Performance analysis of different checkpointing and recovery schemes using stochastic model

Partha Sarathi Mandal, Krishnendu Mukhopadhyaya*

Advanced Computing and Microelectronics Unit, Indian Statistical Institute, 203 B.T. Road, Kolkata 700108, India

Abstract

Several schemes for checkpointing and rollback recovery have been reported in the literature. In this paper, we analyze some of these schemes under a stochastic model. We have derived expressions for average cost of checkpointing, rollback recovery, message logging and piggybacking with application messages in synchronous as well as asynchronous checkpointing. For quasi-synchronous checkpointing we show that in a system with n processes, the upper bound and lower bound of selective message logging are $O(n^2)$ and $O(n)$, respectively.

Keywords: Checkpointing; Message logging; Rollback recovery; Performance evaluation

1. Introduction

The technique of checkpointing and rollback recovery is a well-known method to achieve fault tolerance in distributed computing systems. In case of a fault, the system can rollback to a consistent global state, and resume computation without requiring additional efforts from the programmer. A *checkpoint* is a snapshot of the current state of a process. It saves enough information in non-volatile stable storage such that, if the contents of the volatile storage are lost due to process failure, one can reconstruct the process state from the saved information. The action of the receiver of a message may depend on the content of the message. Thus the receiver is considered to be dependent on the sender of the message. This dependency relation is transitive. If the processes communicate with each other through messages, rolling back a process may cause some inconsistency. Within the time since its last checkpoint, a process may have sent some messages. If it is rolled back and restarted from the point of its last checkpoint, it may create *orphan messages*, i.e., messages whose *receive* events are recorded in the states of the destination processes but the *send* events are lost. The process, that received the original message, now orphaned, is called an *orphan process*.

Similarly, messages received during the rolled back period, may also cause problem. Their sending processes will have no idea that these messages are to be sent again. Such a message, whose *send* event is recorded in the state of the sender process but the *receive* event is lost, is called a *missing message*.

A set of checkpoints, with one checkpoint for every process, is said to be *Consistent Global checkpointing State* (CGS), if it does not contain any orphan message or missing message. However, generation of missing messages may be acceptable, if messages are logged by sender.

Checkpointing algorithms may be classified into three broad categories: (a) *synchronous*, (b) *asynchronous* and (c) *quasi-synchronous* [15]. In asynchronous checkpointing [17,26] each process takes checkpoints independently. In case of a failure, after recovery, a CGS is found among the existing checkpoints and the system restarts from there. Here, finding a CGS can be quite tricky. The choice of checkpoints for the different processes is influenced by their mutual causal dependencies. The common approach is to use *rollback-dependent graph* or *checkpoint graph* [7,29,3,20].

* Corresponding author. Fax: +91 33 2577 3035.
E-mail addresses: partha_r@isical.ac.in (P.S. Mandal), krishnendu@isical.ac.in (K. Mukhopadhyaya).

If all the processes take checkpoints at the same time instant, the set of checkpoints would be consistent. But since globally synchronized clocks are very difficult to implement, processes may take checkpoints within an interval. In synchronous checkpointing [3,4,6,12–15,17,20,22] processes synchronize through system messages before taking checkpoints. These synchronization messages contribute to extra overhead. On the other hand, in asynchronous checkpointing some of the checkpoints taken may not lie on any CGS. Such checkpoints are called *useless checkpoints*. Useless checkpoints degrade system performance. Unlike asynchronous checkpointing, synchronous checkpointing does not generate useless checkpoints.

To overcome the above tradeoff of *synchronous* and *asynchronous* checkpointing, *quasi-synchronous* checkpointing algorithms were proposed by Manivannan and Singhal [15]. Processes take checkpoints asynchronously. So there is no overhead for synchronization. Generation of useless checkpoints is reduced by forcing processes to take additional checkpoints at appropriate times.

Some works on performance evaluation of checkpointing and rollback recovery algorithms have been reported in the literature. Plank and Thomason [19] calculated the average availability of parallel checkpointing systems and used them in selecting runtime parameters like the number of processors and the checkpointing interval. These can minimize the expected execution time of a long-running program in the presence of failures. Vaidya [27] proposed a two level distributed recovery scheme and analyzed it to show that it achieves better performance than the traditional recovery schemes. The same algorithm was also analyzed by Panda and Das [18] taking the probability of task completion on a system with limited repairs as the performance metric. Rao et al. [21] presented an experimental evaluation of the performance of different message logging protocols during recovery.

Section 2 describes the stochastic model used for the analysis. In Sections 3–5 expressions for checkpointing and recovery cost are derived for synchronous, asynchronous and quasi-synchronous checkpointing, respectively. In section 6, the different schemes are compared.

2. The underlying model

For the purpose of analysis, we consider the following stochastic model:

(1) Time is assumed to be discrete.
(2) The system consists of a loosely coupled system with n processes.
(3) Inter-process communication is through message passing.
(4) Message sending, checkpointing and faults occur independent of each other.
(5) At any point of time, a process may generate a message with probability λ_m. The destination of the message can be any one of the remaining $(n-1)$ processes with equal probabilities.
(6) At any point of time, a process that is not involved in taking checkpoints, may start checkpointing with probability λ_c.
(7) At any point of time, a process may fail with probability λ_f.

3. Synchronous checkpointing

In synchronous checkpointing algorithms, processes communicate through system messages and make sure that the checkpointing scheme yields a CGS. In the schemes proposed by Prakash and Singhal [20], and Cao and Singhal [3], the initiator of the checkpointing process forces the dependent process to take checkpoints. The dependency relations are maintained by attaching an n-bit vector with every application message. Every message sent makes the receiver dependent on the sender. In the worst case, checkpointing initiator may directly or transitively depend on the remaining $(n-1)$ processes. In that case, all processes take checkpoints for the checkpointing initiator. We consider the algorithms [13,14,6,22] where the checkpointing initiator forces all processes in the system to take checkpoints. The results of our analysis gives an upper bound for the overhead in the other algorithms (where only dependent processes take checkpoints).

3.1. Checkpointing overhead

At any point of time, a process initiates checkpoint with probability λ_c. It also takes checkpoint if at least one of the other process initiates checkpointing and propagates a checkpointing request to all other processes. Probability that at least one process initiates checkpointing is $(1-(1-\lambda_c)^n)$. Expected inter-checkpoint gap $= \frac{1}{1-(1-\lambda_c)^n}$. Suppose t_c denotes the average cost of taking a checkpoint. Over and above the cost of taking a checkpoint, there is also the overhead of message communication for synchronization. An initiator generates $(n-1)$ checkpoint request messages and another $(n-1)$ *commit* messages after the acknowledgment comes back. A non-initiator generates only an acknowledgment message. Since one in n of the checkpoints taken by a process is initiated by itself, the average number of messages generated per checkpoint taken is $2(n-1)\cdot\frac{1}{n}+1\cdot(1-\frac{1}{n}) = \frac{3(n-1)}{n}$. Let C_{snr} denote the average cost for sending and receiving a message alongwith the network congestion cost of a message. So the average cost per checkpoint is $t_c' = t_c + \frac{3(n-1)}{n}C_{\text{snr}}$.

$$\text{Checkpointing overhead for a process per unit time} = \frac{(1-(1-\lambda_c)^n)t_c'}{1+(1-(1-\lambda_c)^n)t_c'}.$$

3.2. Rollback recovery overhead

If failures are rare, we can safely assume that a failure may occur at any point between two successive checkpoints, with equal probabilities. Let C_{reco} denote the recovery cost for every unit time rolled back.

Thus, the average rollback recovery overhead for a process is

$$= \tfrac{1}{2} \text{ (average inter-checkpoint gap) } C_{\text{reco}} = \frac{C_{\text{reco}}}{2(1-(1-\lambda_c)^n)}.$$

4. Asynchronous checkpointing with message logging

In checkpointing and message logging protocols, each process typically records both the content and the *receive* sequence number of all the messages it has processed in a location that will survive the failure of the process. In case the process has to rollback, the logged messages are replayed from the stable storage; they need not be retransmitted by the sender. The messages which are not logged will have to be resent, and may force the sender to rollback too. A process may also periodically create checkpoints of its local state, thereby allowing message logs to be removed. The periodic checkpointing of a process state is only needed to bound the length of its message log. There are sender and receiver based message logging algorithms in literature. Here only receiver based message logging protocols are considered.

4.1. Pessimistic message logging

A *pessimistic* protocol [9,25] is one in which a process P_i never sends a message until it knows that all messages received and processed so far are logged. Thus, pessimistic protocols will never create an orphan message. The reconstruction of the state of a crashed process is also straightforward compared to the optimistic protocols. Received messages are logged synchronously. This may be achieved by blocking the receiver until the message is logged to a stable storage. The other option is to block the receiver only if it attempts to send a new message before this received message is logged. Blocking the receiver can slow down the throughput of the processes even when no process ever crashes. On the other hand, during recovery, only the faulty process rolls back to its latest checkpoint. All messages received in the time between the latest checkpoint and the fault are replayed to it from the stable storage in the same order as they were received before the fault. Messages sent by the process during recovery are ignored since they are duplicates of the ones sent before the failure.

Overhead due to this protocols may be partitioned into

(1) blocking time for logging received messages and
(2) rollback overhead due to fault.

Expected number of application messages ($\mathbf{E_{msg}(T_p)}$) received by a process in T_p unit of time is

$$\mathbf{E_{msg}(T_p)} = \lambda_m T_p.$$

Total message overhead due to *pessimistic message logging* ($\mathbf{E_{pessimistic\_cost}}$) depends on two parameters; C_{snr} and $C_{\text{pessi\_log}}$, the cost of storing a message. Total pessimistic message logging cost per unit time is $\lambda_m(C_{\text{snr}} + C_{\text{pessi\_log}})$.

Total cost (checkpointing and message logging) per unit time

$$\mathbf{E_{pessi\_ckpt\_msg}} = \frac{t_c}{T_p + t_c} + \lambda_m(C_{\text{snr}} + C_{\text{pessi\_log}}) \quad \text{where } T_p = \frac{1}{\lambda_c}.$$

If failures are rare, we can safely assume that a failure may occur at any point between two successive checkpoints, with equal probabilities.

The average rollback recovery overhead ($\mathbf{E_{pessi\_reco}}$) for a process is the sum of recovery cost and message-replaying cost from stable storage. Let C_{replay} denote the cost of replaying a logged message from stable storage.

$$\mathbf{E_{pessi\_reco}} = \frac{C_{\text{reco}}}{2\lambda_c} + \frac{\lambda_m C_{\text{replay}}}{2\lambda_c} = \frac{1}{2\lambda_c}(C_{\text{reco}} + \lambda_m C_{\text{replay}}).$$

4.2. Optimistic message logging

In optimistic message logging protocols [5,10,23,26,28], messages may not be logged immediately. The receiver continues its normal actions. The messages are logged at some point of time in the future so as to minimize logging overhead. This may be achieved by grouping several messages or logging during idle time of the system. Checkpoints are taken asynchronously.

Let $C_{\text{opti\_log}}$ denote the average cost for logging a message in this scheme. Note that $C_{\text{opti\_log}} < C_{\text{pessi\_log}}$.

Expected total cost (checkpointing and message logging) per unit time is

$$\mathbf{E_{opt\_ckpt\_msg}} = \frac{t_c}{T_p + t_c} + \lambda_m(C_{\text{snr}} + C_{\text{opti\_log}}).$$

The receiver, P_i, of a message m_1 depends on the state of the sender, P_j. Suppose P_j received a message m_2 from P_k, before sending m_1. If P_j fails without logging m_2, P_i will become orphan. Thus rollback of P_j may cause a rollback of P_i too.

This problem can be solved by optimistic recovery. In this case, the faulty process restarts by restoring its latest checkpoint and replays the logged messages which were received after the restored state. Since in this scheme of message logging, messages are logged asynchronously, on a failure, a process loses all the messages it received but did not log before the failure. Such processes are said to be in *lost states* [5]. Other processes which are dependent on the lost states must be rolled back. Before rolling back they log all unlogged messages in a stable storage. Thus no message is lost in this rollback. It is important to find out the expected number of dependent processes to calculate the overhead due to single fault.

In an interval of length t, $\left(1-\left[1-\frac{\lambda_{\text{msg}}}{n-1}\right]^t\right)$ is the probability that process P_i sends at least one message to process P_j, $j \neq i$, $j \in \{0, 1, 2, 3, \ldots, n-1\}$. The expected number of distinct processes which receive message(s) from process P_i during this period is

$$(n-1)\left[1-\left[1-\frac{\lambda_{\text{msg}}}{n-1}\right]^t\right].$$

If process P_i fails at this point, this would be the expected number of orphan processes.

Let $C_{\text{roll\_back}}$ be the rollback cost for every time unit rolled back. So the expected total rollback cost for all orphan processes is

$$\frac{(n-1)}{2\lambda_c}\left[1-\left[1-\frac{\lambda_{\text{msg}}}{n-1}\right]^t\right] C_{\text{roll\_back}}.$$

Let the average gap between two loggings be $\frac{1}{\lambda_l}$. The average rollback recovery overhead ($\mathbf{E_{opt\_reco}}$) for a process is the sum of the recovery cost, the message replaying cost, the message resend cost and the rollback cost ($C_{\text{roll\_back}}$) of orphan processes.

$$\begin{aligned}
\mathbf{E_{opt\_reco}} &= \left[\frac{1}{2\lambda_c}-\frac{1}{2\lambda_l}\right] C_{\text{reco}} + \left[\frac{1}{2\lambda_c}-\frac{1}{2\lambda_l}\right]\lambda_m C_{\text{replay}} \\
&\quad + \frac{1}{2\lambda_l}\lambda_m C_{\text{snr}} \\
&\quad + \frac{(n-1)}{2\lambda_c}\left[1-\left[1-\frac{\lambda_{\text{msg}}}{n-1}\right]^t\right] C_{\text{roll\_back}} \\
&\quad \text{where } t = \left[\frac{1}{2\lambda_l}\right] \\
&= \frac{1}{2\lambda_c}\left[C_{\text{reco}} + \lambda_m C_{\text{replay}}\right] \\
&\quad + \frac{1}{2\lambda_l}\left[(C_{\text{snr}} - C_{\text{replay}})\lambda_m - C_{\text{reco}}\right] \\
&\quad + \frac{(n-1)}{2\lambda_c}\left[1-\left[1-\frac{\lambda_{\text{msg}}}{n-1}\right]^t\right] C_{\text{roll\_back}}.
\end{aligned}$$

4.3. Causal message logging

Causal message logging protocols [1,2,8] neither create orphans when there are failures nor do they ever block a process when there is no failure. Dependency information is piggybacked on application messages. In order to make the system *f*-fault tolerant $(f+1)$ processes log the dependency information in their volatile storage. In this protocol, message contents are logged only in the volatile memory of the sender. Total message overhead due to *causal message logging* depends on $C_{\text{caus\_log}}$, the cost of storing a message in volatile memory. Total causal message logging cost per unit time is $\lambda_m[C_{\text{caus\_log}} + C_{\text{snr}}]$.

At any point of time, the probability of P_i sending a message to P_j is $\frac{\lambda_m}{n-1}$. Suppose the current time is τ. Probability that the last checkpoint before τ was taken at time $(\tau - t)$ is $(1-\lambda_c)^{t-1}\lambda_c$.

P (last message was sent to P_j at $\tau - i$| last checkpoint was taken at $\tau - t$) $= \left(1-\frac{\lambda_m}{n-1}\right)^{i-1}\frac{\lambda_m}{n-1} = q_i^t$ (say) for $i = 1, 2, 3, \ldots, t-1$.

P (there was no message to P_j since the last checkpoint | last checkpoint was taken at $\tau - t$) $= 1 - \sum_{i=1}^{t-1} q_i^t = \left(1-\frac{\lambda_m}{n-1}\right) = r_i^t$ (say) for $i = 1, 2, \ldots, t-1$.

E (time lapsed since the last message to P_j *or* since the last checkpoint, if there was no message | last checkpoint was taken at $\tau - t$) $= \sum_{i=1}^{t-1}\left[iq_i^t\right] + tr_i^t = \frac{1}{p^2}\left[2 + p + tp^2 - tp^3 - tp(1-p)^{t-1}\right] = s^t$, (say) where $p = \frac{\lambda_m}{n-1}$.

Therefore, **E** (time lapsed since the last message to P_j *or* since the last checkpoint, if there was no message) $= \sum_{t=1}^{\infty} s^t(1-\lambda_c)^{t-1}\lambda_c$.

Let C_{pgb} be the cost for one piggybacking information. Let $\mathbf{E_{pgb}}$ be the expected cost of piggybacking information. Therefore,

$$\mathbf{E_{pgb}} = C_{\text{pgb}}\lambda_m \sum_{t=1}^{\infty} s^t(1-\lambda_c)^{t-1}\lambda_c.$$

The average rollback recovery overhead ($\mathbf{E_{causal\_reco}}$) for a process is the sum of recovery cost, messages and determinants [2] collection and message replaying cost from the logs of from another process. Let C'_{replay} denote the cost of replaying a logged message from another process.

$$\begin{aligned}
\mathbf{E_{causal\_reco}} &= \frac{C_{\text{reco}}}{2\lambda_c} + \frac{\lambda_m(C'_{\text{replay}} + C_{\text{snr}})}{2\lambda_c} \\
&= \frac{1}{2\lambda_c}\left[C_{\text{reco}} + \lambda_m(C'_{\text{replay}} + C_{\text{snr}})\right].
\end{aligned}$$

5. Quasi-synchronous checkpointing with message logging

5.1. Checkpointing overhead

There are three factors contributing to checkpointing overhead in quasi-synchronous checkpointing protocol.

(1) Processes are allowed to take checkpoints asynchronously.

(2) Processes take forced checkpoints on receiving some application messages.
(3) Process may take checkpoint on receiving checkpoint request message from a process that wants to establish a CGS.

According to the algorithm proposed by Manivannan and Singhal [16] each checkpoint is assigned a unique sequence number. The sequence number assigned to a checkpoint is the current value of a counter. The local counters maintained by the individual processes are incremented periodically. The time period, T_{period}, is the same for all processes. Since the sequence numbers assigned to checkpoints of a process are picked from the local counters, the sequence numbers of the latest checkpoints of all the processes will remain close to each other. For simplicity, we assume that each process takes checkpoints periodically with fixed time period. The gap between two checkpoints T_{period} is the same as the period for incrementing the counters. The differences in the times for checkpoints in different processes will be due to the skew in their clocks. So the latest checkpoints of all processes are very likely to be in CGS. In this situation probability for forced checkpoint is very low. We can ignore the checkpointing overhead cost due to forced checkpoints.

In this protocol, checkpointing cost for a process is the sum of asynchronous checkpointing cost and cost of extra checkpoints which may be needed for establishing a CGS. Let $\lambda_c^{'}$ be the probability of taking a checkpoint for establishing a forced CGS. Since the processes do not establish forced CGS very frequently, we can safely assume that $\lambda_c^{'} << \lambda_c$.

Expected total checkpointing cost per unit time is

$$\mathbf{E}_{\mathbf{quasi\_ckpt}} = \frac{t_c}{T_p + t_c} + \frac{(1-(1-\lambda_c^{'})^n)t_c'}{1+(1-(1-\lambda_c^{'})^n)t_c'}.$$

5.2. Selective message logging

A *recovery line* (a globally consistent set of checkpoints) divides the set of all events of the computation into two disjoint parts. When a process rolls back, all those application messages whose *send* events lie to the left and the corresponding *receive* events lie to the right of the current recovery line are *lost messages*. All such messages should be replayed. To cope with messages lost due to a rollback, all such messages should be logged into stable storage. Manivannan and Singhal [16] proposed *selective message logging protocol* that logs only these messages instead of all messages.

In a distributed computing system processors are connected through communication links. We assume that a single process runs in a processor. The topology of the system may be represented by a graph. A node represents a process and an edge represents a communication link between a pair of nodes. The time for one hop message passing is assumed to be constant (t_{hop}) for all edges. Edges are bidirectional. The *distance* $d(i, j)$ between P_i and P_j is the length of the shortest path between them.

Definition 1. Let $G = (V, E)$ be any connected graph. For every node $v \in V$, we define the pathsum of v, $pathsum(v) \overset{def}{=} \sum_{u \in V} d(u, v)$. The maximum pathsum of G is defined as $MPS(G) \overset{def}{=} \max_{u\in V}\{pathsum(u)\}$.

Lemma 1. *Let $T = (V, E)$ be a tree. If $MPS(T) = pathsum(v)$ for some $v \in V$, then v is a leaf node of T.*

Proof. If possible, let v be a non-leaf node such that $MPS(T) = pathsum(v)$. Let the nodes adjacent to v be $u_1, u_2, \ldots, u_k$ for some $k \geqslant 2$. The removal of v splits T into k different trees, with $u_1, u_2, \ldots, u_k$ in different trees.

Let the number of nodes in the tree having u_i be n_i for $i = 1, 2, \ldots, k$. Without loss of generality, let $n_1 \leqslant n_2 \leqslant \cdots \leqslant n_k$. Let $|V| = n$. $\sum_{i=1}^{k} n_i = n - 1$. Then $n - n_1 = ((n-1) - n_1) + 1 = \sum_{i=2}^{k} n_i + 1 \geqslant n_2 + 1 > n_1$.

$pathsum(u_1) = pathsum(v) - n_1 + (n - n_1) > pathsum(v)$, which is a contradiction as $MPS(T) = pathsum(v) \geqslant pathsum(u)$ for any $u \in V$. □

Lemma 2. *For a path graph P_n with n nodes, $MPS(P_n) = \frac{n(n-1)}{2}$.*

Proof. By *Lemma* 1, $MPS(P_n) = pathsum(v)$ where v is a leaf node of the path. For a leaf node v,
$pathsum(v) = 1+2+3+\cdots+(n-1) = \frac{n(n-1)}{2}$. □

Lemma 3. *Let T_n be a tree with n nodes. Then $MPS(T_n) \leqslant MPS(P_n) = \frac{n(n-1)}{2}$.*

Proof. The result is true for $n = 1$. Suppose the result is true for $n = m$. Let v be any leaf node in T_{m+1}. Let $T_m = T_{m+1} - \{v\}$. $MPS(T_{m+1}) = MPS(T_m) + m \leqslant \frac{m(m-1)}{2} + m$ (induction hypothesis) $= \frac{m(m+1)}{2}$. □

Definition 2. For a connected graph G, we define the total pathsum of G to be $TPS(G) \overset{def}{=} \frac{1}{2}\sum_{v\in V(G)} pathsum(v)$.

Lemma 4. *For a path graph P_n, $TPS(P_n) = \frac{1}{6}(n^3 - n)$.* □

Theorem 1. *Let T_n be a tree with n nodes. Then $TPS(T_n) \leqslant \frac{1}{6}(n^3 - n)$.*

Proof. The result is true for $n = 1$. Suppose the result is true for $n = m$. Let v be a leaf node in T_{m+1}. $T_m = T_{m+1} - \{v\}$ is also a tree.

$$\begin{aligned} TPS(T_{m+1}) &= TPS(T_m) + pathsum(v) \\ &\leqslant \tfrac{1}{6}(m^3 - m) + MPS(T_{m+1}) \\ &\qquad \text{(induction hypothesis)} \\ &\leqslant \tfrac{1}{6}(m^3 - m) + \frac{m(m+1)}{2} \quad (\textit{Lemma } 3) \\ &= \tfrac{1}{6}((m+1)^3 - (m+1)). \qquad \square \end{aligned}$$

Theorem 2. *Suppose P_i and P_j are two processes which take checkpoints at t_i and t_j ($t_i \leqslant t_j$), respectively. Let $d(i, j)$*

denote the distance between them. If $d(i, j) > t_j - t_i$, P_j will log the messages, sent by P_i during the interval $[t_j - d(i, j),\ \ t_i)$; otherwise P_j will not log any message sent by P_i. P_j will log the messages, sent by P_i during the interval $[t_i - d(i, j),\ \ t_j)$.

Proof. Since $d(i, j)$ is the time taken by a message to reach P_j from P_i, the messages sent in the interval $[t_i - (d(i, j) - t_j + t_i),\ \ t_i)$ are the only ones which were sent before t_i (the checkpoint time for P_i) and reached after t_j (the checkpoint time for P_j). Hence, these are the only messages which are logged.

Similarly, the messages sent by P_j to P_i are logged if and only if they are sent in the given interval. □

Let us consider a distributed system with underlying topology $G = (V, E)$. Suppose process P_0 initiates checkpointing at t_0. Without loss of generality, let $\mathrm{d}(0,1) \leqslant \mathrm{d}(0,2) \leqslant \cdots \leqslant \mathrm{d}(0, n-1)$. We assume that all messages take a shortest path to the destination and each hop takes t_{hop} units of time with no congestion delay. For the checkpoint initiated by P_0, a process P_i $(1 \leqslant i \leqslant n-1)$ receives checkpoint request and takes checkpoint at $t_i = t_0 + \mathrm{d}(i, 0)t_{\text{hop}}$. It is also assumed that there is no other new request for checkpointing. Let E_{logged} be the expected number of messages logged by all processes.

Theorem 3. $t_{\text{hop}}\ \lambda_m\ n \leqslant E_{\text{logged}} \leqslant \frac{2}{3}\ t_{\text{hop}}\ \lambda_m\ n(n+1)$.

Proof. Applying Theorem 2, we see that P_i will log a message sent by P_j if and only if $i < j$. P_0 will log messages sent by P_i during $[t_i - \mathrm{d}(i,0)t_{\text{hop}}, t_i + \mathrm{d}(i,0)t_{\text{hop}})$, $1 \leqslant i \leqslant n-1$. So, the expected number of messages to be logged by P_0 is

$$\frac{2}{n-1}\ t_{\text{hop}}\ \lambda_m \sum_{i=1}^{n-1} \mathrm{d}(i, 0).$$

Similarly, process P_i is expected to log

$$\frac{2}{n-1}\ t_{\text{hop}}\ \lambda_m \sum_{k=i+1}^{n-1} \mathrm{d}(k, i)$$

messages from processes $P_{i+1}, P_{i+2}, \cdots, P_{n-1}$.

$$\begin{aligned} E_{\text{logged}} &= \frac{2}{n-1}\ t_{\text{hop}}\ \lambda_m \left[\sum_{i=1}^{n-1} \mathrm{d}(i,0) + \sum_{i=2}^{n-1} \mathrm{d}(i,1) \right. \\ &\quad + \cdots + \sum_{i=n-2}^{n-1} \mathrm{d}(i, n-3) \\ &\quad \left. + \ \mathrm{d}(n-1, n-2) \right] \\ &= \frac{4}{n-1}\ t_{\text{hop}}\ \lambda_m\ TPS(G) \quad \text{(Definition 2)} \\ &\leqslant \frac{2}{3}\ t_{\text{hop}}\ \lambda_m\ n(n+1) \quad \text{(Theorem 1).} \end{aligned}$$

Table 1
Checkpointing, message logging, recovery and piggybacking costs of different checkpointing, recovery and message logging schemes

	Synchronous checkpointing	Quasi-synchronous checkpointing	Asynchronous checkpointing		
		Selective logging	Pessimistic logging	Optimistic logging	Causal logging
Checkpointing cost	$\frac{(1-(1-\lambda_c)^n)t'_c}{1+(1-(1-\lambda_c)^n)t'_c}$	$\frac{t_c}{T_p+t_c} + \frac{(1-(1-\lambda'_c)^n)t'_c}{1+(1-(1-\lambda'_c)^n)t'_c}$	$\frac{t_c}{T_p+t_c}$	$\frac{t_c}{T_p+t_c}$	$\frac{t_c}{T_p+t_c}$
Message logging cost (C)	0	$t_{\text{hop}}\lambda_m \leqslant \mathrm{C} \leqslant \frac{2}{3}t_{\text{hop}}\lambda_m(n+1)$	$\lambda_m(C_{\text{snr}} + C_{\text{pessi\_log}})$	$\lambda_m(C_{\text{snr}} + C_{\text{opti\_log}})$	$\lambda_m(C_{\text{snr}} + C_{\text{caus\_log}})$
Recovery cost (R)	$\frac{C_{\text{reco}}}{2(1-(1-\lambda_c)^n)}$	$\frac{1}{2(1-(1-\lambda_c)^n)}[C_{\text{reco}} + \lambda_m t_{\text{hop}} C_{\text{replay}}] \leqslant R \leqslant \frac{1}{6(1-(1-\lambda_c)^n)} [3C_{\text{reco}} + 2(n+1)\lambda_m t_{\text{hop}} C_{\text{replay}}]$	$\frac{1}{2\lambda_c}(C_{\text{reco}} + \lambda_m C_{\text{replay}})$	$\frac{1}{2\lambda_c}[C_{\text{reco}} + \lambda_m C_{\text{replay}}] + \frac{1}{2\lambda_l}\left[(C_{\text{snr}} - C_{\text{replay}})\lambda_m - C_{\text{reco}}\right] + \frac{(n-1)}{2\lambda_c}\left[1 - \left[1 - \frac{\lambda_{msg}}{n-1}\right]^t\right] C_{\text{roll\_back}}$	$\frac{1}{2\lambda_c}(C_{\text{reco}} + \lambda_m(C'_{\text{replay}} + C_{\text{snr}}))$
Piggybacking cost	0	Constant	0	$C_{\text{pgb}}n$	$C_{\text{pgb}}\lambda_m \sum_{t=1}^{\infty} s^t(1-\lambda_c)^{t-1}\lambda_c$

Table 2
The recovery costs of different checkpointing and message logging algorithms for different values of λ_m

$1/\lambda_m$	Synchronous checkpointing	Quasi-synchronous checkpointing		Asynchronous checkpointing		
		Selective logging				
		Minimum	Maximum	Pessimistic logging	Optimistic logging	Causal logging
2	30.6601	38.3251	362.8110	2250.0000	3349.5680	4950.0000
10	30.6601	32.1931	97.0902	1890.0000	2090.0000	2430.0000
50	30.6601	30.9667	93.9461	1818.0000	1838.2400	1926.0000
100	30.6601	30.8134	37.3031	1809.0000	1806.6200	1863.0000
500	30.6601	30.6907	31.9887	1801.8000	1781.3200	1812.0000
1000	30.6601	30.6754	31.3244	1800.9000	1778.1600	1806.0000
5000	30.6601	30.6631	30.7929	1800.1800	1775.3000	1801.2600

$C_{\text{snr}} = 10$, $C_{\text{replay}} = 5.0$, $C'_{\text{replay}} = 25.0$, $C_{\text{reco}} = 10$, $C_{\text{roll\_back}} = 5$, $\lambda_l = \frac{1}{5}$, $\lambda_c = \frac{1}{360}$, $t_{\text{hop}} = 1.0$, $n = 64$.

Table 3
The recovery costs of different checkpointing and message logging algorithms for different values of λ_c

$1/\lambda_c$	Synchronous checkpointing	Quasi-synchronous checkpointing		Asynchronous checkpointing		
		Selective logging				
		Minimum	Maximum	Pessimistic logging	Optimistic logging	Causal logging
720	58.748	61.685	186.035	3780.000	4250.710	4860.000
1440	114.979	120.728	364.102	7560.000	8435.180	9720.000
2160	171.223	179.784	542.207	11340.000	12664.600	14580.000
2880	227.470	238.844	720.322	15120.000	16894.100	19440.000
3600	283.718	297.904	898.441	18900.000	21123.600	24300.000

$C_{\text{snr}} = 10$, $C_{\text{replay}} = 5.0$, $C'_{\text{replay}} = 25.0$, $C_{\text{reco}} = 10$, $C_{\text{roll\_back}} = 5$, $\lambda_l = \frac{1}{5}$, $\lambda_m = \frac{1}{10}$, $t_{\text{hop}} = 1.0$, $n = 64$.

Table 4
Message logging costs for different values of λ_m

$1/\lambda_m$	Selective logging		Pessimistic logging	Optimistic logging	Causal logging
	Minimum	Maximum			
5	0.2000	13.0000	22.0000	14.0000	4.0000
7	0.1429	9.2857	15.7143	10.0000	2.8571
12	0.0833	5.4167	9.1667	5.8333	1.6667
15	0.0667	4.3333	7.3333	4.6667	1.3333

$C_{\text{snr}} = 10$, $C_{\text{pessi\_log}} = 100$, $C_{\text{opti\_log}} = 60$, $C_{\text{caus\_log}} = 10$, $\lambda_m = \frac{1}{3600}$, $t_{\text{hop}} = 1.0$, $n = 64$.

It is easy to see that in a complete graph the least number of messages would be logged. Checkpointing message reaches all other processes in the very next moment. A message would be logged only if the message is sent during the time when the message travels. So, $E_{\text{logged}} \geqslant t_{\text{hop}}\, \lambda_m\, n$. □

5.3. Rollback recovery overhead

While recovering from a failure, the failed process P_i rolls back to its latest checkpoint, and all other processes P_j, $j \neq i$, $j \in \{0, 1, 2, 3, \ldots, n-1\}$, rollback to their last checkpoint with checkpoint sequence number greater than or equal to the checkpoint sequence number of the failed process. If such a checkpoint does not exist, P_j takes a checkpoint with checkpoint sequence number equal to that of the failed process, P_i.

The average rollback recovery overhead for a process is the sum of the recovery cost and the message replaying cost from the stable storage which have been logged selectively. Expected minimum message-replaying cost for all processes is

$$\frac{n\lambda_m}{2(1-(1-\lambda_c)^n)}\, t_{\text{hop}}\, C_{\text{replay}}$$

Table 5
Maximum and minimum numbers of messages logged in selective message logging protocol for different values of λ_m

$1/\lambda_m$	Minimum number of message logging	Maximum number of message logging
5	12.800	832.0
10	6.400	416.0
20	3.200	208.0
30	2.133	138.6
40	1.600	104.0
50	1.280	83.2
100	0.640	41.6

$t_{\text{hop}} = 1.0,\ n = 64.$

Table 6
Message logging costs for different values of n

n	Minimum number of message logging	Maximum number of message logging
16	1.6	72.2
32	3.2	105.6
64	6.4	416.0
128	12.8	1651.2
256	25.6	6579.2
512	51.2	26265.6
1024	102.4	104960.0

$\lambda_m = \frac{1}{10},\ t_{\text{hop}} = 1.0.$

Table 7
Checkpointing cost of different checkpointing schemes for different values of λ_c

$1/\lambda_c$	Synchronous checkpointing	Quasi-synchronous checkpointing	Asynchronous checkpointing
360	0.9548	0.6698	0.2173
720	0.9168	0.5744	0.1219
1440	0.8492	0.5174	0.0649
2160	0.7909	0.4967	0.0442
2880	0.7400	0.4860	0.0335
3600	0.6953	0.4795	0.0270

$C_{\text{snr}} = 10,\ t_c = 100.0,\ \lambda'_c = \frac{1}{10000},\ T_p = \frac{1}{\lambda_c},\ n = 64.$

and maximum message-replaying cost for all processes is

$$\frac{n(n+1)\lambda_m}{3(1-(1-\lambda_c)^n)}\, t_{\text{hop}}\, C_{\text{replay}}.$$

Minimum rollback recovery overhead for a process is

$$\mathbf{E}_{\textbf{Min\_quasi\_reco}} = \frac{1}{2(1-(1-\lambda_c)^n)}\big[C_{\text{reco}} + \lambda_m\, t_{\text{hop}}\, C_{\text{replay}}\big].$$

Maximum rollback recovery overhead for a process is

$$\mathbf{E}_{\textbf{Max\_quasi\_reco}} = \frac{1}{6(1-(1-\lambda_c)^n)}\big[3C_{\text{reco}} + 2(n+1)\lambda_m\, t_{\text{hop}}\, C_{\text{replay}}\big].$$

Table 1 shows the analytical expressions for different types of overheads under different checkpointing schemes. These expressions have been used to evaluate the overheads for different checkpointing schemes. Tables 2 and 3 show the recovery costs of different checkpointing and message logging schemes for different values of λ_m and λ_c, respectively. Table 2 shows that with decreasing message sending rate λ_m, the recovery cost of optimistic logging decreases faster than the recovery costs of pessimistic and causal logging.

Table 4 compares the message logging costs of quasi-synchronous and asynchronous algorithms for different values of λ_m. In selective message logging, maximum message logging cost is less than the message logging cost of pessimistic and optimistic ones but it is greater than the cost of causal logging for different values of λ_m. Minimum message logging cost in selective logging is very less compared to any other message logging cost for different values of λ_m.

Table 5 shows maximum and minimum message logging cost for different values of λ_m in selective message logging protocol. Table 6 compares the message logging costs for selective message logging protocol for different values of n, the number of processes. Table 7 shows checkpointing cost of synchronous, quasi-synchronous and asynchronous checkpointing schemes for different values of checkpointing rate λ_c. Checkpointing cost of quasi-synchronous scheme always lies between the checkpointing costs of synchronous and asynchronous schemes for different values of λ_c.

6. Conclusion

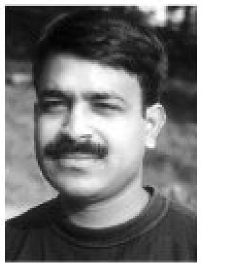
In this work, we have calculated expected costs of different types of checkpointing algorithms such as synchronous, asynchronous and quasi-synchronous alongwith their rollback recovery algorithms with message logging and without message logging. These formulae have been used to evaluate the overheads of checkpointing, rollback recovery, message logging, and message piggybacking for different techniques. It has been found that with decreasing message sending rate λ_m, the recovery cost of optimistic logging decreases faster than the recovery costs of pessimistic and causal logging. In selective message logging, maximum message logging cost is less than the message logging costs of pessimistic and optimistic ones, but it is greater than the cost of causal logging for different values of λ_m. Minimum message logging cost in selective logging is much less than any other message logging cost, for different values of λ_m. Checkpointing cost of synchronous checkpointing algorithm is greater than the asynchronous checkpointing algorithm for different values of λ_c. But the checkpointing cost of quasi-synchronous algorithm lies between the checkpointing costs of synchronous and asynchronous checkpointing algorithms.

Acknowledgments

The first author is thankful to *Council of Scientific and Industrial Research* (*CSIR*), *India*, for financial support during this work. The authors are grateful to the anonymous reviewers and Professor N. Das of ACM Unit, Indian Statistical Institute, Kolkata, for their many helpful comments and suggestions.

References

[1] L. Alvisi, K. Bhatia, K. Marzullo, Nonblocking and orphan free message logging protocols, in: Proceedings of 23rd Fault-Tolerant Computing Symposium, June 1993, pp. 145–154.
[2] L. Alvisi, B. Hoppe, K. Marzullo, Causality tracking in causal message-logging protocols, Distrib. Comput. 15 (2002) 1–15.
[3] G. Cao, M. Singhal, On coordinated checkpointing in distributed systems, IEEE Trans. Parallel Distrib. Syst. 9 (12) (1998) 1213–1225.
[4] K.M. Chandy, L. Lamport, Distributed snapshots: determining global states of distributed systems, ACM Trans. Comput. Syst. 3 (1) (1985) 63–75.
[5] O.P. Damani, V.K. Garg, How to recover efficiently and asynchronously when optimism fails, in: Proceedings of IEEE International Conference on Distributed Computing Systems, 1996, pp. 108–115.
[6] E.N. Elnozahy, D.B. Johnsone, W. Zwaenepoel, The performance of consistent checkpointing, in: Proceedings of 11th Symposium on Reliable Distributed Systems, 1992, pp. 86–95.
[7] E.N. Elnozahy, L. Alvisi, Y.-M. Wang, D.B. Johnson, A survey of rollback-recovery protocols in message-passing systems, ACM Comput. Surveys 34 (3) (2002) 375–408.
[8] E.N. Elnozahy, W. Zwaenepoel, Manetho: transparent rollback recovery with low overhead, limited rollback and fast output commit, IEEE Trans. Comput. 41 (5) (1992) 526–531.
[9] D. Johnson, W. Zwaenepoel, Sender-based message logging and checkpointing, in: Proceedings of 17th Annual International Symposium on Fault-Tolerant Computing, IEEE Computer Society, June 1987, pp. 14–19.
[10] D. Johnson, W. Zwaenepoel, Recovery in distributed systems using optimistic message logging and checkpointing, J. Algorithms 3 (11) (1990) 462–491.
[11] J.L. Kin, T. Park, An efficient protocol for checkpointing recovery in distributed systems, IEEE Trans. Parallel Distrib. Syst. 5 (8) (1998) 955–960.
[12] R. Koo, S. Toueg, Checkpointing and rollback-recovery for distributed systems, IEEE Trans. Software Engrg. 13 (1) (1987) 23–31.
[13] P.S. Mandal, K. Mukhopadhyaya, Mobile agent based checkpointing and recovery algorithms on a distributed system, in: Proceedings of Sixth International Conference Exhibition on High Performance Computing in Asia Pacific Region, Bangalore, India, 2, December 2002, pp. 492–499.
[14] P.S. Mandal, K. Mukhopadhyaya, Concurrent checkpoint initiation and recovery algorithms on asynchronous ring networks, J. Parallel Distrib. Comput. 64 (5) (2004) 649–661.
[15] D. Manivannan, M. Singhal, Quasi-synchronous checkpointing: models, characterization, and classification, IEEE Trans. Parallel Distrib. Syst. 10 (7) (1999) 703–713.
[16] D. Manivannan, M. Singhal, Asynchronous recovery without using vector timestamps, J. Parallel Distrib. Comput. 62 (2002) 1695–1728.
[17] K.Z. Meth, W.G. Tuel, Parallel checkpoint/restart without message logging, in: Proceedings of IEEE 28th International Conference on Parallel Processing (ICPP '00), August 2000, pp. 253–258.
[18] B.S. Panda, S.K. Das, Performance evaluation of a two level error recovery scheme for distributed systems, in: Proceedings of Fourth International Workshop on Distributed Computing, Springer, December 2002, pp. 88–97.
[19] J.S. Plank, M.G. Thomason, Processor allocation and checkpoint interval selection in cluster computing systems, J. Parallel Distrib. Comput. 61 (2001) 1570–1590.
[20] R. Prakash, M. Singhal, Low-cost checkpointing and failure recovery in mobile computing systems, IEEE Trans. Parallel Distrib. Syst. 7 (10) (1996) 1035–1048.
[21] S. Rao, L. Alvisi, H.M. Vin, The cost of recovery in message logging protocols, IEEE Trans. Knowledge Data Engrg. 12 (2) (2000) 160–173.
[22] L.M. Silva, J.G. Silva, Global checkpointing for distributed systems, in: Proceedings of 11th Symposium on Reliable Distributed Systems, 1992, pp. 155–162.
[23] A.P. Sistla, J. Welch, Efficient distributed recovery using message logging, in: Proceedings of the ACM Symposium on Principle of Distributed Computing, 1989, pp. 223–238.
[24] M. Spezialetti, P. Kearns, Efficient distributed snapshots, in: Proceedings of the Sixth ICDCS, 1986, pp. 382–388.
[25] R.E. Strom, D.F. Bacon, S. Yemini, Volatile logging in *n*-fault-tolerant distributed systems, in: Proceedings of 18th Annual International Symposium on Fault-Tolerant Computing, 1988, pp. 44–49.
[26] R.E. Strom, S. Yemini, Optimistic recovery in distributed systems, ACM Trans. on Computer Syst. 3 (3) (1985) 204–226.
[27] N.H. Vaidya, A case for two-level recovery schemes, IEEE Trans. Computers 47 (1998) 656–666.
[28] S. Venkatesan, T.-Y. Juang Tony, Efficient algorithms for optimistic crash recovery, Distrib. Comput. 8 (2) (1994) 105–114.
[29] Y.M. Wang, Consistent global checkpoints that contain a given set of local checkpoints, IEEE Trans. Comput. 46 (4) (1997) 456–468.



Partha Sarathi Mandal received a Bachelor of Science (Hons.) degree in Mathematics from the University of Calcutta, India, a Master of Science degree in Mathematics from Jadavpur University, India, in 1995 and 1997, respectively. He is awarded Junior and Senior Research Fellowship by the Council of Scientific & Industrial Research (CSIR), India. He is currently working towards his Ph.D. degree in Computer Science at the Advanced Computing and Microelectronics Unit of the Indian Statistical Institute, Kolkata. His current research interests include parallel and distributed computing, fault tolerance, mobile agents, performance analysis, self-stabilization, etc.

Krishnendu Mukhopadhyaya received his Bachelor of Statistics (Hons.), Master of Statistics, Master of Technology in Computer Science, and Ph.D. in Computer Science all from the Indian Statistical Institute, Kolkata, in 1985, 1987, 1989 and 1994, respectively. From 1993 to 1999, he worked as a Lecturer in the Department of Mathematics, Jadavpur University. Since 1999, he is working at the Indian Statistical Institute, Kolkata as an Associate Professor. He was a recipient of the Young Scientist Award of the Indian Science Congress Association and the BOYSCAST Fellowship of the Department of Science and Technology, Government of India. His current research interests include mobile computing, parallel and distributed computing, sensor networks, etc. He has served as a member of the technical program committees of international conferences like HiPC, VTC, etc.