Experimental Evaluation of Concurrent Checkpointing 
and Rollback-Recovery Algorithms*

Bharat Bhargava  
Shy-Renn Lian  
Pei-Jyun Leu

Department of Computer Sciences  
Purdue University  
West Lafayette, IN 47907

ABSTRACT

This paper evaluates the performance of two distributed checkpointing and recovery algorithms, synchronous checkpointing algorithm (SA) and independent checkpointing algorithm (ICA). The performance is based on a detailed implementation of algorithms (that we have published earlier) in C language. A benchmark that simulates a variety of application requirements and considers variations in the number of processes running on one or more machines, size of processes to be checkpointed, size of control messages, and the frequency of message exchanges for normal processing is used to conduct the experiments. We make measurements for the elapsed time and the cpu time (includes computation and message processing time) to run a single instance of the checkpoint or rollback. We repeat the experiments for various combinations of concurrent checkpoint and rollback executions. We compute the messages needed for synchronization. We find that the time that a process spends in processing control messages contributes significantly to the elapsed time in both algorithms. We note that elapsed times for recovery for both algorithms are comparable when the number of checkpoints is small. The overhead of checkpointing is higher than for rollback in the SA algorithms. As the number of independent checkpoints increases in the ICA algorithm, elapsed time for recovery becomes larger than for an SA algorithm.

1. Introduction

Checkpointing and rollback recovery in distributed systems has been studied in the literature [1, 2, 6, 7, 8, 9, 10, 11, 12, 13] over the last decade. This research allows the system to recover and restart from a consistent state after a failure. Two main approaches that have been taken to design algorithms in this area are the synchronous and the asynchronous approach. We have made a serious attempt to evaluate the performance of these algorithms. We have selected one algorithm from each approach. The algorithms studied in this experiment have been published in [2] and [9]. The implementation and measurements are performed in the mini-Raid system[4] which runs on SUN-3/50\(^1\) workstations. We measure the total time a process spends in executing one instance of the algorithm, and the computational and message traffic overload introduced by the algorithms. These measurements quantify the efficiency of the algorithms and the overheads contributed to the response time of an application. We identify and measure various components that make up the execution time of the algorithms. Based on the experiment results, we also propose guidelines for efficient applications of the algorithms. This research should be useful in achieving efficient fault-tolerance in distributed databases.

In the synchronous algorithm (SA) [9], distributed checkpointing and rollback operations are synchronized among multiple processes to ensure global consistency. Each instance of checkpointing or rollback follows a hierarchical two-phase commit protocol. The initiator of an instance sends out a checkpoint or rollback request to processes that have exchanged messages with it since their last checkpoints were taken. Upon receipt of a checkpoint or rollback request, a process propagates the request to other processes that have exchanged messages with it since their last checkpoints were taken. Since all the processes in a system are synchronized at each checkpointing instance, each process will rollback only to the prior checkpoint when recovering from system failure. Concurrent execution of multiple instances initiated by different coordinators is allowed.

In the independent checkpointing algorithm (ICA) [2] based on the asynchronous approach, each process takes checkpoints independently according to its own need and without any synchronization with other processes. Multiple checkpoints have to be kept in local stable storage. For recovery, each process \(p\) maintains an input information table \(IIT_p\). Each entry \(IIT_p[p, q]\) of \(IIT_p\) contains the set of checkpoint intervals of \(q\) during which \(q\) sent some messages later received by \(p\) in checkpoint interval \(I_p\). Only the entry for the current checkpoint interval is stored in main memory, the rest of the table can be in the stable secondary storage. When a process is recovering from a failure, it collects system message flow information from all other processes, computes to establish a consistent recovery line and informs this to all other processes. Note that recovery line represents a consistent global state from which the system can recover from the failure.

In Section 2, we describe the experimental design, the input parameters, and the measured data. We choose parameters based on our working environment but have tried different values for input parameters to get a broad spectrum of measurements.

---

* This research is supported by NASA and AIRMICS under grant number NAG 1-676 and UNISYS.
† Currently at Tandem Corporation.

---

1. SUN-3 is a trademark of Sun Microsystems, Inc.
Section 3 contains the experiment results about the performance of these algorithms. We first measure the time to execute a single instance of the algorithms without concurrent execution. We identify and measure each component in the execution time.

In Section 4, we describe how the concurrent execution of several instances of the algorithms improves their performance. For the synchronous algorithm, experiments have been done for the cases of concurrent checkpointing, concurrent rollback, and concurrent execution of checkpointing and rollback. In the independent checkpointing algorithm, performance analysis of the algorithm in concurrent rollback execution is derivable quantitatively. So instead of data, we provide quantitative statements.

Section 5 contains the overhead contributed by the algorithms. The overhead is evaluated in terms of CPU usage and the number of synchronization messages.

In Section 6, we report the effect of multiprogramming level on the performance of the algorithms. Two factors, interprocess communication delay and time sharing delay, affect the performance.

Section 7 contains the comparison of the two algorithms and some concluding remarks.

2. Experimental Design

The experiments were conducted to measure the performance of both checkpointing and rollback recovery actions in SA and ICA. Performance was studied by the following factors:

- the duration of the execution of the algorithms.
- the overhead induced in the system by executing the algorithms.
- the effect of concurrent execution of more than one instance of the algorithms in the system.
- the effect of frequency of message exchange.

2.1. Experimental Environment

Implementation/measurements/experimentations were done in the mini-Raid software [4,5]. The system runs on SUN-3/50 workstations connected to a 10 Mbps Ethernet in our laboratory. In this system, communication delay is approximately 5-10 ms (milliseconds) for processes on a single machine and 20 ms across machines [3]. Most of experiments were carried out on a single machine so that the effects of network communications were factored out. For SA, since each checkpointing and rollback action requires a coordination among processes, we extended the experiments on multiple machines to observe the effect of different environments on the execution of a synchronous instance. For local communication, processes communicate through message queues in Sun Unix. Each process is equipped with two queues for incoming messages, one for normal messages, and the other for synchronization messages. For remote communication, UDP communication facilities are used.

2.2. Input Parameters

The following parameters were used to simulate a variety of processing benchmarks:

- number of processes
- size of processes
- size of control messages
- frequency of normal system message sent

Process size in this study is expressed in the form of a local checkpoint/rollback delay because a checkpoint (rollback) delay is the time to write (read) the image of a process into (from) the disk. Instead of limiting the measurements on the existing processes such as concurrency controller, atomicity controller, replication controller, etc in the Raids [5], we create dummy processes with varying sizes so that the performance can be applied to different distributed applications. These processes periodically send messages to one another. Error recovery is introduced to the system by sending a control message to a process.

The values of parameters used in these experiments are based on the following observations:

- Local checkpoint/rollback delay: We have examined 900 object files in the UNIX2 system, some of which are system files, while others are user files. An object file is the memory image of a process, and has three segments: text, data, and bss. In taking a checkpoint, we need only write the data and bss segments to the disk, while in rollback, we only read the data and bss segments. The size of these object files (excluding their text segments) in the UNIX system ranges from 4K bytes to 48K bytes. The checkpoint and rollback operations were measured to take time ranging from 89 ms to 496 ms.
- Number of processes: The algorithms are executed by 2 to 10 processes each time. We are not certain about the effect when large number (say 50 processes) are involved. Since in a system, checkpointing would be limited to a few processes to avoid excessive overhead, the results of our experiments can be utilized in most applications.
- Control message size: In SA, each synchronization message only needs to contain the sender, the receiver, and the message type. The message size is determined to be 22 bytes. In ICA, message size depends on the number of checkpoints taken by each process. We choose for each process to take 4 to 10 checkpoints. We do not intend to choose a very large checkpoint number because the algorithm allows old checkpoints to be discarded. In our implementation, a control message containing system message flow information has the size of \((#\_of\_checkpoints)\times(#\_of\_processes)\). Hence the size of control message ranges from 32 bytes to 1000 bytes. In this implementation, we use a two-dimensional bit vector for each checkpoint interval of process p, to indicate the message flow from every checkpoint interval of every process q. It is possible to reduce the message size by storing only the non-zero entries of the bit vector in the control message.
- Frequency of normal system message: For SA, during rollback recovery, processes always roll back to the last checkpoint, the message exchange pattern does not affect the rollback distance of each process, we let the dummy processes

2. UNIX is a trademark of AT&T Bell Laboratories.
randomly send messages to one another. For ICA, processes may roll back to the checkpoint older than the last one and rollback distance may be dependent on the message exchange rate. Since message exchange rate depends on the application and no empirical data about message exchange pattern is available in the literature, we choose the uniform distribution of message exchange. Uniform message exchange accounts for the worst case for ICA in the study of rollback distance. We assign the probability for a process to send every other process at least one message during each checkpoint interval to be 1, 1/2, 1/3, 1/4, 1/5, or 1/6.

2.3. Experimental Setup and Implementation

The experiments were run in the system repeatedly over a two month period and execution times of the algorithms were recorded after a stable state of the processing was achieved. The times presented in this paper are the averages of the recorded times. Execution times were measured in the software by referencing the processor clock with a granularity of 20 ms.

Each measurement is carried out in the following steps: 1) We choose values of parameters, type of instance (i.e. rollback or checkpointing), and a process to be the coordinator of the instance. Then input this information to the driver (to produce a benchmark) of the experiment. 2) Accordingly, the driver first initiates a number of dummy processes in the system. These processes send messages to one another. 3) The driver then invokes a starter to be the coordinator of the instance. This starter sends a control message to other processes to initiate a checkpoint instance or a rollback instance accordingly. 4) The processes involved in the instance record the times when it begins and when it finishes the execution of this instance. 5) For concurrent execution, steps 3 and 4 are repeated for different coordinator. 6) We send a termination command to the driver and the driver terminates the measurement by killing all the processes.

During measurements, the checkpointing or rollback actions were simulated by requiring a process to sleep for various values of checkpoint or rollback delay. Note that reading/restoring memory images can be done by a back-end processor in a real application.

For SA, the two-phase commit protocol for a checkpointing/rollback instance follows a spanning tree which records the message exchanges. We implement the spanning tree as arrays of integers. Each process maintains two arrays: one records the senders of the messages it receives, the other records the receivers of the messages it sends. These two arrays are reset when a new checkpoint is made. For ICA, the input_information table is implemented by a three-dimensional bit vector. Each entry $ITP(i,j,k)$ indicates whether during $p$'s checkpoint interval $i$, $p$ received a message from $j$ sent during $j$'s checkpoint interval $k$. A value of 1 indicates a message receipt, 0 otherwise.

2.4. Measured Data

Output parameters chosen for these experiments are elapsed time and cpu utilization. In addition, average rollback distance for ICA is measured. It is important that a checkpointing/rollback recovery algorithm does not greatly overload a system. Besides, the execution time of such an algorithm should not be long so that a system can soon return to a stable state for normal execution. We measure elapsed time for the duration of executing one instance of the algorithms and cpu utilization for the overhead induced in the system. Rollback distance is a method to measure the amount of computation lost in rollback recovery. A large rollback distance means a process must repeat large amount of execution to reach the local state right before error occurs. For SA, a process always returns to the last checkpoint instance and rollback recovery; while for ICA, it is a parameter to measure. Note that the data collected in this study are not intended to represent the absolute performance of a system but rather the performance of the system for a particular set of system parameters. The comparison of data collected in different cases is of more interest than the numerical value of each.

2.4.1. The Synchronous Algorithm

We measure elapsed time and cpu utilization for both the coordinator and the participants during the execution of a checkpoint instance or a rollback instance. In our study, an instance consists of several processes and a process can be a participant of more than one concurrent instances. Elapsed time starts from the time when the process receives a checkpoint request or a rollback request until it receives a commit or an abort decision from the coordinator. We have measured elapsed times for the execution of both a single instance and concurrent instances. Elapsed time contains three components: a) time to take a local checkpoint or roll back to the last checkpoint, b) cpu time, and c) idle time waiting for messages. cpu time consists of communication cost and computation cost in executing the algorithm. Communication cost is the time spent in sending/receiving and processing synchronization messages. Computation cost aggregates the times for the coordination among processes in checkpointing and rollback synchronization.

The following notation has been used in this paper for SA.

Notation.

- $elap_c$ elapsed time of the coordinator of a single instance.
- $elap_p$ elapsed time of a participant of a single instance.
- $elap_{cp}$ elapsed time of a process that is the coordinator of one instance and also a participant of the other instance. In this case, two instances are executed concurrently.
- $elap_pp$ elapsed time of a process that is a common participant of two instances.
- $elap_{cpp}$ elapsed time of a process that is the coordinator of one instance and also a common participant of the other two instances. In this case, three instances are executed concurrently.
- $elap_{cppp}$ elapsed time of a process that is a common participant of three instances.

Similarly, $CPU_c$, $CPU_p$, $CPU_{cp}$, $CPU_{cpp}$, $CPU_{cppp}$, and $CPU_{cpppp}$ denote the corresponding cpu times during the execution of a single instance and that of concurrent instances.

2.4.2. The Independent Checkpointing Algorithm

Due to the freedom in checkpointing, the independent checkpointing algorithm does not incur synchronization overhead during the checkpointing phase [2]. Since most of the overhead is during the rollback recovery phase of the algorithm, We focus on this phase. We measure elapsed time and cpu utilization for a rollback coordinator, elapsed time for a rollback participant, and average rollback distance of all processes. For a rollback coordinator,
elapsed time consists of three components: 1) time in collecting system message flow information. 2) local computation time for a recovery line. 3) time for restoring a previously saved state. For a rollback participant, elapsed time is the period between the instance it receives a rollback initiating message and the instance it finishes the rollback operation. cpu usage for a rollback coordinator is the second component contained in elapsed time. Rollback distance is the count of checkpoint intervals between the last checkpoint and the one to which a process rolls back for recovery. We measure the average rollback distance of all processes participating in a rollback instance for different message exchange patterns.

The independent checkpointing algorithm requires that each process keep a number of old checkpoints. Since stable storage is inexpensive, space cost is not a major concern. However, as processes keep multiple checkpoints, the size of control message for rollback recovery increases. We investigate the effect of the number of checkpoints kept by each process on the performance of rollback recovery.

3. Performance of SA and ICA without concurrent execution

The performance of SA and ICA is first evaluated in the absence of concurrent execution so that the interactions among different instances are factored out. The effect of blocking due to the execution of more than one instance is brought into consideration in Section 4.

3.1. Performance of SA

In SA, each instance of checkpointing or rollback recovery follows a two-phase commit protocol. The elapsed time measured for a coordinator in a checkpointing or rollback instance is the time for the two-phase commit protocol. Two-phase commit protocol used in executing a rollback instance has a higher degree of parallelism than that in a checkpointing instance, hence the elapsed time of processes is longer in checkpointing than in rollback instance. Table 1 contains the measurements of the elapsed time of checkpointing processes.

Table 1. Elapsed times for checkpointing in SA (in milliseconds).

<table>
<thead>
<tr>
<th>number of processes in the checkpointing instance</th>
<th>Elapsed times (in milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4, 5</td>
<td></td>
</tr>
<tr>
<td>local checkpoint delay: 251 ms</td>
<td></td>
</tr>
</tbody>
</table>

A rollback instance can always commit without delay: If there is concurrent execution of checkpointing and rollback, the checkpointing instance is blocked. If there is concurrent rollback, the different rollback instances share the rollback points. Therefore, we can allow a process to roll back without waiting until other participants agree to roll back. A process can thus recover from transient error faster. If processes roll back only after other participants agree to do so, their normal operations will be suspended for a long period of time. In our algorithm, the period of time for normal operations to be suspended is about the same as the time for a process to roll back. Table 2 shows the elapsed time of a process in the execution of a rollback instance with respect to three different local rollback delays in the single site environment. This period of time is about 1.4 to 4 times the local rollback delay. Additional data about elapsed time, with different values of parameters, for a rollback instance are given in Table 5.

Table 2. Elapsed times of rollback recovery in SA (in milliseconds).

<table>
<thead>
<tr>
<th>number of processes of the rollback instance: 8</th>
<th>local rollback delays: 89 ms, 251 ms, 496 ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>coordinator</td>
<td>363 472 719</td>
</tr>
<tr>
<td>participant</td>
<td>316 438 684</td>
</tr>
</tbody>
</table>

3.2. The Independent Checkpoint Algorithm

Elapsed time of checkpointing in ICA is merely the time taken by a process to restore its image into stable storage. It does not consume any time for computation or synchronization. We measure the elapsed time for both a rollback coordinator and a rollback participant and the results are shown in Table 3. Elapsed time of a rollback coordinator consists of three components: time for collecting system message flow information, local computation for a recovery line, and the local rollback delay. From Table 3, elapsed time is about 2 to 4 times the delay due to a rollback. Elapsed times for both rollback coordinator and participants increase as the size of control message increases. This shows that as the number of checkpoints maintained by each process increases, the rollback recovery takes longer time to complete.

Table 3. Elapsed times of rollback recovery in ICA (in milliseconds).

<table>
<thead>
<tr>
<th>number of processes of the rollback instance: 8</th>
<th>local rollback delays: 89 ms, 251 ms, 496 ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case A: Each process keeps 4 checkpoints.</td>
<td>message size = 160 bytes</td>
</tr>
<tr>
<td>local rollback delays: 89 ms, 251 ms, 496 ms</td>
<td></td>
</tr>
<tr>
<td>coordinator</td>
<td>89 251 496</td>
</tr>
<tr>
<td>participant</td>
<td>328 446 700</td>
</tr>
<tr>
<td>Case B: Each process keeps 10 checkpoints.</td>
<td>message size = 1000 bytes</td>
</tr>
<tr>
<td>local rollback delays: 89 ms, 251 ms, 496 ms</td>
<td></td>
</tr>
<tr>
<td>coordinator</td>
<td>880 920 1140</td>
</tr>
<tr>
<td>participant</td>
<td>689 851 1043</td>
</tr>
</tbody>
</table>

The average rollback distance are measured under the following assumptions: a) Message delay is smaller than checkpoint interval. b) The checkpoint interval of each process is of the same size. c) Each process keeps 10 checkpoints in the stable storage. We use the uniform distribution of message exchange for the measurements. The uniform distribution accounts for the worst case since the effect of an undone message is likely to propagate through the whole system. We assign the probability for a process to send every other
process at least one message during each checkpoint interval to be 1, 1/2, 1/3, 1/4, 1/5, or 1/6.

<table>
<thead>
<tr>
<th>probability of exchanging messages</th>
<th>1</th>
<th>1/2</th>
<th>1/3</th>
<th>1/4</th>
<th>1/5</th>
<th>1/6</th>
</tr>
</thead>
<tbody>
<tr>
<td>average rollback distance</td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>2.6</td>
<td>2.5</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4. Rollback distances in ICA.

We have found that the average rollback distance is independent of the number of processes, but depends on the message exchange pattern among the processes. The rollback distances in different message exchange rates are shown in Table 4. In general, the average rollback distance is small. As the probability of exchanging messages increases, the average rollback distance also increases. The worst case occurs when each process, during each checkpoint interval, sends at least a normal message to every other process. Then a rollback recovery may cause all processes to rollback to the beginning. In most applications, only a proper subset of processes has frequent message exchange. Moreover, probability of exchanging messages can be reduced by reducing the checkpoint interval. Thus the worst case is avoidable.

4. Performance in Concurrent Execution of the Algorithms

Both SA and ICA allow concurrent execution of combinations of several checkpointing and rollback recovery actions. Besides better performance, concurrent rollback recovery allows the system to have additional fault tolerance.

In ICA, each process takes checkpoints independently, hence ICA allows a system to have any degree of concurrent checkpointing operations. In rollback recovery, a recovering process has to collect system message flow information to compute a recovery line. For concurrent rollback recovery, to avoid collecting inconsistent information, message flow information has to be collected serially, but a local rollback by each participant can be done concurrently. Suppose that rollback instances $I_1, I_2, ..., I_k$ are initiated at the same time by processes $P_1, P_2, ..., P_k$, respectively. Let the priority of $P_i$ be greater than that of $P_{i+1}$, where $1 \leq i \leq k$. A process with lower priority collects message flow information after processes with higher priorities do so and computes a recovery line. We assume that it takes time $x$ for a process to finish the execution of rollback recovery algorithm and $y$ is the time for a process to make a local rollback. Then the finish time for process $P_i$ in concurrent rollback recovery is $(i-1)(x+y)+x$. For example, $P_1$ finishes at time $x$, $P_2$ finishes at time $(x+y) + x$, etc.

In a system executing SA, it is possible that there is more than one coordinator at a time. Each coordinator initiates a synchronization instance. Without concurrent execution, one instance would have to wait for the other to finish. The delay will be accumulated as there are more instances running. For example, suppose instances $I_1, I_2, \ldots, I_k$ are initiated at the same time, and they are of the same size. Each process takes time $\Delta y$ to make a checkpoint and propagate the checkpoint request. Assume that $I_1$ finishes at time $X$. Also assume that process $P_i$ is a common participant of instances $I_i$ and $I_{i+1}$, for $1 \leq i \leq k$. If $P_i$ can execute a checkpoint operation for $I_{i+1}$ only after $I_i$ has terminated, instance $I_{i+1}$ will finish $\Delta y$ time later than $I_i$. Then the finish time of instance $I_i$ is $X + (i-1)\Delta y$. If every common participant can execute checkpoint operations concurrently for two instances, the two instances can precede simultaneously. But the common participant still needs to spend $2\Delta y$ executing two checkpoint operations. The finish time of each instance will be the same.

An optimization has been done in SA for concurrent execution so that concurrent instances can share checkpoints or rollback points. Therefore, each common participant spends less than $2\Delta y$ executing checkpoint operations for the two instances. We have studied experimentally the effect of sharing checkpoints and rollback points on the elapsed time.

Table 5. Elapsed times of concurrent execution on the same machine in SA (in milliseconds).

<table>
<thead>
<tr>
<th>number of processes of each instance: 5</th>
<th>checkpoint instances</th>
<th>rollback instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>single instance</td>
<td>elap_s</td>
<td>583</td>
</tr>
<tr>
<td>two concurrent instances</td>
<td>elap_sp</td>
<td>665</td>
</tr>
<tr>
<td>three concurrent instances</td>
<td>elap_spp</td>
<td>701</td>
</tr>
<tr>
<td>instances</td>
<td>elap_spp</td>
<td>463</td>
</tr>
</tbody>
</table>

Tables 5 and 6 show the elapsed time of a coordinator and that of a participant for the execution of both a single instance and concurrent instances. In this experiment, each instance is executed by the same five processes.

Table 6. Elapsed times of concurrent execution on different machines in SA (in milliseconds).

<table>
<thead>
<tr>
<th>number of processes of each instance: 5</th>
<th>checkpoint instances</th>
<th>rollback instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>single instance</td>
<td>elap_s</td>
<td>559</td>
</tr>
<tr>
<td>two concurrent instances</td>
<td>elap_sp</td>
<td>588</td>
</tr>
<tr>
<td>three concurrent instances</td>
<td>elap_spp</td>
<td>617</td>
</tr>
<tr>
<td>instances</td>
<td>elap_spp</td>
<td>389</td>
</tr>
</tbody>
</table>

Due to the sharing of checkpoints and rollback points, the elapsed time of a process that executes operations for concurrent instances will be smaller. Our observations from Table 5 and 6 are as follows.
elap\_cp = elap\_c + elap\_p - d
elap\_pp = elap\_p + elap\_p - d
elap\_cpp = elap\_cp + elap\_p - d
elap\_ppp = elap\_pp + elap\_p - d
d = 251\ ms, which is the delay of taking a
local checkpoint or rollback.

Data on the left side of = are measured experimentally. The
expressions on the right side represent expected values. When a
process executes checkpoint operations for two concurrent check-
point instances, the process takes one local checkpoint instead of
two. Therefore, elap\_cp and elap\_pp can be expected to be d mil-
iseconds shorter than if the process makes two check-points. When
a process executes operations for two concurrent rollback instances,
the process rolls back twice instead of twice. Therefore, elap\_cp and
elap\_pp can be expected to be d milliseconds shorter than if the
process rolls back twice. For checkpoint processes, the experimen-
tal data are even smaller than expected, because processes tend to
utilize cpu idle time more efficiently in concurrent processing.

5. Overheads of the Algorithms

The overheads of these algorithms are evaluated in terms of
cpu time used and message traffic induced by the algorithms.

5.1. The Synchronous Algorithm

Processes synchronize their checkpoint operations and roll-
back operations by sending messages. In SA, the message overhead
is not uniformly distributed. The total number of messages sent and
received by the coordinator is in the order \(O(n^2)\). The total
number of messages sent and received by a participant is \(O(n)\),
where \(n\) is the number of the processes of the instance. We present
in Table 7 the maximum number of synchronization messages sent
among processes in executing a single instance. This number only
depends on the number of processes in the instance.

Table 7. Maximum number of synchronization messages in SA.

<table>
<thead>
<tr>
<th>number of processes in the instance: 5</th>
<th>process of checkpoint instances</th>
<th>process of rollback instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 coordinator and 4 participants.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>number of messages sent</td>
<td>13</td>
<td>26</td>
</tr>
<tr>
<td>number of messages received</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>total</td>
<td>13 + 4 \times 8 = 45</td>
<td>26 + 4 \times 5 = 46</td>
</tr>
</tbody>
</table>

We have measured the cpu cost for a process in the execution
of a single instance and that of concurrent instances. Each instance
is executed by the same five processes. Table 8 contains the cpu
costs when all five processes are on a single site. Table 9 contains
the cpu costs when each process is on a different site.

Table 8. cpu overhead of SA in multiprocessing environment (in milliseconds).

<table>
<thead>
<tr>
<th>number of processes of each instance: 5</th>
<th>process of checkpoint instances</th>
<th>process of rollback instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>single instance</td>
<td>CPU_c</td>
<td>47.0</td>
</tr>
<tr>
<td></td>
<td>CPU_p</td>
<td>18.3</td>
</tr>
<tr>
<td>two concurrent instances</td>
<td>CPU_cpp</td>
<td>67.1</td>
</tr>
<tr>
<td></td>
<td>CPU_pp</td>
<td>36.3</td>
</tr>
<tr>
<td>three concurrent instances</td>
<td>CPU_cpp</td>
<td>80.9</td>
</tr>
<tr>
<td></td>
<td>CPU_ppp</td>
<td>55.5</td>
</tr>
</tbody>
</table>

Table 9. cpu overhead of SA in distributed environment (in milliseconds).

<table>
<thead>
<tr>
<th>number of processes of each instance: 5</th>
<th>process of checkpoint instances</th>
<th>process of rollback instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>single instance</td>
<td>CPU_c</td>
<td>102.0</td>
</tr>
<tr>
<td></td>
<td>CPU_p</td>
<td>41.6</td>
</tr>
<tr>
<td>two concurrent instances</td>
<td>CPU_cpp</td>
<td>135.1</td>
</tr>
<tr>
<td></td>
<td>CPU_pp</td>
<td>87.7</td>
</tr>
<tr>
<td>three concurrent instances</td>
<td>CPU_cpp</td>
<td>173.0</td>
</tr>
<tr>
<td></td>
<td>CPU_ppp</td>
<td>138.8</td>
</tr>
</tbody>
</table>

From these data, we observe that the cpu cost in executing a
checkpoint instance is about the same as that in executing a rollback
instance with the same number of participants. For a process
involved in concurrent execution, the cpu cost is about the same as
if the events are executed sequentially on the process. The algo-

From these data, we observe that the cpu cost in executing a
checkpoint instance is about the same as that in executing a rollback
instance with the same number of participants. For a process
involved in concurrent execution, the cpu cost is about the same as
if the events are executed sequentially on the process. The algo-

5.2. The Independent Checkpointing Algorithm

In ICA, checkpointing actions incur no message overhead. In
rollback recovery, message complexity is \(O(3n)\), where \(n\) is the
number of processes in the instance. The rollback initiator sends $O(n)$ rollback initiating messages and $O(n)$ rollback request messages to other processes. All other processes totally send $O(n)$ input information messages to the rollback initiator.

The cpu overhead for rollback recovery is an important parameter to measure. Table 10 shows the cpu time spent by the coordinator in rollback recovery. In Table 10, we notice that the cpu time increases as the message size increases but the increases are slow. The message size depends on the number of checkpoints maintained by each process. From this we infer that discarding old checkpoints is not very critical since a larger message size does not significantly increase the cpu overhead. Hence, processes can discard old checkpoints when convenient.

Table 10. cpu overhead of rollback coordinator in ICA (in milliseconds).

<table>
<thead>
<tr>
<th>Control message sizes: 160, 360, 640, 1000 bytes</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>160</td>
<td>20</td>
<td>35</td>
<td>50</td>
<td>100</td>
<td>132</td>
</tr>
<tr>
<td>360</td>
<td>17</td>
<td>38</td>
<td>78</td>
<td>130</td>
<td>210</td>
</tr>
<tr>
<td>640</td>
<td>20</td>
<td>40</td>
<td>108</td>
<td>193</td>
<td>278</td>
</tr>
<tr>
<td>1000</td>
<td>30</td>
<td>90</td>
<td>160</td>
<td>250</td>
<td>376</td>
</tr>
</tbody>
</table>

In Table 10, we also observe that cpu time increases as the number of processes increases. This increase is much larger than that of the number of processes. Comparing the cpu time with elapsed time ( in Table 3 ) for rollback coordinator, we find that cpu time for local computation weighs about 17% of the elapsed time for rollback recovery. Since cpu time is a minor portion of the total delay during the execution of the rollback recovery algorithm, the increasing rate of cpu does not significantly degrade the performance of the rollback recovery algorithm.

6. Effect of Multiprogramming Level on Execution Time

In this section, we study the effect of different environments on the performance of SA. Since every checkpointing and rollback instance in SA produce heavy message traffic among processes, it is important to evaluate its performance in environments that utilize different communication facilities. In the environment where processes are distributed on different machines, communication cost is usually more expensive than if the processes are all on the same site. On the other hand, delay caused by time sharing is longer for processes being run on the same site than distributed in different sites. The total effect of these two factors is not easy to see. We evaluate the execution time of SA by running 4 processes in two different cases: 1) Processes are all on a single site, and 2) processes are distributed over multiple sites. When each site has one process, we use four sites. If each site has two processes, we use two sites. In the third case, one site has all the four processes. For the rollback recovery, we expect that the multiprogramming level affects the coordinator more than a participant. This is because the coordinator has higher cpu cost, which incurs more time sharing delay. Table 11 shows the elapsed times of the coordinator and that of a participant.

These results show that when all processes of an instance are on the same site, the elapsed time is the longest. When there is one process per site, the elapsed time is the shortest. We also observe that multiprogramming level affects processes of a checkpoint instance more than those of a rollback instance. This is because the two-phase commit protocol used in executing a rollback instance has a higher degree of parallelism than that in a checkpoint instance. Upon a rollback request, a process replies to the coordinator and propagates the rollback request before it rolls back to its last checkpoint. The coordinator can process some messages while other participants are rolling back, which does not consume the cpu resource. Therefore, rollback processes can utilize cpu idle time more efficiently than checkpoint processes. On the other hand, upon a checkpoint request, a process replies to the coordinator and propagates the checkpoint request only after it has made a checkpoint.

>From Table 11, we observe that the elapsed time of a checkpoint process increases faster than that of a rollback process, also that as the multiprogramming level increases, the elapsed time of the coordinator increases faster than that of a participant.

Table 11. Elapsed times of SA at different multiprogramming levels.

<table>
<thead>
<tr>
<th>Multiprogramming levels: 1, 2, 4 ( # of processes per site)</th>
<th>Number of processes: 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local checkpoint/rollback delay: 251 ms</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Checkpoint</th>
<th>Coordinator</th>
<th>Participant</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>542</td>
<td>587</td>
</tr>
<tr>
<td>2</td>
<td>308</td>
<td>344</td>
</tr>
<tr>
<td>4</td>
<td>299</td>
<td>327</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rollback</th>
<th>Coordinator</th>
<th>Participant</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>291</td>
<td>302</td>
</tr>
</tbody>
</table>

7. Comparison and Concluding Remarks

In this section, we summarize our observations based on the measurements, comment on the overall performance of SA and ICA, and compare these two approaches.

For SA, we have the following observations:

(1) Tables 1, 2, 5 and 6 show that the elapsed time for a checkpointing instance is longer than for a rollback instance. This is because the rollback algorithm allows a higher parallelism than the checkpointing algorithm in the two-phase commit protocol.

(2) Tables 8 and 9 indicate that the cpu overhead for checkpointing and rollback recovery is about the same.

(3) Message overhead is $O(2n^2)$ for both a checkpointing and a rollback instance, where $n$ is the number of processes involved in an instance.

(4) Tables 1, 2, 8, and 9 show that for the three components that make up the elapsed time of a checkpoint/rollback instance, cpu time contributes less than 10%, local checkpoint/rollback delay contributes between 25% to 60%, and collecting the synchronization messages contributes between 30% to 60%.
In summary, SA has two advantages: (1) the cpu overhead is very small, and (2) the nature of SA makes the rollback distance be always 1. The disadvantages are as follows: (1) checkpointing overhead is large. This is undesirable especially because this overhead takes longer time than for rollback recovery, and (2) message overhead \( O(2n^{2.5}) \) is large.

For ICA, we have the following observation:

(1) No message or cpu overhead for a checkpointing instance. Message overhead is \( O(3n) \) for a rollback instance.
(2) Table 3 and 10 show that, on the average, among the three components that make up the elapsed time of a rollback instance, cpu time contributes about 17%, local rollback delay contributes about 50%, and collecting messages of system flow information contributes about 33%.
(3) From Table 3, we observe that elapsed time for a rollback instance moderately increases as the size of control message increases. From Table 10, it shows that cpu time slightly increases as the size of control message increases.
(4) From Table 4, we observe that the message exchange rate has direct impact on the rollback distance.

In summary, ICA has the advantage of minimal checkpointing overhead. The disadvantage is that its performance in rollback recovery is dependent on message exchange rate and the number of checkpoints maintained by processes.

The SA and ICA are similar in two respects: (1) The execution time elapsed in rollback recovery is dependent on the number of processes in a system. (2) The time a process spends in collecting control messages weights a significant portion of the total elapsed time of an instance. Comparing the performance of rollback recovery in SA and ICA (Sections 3.1 and 3.2), we found that when the number of checkpoints maintained by each process is small, such as 4, the elapsed times for rollback recovery in both SA and ICA are about the same. When the number of checkpoints grows, elapsed time for recovery in ICA becomes longer than in SA. For the number of checkpoints increased to 10, the elapsed time in ICA is about double the time in SA. Since in SA, checkpointing time takes longer than rollback recovery, checkpointing would be paid off only if the system has a fairly high rate of failure. If ICA is to be used in a system, the user should use a more efficient implementation in which only nonempty entries in input information table are sent so that the control message can be shortened. It may increase cpu time in processing control messages but shorten the total elapsed time for recovery. This is not only because cpu time weighs a small portion of the elapsed time but also because cpu time does not increase as much as elapsed time does when message size increases. The average rollback distance in Table 4 was obtained under the assumption of uniform distribution of normal system messages. When a large value is assigned to the probability of the uniform distribution, it accounts for a more pessimistic situation than ordinary cases. However, in general, if processes in a system exchange messages very frequently, some of the processes may need to roll back to some previous checkpoints at distance in ICA. One way to cope with this kind of problems is to make the checkpoint interval adaptable. In other words, smaller checkpoint interval should be used when the system has a higher rate of failure or a higher rate of message exchange.

References