An Index-Based Checkpointing Algorithm for Autonomous Distributed Systems [*]

Roberto Baldoni   Francesco Quaglia   Paolo Fornara

Technical Report n. 07-97
March 1997

Dipartimento di Informatica e Sistemistica, Universitá "La Sapienza",
Via Salaria 113, 00198 Roma, Italy.
E.mail: {baldoni,quaglia,fornara}@dis.uniroma1.it.

Abstract

This paper presents an index-based checkpointing algorithm for distributed systems which minimizes the total number of checkpoints while ensuring that each checkpoint belongs to at least one *consistent global checkpoint* (or *recovery line*). The algorithm is based on an *equivalence relation* defined between pairs of successive checkpoints of a process which allows, in some cases, to advance the recovery line of the computation without forcing checkpoints in other processes. This protocol shows good performance expecially in *autonomous* environments where each process does not have any private information about other processes.

1 Introduction

Checkpointing is one of the techniques for providing fault-tolerance in distributed systems [6]. A *global checkpoint* consists of a set of local checkpoints, one for each process, from which a distributed computation can be restarted after a failure. A *local checkpoint* is a state of a process saved on stable storage. Resuming an application from a global checkpoint is correct if there not exist a message m whose receipt is recorded in the global checkpoint but not its transmission. In that case the global checkpoint is said to be *consistent* [4, 10, 13, 17].

Three classes of algorithms have been proposed in the literature to determine consistent global checkpoints: *uncoordinated*, *coordinated* and *communication induced* [6]. In the first class, processes take local checkpoints independently by each other and upon the occurrence of a failure, a procedure of rollback-recovery tries to build a consistent global checkpoint. Note that, a consistent global

[*]This work is partially supported by Scientific Cooperation Network of the European Community "OLOS" under contract No. ERB4050PL932483.

checkpoint might not exist producing a *domino effect* [1, 14] which, in the worst case, forces the computation to its initial state.

In the second class, an initiator process forces other processes, during a failure-free computation, to take a local checkpoint by using control messages. The coordination can be either blocking [4] or non-blocking [8]. However, in both cases, the last local checkpoint of each process belongs to a consistent global checkpoint built on-the-fly.

In the third class, the coordination to determine consistent global checkpoints on-the-fly is done in a lazy fashion by piggybacking control information on application messages. Communication-induced checkpointing algorithms can be classified in two distinct categories: *model-based* and *index-based* [6]. Algorithms in the first category [2, 15, 17] have the target to mimic a piece-wise deterministic behavior for each process [7, 16] as well as providing the domino-free property. Index-based algorithms associate each local checkpoint with a sequence number and try to enforce consistency among local checkpoints with the same sequence number [3, 5, 11, 18]. Index based algorithms usually ensure domino-free rollback with, generally, less overhead, in terms of number of checkpoints and control information, than model based ones.

In this paper we present an index-based checkpointing protocol that reduces the global checkpointing overhead in terms of total checkpoints compared to previous index-based algorithms while ensuring that each checkpoint belongs to at least one consistent global checkpoint[1] (i.e., each local checkpoint is useful [13]). Our protocol is well suited for *autonomous* environments where each process does not have any private information of other processes[2].

To design our algorithm, we derive the rules, used by index-based algorithms, to update sequence numbers. This points out that the cause of induced checkpoints is due to the increasing of the value of the sequence number in each process. Hence, we derive an algorithm that using an equivalence relation between pair of successive checkpoints of a process allows a recovery line to advance without always increasing its sequence number. In the worst case, our algorithm takes the same number of checkpoints as in [11]. The advantages of our algorithm are quantified by an extensive simulation study showing that the overall checkpointing overhead can be reduced up to 30% compared to the best previous solution. The price we pay is that each application message piggybacks one vector of integer more than previous proposals whose size of the control information is one integer.

The paper is organized as follows. Section 2 presents the system model. Section 3 shows the class of index–based checkpointing algorithms in the particular context of autonomous environments. In Section 4 we describe the equivalence relation, the algorithm and provide its correctness proof. In Section 5 a simulation study is presented.

[1]In the following, we use the term consistent global checkpoint and recovery line interchangeably.

[2]The index-based algorithm presented in [5] assumes, for example, the existence of a global common clock.

2 Model of the Distributed Computation

We consider a distributed computation consisting of n processes $(P_1, P_2, \ldots P_n)$ which interact by messages exchanging. Each pair of processes is connected by a two-ways reliable channel whose transmission delay is unpredictable but finite.

Processes are *autonomous* in the sense that: they do not share memory, do not share a common clock value and do not have access to private information of other processes such as clock drift, clock granularity, clock precision and speed. Process failure is not considered in this paper.

A process produces a sequence of events; each event moves the process from one state to another. A local checkpoint C is an event which dumps the current process state onto stable storage. We assume events are produced by the execution of internal, send or receive statements as well as local checkpoint operations. The send and receive events of a message m are denoted respectively by $send(m)$ and $receive(m)$. A *distributed execution* $\hat{E}$ can be modeled as a partial ordering of events $\hat{E} = (E, \rightarrow)$ where E is the set of all events and $\rightarrow$ is the *happened-before relation* [9].

A checkpoint in process P_i is denoted as $C_{i,sn}$ where sn is called the *index*, or *sequence number*, of a checkpoint. Each process takes local checkpoints either at its own pace (*basic* checkpoints) or induced by some communication pattern (*forced* checkpoints). We assume that each process P_i takes an initial basic checkpoint $C_{i,0}$ and that, for the sake of simplicity, basic checkpoints are taken by a periodic algorithm. We use the notation $next(C_{i,sn})$ to indicate the successive checkpoint, taken by P_i, after $C_{i,sn}$. A checkpoint interval $I_{i,sn}$ is the set of events between $C_{i,sn}$ and $next(C_{i,sn})$.

A message m sent by P_i to P_j is called *orphan* with respect to a pair (C_{i,sn_i}, C_{j,sn_j}) iff its receive event occurred before C_{j,sn_j} while its send event occurred after C_{i,sn_i}. A *global checkpoint* $\mathcal{C}$ is a set of local checkpoints $(C_{1,sn_1}, C_{2,sn_2}, \ldots, C_{n,sn_n})$ one for each process. A global checkpoint $\mathcal{C}$ is *consistent* if no orphan message exists in any pair of local checkpoints belonging to $\mathcal{C}$. In the following, we denote with $\mathcal{C}_{sn}$ a global checkpoint formed by checkpoints with sequence number sn.

3 Index-based Checkpointing Algorithms

The simplest way to form a consistent global checkpoint is, each time a basic checkpoint $C_{i,sn}$ is taken by process P_i, to start an explicit coordination. This coordination results in a consistent global checkpoint $\mathcal{C}_{sn}$ associated to that local checkpoint and in the fact that no local checkpoint is useless (i.e., each local checkpoint belongs to at least one consistent global checkpoints [13, 19]). This strategy induces $n-1$ forced checkpoints, one for each process, per basic checkpoint.

Briatico at al. [3] argued that the previous "centralized" strategy can be "decentralized" in a lazy fashion by piggybacking on each application message m the index sn of the last checkpoint taken (denoted $m.sn$).

Let us assume each process P_i endows a variable sn_i which represents the sequence number of the last checkpoint. Then, the Briatico-Ciuffoletti-Simoncini (BCM) algorithm can be sketched by using the following rules associated with the action to take a local checkpoint:

`take-basic(BCS)` : When a basic checkpoint is scheduled, sn_i is increased by one and a checkpoint $C_{i,sn}$ is taken with $sn = sn_i$;

`take-forced(BCS)` : Upon the receipt of a message m, if $sn_i < m.sn$, a checkpoint $C_{i,m.sn}$ is taken and sn_i is set to $m.sn$, then the message is processed.

By using the above rules, it has been proved that $\mathcal{C}_{sn}$ is consistent [3], and that BCS does not produce useless checkpoints [12]. Note that, due to the rule `take-forced(BCS)`, there could be some gap in the index assigned to checkpoints by a process. Hence, if a process has not assigned the index sn, the first local checkpoint of the process with sequence number greater than sn can be included in the consistent global checkpoint $\mathcal{C}_{sn}$ (or $\mathcal{L}_{sn}$ in terms of recovery line).

In the worst case of BCS algorithm, the number of forced checkpoints induced by a basic one is $n - 1$. In the best case, if all processes would take a basic checkpoint at the same physical time, the number of forced checkpoints per basic one would be zero. However, in an *autonomous environment*, local indices of processes may diverge due to many causes (process speed, different period of the basic checkpoint etc). This pushes the indices of some processes higher and each time one of such processes sends a message to another one, it is extremely likely that a number of forced checkpoint, close to $n - 1$, will be induced.

Manivannan and Singhal present an algorithm [11] which tries to keep the indices of all the processes close to each other and to push the recovery line as close as possible to the end of the computation. In this way, it reduces the probability that $sn_i < m.sn$ that, in turn, decreases the number of the local checkpoint forced by a basic one. The Manivannan-Singhal (MS) algorithm is based on the following two observations:

1. Let each process be endowed with a counter incremented each x time unit (where x is the smallest period between two basic checkpoints overall processes). If each tx (with $t \geq 1$) time units a process takes a basic checkpoint $C_{i,tx}$, $\mathcal{L}_{tx}$ is the closest recovery line to the end of the computation.

2. If the last checkpoint taken by a process is a forced one and its sequence number is greater than or equal to the next scheduled basic checkpoint, then the basic checkpoint is skipped.

Point 1 would allow a heavy reduction of forced checkpoints, by synchronizing actually the action to take basic checkpoints in distinct processes, if *it did not violate autonomy of processes* (no private information about other processes is available). Point 2 reduces the number of basic checkpoints. In particular, it has been shown that, the number of total checkpoints of a computation (i.e., forced plus basic checkpoints) cuts down compared to BCS algorithm [11]. So, let us assume process P_i endows a flag $skip_i$ which indicates if at least one forced checkpoint is taken in the current checkpoint period (this flag is set to FALSE each time a basic checkpoint is scheduled, and set to TRUE each time a forced checkpoint is taken). A version of Manivannan-Singhal algorithm well suited for autonomous environment can be sketched by the following rules:

```
take-basic(MS) : When a basic checkpoint is scheduled
    if $skip_i$ then $skip_i = FALSE$
    else $sn_i$ is increased by one and a checkpoint $C_{i,sn}$ is taken with $sn = sn_i$;

take-forced(MS) : Upon the receipt of a message $m$
    if $sn_i < m.sn$ then a checkpoint $C_{i,m.sn}$ is taken, $sn_i$ is set to $m.sn$ and $skip_i = TRUE$;
    the message is processed.
```

Hence, in an autonomous environment, even though MS algorithm produces a reduction of the total number of checkpoints, the number of forced checkpoints caused by a basic one is equal to BCS as `take-forced(MS)` is actually similar to `take-forced(BCS)`.

4 An Index-Based Algorithm

In this section we propose an algorithm that includes the `take-basic(MS)` rule, however, when a basic checkpoint is taken, the local sequence number is increased only if there was the occurrence of a particular checkpoint and communication pattern. The rationale behind this solution is that each time a basic checkpoint is taken without increasing the sequence number, it does not force any checkpoint and this will reduce the total number of checkpoints. At this end, we first introduce a relation of equivalence on pairs of successive local checkpoints and then a checkpoint index which includes an equivalence number as well as a sequence one.

4.1 The Relation of Equivalence Between Checkpoints

Definition 4.1 *Two local checkpoints $C_{i,sn}$ and $next(C_{i,sn})$ of process P_i are equivalent with respect to the recovery line $\mathcal{L}_{sn}$, denoted $C_{i,sn} \overset{\mathcal{L}_{sn}}{\equiv} next(C_{i,sn})$, if*

$$(\forall C_{j,sn} \in \mathcal{L}_{sn}) \quad \nexists receive(m) \in I_{i,sn} : \ C_{j,sn} \rightarrow send(m)$$

As an example, let consider the recovery line $\mathcal{L}_{sn}$ depicted in Figure 1.a. If in $I_{2,sn}$ process P_2 performs either send events or receive events of messages which have been sent before the checkpoints included in the recovery line $\mathcal{L}_{sn}$, then $C_{2,sn} \overset{\mathcal{L}_{sn}}{\equiv} next(C_{2,sn})$ and a recovery line $\mathcal{L'}_{sn}$ is created by replacing $C_{2,sn}$ with $next(C_{2,sn})$ from $\mathcal{L}_{sn}$. Figure 1.b shows the construction of the recovery line $\mathcal{L''}_{sn}$ starting from $\mathcal{L'}_{sn}$ by using the equivalence between $next(C_{1,sn})$ and $C_{1,sn}$ with respect to $\mathcal{L'}_{sn}$.

As shown in the above examples, the equivalence relation has a simple property (see Lemmas 4.1 and 4.2 of Section 4.4): if $C_{i,sn} \overset{\mathcal{L}_{sn}}{\equiv} next(C_{i,sn})$, the set $\mathcal{L}'_{sn} = \mathcal{L}_{sn}$-$\{C_{i,sn}\} \cup \{next(C_{i,sn})\}$ is a recovery line. Hence, the presence of a pair of equivalent checkpoints allows a process to locally advance a recovery line without increasing the sequence number.

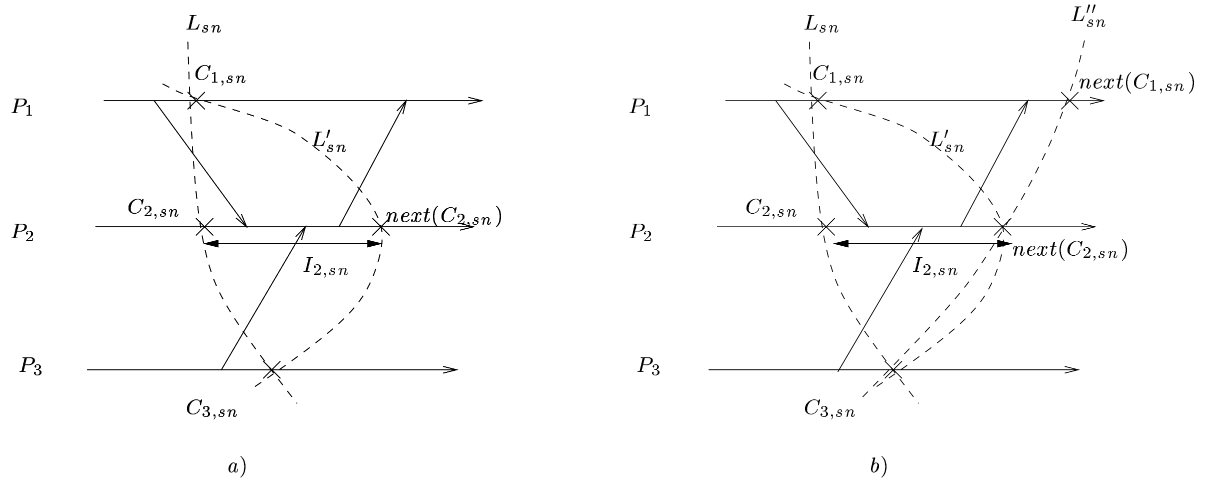

Figure 1: $C_{2,sn}$ and $next(C_{2,sn})$ are equivalent with respect to the recovery line $\mathcal{L}_{sn}$ (a); $C_{1,sn}$ and $next(C_{1,sn})$ are equivalent with respect to the recovery line $\mathcal{L}_{sn'}$ (b).

4.2 Sequence and Equivalence Numbers of a Recovery line

We suppose process P_i owns two local variables: sn_i (sequence number) and en_i (equivalence number). The variable sn_i stores the number of the current recovery line. The variable en_i represents the number of equivalent local checkpoints with respect to the current recovery line (both sn_i and en_i are initialized to zero).

Hence, we denote as $C_{i,sn,en}$ the checkpoint of P_i with the sequence number sn and the equivalence number en; the pair $< sn, en >$ is also called the index of a checkpoint. Thus, the initial checkpoint of process P_i will be denoted as $C_{i,0,0}$. The index of a checkpoint is updated according to the following rule:

$$\textbf{if } C_{i,sn,en} \overset{\mathcal{L}_{sn}}{\equiv} next(C_{i,sn,en}) \textbf{ then } next(C_{i,sn,en}) = C_{i,sn,en+1} \textbf{ else } next(C_{i,sn,en}) = C_{i,sn+1,0};$$

Process P_i also endows a vector EQ_i of n integers initialized to zero each time $next(C_{i,sn,en}) = C_{i,sn+1,0}$. The j–th entry of the vector represents the knowledge of P_i about the equivalence number of P_j associated to the current sequence number sn_i (thus the i–th entry corresponds to en_i).

EQ_i is updated according to the following rule: each application message m sent by process P_i piggybacks the current sequence number sn_i ($m.sn$) and the current EQ_i vector ($m.EQ$). Upon the receipt of a message m, if $m.sn = sn_i$, EQ_i is updated from $m.EQ$ by taking a component-wise maximum. If $m.sn > sn_i$, the values in $m.EQ$ and $m.sn$ are copied in EQ_i and sn_i.

Let us remark that the set $\mathcal{L}_{sn} = \cup_{\forall j} C_{j,sn,EQ_i[j]}$ is a recovery line (a formal proof of this property is given in Lemma 4.4). So, to the knowledge of P_i, the vector EQ_i actually represents the most recent recovery line with sequence number sn_i.

4.3 Tracking the Equivalence Relation On-The-Fly

Upon the arrival of a message m sent by P_j at P_i in the checkpoint interval $I_{i,sn,en}$, one of the following three cases is true:

1) $(m.sn < sn_i)$ or $((m.sn = sn_i)$ and $(m.EQ[j] < EQ_i[j]))$;
% m has been sent from the left side of the recovery line $\cup_{\forall j} C_{j,sn,EQ_i[j]}$ %
2) $(m.sn = sn_i)$ and $(m.EQ[j] \geq EQ_i[j])$;
% m has been sent from the right side of the recovery line $\cup_{\forall j} C_{j,sn,EQ_i[j]}$ %
3) $(m.sn > sn_i)$;
% m has been sent from the right side of a recovery line of which P_i was not aware %

According to previous cases, at the time of the insertion of the checkpoint $next(C_{i,sn,en})$, P_i falls in one of the following three alternatives:

- If no message m is received in $I_{i,sn,en}$ that falls in case 2 or 3, then $C_{i,sn,en} \overset{\mathcal{L}_{sn}}{\equiv} next(C_{i,sn,en})$. This equivalence can be tracked by a process using its local context at the time of taking $next(C_{i,sn,en})$. Thus $next(C_{i,sn,en}) = C_{i,sn,en+1}$ (the equivalence between $C_{2,sn,0} \overset{\mathcal{L}_{sn}}{\equiv} next(C_{2,sn,0})$, shown in Figure 2, is an example of such behavior).

- If there exists a message m which falls in case 3 then $C_{i,sn,en}$ is not equivalent to $next(C_{i,sn,en})$ and thus $next(C_{i,sn,en}) = C_{i,sn+1,0}$.

- If no message falls in case 3 and there exists at least a message m received in $I_{i,sn,en}$ which falls in case 2, then the checkpoint $next(C_{i,sn,en})$ is causally related to one checkpoint belonging to the recovery line formed by $\cup_{\forall j} C_{j,sn,EQ_i[j]}$ (this communication pattern is shown in Figure 2 where, due to m, $C_{2,sn,0} \rightarrow next(C_{1,sn,0})$). The consequence is that process P_i cannot determine at the time of taking the checkpoint $next(C_{i,sn,en})$ if $C_{i,sn,en} \overset{\mathcal{L}_{sn}}{\equiv} next(C_{i,sn,en})$.

  Hence, we assume *optimistically* (and *provisionally*) that $C_{i,sn,en} \overset{\mathcal{L}_{sn}}{\equiv} next(C_{i,sn,en})$. If at the time of the first send event after $next(C_{i,sn,en})$ the equivalence is still undetermined, as provisional indices cannot be propagated in the system, then $next(C_{i,sn,en}) = C_{i,sn+1,0}$ (thus, $sn_i = sn_i + 1$, $en_i = 0$, and $\forall j : EQ_i = 0$). Otherwise, the provisional index becomes permanent. Figure 2 shows a case in which message m' brings the information (encoded in $m'.EQ$) to P_1 (before the sending of m'') that $C_{2,sn,0} \overset{\mathcal{L}_{sn}}{\equiv} next(C_{2,sn,0})$ and the recovery line is advanced, by P_2, from $\mathcal{L}_{sn}$ to $\mathcal{L}'_{sn}$. In such a case, process P_1 can determine $C_{1,sn,0}$ is equivalent to $next(C_{1,sn,0})$ with respect to the recovery line $\mathcal{L}'_{sn}$ and, then, advances the recovery line to $\mathcal{L}''_{sn}$

4.4 The Algorithm

The checkpointing algorithm we propose (hereafter BQF) takes basic checkpoints by using the `take-basic(MS)` rule, but it does not increase the sequence number by optimistically assuming that a basic checkpoint is equivalent to the last one. So we have:

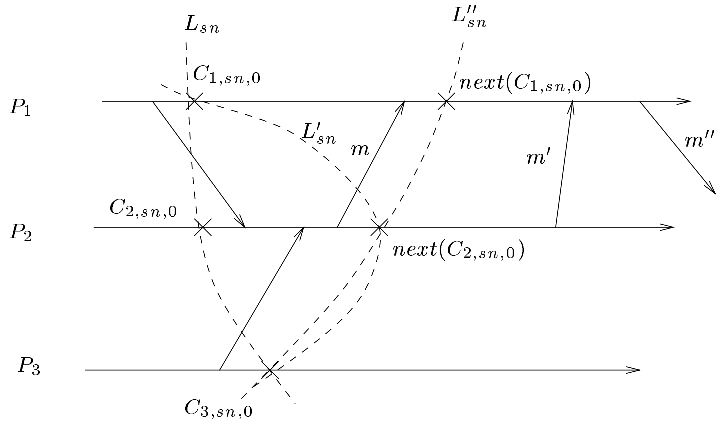

Figure 2: Upon the receipt of m', P_1 detects $C_{1,sn,0} \overset{\mathcal{L}'_{sn}}{\equiv} next(C_{1,sn,0})$ due to $C_{2,sn,0} \overset{\mathcal{L}_{sn}}{\equiv} next(C_{2,sn,0})$ which is encoded in $m'.EQ$.

```
take-basic(BQF): When a basic checkpoint is scheduled:
  If $skip_i$ then $skip_i = FALSE$
  else $en_i = en_i + 1$, a checkpoint $C_{i,sn,en}$ is taken with index provisionally set to $< sn_i, en_i >$;
```

Due to the presence of provisional indices caused by the equivalence relation, our algorithm needs a rule, when sending a message, in order to disseminate only permanent indices of checkpoints.

```
send-message(BQF): before sending a message $m$ in $I_{i,sn,en}$:
  if there has been no send event in $I_{i,sn,en}$ and the index is provisional then
    if $C_{i,sn,en-1} \overset{\mathcal{L}_{sn}}{\equiv} C_{i,sn,en}$ then the index $< sn, en >$ becomes permanent
    else $sn_i = sn_i + 1$, $en_i = 0$ and the index of $C_{i,sn,en}$ is replaced permanently with $< sn_i, en_i >$;
  the message $m$ is sent;
```

The last rule of our algorithm `take-forced(BQF)` refines BCS's one by using a simple observation. There is no reason to take a forced checkpoint if there has been no send event in the current checkpoint interval $I_{i,sn,en}$ till the receipt of message m piggybacking the sequence number $m.sn > sn_i$ (point (b) of `take-forced(BQF)`). Indeed, no causal relation can be established between the last checkpoint $C_{i,sn,en}$ and any checkpoint belonging to the recovery line $\mathcal{L}_{m.sn}$ and, thus, the index of $C_{i,sn,en}$ can be replaced permanently with the index $< m.sn, 0 >$.

```
take-forced(BQF): Upon the receipt of a message $m$ in the checkpoint interval $I_{i,sn,en}$:
  (a) If $sn_i < m.sn$ and there has been at least a send event in $I_{i,sn,en}$ then
      begin
        a forced checkpoint $C_{i,m.sn,0}$ is taken and its index is permanent;
        $sn_i = m.sn$; $en_i = 0$; $skip_i = TRUE$
      end;
      the message $m$ is processed;
```

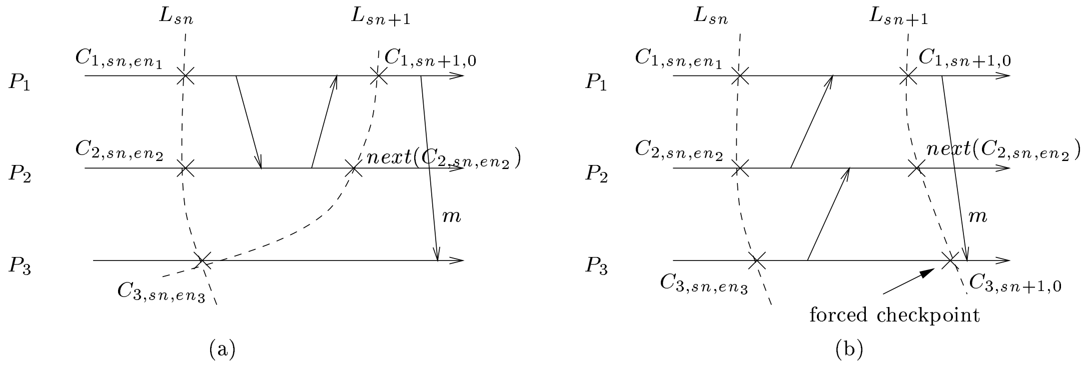

Figure 3: Upon the receipt of m, C_{3,sn,en_3} can be a part of the recovery line $\mathcal{L}_{sn+1}$ (a); C_{3,sn,en_3} cannot belong to the recovery line $\mathcal{L}_{sn+1}$, then the forced checkpoint $C_{3,sn+1,0}$ is taken (b).

```
(b) If $sn_i < m.sn$ and there has been no send event in $I_{i,sn,en}$ then
    begin
        the index of the last checkpoint $C_{i,sn,en}$ is replaced permanently with $< m.sn, 0 >$;
        $sn_i = m.sn$; $en_i = 0$
    end;
    the message $m$ is processed;
```

For example, in Figure 3.a, the local checkpoint C_{3,sn,en_3} can belong to the recovery line $\mathcal{L}_{sn+1}$ (so the index $< sn, en_3 >$ can be replaced with $< sn+1, 0 >$). On the other hand, if a send event has occurred in the checkpoint interval I_{3,sn,en_3} shown in Figure 3.b, a forced checkpoint with index $< sn+1, 0 >$, that will belong to the recovery line $\mathcal{L}_{sn+1}$, has to be taken before the processing of message m.

Point (b) of `take-forced(BQF)` decreases the number of forced checkpoints taken compared to `take-forced(BCS)`. The rule `take-basic(BQF)` does not increase the sequence number while taking a basic checkpoint, so no forced checkpoint will be induced in other processes. The **else** alternative of `send-message(BQF)` and `take-forced(BQF)` part (a), represent the cases in which the action to take a basic checkpoint leads to permanently increase the sequence number.

Data Structures and Process Behavior

We assume each process P_i has the following data structures:
sn_i, en_i: **integer**;
$after\_first\_send_i$, $skip_i$, $provisional_i$: **boolean**;
$past_i$, $present_i$, EQ_i: ARRAY[1,n] of **integer**.
The boolean variable $after\_first\_send_i$ is set to TRUE if at least one send event has occurred in the current checkpoint interval. It is set to FALSE each time a local checkpoint is taken.

The boolean variable $provisional_i$ is set to TRUE when a provisional index assignment is executed. It is set to FALSE each time the index becomes permanent.

$present_i[j]$ represents the maximum equivalence number en_j piggybacked on a message m received in the current checkpoint interval by P_i from P_j and that falls in the case 2 shown in Section 4.3. Upon taking a checkpoint or when increasing the sequence number, $present_i$ is initialized to -1. If the checkpoint is basic, $present_i$ is copied in $past_i$ before its initialization. $past_i[j]$ is set to -1 each time a message m is received such that $past_i[h] < m.EQ[h]$. So, the predicate $(\exists h : past_i[h] > -1)$ indicates that there is a message received in the past checkpoint interval that has been sent from the right side of the recovery line (case 2 of Section 4.3) *currently* seen by P_i.

Below the process behavior is shown (the procedures and the message handler are executed in atomic fashion). This implementation assumes that there exist at most one provisional index in each process. So each time two successive provisional index are detected, the first index is permanently replaced with $< sn_i + 1, 0 >$.

```
**init** $P_i$:
$sn_i := 0$; $en_i := 0$; $after\_first\_send_i := FALSE$; $skip_i := FALSE$; $provisional_i := FALSE$;
$\forall h\ EQ_i[h] := 0$; $\forall h\ past_i[h] := -1$; $\forall h\ present_i[h] := -1$;

**when** $(m)$ **arrives at** $P_i$ from $P_j$:
if $m.sn > sn_i$ then                       % $P_i$ is not aware of the recovery line with sequence number $m.sn$ %
begin
  if $after\_first\_send_i$ then
  begin
    take a checkpoint C;                                        % taking a forced checkpoint %
    $skip_i := TRUE$;
    $after\_first\_send_i := FALSE$;
  end;
  $sn_i := m.sn$; $en_i := 0$;
  assign the index $< sn_i, en_i >$ to the last checkpoint C;
  $provisional_i := FALSE$;                                        % the index is permanent %
  $\forall h\ past_i[h] := -1$; $\forall h\ present_i[h] := -1$;
  $present_i[j] := m.EQ[j]$;
  $\forall h\ EQ_i[h] := m.EQ[h]$;
end
else if $m.sn = sn_i$ then
     begin
       if $present_i[j] < m.EQ[j]$ then $present_i[j] := m.EQ[j]$;
       $\forall h\ EQ_i[h] := max(EQ_i[h],\ m.EQ[h])$;            % a component-wise maximum is performed %
       $\forall h$ if $past_i[h] < m.EQ[h]$ then $past_i[h] := -1$;
     end;
process the message m;

**when** $P_i$ sends $data$ to $P_j$:
if $\neg after\_first\_send_i \wedge provisional_i \wedge (\exists h : past_i[h] > -1)$ then % last checkpoint not equivalent to the previous one %
begin
  $sn_i := sn_i + 1$; $en_i := 0$;
  assign the index $< sn_i, en_i >$ to the last checkpoint C;
  $provisional_i := FALSE$;                                        % the index is permanent %
  $\forall h\ past_i[h] := -1$; $\forall h\ present_i[h] := -1$; $\forall h\ EQ_i[h] := 0$;
end;
$m.content = data$; $m.sn := sn_i$; $m.EQ := EQ_i$;                    % packet the message %
send $(m)$ to $P_j$;
$after\_first\_send_i := TRUE$;
```

```
when a basic checkpoint is scheduled from $P_i$:
if $skip_i$ then $skip_i := FALSE$                              % the basic checkpoint is skipped as in [11]%
else
begin
  if $provisional_i$ then                                          % two successive provisional indices %
    if $(\exists h : past_i[h] > -1)$ then                   % last checkpoint not equivalent to the previous one %
    begin
      $\forall h \ past_i[h] := -1$;
      $sn_i := sn_i + 1$; $en_i := 0$;
      assign the index $< sn_i, en_i >$ to the last checkpoint C;        % the index is permanent %
      $\forall h \ EQ_i[h] := 0$;
    end
    else $\forall h \ past_i[h] := present_i[h]$; % last checkpoint is equivalent to the previous one %
  take a checkpoint C;                                                % taking a basic checkpoint %
  $en_i := en_i + 1$;
  $EQ_i[i] := en_i$;
  assign the index $< sn_i, en_i >$ to the last checkpoint C;
  $provisional_i := TRUE$;                                            % the index is provisional %
  $\forall h \ present_i[h] := -1$;
  $after\_first\_send_i := FALSE$;
end
```

4.5 Correctness Proof

Let us first introduce the following simple observations that derive directly from the algorithm:

Observation 1 For any checkpoint $C_{i,sn,0}$, there not exists a message m with $m.sn \geq sn$ such that $receive(m) \rightarrow C_{i,sn,0}$ (this observation derives from rule `take-forced(BQF)` when considering $C_{i,sn,0}$ is the first checkpoint with sequence number sn).

Observation 2 For any checkpoint $C_{i,sn,en}$, there not exists a message m with $m.sn > sn$ such that m is received in $I_{i,sn,en}$ (this observation derives from rule `take-forced(BQF)`).

Observation 3 For any message m sent by P_i: if $C_{i,sn,en} \rightarrow send(m)$ then $m.sn \geq sn$ (this observation derives from the rule `send-message(BQF)`).

Lemma 4.1 *The set of checkpoints $S = (C_{1,sn,0}, C_{2,sn,0}, \ldots, C_{n,sn,0})$ with $sn \geq 0$ is a recovery line. If process P_i does not have a checkpoint with index $< sn, 0 >$, the first checkpoint $C_{i,sn',0}$ with $sn' > sn$ must be included in the set S.*

Proof If $sn = 0$, S is a recovery line by definition. Otherwise suppose, by the way of contradiction that S is not a recovery line. Then, there exists a message m, sent by some process P_j to a process P_k, that is orphan with respect to the pair $(C_{j,sn,0}, C_{k,sn,0})$. Hence, we have: $C_{j,sn,0} \rightarrow send(m) \rightarrow receive(m) \rightarrow C_{k,sn,0}$ and $m.sn = sn$. This contradicts the observation 1.

Suppose process P_k does not have a checkpoint with sequence number sn, in this case, from lemma's assumption, we replace $C_{k,sn,0}$ with $C_{k,sn',0}$ where $sn' > sn$. As m is orphan wrt the pair $(C_{j,sn,0}, C_{k,sn',0})$, m is received by P_k in a checkpoint interval $I_{k,sn'',en}$ such that $m.sn > sn''$ contradicting the observation 2.

Suppose process P_j does not have a checkpoint with sequence number sn, in this case, from lemma's assumption, we replace $C_{j,sn,0}$ with $C_{j,sn',0}$ where $sn' > sn$. As m is orphan wrt the pair $(C_{j,sn',0}, C_{k,sn,0})$, $m.sn' > sn$ and $receive(m) \rightarrow C_{k,sn',0}$. This contradicts observation 1.

Hence, in all cases the assumption is contradicted and the claim follows. □

Lemma 4.2 *Let C_{i,sn,en_i} and $next(C_{i,sn,en_i})$ be two local checkpoints such that $C_{i,sn,en_i} \overset{\mathcal{L}_{sn}}{\equiv} next(C_{i,sn,en_i})$. If the set of checkpoints $S = (C_{1,sn,en_1}, C_{2,sn,en_2}, \ldots, C_{i,sn,en_i}, \ldots, C_{n,sn,en_n})$, with $en_i \geq 0$, is a recovery line $\mathcal{L}_{sn}$ then the set $S' = S - \{C_{i,sn,en_i}\} \cup \{next(C_{i,sn,en_i})\}$ is a recovery line.*

Proof If $C_{i,sn,en_i} \overset{\mathcal{L}_{sn}}{\equiv} next(C_{i,sn,en_i})$ then from definition 4.1, for each message m sent by P_j such that $receive(m) \rightarrow next(C_{i,sn,en_i})$ we have $send(m) \rightarrow C_{j,sn,en_j}$, thus no orphan message can ever exist with respect to any pair of checkpoints in S'. □

From Lemma 4.1 and Lemma 4.2, it follows that each checkpoint belongs to at least one recovery line. In particular, $C_{i,sn,en} = next(C_{i,sn',en'})$ belongs to all recovery lines $\mathcal{L}_{sn''}$ with $sn' < sn'' \leq sn$.

Lemma 4.3 *The set $\mathcal{L}_{sn} = \cup_{\forall j} C_{j,sn,EQ_i[j]}$ with $sn \geq 0$ and $\forall j\ EQ_i[j] = 0$ is a recovery line.*

Proof It follows from lemma 4.1 and from the semantic of the vector EQ_i explained in Section 4.2. □

Lemma 4.4 *The set $S = \cup_{\forall j} C_{j,sn,EQ_i[j]}$ is a recovery line.*

Proof (Sketch) Let us assume, by the way of contradiction, S is not a recovery line. If $\forall j\ EQ_i[j] = 0$, from lemma 4.3, the assumption is contradicted. Otherwise, there exists a message m, sent by some process P_j to a process P_k, that is orphan with respect to the pair $(C_{j,sn,EQ_i[j]}, C_{k,sn,EQ_i[k]})$ and there exists a causal message chain μ that brings this information to P_i encoded in EQ_i. Hence, we have: $C_{j,sn,,EQ_i[j]} \rightarrow send(m) \rightarrow receive(m) \rightarrow C_{k,sn,EQ_i[k]}$, $m.sn = sn$ and $m.EQ[j] \geq EQ_k[j]$. So, upon arrival message m falls in case 2 of Section 4.3. In this case, the index associated to $C_{k,sn,EQ_k[j]}$ is provisional (see the third point of Section 4.3). Before P_k sends the first message m' forming the causal message chain μ, the index has to be permanent. Hence, according to the algorithm, the index is replaced by $< sn+1, 0 >$, EQ_k is reset and piggybacked on m'. As soon as the last message of the causal message chain m'' arrives at P_i, $m''.sn > sn_i$ then $\forall j\ EQ_i[j] = 0$ which is consistent by lemma 4.3. So the initial assumption is contradicted and the claim follows. □

5 A Performance Study

5.1 Simulation Model

This section presents simulation results of a performance study to compare BCS, MS and the proposed algorithm (BQF). The simulation has been carried for the *uniform point-to-point* environment in which each process can send a message to any other and the destination of each message is a uniformly distributed random variable. We assume a system with $n = 8$ processes, each process executes internal, send and receive operations with probability $p_i = 0.8$, $p_s = 0.1$ and $p_r = 0.1$, respectively. The *time to execute an operation* in a process and *the message propagation time* are exponentially distributed with mean value equal to 1 and 100 time units respectively.

We also consider a *bursted point-to-point environment* in which a process with probability $p_b = 0.1$ enters a burst state and then executes only internal and send events (with probability $p_i = 0.8$, $p_s = 0.2$ respectively) for B checkpoint interval (when B=0 we have the uniform point-to-point environment described above).

Basic checkpoints are taken periodically. Let *bcf* (basic checkpoint frequency) be the percentage of the ratio t/T where t is the time elapsed between two successive periodic checkpoints and T is the total execution time. For example, *bcf*= 100% means that only the initial local checkpoint is a basic one, while *bcf*= 0.1% means that each process takes 1000 basic checkpoints.

We also consider a degree of heterogeneity among processes H. For example, $H = 0\%$ (resp. $H = 100\%$) means all processes have the same checkpoint period $t = 100$ (resp. $t = 10$), $H = 25\%$ (resp. $H = 75\%$) means 25% (resp. 75%) of processes have the checkpoint period $t = 10$ while the remaining 75% (resp. 25%) has a checkpoint period t=100.

A first series of simulation experiments were conducted by varying bcf from 0,1% to 100% and we measured (a) the ratio Tot between the total number of checkpoints taken by an algorithm and the total number of checkpoints taken by BCS and (b) the average number of checkpoints F forced by each basic checkpoint.

In a second series of experiments we varied the degree of heterogenity H of the processes and then we measured (c) the ratio E between the total number of checkpoints taken by BQF and MS.

As we are interested only in counting how many local states are recorded as checkpoints, the overhead due to the taking of checkpoints is not considered. Each simulation run contains 8,000 message deliveries and for each value of *bcf* and H, we did several simulation runs with different seeds and the result were within four percent of each other, thus, variance is not reported in the plots.

5.2 Results of the Experiments

Figure 4 shows the ratio Tot of MS and BQF in an uniform point-to-point environment. For small values of *bcf* (below 1.0%), there are only a few send and receive events in each checkpoint interval, leading to high probability of equivalence between checkpoints. Thus BQF saves from 2% to 10% of checkpoints compared to MS. As the value of *bcf* is higher than 1.0%, MS and BQF takes the same number of checkpoints as the probability that two checkpoints are equivalent tends to zero. An important point lies in the plot of the average number of forced checkpoints per basic one taken by MS and BQF shown in Figure 6. For small value of *bcf*, BQF induces up to 70% less than MS.

The reduction of the total number of checkpoints and of the ratio F is amplified by the bursted environment (Figure 5 and 7) in which the equivalences between checkpoints on processes running

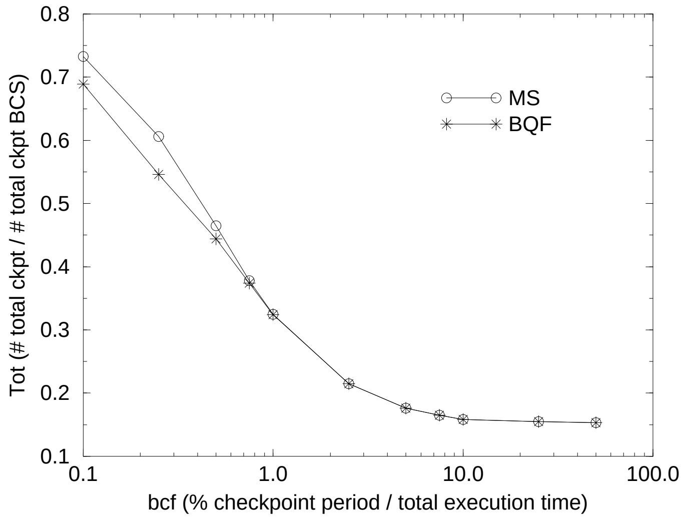

Figure 4: *Tot* versus *bcf* in the *uniform point-to-point* environment (B=0).

in the burst mode are disseminated to the other processes causing other equivalences. In this case, for all values of *bcf*, BQF saves from a 7% to 18% checkpoints compared to MS, and induces up to 77% less than MS.

The low values of F shown by BQF suggested that its performance could be particularly good in an heterogeneous environment in which there are some processes with a shorter checkpointing period. These processes would push higher the sequence number leading to a very high checkpointing overhead using either MS or BCF.

In Figure 8, the ratio E as a function of the degree of heterogeneity H of the system is shown in the case of uniform (B=0) and bursted point-to-point environment (B=2). The best performance (about 30% less checkpointing than MS) are obtained when $H = 12.5\%$ (i.e., when only one process has a checkpoint frequency ten times greater than the others) and $B = 2$.

In Figure 9 we show the ratio Tot as a function of *bcf* in the case of B=2 and $H = 12.5\%$ which is the environment where BQF got the maximum gain (see Figure 8). Due to the heterogenity, *bcf* is in the range between 1% and 10% of the slowest processes. We would like to remark that in all the range the checkpointing overhead of BQF is constantly around 30% less than MS.

6 Conclusion

In this paper we presented an index-based checkpointing algorithm well suited for autonomous distributed systems that reduces the checkpointing overhead compared to previous algorithms. It lies on an equivalence relation that allows to advance the recovery line without increase its sequence number.

The algorithm optimistically assumes that a basic checkpoint is equivalent to the last one. In

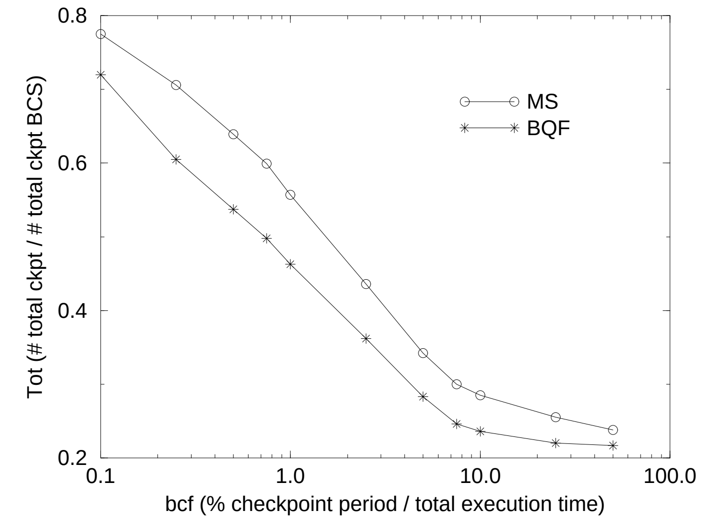

Figure 5: *Tot* versus *bcf* in the *bursted point-to-point* environment (B=2).

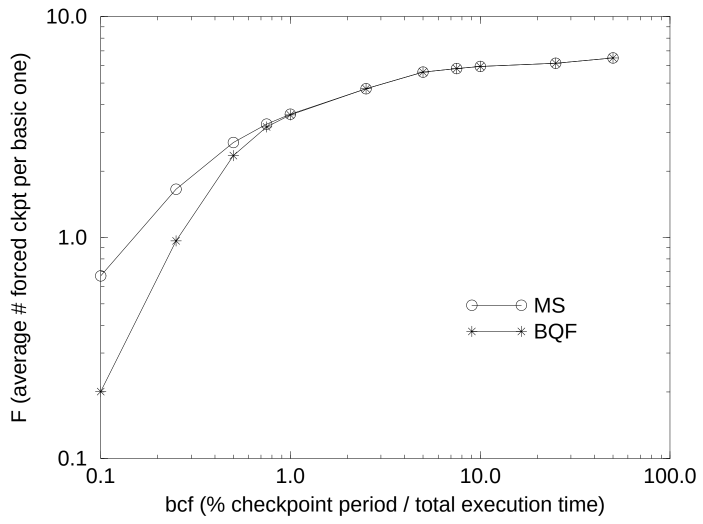

Figure 6: *F* versus *bcf* in the *uniform point-to-point* environment (B=0).

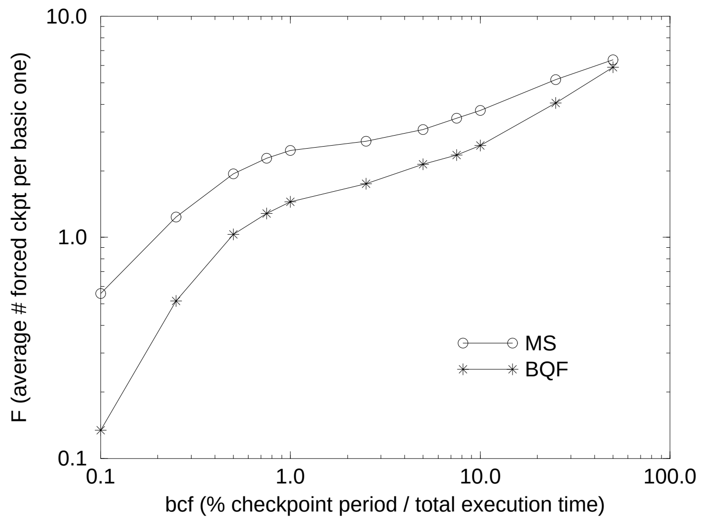

Figure 7: F versus *bcf* in the *bursted point-to-point* environment (B=2).

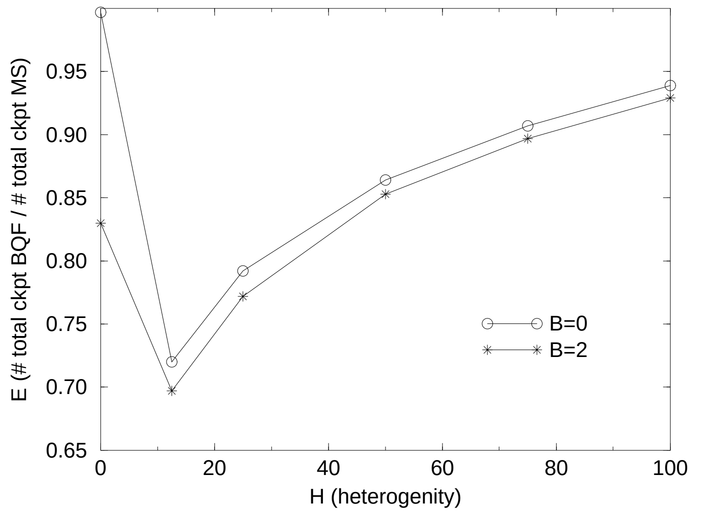

Figure 8: E versus *Heterogeneity* in both the *uniform point-to-point* environment (B=0) and the *bursted point-to-point* environment (B=2).

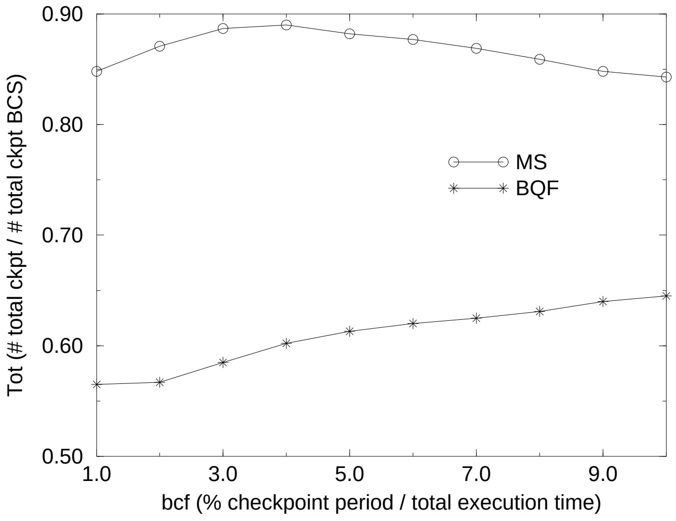

Figure 9: *Tot* versus *bcf* of the slowest processes in a *bursted point-to-point* environment (B=2) with H=12.5%

such a case, the index of the basic checkpoint has the equivalence number incremented by one. Hence, the assumption is confirmed or denied before the first send after the basic checkpoint. If the assumption is confirmed the provisional index becomes permanent, otherwise the sequence number is increased, as in [3, 11], and this will induce forced checkpoints in other processes.

We presented an extensive simulation study which quantifies the saving of checkpointing overhead in different environments compared to previous proposals. The price to pay is each application message piggybacks $n+1$ integers as control information compared to one integer used by previous algorithms.

The proposed algorithms can be also easily adapted to index-based algorithms that does not ensure each checkpoint belongs to a recovery line as the one presented in [18]. In this algorithm, a recovery line has a sequence number which is a multiple of a parameter of the laziness Z of the system. If Z=1 [18] boils down to [3].

References

[1] R.Baldoni, J. Brezinski, J.M.Helary, A.Mostefaoui, M.Raynal, On modeling consistent checkpoints and the domino effect in distributed systems, *in Proc. IEEE Int. Conference on Future Trends in Distributed Computing Systems*, 1995, pp.314–323.

[2] R.Baldoni, J.M.Helary, A.Mostefaoui, M.Raynal, A communication-induced checkpointing protocol that ensures rollback-dependency trackability, *Technical Report No.2564, INRIA, France, June 1995.*

[3] D.Briatico, A.Ciuffoletti, L.Simoncini, A distributed domino-effect free recovery algorithm, *in Proc. IEEE Int. Symposium on Reliability Distributed Software and Database*, 1984, pp.207-215.

[4] K.M.Chandy, L.Lamport, Distributed snapshots: determining global states of distributed systems, *ACM Transactions on Computer Systems*, Vol.3, No.1, 1985, pp.63-75.

[5] F.Cristian, F.Jahanian, A timestamp-based checkpointing protocol for long-lived distributed computations, *in Proc. IEEE Int. Symposium on Reliable Distributed Systems*, 1991, pp.12-20.

[6] E.N.Elnozahy, D.B.Johnson, Y.M.Wang, A survey of rollback-recovery protocols in message-passing systems, *Technical Report No.CMU-CS-96-181, School of Computer Science, Carnegie Mellon University*, 1996.

[7] E.N.Elnozahy, W.Zwaenepoel, Manetho: transparent rollback recovery with low overhead, limited rollback and fast output commit, *IEEE Transactions on Computers*, Vol.41, No.5, 1992, pp.526-531.

[8] R.Koo, S.Toueg, Checkpointing and rollback- recovery for distributed systems, *IEEE Transactions on Software Engineering*, Vol.13, No.1, 1987, pp.23-31.

[9] L.Lamport, Time, clocks and the ordering of events in a distributed system, *Communications of the ACM*, Vol.21, No.7, 1978, pp.558-565.

[10] D.Manivannan, R.H.B.Netzer, M.Singhal, Finding consistent global checkpoints in a distributed computation, *to appear in IEEE Transactions on Parallel and Distributed Systems.*

[11] D.Manivannan, M.Singhal, A low-overhead recovery technique using quasi synchronous checkpointing, *in Proc. IEEE Int. Conference on Distributed Computing Systems*, 1996, pp.100-107.

[12] D.Manivannan, M.Singhal, Quasi-synchronous checkpointing: models, characterization, and classification, *Technical Report No.OSU-CISRC-5/96-TR33, Dept. of Computer and Information Science, The Ohio State University*, 1996.

[13] R.H.B.Netzer, J.Xu, Necessary and sufficient conditions for consistent global snapshots, *IEEE Transactions on Parallel and Distributed Systems*, Vol.6, No.2, 1995, pp.165-169.

[14] B.Randell, System structure for software fault tolerance, *IEEE Transactions on Software Engineering*, Vol.SE1-2, 1975, pp.220-232.

[15] D.L. Russell, D.L. State restoration in systems of communicating processes, *IEEE Transactions on Software Engineering*, Vol. SE6, No. 2, 1980, pp. 183-194.

[16] R.E.Strom, D.F.Bacon, S.A.Yemini. Volatile logging in n-fault-tolerant distributed systems, *In proc. IEEE Int. Symposium on Fault Tolerant Computing*, 1988, pp.44-49.

[17] Y.M.Wang, Consistent global checkpoints that contains a set of local checkpoints, *to appear in IEEE Transactions on Computers (April 1997).*

[18] Y.M.Wang, W.K.Fuchs, Lazy checkpoint coordination for bounding rollback propagation, *in Proc. IEEE Int. Symposium on Reliable Distributed Systems*, 1993, pp.78-85.

[19] J.Xu, R.H.B.Netzer, Adaptive independent checkpointing for reducing rollback propagation, *in Proc. IEEE Int. Symposium on Parallel and Distributed Processing*, 1993, pp.754-761.