>250</td>
<td>16</td>
<td>21462</td>
<td>34.14</td>
</tr>
<tr>
<td>26</td>
<td>1700</td>
<td>260</td>
<td>17</td>
<td>22917</td>
<td>34.81</td>
</tr>
<tr>
<td>27</td>
<td>1800</td>
<td>270</td>
<td>18</td>
<td>24121</td>
<td>34.01</td>
</tr>
<tr>
<td>28</td>
<td>1900</td>
<td>280</td>
<td>19</td>
<td>25683</td>
<td>35.17</td>
</tr>
<tr>
<td>29</td>
<td>2000</td>
<td>290</td>
<td>20</td>
<td>27164</td>
<td>35.82</td>
</tr>
<tr>
<td>30</td>
<td>2100</td>
<td>300</td>
<td>21</td>
<td>28908</td>
<td>37.66</td>
</tr>
<tr>
<td>31</td>
<td>2200</td>
<td>310</td>
<td>22</td>
<td>30166</td>
<td>37.12</td>
</tr>
</tbody>
</table>

![Fig. 3: Graphical representation of results](image)

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 \( P_i \) sends checkpoint-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


