Evaluating Distributed Checkpointing Protocols

Adnan Agbaria
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
adnan@crhc.uiuc.edu

Ari Freund  Roy Friedman
Computer Science Department
Technion - Israel Institute of Technology
Haifa 32000, Israel
{arief,roy}@cs.technion.ac.il

Abstract

This paper presents an objective measure, called overhead ratio, for evaluating distributed checkpointing protocols. This measure extends previous evaluation schemes by incorporating several additional parameters that are inherent in distributed environments. In particular, we take into account the rollback propagation of the protocol, which impacts the length of the recovery process, and therefore the expected program run-time in executions that involve failures and recoveries. The paper also analyzes several known protocols and compares their overhead ratio.

1. Introduction

Checkpoint/Restart (C/R) is one of the most prominent techniques for providing fault-tolerance, and can also be used for debugging and migration in both uniprocessor and distributed systems [8, 13]. Specifically, checkpointing is the act of saving a program's state on stable storage, and restart is the act of restarting an application from its saved state. In particular, if an application takes periodic checkpoints, then in case of a failure, it is possible to restart it from the latest checkpoint, thereby avoiding losing all the computation that was carried before that checkpoint.

One of the main challenges in implementing C/R mechanisms is maintaining low overhead, since otherwise the cost of taking a checkpoint will outweigh its potential benefit. For programs executing on a single computer, the main focus is on interleaving the task of saving the program state with the program execution and on reducing the total size of the file being saved. For parallel or distributed applications, the situation is more complicated. If each process saves its state in a completely independent manner, it is possible that no collection of checkpoints, one from each process, will correspond to a consistent application state, also known as a recovery line. Thus, the main research focus in this area is on devising techniques that guarantee the existence of a recovery line while minimizing the coordination between processes. This coordination overhead includes the amount of control information exchanged between processes and the number of times some process \( p \) is forced to take a checkpoint to ensure that a recovery line exists. Such checkpoints are called induced or forced checkpoints.

Over the years, a multitude of C/R techniques have been proposed. This has created a need for objective quantitative evaluation measures that allow one to compare various approaches on the same scale. One such scheme has been developed by Vaidya in the context of applications running on a single computer [17], but it does not extend trivially to most distributed C/R mechanisms. There are two reasons for this. First, Vaidya's work does not take into account the communication costs associated with distributed checkpointing, and second, it assumes that each process always restarts from its most recent checkpoint. This assumption cannot always be made in distributed C/R schemes, since these schemes often require a process to rollback after a failure to a more distant checkpoint [1].

In this paper we generalize Vaidya’s framework [17] to distributed checkpointing schemes, by incorporating into it the communication overhead and rollback propagation. In particular, we show that when a checkpoint mechanism guarantees that a process will always rollback to its most recent checkpoint, as Vaidya’s assumption, our evaluation measure agrees with Vaidya's results.

2. System Model and Definitions

We assume the typical distributed computing model, in which processes communicate by exchanging messages, and are modeled as automata. We also assume that each event in the system is associated with a time \( t(event) \). Each checkpoint taken by a process is assigned a unique sequence number. The \( i \)th checkpoint of process \( p \) is denoted \( C_{p,i} \). The \( i \)th checkpoint interval of process \( p \), de-
Definition 2.1: The *checkpoint overhead*, denoted $o$, is the increase in the execution time of a process $p$ because of a single checkpoint.

Definition 2.2: The *checkpoint latency*, denoted $L$, is the duration required to take a single checkpoint.

For the purpose of evaluation, we further assume that every process takes an independent checkpoint every duration required to take a single checkpoint. Thus, not all cuts of checkpoints are consistent, i.e., correspond to a state that could have been reached in the execution. A consistent cut of checkpoints is called a *recovery line*.

Definition 2.4: A cut of checkpoints $S$ is a *recovery line* if for each message received in $S$, the corresponding send event is also included in $S$.

We are now ready to explain the notion of k-rollback [1] which we use for defining our evaluation tools.

Definition 2.5: Given two cuts of checkpoints $S_1$ and $S_2$, $S_1S_2$ denotes the cut of process $p$ in $S_1$. We denote $S_1 \leq S_2$ if for every process $p$, either $S_1[p] \triangleq S_2[p]$ or $S_1[p] = S_2[p]$.

Definition 2.6: Given two checkpoints $C_{p,i}$ and $C_{p,j}$ of the same process $p$, the *distance* between them, denoted $\text{dist}(C_{p,i}, C_{p,j})$, is $|i - j|$. The distance between two cuts of checkpoints $S_1$ and $S_2$, denoted $\text{dist}(S_1, S_2)$, is $\max_p(\text{dist}(S_1[p], S_2[p]))$.

Definition 2.7: The cut of checkpoints derived from $C_{p,i}$, denoted $\text{Cut}(C_{p,i})$, is the set of the latest checkpoints from each process that occurred at or before time $t(C_{p,i})$.

Definition 2.8: The *checkpoint interleaving level* of an execution $E$, denoted $\text{IL}(E)$, is the minimal number $l$ such that for all processes $p$, all pairs of consecutive checkpoints $(C_{p,i}, C_{p,i+1})$, no process $q \neq p$ takes more than $l$ checkpoints in the time interval $[t(C_{p,i}), t(C_{p,i+1})]$.

Definition 2.9: An execution $E$ is a k-rollback for a given integer $k \geq 0$, if for every checkpoint $C_{p,i}$ there is a recovery line $R \in E$ such that $R \leq \text{Cut}(C_{p,i})$ and $\text{dist}(R, \text{Cut}(C_{p,i})) \leq k \cdot \text{IL}(E)$. If there is no $k$ such that $E$ is k-rollback, then $E$ is unbounded-rollback. The k-rollback class is the set of all k-rollback executions. We slightly abuse the terminology by omitting the word “class”.

Many distributed checkpointing protocols produce control overhead [8]. Control overhead is the overhead due to control information. Given a checkpointing protocol $\mathcal{P}$, we denote by $\text{expctMsgsNum}(\mathcal{P})$ the expected number of control messages in a single checkpoint interval, and by $\text{expctMsgsSize}(\mathcal{P})$ the expected total size of the control information for a single checkpoint interval. The expected control overhead of $\mathcal{P}$ is therefore $M(\mathcal{P}) = \text{expctMsgsNum}(\mathcal{P}) \cdot w_m + \text{expctMsgsSize}(\mathcal{P}) \cdot w_b$, where $w_m$ is the “setup” time for sending a message, and $w_b$ is the additional per-bit delay associated with sending a message.

The *total checkpoint overhead*, denoted $O$, is the increase in the execution time of a process $p$ because of a checkpoint $C_{p,i}$ and the control overhead corresponding to $C_{p,i}$. Namely, $O = o + M$.

If the control information is piggybacked on application messages, then the communication-pattern of the execution determines the control overhead. Therefore, for some executions we need to determine the message rate, as defined below, to compute the control overhead.

Definition 2.10: Given an execution $E$, the *message rate* of $E$, denoted $\text{MR}(E)$, is the expected number of data messages sent in a checkpoint interval $I_{p,i} \in E$.

Given an execution $E$, we assume that there is a recovery mechanism that finds the most advanced recovery line in $E$. A recovery line $R$ is said to be the most advanced recovery line if there is no other recovery line $R'$ such that $R \leq R'$.

3. The Overhead Ratio

Consider an execution $E \in k$-rollback, and a process $p \in E$. Suppose that $p$ is running in the checkpoint interval $I_{p,i+1}$, namely, it has taken the checkpoint $C_{p,i}$ but not yet $C_{p,i+1}$. If a failure occurs during $I_{p,i+1}$, then $p$ needs to recover from the newest exploited checkpoint. By [1], since $E \in k$-rollback, $p$ needs to rollback no more than $k \cdot \text{IL}(E)$ checkpoints backwards.

Consider an execution $E$ with a checkpointing protocol $\mathcal{P}$. Denote by $F(\mathcal{P})$ the expected number of forced checkpoints that occur during $T$ when there are no failures, then the expected time of a checkpoint interval $I_{p,i} \in E$ is $\tau(\mathcal{P}) = F(\mathcal{P}) + T$. Let $\tau'$ be the expected time of a checkpoint interval without the checkpoint and control overheads, namely, $\tau' = \tau - O$. 
On the other hand, if one or more failures occur during $I_{p,i}$, then the expected time of $I_{p,i}$ is more than $\tau$. After a failure, process $p$ must rollback, incurring $r$ units of time due to recovery. Moreover, after the rollback, $L - o$ units of computation that were performed during the checkpoint latency should be performed again. This is necessary because the computation that happened concurrently with the checkpoint is not part of the saved data [14, 17]. Therefore, in the presence of one failure, $\tau + r + (L - o)$ units of time are required to complete the checkpoint interval.

We augment the definition of the overhead ratio as defined in [17, 20] to our generalized model as follows.

**Definition 3.1:** Consider an execution $E \in k$-rollback with a checkpointing protocol $P$. Let $\Gamma_k(P)$ be the expected execution time of a checkpoint interval $I_{p,i} \in E$. The overhead ratio of $P$, denoted $v(k, P)$, is $v(k, P) = \frac{\Gamma_k(P) - \tau(P)}{\tau(P)} = \frac{\Gamma_k(P) - \tau_k(P)}{\tau_k(P)} - 1 = \frac{\Gamma_k(P)(F(P)+1)}{F(P)+1} - 1$.

When we do not care about the checkpointing protocol, we denote the overhead ratio as $v(k) = \frac{\Gamma_k(F+1)}{P+1} - 1$. Notice that $v(k) \geq 0$ for every $k \geq 0$. Moreover, a smaller overhead ratio corresponds to a better execution. The overhead ratio is the ratio between the total overhead of C/R and the computation events in a checkpoint interval. Therefore, for $v(k) \geq 1$, we have that the total overhead of C/R is bigger than the computation events in a checkpoint interval.

Recall that $T$ is a constant and $F$ can be computed by either theoretical analysis or experimental work [4]. Therefore, we only need to compute $\Gamma_k$, which depends on the $k$-rollback class that the execution belongs to.

### 3.1. Computing $\Gamma_k$ Using Markov Chains

We compute $\Gamma_k$ by constructing a finite-state Markov chain [15] for the $k$-rollback class of executions. For simplicity, we assume that $IL(E) = 1$ for an execution $E$. We then extract $\Gamma_k$ from the Markov chain.

Consider process $p$ running in $I_{p,i+1}$ of a 0-rollback execution. $\Gamma_0$ can be computed using the Markov chain presented in Figure 1. Process $p$ starts the interval in the start state $i$ (related to the checkpoint $C_{p,i}$). A transition from state $i$ to the sink state $i + 1$ occurs if $I_{p,i+1}$ is completed without failures. If a failure occurs during $I_{p,i+1}$, then $p$ recovers from $C_{p,i}$. In this case, we have a transition from state $i$ to $R_i$. After $R_i$ is entered, a transition is made to state $i + 1$ if no further failure occurs in $I_{p,i+1}$ after a recovery. Otherwise, a transition is made from state $R_i$ to itself.

Let $M$ be a Markov chain that represents the interval $I_{p,i+1}$ in a $k$-rollback execution. Let $s, t$ be states in $M$ such that there is a transition from $s$ to $t$ in $M$. We denote by $P_{s,t}$ the probability of the transition from $s$ to $t$ and by $W_{s,t}$ the expected execution time spent in state $s$ before moving to state $t$. State $s \in M$ is called recovery if a transition to this state is due to a failure. For instance, in Figure 1, only state $R_i$ is a recovery state. Furthermore, for each state $s \in M$, we define the variable $X_s$ to be the expected cost of reaching the sink state $i + 1$ from state $s$. Actually, $X_s$ equals the expected cost of all possible paths in $M$ from state $s$ to state $i + 1$ weighted by their probabilities. Therefore, $\Gamma_k = X_i$.

In Figure 1, since there are two possible paths from state $i$ to state $i + 1$, then $X_i = P_{i,i+1}W_{i,i+1} + P_{i,R_i}(W_{i,R_i} + X_{R_i})$. Therefore, we have the following two linear equations: $X_i = P_{i,i+1}W_{i,i+1} + P_{i,R_i}(W_{i,R_i} + X_{R_i})$ and $X_{R_i} = P_{R_i,R_i}(W_{R_i,R_i} + X_{R_i}) + P_{R_i,R_i+1}W_{R_i,R_i+1}$. After solving these two linear equations and substituting $P_{R_i,R_i+1} = 1 - P_{R_i,R_i}$, we have that $\Gamma_0 = X_i = P_{R_i,R_i}(W_{i,R_i} + X_{R_i}) + P_{i,i+1}W_{i,i+1}$.

Given an execution $E \in 1$-rollback, if a failure occurs in process $p$ during $I_{p,i+1}$, then $p$ can rollback either to $C_{p,i}$ or $C_{p,i-1}$ [1]. The Markov chain for $\Gamma_1$ presented in Figure 2.

![Figure 1. $I_{p,i+1} \in 0$-rollback](image1)

A transition to state $R_i$ represents that a recovery is made from $C_{p,i}$. Since we consider a recovery mechanism that finds the most recent recovery line, after entering state $R_i$, if there is an additional failure, then the recovery will be made from $C_{p,i}$. On the other hand, a transition to state $R_{i-1}$ represents that a recovery is made from $C_{p,i-1}$. In this case, process $p$ is rolled back to re-execute the checkpoint interval $I_{p,i}$. After state $R_{i-1}$ is entered, a transition is made to state $i$ after completing $I_{p,i}$. When a further failure occurs, a transition is made from state $R_{i-1}$ to itself. The corresponding linear equations were dropped due to lack of space, and appear in the full version of this paper.

![Figure 2. $I_{p,i+1} \in 1$-rollback](image2)

Given an execution $E \in 2$-rollback, if a failure occurs in process $p$ during $I_{p,i+1}$, then the rollback could be from $C_{p,i-2}$. Therefore, the Markov chain of $2$-rollback computing $\Gamma_2$ should contain transitions from state $i$ to the recovery states $R_i$, $R_{i-1}$, and $R_{i-2}$, as presented in Figure 3.

The states $i$ and $i_1$ in Figure 3 represent the same state of the execution, where process $p$ runs in $I_{p,i+1}$. However, we use two different states in the Markov chain to capture the situation that if process $p$ has rolled back to $C_{p,i-1}$, then it...
will not rollback to $C_{p,i-2}$. This means that there is no transition path in Figure 3 from state $R_{i-1}$ to state $R_{i-2}$. The probabilities and costs from states $i$ and $i_1$ to state $i + 1$ are identical. Thus, $P_{i,i+1} = P_{i_1,i+1}$ and $W_{i,i+1} = W_{i_1,i+1}$. Due to space limitation, we dropped the corresponding calculation of $\Gamma_2$, based on these diagrams, but the details can be found in the full version of this paper.

Figure 4 presents the Markov chain for $k$-rollback. There are $m = \frac{(k+2)(k+1)}{2}$ states apart from the sink state $i + 1$.

3.2. Computing the Overhead Ratio

A transition from state $i$ to $i + 1$ occurs if $I_{p,i+1}$ is completed without failures. If this transition is made, then $\tau$ units of time are spent where $\tau = \frac{I}{k+1}$, namely, $W_{i,i+1} = \tau$. We assumed that failures are governed by a Poisson process with rate $\lambda$. Therefore, the probability that there is no failure during $\tau$ units of time is $P_{i,i+1} = e^{-\lambda \tau}$. Since for every $l$, $0 \leq l \leq k - 1$, the probability and expected cost from state $i - l$ to state $i - l + 1$ is equal, we have that $P_{i-l,i-l+1} = e^{-\lambda \tau}$, $W_{i-l,i-l+1} = \tau$, where $0 \leq l \leq k - 1$.

On the other hand, if a failure occurs during $I_{p,i+1}$, then the rollback is made to the newest exploited checkpoint $C_{p,i-l}$ for $0 \leq l \leq k$. Therefore, a transition is made from state $i$ to one of the states $\{R_{i-l} \mid 0 \leq l \leq k\}$. The probability of such a transition is equal to $1 - P_{i,i+1}$. We denote $P_{i,R_{i-l}} = \mu_l(1 - P_{i,i+1})$ for $0 \leq l \leq k$ and $\sum_{l=0}^{k} \mu_l = 1$, where the value of $\mu_l$ depends on the communication pattern and the checkpointing protocol. The cost of this transition, $W_{i,R_{i-l}}$, is the expected execution time for $I_{p,i+1}$ until a failure occurs. Given that a failure occurs in the interval $[0, \tau]$ during the execution of $I_{p,i+1}$, the time to failure (TTF) is a random variable $x$ in the interval $[0, \tau]$ [15]. Moreover, its probability density function (PDF) is $\lambda e^{-\lambda x}$ for $0 \leq x < \tau$, and its conditional density function is $f_{i}(x) = \frac{\lambda e^{-\lambda x}}{P_{i,R_{i-l}}}$, where $0 \leq x < \tau$ and $0 \leq l \leq k$. Implying that $W_{i,R_{i-l}} = \int_0^\tau x \cdot f_{i}(x) dx = \frac{1}{\lambda} - \frac{e^{-\lambda x}}{1 - e^{-\lambda x}}$, where $0 \leq l \leq k$.

Therefore, for every $m$, $0 \leq m \leq k - 1$, the probability of transition from every state $i - m$ to state $R_{i-m-1}$ is $P_{i-m,R_{i-m-1}} = \mu_l[1 - e^{-\lambda \tau}]$, where $\sum_{l=0}^{k-m} \mu_l = 1$, and $W_{i-m,R_{i-m-1}} = \frac{1}{\lambda} - \frac{e^{-\lambda \tau}}{1 - e^{-\lambda \tau}}$, where $0 \leq l \leq k - m$.

For every $l$, $0 \leq l \leq k$, after state $R_{i-l}$ of Figure 4 is entered, a transition to state $i + 1 - l$ is made if no further failure occurs before $I_{p,i+1}$ is completed. The execution time required to reach state $i + 1 - l$ is $W_{R_{i-l},i+1-l} = \tau + r + L - o$. Therefore, the probability that no additional failure occurs is $P_{R_{i-l},i+1-l} = e^{-\lambda (\tau + r + L - o)}$. It follows that $P_{R_{i-l},i+1-l} = e^{-\lambda (\tau + r + L - o)}$, $W_{R_{i-l},i+1-l} = \tau + r + L - o$, where $0 \leq l \leq k$.

If another failure occurs after being in state $R_{i-l}$, $0 \leq l \leq k$, a transition is made from state $R_{i-l}$ to itself. For this transition we have $P_{R_{i-l},R_{i-l}} = 1 - P_{R_{i-l},i+1-l} = 1 - e^{-\lambda (\tau + r + L - o)}$. As discussed above, $W_{R_{i-l},R_{i-l}}$ can be obtained in a similar way to $W_{i,R_{i-l}}$. The PDF of the TTF is $\lambda e^{-\lambda x}$ for $x \in [0, \tau + r + L - o]$. Thus we have, $P_{R_{i-l},R_{i-l}} = 1 - e^{-\lambda (\tau + r + L - o)}$, $W_{R_{i-l},R_{i-l}} = \frac{1}{\lambda} - \frac{(\tau + r + L - o)e^{-\lambda (\tau + r + L - o)}}{1 - e^{-\lambda (\tau + r + L - o)}}$, where $0 \leq l \leq k$. By solving these equations, we can obtain $\Gamma_k$ and $v(k)$.

We would like to use the optimum value of $T$ that will yield a minimum overhead ratio. We consider two computing environments: SMALL and BIG. In both environments we consider seconds as time units. SMALL represents small and short-running applications executed in the Starfish system [2]. In previous experimental work, presented in [3], we ran a matrix multiplication application in Starfish with checkpointing. From this experimental work we obtained the following parameters: $o = 1.78$, $L = 4.292$, and $r = 3.32$. On the other hand, BIG represents a long-running distributed application. In this environment, applications run several days and produce large checkpoint files. For this environment, we use the simulation and experimental work performed by Plank and Thomason [14] where they obtained the following parameters: $o = 316.7$, $L = 5375.5$, and $r = 5375.5$. Moreover, both environments use a Fast Ethernet network (100 Mb/s). Hence, $w_m = 0.000276$ and $w_b = 10^{-9}$.

In both environments we assume that the failure rate of a single process is $\lambda = 1.23 \cdot 10^{-6}$ [14, 17]. In addition, for $k$-rollback execution $E$ and process $p \in E$, we first assume equal probabilities of recovering from the possible checkpoints, that means the probability to rollback to any $C_{p,i-j}$, $1 \leq j < k$, is the same. We also consider decreasing probabilities of recovering from the possible checkpoints, that is, if process $p$ fails during the checkpoint interval $I_{p,i+1}$, then the probability of recovering $p$ from $C_{p,j}$ is twice as big as the probability of recovering from $C_{p,j-1}$ for $i - k < j \leq i$ (given that $i \geq k$).
Figure 5 depicts $v(k)$ vs. $T$ in SMALL and BIG. Where $F = 0$, $n = 8$, and $M = w_{\text{pr}} + \mu_b$ (we discuss the impact of this choice of values later). The figure shows that for every value of $k$ we have an optimum $T$ that achieves the minimum $v(k)$. Notice that since BIG has long-running applications, the optimum $T$ values, as presented in Figure 5(b), are larger than in SMALL, as presented in Figure 5(a). Moreover, in both environments we figure out that $v(k)$ is a monotonic function with $k$, and for larger $k$ the minimum value for $v(k)$ is obtained when $T$ is small. This is because with larger $k$ the execution loses less computation events with smaller $T$.

Figure 5. A minimum $v(k)$ could be achieved.

Figure 6 depicts the value of $v(k)$ vs. $T$ as in Figure 5(a) but with decreasing probabilities of recovering. We also have here an optimum $T$ for every value of $k$. However, $v(k)$ is a slow monotonic function with $k$, and as can be seen, $v(16)$ is almost the same as $v(25)$.

Consider figures 5(a) and 6(a). For every $k$, $v(k)$ obtains the minimum value in the interval $[0, 500]$ of $T$. In this interval the value of $v(k)$ drops exponentially while $T$ increases. This is because in SMALL $o$ and $I$ are small, so we need longer checkpoint interval for performing useful computation and thus achieving better overhead ratio. However, since with large values of $T$ we can lose computation events due to rollback, in the interval $(500, \infty)$ the overhead ratio becomes much worse as long as $T$ becomes longer (particularly in Figure 5(a) due to equal probabilities). Therefore, there is a value of $T$, where the lose of overhead ratio due to the checkpoint overhead and latency on one hand and the lose due to rollback propagation on the other hand are similar. This value of $T$ yields the minimal overhead ratio.

Figure 6(b) shows the tradeoff between the optimum value of $T$ and $\lambda$. With no doubt, the overhead ratio increases as the failure rate increases too. Thus, to limit the losing of useful computation with large values of $\lambda$, we need to set smaller values of $T$.

Figure 7(a) considers the effect of $F$ and $n$ on $v(k)$. It shows that the optimum value of $T$ that is obtained in different points in respect with $F$. We have that the overhead ratio with $F = 2$ and $T = 3000$ is equivalent to the overhead ratio with $F = 0$ and $T = 1000$. In general, we have that the overhead ratio with $F = f$ and $T = t$ is equivalent to the overhead ratio with $F = 0$ and $T = \frac{t}{1-F}$. Therefore, there is no reason to present the overhead ratio with $F > 2$. On the other hand, Figure 7(b) shows that the overhead ratio increases proportionally with number of processes. Under the assumption that process failures occur randomly and independently with probability of $p$, the probability of a failure in a system with $n$ processes is $1 - (1-p)^n$. Therefore, the overhead ratio increases proportionally with $n$. 
ZPF

Wang showed that FDI is Z-path free (ZPF) [19]. By [1], connected network with therefore, where it should distinguish between different runs of C-L. messages per checkpoint [13] and the marker since is 8-bit, therefore, FDI \(= c \cdot MR(E)\) for 0 \(\leq c \leq 1\). In the measurements below \(c = 0.1\).

**Briatico, Ciuffoletti and Simoccini** (BCS) [6] is a communication induced checkpointing protocol. Manivannan and Singhal [10] proved that BCS belongs to ZCF, and from [1] we know that ZCF \(\subseteq n\)-rollback. It can be claimed that the number of forced checkpoints depends on the communication pattern of an execution \(E\). Alvisi et al [4] showed that the expected number of forced checkpoints in BCS is 2 for \(T = 360\) and \(T = 480\). Lastly, since BCS piggybacks a 32-bit logical clock (assuming a 32-bit architecture) on every application message, then \(M = MR(E)(32 \cdot w_b + \epsilon)\), where \(\epsilon\) is the delay for intercepting every data message. E.g., in our measurements below, \(\epsilon = 50\) microseconds.

**Baldivi, Quaglia and Ciciani** (BQC) [5] is another communication induced checkpointing protocol ensuring ZCF by preventing potential Z-cycles from being created. By [1], BQC \(\in n\)-rollback. Moreover, Alvisi et al [4] showed that BQC is worse than BCS but \(F(BQC) = 2\). Lastly, the protocol propagates \(n^2\) 32-bit values on each application message to help processes detect suspected Z-cycles. Therefore, we have that \(M(BQC) = MR(E)(32 \cdot n^2 \cdot w_b + \epsilon)\), where \(\epsilon\) is the delay for intercepting every data message.

**d-Bounded Cycles (d-BC)** is a communication induced checkpointing protocol that allows bounded cycles to be formed [1]. By [1], d-BC belongs to \((n - 1)d\)-rollback. Upon a new checkpoint \(C_{p,i}\), process \(p\) broadcasts a cut of size no more that \(d \cdot n\), therefore, \(M(d-BC) = n \cdot w_m + d \cdot n^2 \cdot w_b\). Moreover, d-BC forces checkpoints by calling C-L only if a cycle of size \(d\) is generated. Since a Z-cycle is a special case of a cycle, then the conditions of generating cycles and Z-cycles are almost equivalent. Also since ZCF \(= 1\)-BC, then by [4] we have that \(F(1\text{-}BC) = 2\).

**4. Performance Analysis of Distributed Checkpointing Protocols**

**Sync-and-Stop (SaS)** is a coordinated checkpointing protocol [13]. It was shown in [1] that SaS \(\in 1\)-rollback. In this protocol there are no forced checkpoints, therefore, \(F(SaS) = 0\). Regarding the control overhead, in each phase of SaS, the coordinator broadcasts three messages and the other \(n - 1\) processes send two reply messages. Notice that the protocol needs an 8-bit control messages. Therefore, \(M(SaS) = 5(n - 1)(w_m + 8 \cdot w_b)\).

**Chandy-Lamport (C-L)** [7] is a coordinated checkpointing protocol in which there is no need to block the application execution. C-L belongs to 1-rollback and since there are no forced checkpoints, \(F(C-L) = 0\). In a fully connected network with \(n\) nodes, C-L generates \(2n(n - 1)\) messages per checkpoint [13] and the marker since is 8-bit, where it should distinguish between different runs of C-L. Therefore, \(M(C-L) = 2n(n - 1)(w_m + 8 \cdot w_b)\).

**Fixed-Dependency-Interval (FDI)** was suggested in [19]. Wang showed that FDI is Z-path free (ZPF) [19]. By [1], ZPF \(\subseteq 1\)-rollback. Therefore, FDI \(\in 1\)-rollback. Also, the dependency vector is piggybacked on each message. Thus, \(M(FDI) = n \cdot MR(E)\) for an execution \(E\). However, the number of forced checkpoints clearly depends on the communication pattern. To guarantee ZPF, this protocol takes many forced checkpoints depending on the communication pattern. We assumed that \(F(FDI) = c \cdot MR(E)\) for 0 \(\leq c \leq 1\). In the measurements below \(c = 0.1\).

![Figure 7. \(v(k)\) is affected by other parameters](image)

**Figure 7. \(v(k)\) is affected by other parameters**

**4.1. Comparing Checkpointing Protocols**

In this section we compare the overhead ratio of the checkpointing protocols presented in the previous section. First, we compare the coordinated checkpointing protocols: C-L and SaS. Figure 8 depicts the overhead ratio of these protocols in the SMALL environment. The full version of the paper also presents the results for the BIG environment, yet qualitatively they are very similar. We used the SMALL environment with \(T = 1024\), which is the optimum value of \(T\). We can see in Figure 8(a) that C-L is better if the number of processes is less than 100 and it becomes worse for large number of processes. This is because C-L incurs more control overhead than SaS, where \(M(C-L) = O(n^2)\) and \(M(SaS) = O(n)\). Notice here that the checkpoint overhead in SaS is bigger than in C-L. This is because SaS pauses processes for checkpointing.

In Figure 9, we compare additional checkpointing pro-
protocols. We computed the overhead ratio in the SMALL environment with $T = 1024$. Consider the $n$-rollback protocols: BQC, BCS and 1-BC. The 1-BC protocol is an optimistic protocol which takes fewer forced checkpoints than BQC and BCS. Moreover, 1-BC does not use piggybacking as the other protocols do, therefore, its overhead ratio is not affected by the communication pattern. As depicted in Figure 9(a), in executions with small MR, 1-BC has worse overhead than BCS and BQC. On the other hand, when MR is large, 1-BC performs better than both BQC and BCS. Figure 9(b) shows that with $MR = 2048$, BQC an BCS are worse than in Figure 9(a). Notice that, since BQC piggybacks $O(n^2)$ control information while BCS piggybacks only an 8-bit logical clock, then $v(n, BCS) \leq v(n, BQC)$.

Figure 10 compares the same protocols as Figure 9, with the same MR and $T$ values, but with decreasing probabilities of recovering from the possible checkpoints. The overhead ratio in Figure 10 is smaller than in Figure 9. This is because when the probability of long rollback is small, then the probability of losing useful computation is small too.

Clearly, we can see in Figure 10 that the overhead ratio of executions with protocols that use piggybacking are worse than the others with large MR. These protocols are FDI, BCS, and BQC. Notice that in Figure 10(b), 1-BC has better overhead ratio than the other $n$-rollback protocols. Lastly, we can conclude here that the C-L protocol is the best up to 32 processes. Recall that SaS becomes better than C-L for large number of processes as depicted in Figure 8.

5. Related Work

There has been much work on checkpointing performance analysis [11, 14, 18, 20]. Most of these works do not take into account the rollback propagation. Ours is the first to incorporate all parameters that affect the performance in distributed environments into an analytical measure.

Mishra and Wang [11] evaluated several checkpointing protocols by implementing and running them with test applications. Ziv and Bruck [20] compared four checkpointing protocols by using the Markov Reward Model [15]. Our approach differs from [20] in that we provide a technique for comparing any checkpointing protocol based on rollback propagation. Ziv and Bruck presented in [21] a checkpoint scheme for duplex systems, and conducted a performance analysis for their scheme in the duplex system. However, it is not a general system for distributed executions.

Vaidya defined the overhead ratio for uniprocessor systems as a function of the checkpoint overhead and latency [17], and proved that the optimum checkpoint interval depends on $o$. Additionally, he claimed that the overhead ratio can be computed for distributed systems as in uniprocessor systems by taking the values of parameters either to be the maximum or the average over all processes. In [16], Vaidya computed the overhead ratio for the two-level recovery approach. This approach tolerates single failures with a low overhead and multiple failures with a higher overhead.

Plank and Thomason [14] presented a method for estimating the overhead ratio for coordinated checkpointing. By assuming coordinated checkpointing, they do not care about rollback propagation. Moreover, they do not address the control overhead incurred by control information.

References

Comparing in SMALL − MR = 32

overhead ratio

FDI

C−L

S&S

BCS BQC

1−BC

(a)

Comparing in SMALL − MR = 1024

overhead ratio

BCS 1−BC

SaS

C−L

BQC

FDI

(b)

Figure 10. With decreasing probabilities


