1988

# Experimenal Evaluation of Concurrent Checkpointing and Rollback-Recovery Algorithms

Bharat Bhargava
*Purdue University*, bb@cs.purdue.edu

Pei-Jyun Leu

Shy-Renn Lian

Report Number:
88-790

# EXPERIMENTAL EVALUATION OF CONCURRENT CHECKPOINTING AND ROLLBACK-RECOVERY ALGORITHMS

Bharat Bhargava
Pei-Jyun Leu
Shy-Renn Lian

CSD-TR-790
July 1988
Revised September 1989

# Experimental Evaluation of Concurrent Checkpointing and Rollback-Recovery Algorithms*

Bharat Bhargava
Pei-Jyun Leu†
Shy-Renn Lian

Dept. of Computer Sciences
Purdue University
West Lafayette, IN 47907

## ABSTRACT

We have implemented two classes of distributed checkpointing and rollback recovery algorithms and evaluated their performance in a real processing environment. One algorithm is based on the synchronous approach and the other on the asynchronous approach. The evaluation measures the overhead due to time spent in executing the algorithms and the cost in terms of computational time and message traffic. We identify the components that make up the execution time of these algorithms and study how each of them contributes to the total execution time. These data are validated by quantitative analysis. One objective of this study is to compare these approaches. This evaluation study is useful for a system designer in choosing the appropriate recovery algorithm based on the current application and environment. We believe that our study is the first attempt that implements, evaluates, and compares concurrent checkpointing/recovery algorithms in distributed systems. The knowledge gained by our research can be applied to achieve efficient fault-tolerance in distributed database systems, distributed operating systems, and multiprocess environments.

* This research is supported by NASA and AIRMICS under grant number NAG 1-676 and UNISYS.

* To appear in the Sixth International Conference on Data Engineering.

† Currently at Tandem Corporation.

# 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 have been taken to design algorithms for this problem: the synchronous and the asynchronous approach. It is necessary to evaluate the performance of these algorithms in a real processing environment. We selected one algorithm from each approach to checkpointing and rollback recovery and implemented them to evaluate their performance. The implementation and measurements are performed on SUN–3/50[1] workstations. The algorithms studied in this experiment have been published in [2] and [10]. We measure the total time a process spends in executing one instance of the algorithms, and the computational and message traffic overload introduced by the algorithms. These measurements quantify the efficiency of the algorithms and the cost introduced to the response time of 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.

In the synchronous algorithm ( SA ) [10], distributed checkpointing and rollback operations are synchronized among multiple processes to ensure global consistency. Only processes that have exchanged messages since their last checkpoints need to take checkpoints or roll back together. 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, processes take checkpoints independently according to their own need and without any synchronization. Multiple checkpoints have to be kept in local stable

1. SUN-3 is a trademark of Sun Microsystems, Inc.

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. The recovery line represents a consistent global state from which the system can recover from the failure.

In section 2, we describe the experimental design and measured data. In the absence of the availability of empirical data about failure rates and recovery rates, we choose parameters based on our working environment. In some cases, we make assumptions but try different values for input parameters to get a broad spectrum of experiments.

Section 3 shows 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 evaluation of processes in concurrent execution is straight forward. So instead of data, we provide quantitative statements.

In section 5, we analyze the overhead in the algorithms. The overhead is evaluated in terms of cpu usage and number of synchronization messages.

In section 6, we report the effect of various environments on the performance of the algorithms. Two factors, interprocess communication delay and time sharing delay, affect the performance. Communication delay between processes in different sites is more expensive than in the same site. The time sharing delay is longer for processes being run on the same site than distributed in different sites.

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

## 2. Experimental Design

The implementation/measurements/experimentations are done in the RAID system [5] and in its miniversion, the mini-RAID system [4]. The RAID system runs on SUN-3/50 workstations connected by Ethernet and the mini-RAID system runs on a single SUN-3/50 machine. In these systems, communication delay is approximately 5

ms (milliseconds) for processes on a single machine and 7 ms across machines [3]. In the following subsections, we first describe the environment of the experiment and parameters used in the experiment. Then we explain the set up of the experiment and the data collected for each of the two algorithms.

### 2.1. Experimental Environments and Input Parameters

We conduct the experiments for the synchronous algorithm in two different environments : 1) the mini-RAID system, where processes are all on a single site, and 2) the RAID system, where processes are distributed over multiple sites. For the independent checkpointing algorithm, the experiment is done on only one site. 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.

Since empirical data about failure rates and recovery rates is not available in the literature, we choose parameters based on our own working environment. In some cases, we make assumptions but try various values to get a broad spectrum of experiments. The values of parameters used in these experiments are based on the following observations:

- **Time for taking/restoring a checkpoint:** Each process of a checkpoint (or rollback) instance will make a single checkpoint (or rollback respectively). Checkpoint delay is the time to write the image of a process into the disk, while rollback delay is the time to read the image of a process from the disk. We have examined 900 object files in the UNIX[2] 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 were measured to take time ranging from 89 ms to 496 ms. Reading memory images or restoring images can be done by a back-end processor, which does not consume cpu resource. So we simulate the checkpointing or rollback actions by

2. UNIX is a trademark of AT&T Bell Laboratories.

requiring a process to sleep for the period of time it takes to read or write its image from disk.

- **Number of processes:** The algorithms are executed by 2 to 10 processes each time. We have not done the study on a large system hence we are not sure about the effect when scaled to say 50 processes. Since in a system, checkpointing should be limited to a few processes to avoid excessive overhead, the results of these experiments are also applicable in many systems.
- **Control message size:** In the synchronous algorithm, 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 the independent checkpointing algorithm, 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 chekpoints to be discarded. In this experiment, a control message containing system message flow information has the size of $(\text{numberofcheckpoints})^2 \times (\text{number of processes})$. Hence the size of control message range from 160 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. We can store only the non-zero entries of the bit vector in the control message so that the message size is reduced.

## 2.2. Experimental Procedures

Each measurement is carried out in the following steps: 1) We initiate several processes in the mini-raid system. These processes are assigned a processing task that requires of sending messages to one another. 2) We randomly invoke a checkpoint starter or a rollback starter. This starter sends a message to a set of processes. A process that receives this message initiates a checkpoint instance or a rollback instance accordingly. 3) The measurement is stopped by killing the processes.

The assumptions and details of parameters selected for measurement are given in each case in section 3.

### 2.3. Measured Data

The performance data collected in this experiment are described in the following sections. Measurements were taken over a two month period and the execution times of the algorithms were recorded after a stable state of the processing was achieved. The times presented here are the averages of the recorded times. Execution times were measured in the software by referencing the processor clock. It should be stressed that the average times are not intended to represent the absolute performance of the system but rather the performance of the system for a particular configuration of system parameters. Thus the comparison of average times is of more interest than the numerical value of each average time.

#### 2.3.1. The Synchronous Algorithm

We measure the performance of both the coordinator and the participants during the execution of a checkpoint instance or a rollback instance. In our experiment, an instance is executed by several processes and a process can be a participant of more than one concurrent instances. We measure *elapsed time* and *cpu usage* during the execution of instances. Elapsed time is the total time a process spends during the execution of an instance. This period 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. Elapsed times of processes have been measured for the execution of both a single instance and concurrent instances. Elapsed time contains three components: a) time to take a single 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.

**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.

$elap_{ppp}$ elapsed time of a process that is a common participant of three instances.

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

### 2.3.2. The Independent Checkpointing Algorithm

By the very nature of independence during 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, it is useful to focus on measurements during this phase. We measure *elapsed time* and *cpu usage* for a rollback coordinator, elapsed time for a rollback participant, and *average rollback distance* of all processes. Elapsed time consists of time spent in a rollback recovery instance. 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 rolling back. cpu usage for a rollback coordinator is the second component contained in elapsed time. Rollback distance is the number of checkpoint intervals 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 number of checkpoints kept by each process on the performance of rollback recovery.

# 3. Performance of the checkpointing and rollback recovery algorithms without concurrent execution

We study the efficiency of each algorithm by measuring the execution time in the absence of concurrent execution. In the independent checkpointing algorithm, unlike the synchronous approach, rollback distance depends on the application. We measure the average rollback distance for the independent checkpointing algorithm in a real scenario that is described in section 3.2.

## 3.1. The Synchronous Algorithm

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

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

number of processes of the checkpointing instance: 4, 5
All the processes are on the same site.
single checkpoint delay: 251 ms

| | 4 | 5 |
|---|---|---|
| coordinator | 542 | 583 |
| participant | 308 | 338 |

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 transcient 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. Because this period will include the time to synchronize with other processes, and the time to await decisions from the coordinator. 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 for the execution of a rollback instance with respect to three different rollback delays in a single site environment. This period of time is about 1.4 to 4 times the single rollback delay.

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

number of processes of the rollback instance: 8
All the eight processes are on the same site.
single rollback delays: 89 ms, 251 ms, 496 ms

| | 89 | 251 | 496 |
|---|---|---|---|
| coordinator | 363 | 472 | 719 |
| participant | 316 | 438 | 684 |

### 3.2. The Independent Checkpointing Algorithm

Checkpointing in the independent checkpointing algorithm is the time taken by a process to restore its image into stable storage. It does not comsume any time for computation or synchronization. We measure the elapsed time for both a rollback coordinator and a rollback participant for various values of the parameters 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 single rollback delay. The experimental results show that the single rollback delay dominates the total delay. From Table 3, the single rollback delay weights 20% to 60% of the total delay. On the average, elapsed time increases 19%, 37%, and 88% as the message size increases 125%, 300%, and 525% respectively.

We measure the average rollback distance 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 measure rollback distance under different message exchange patterns among the processes. Message exchange patterns are determined as follows. We assign the probability for a process to send every other process at least one message during each checkpoint interval to be 1/2, 1/3, 1/4, 1/5, or 1/6.

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

number of processes of the rollback instance: 8
All the eight processes are on the same site.
single rollback delays: 89 ms, 251 ms, 496 ms

Each process keeps 4 checkpoints.
message size = 160 bytes

| | 89 | 251 | 496 |
|---|---|---|---|
| coordinator | 420 | 540 | 780 |
| participant | 328 | 446 | 700 |

Each process keeps 10 checkpoints.
message size = 1000 bytes

| | 89 | 251 | 496 |
|---|---|---|---|
| coordinator | 880 | 920 | 1140 |
| participant | 689 | 851 | 1043 |

Table 4. Rollback distances in ICA.

| probability of exchanging messages between two processes in each checkpt. interval | 1/2 | 1/3 | 1/4 | 1/5 | 1/6 |
|---|---|---|---|---|---|
| average rollback distance | 5 | 4 | 2.6 | 2.5 | 0 |

overall average rollback distance: 2.8

The rollback distances in different message exchange patterns are shown in Table 4. The average performance over all the cases is 2.8 when each process keeps 10 checkpoints in the stable storage. 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. 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 every process, during each checkpoint interval, sends at least a normal message to every other process. Then a rollback recovery may cause all processes to roll back to the beginning. The chance for this worst case to occur is very small. Probability of exchanging messages can be reduced by reducing the checkpoint interval. This also reduces the rollback distance.

### 3.3. Remarks on the Performance of the Serial Execution of SA and ICA

A few observations can be made from the experiment results in section 3.1 and 3.2 as follows: 1) It took less time for the processes to roll back in the synchronous algorithm than in the independent checkpointing algorithm. 2) In the synchronous algorithm, checkpointing takes longer time than rollback recovery. Checkpointing in the independent checkpointing algorithm is more efficient than in the synchronous algorithm because no synchronization is required. 3) in the independent checkpointing algorithm, the message exchange pattern among processes was found highly correlated with the rollback distance.

## 4. Performance in Concurrent Execution of the Algorithms

In our study both the synchronous algorithm and the independent checkpointing algorithm allow concurrent execution in checkpointing and rollback recovery actions. One checkpointing (rollback) instance does not need to wait for the completion of another checkpointing (rollback) instance. Both algorithms also allow concurrent execution of any combination of checkpointing and rollback actions. Besides better performance, concurrent rollback recovery also allows the system to have more fault tolerance.

In the independent checkpointing approach, each process takes checkpoints independently without any coordination. Different checkpointing instances do not interfere with each other at all, hence the independent checkpointing algorithm allows a system to have any degree of concurrent checkpointing operations without compromising the performance. In rollback recovery, a recovering process has to collect system message flow information to compute for a recovery line and make a single rollback. For concurrent rollback recovery, the collection of message flow information has to be serialized, while making a single rollback by each participant can be done concurrently. Suppose that instances $I_1, I_2, ..., I_{k+1}$ are initiated at the same time by processes $P_1, P_2, ..., P_{k+1}$, respectively, and the priority of $P_i$ is greater than that of $P_{i+1}$, where $1 \leq i \leq k$.

A process with lower priority collects message flow information after process with higher priority does 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 single 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 the synchronous checkpointing algorithm, it is likely to have more than one coordinator at a time. Each coordinator initiates a synchronization instance. Different instances may interfere with each other. 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 $I_1, I_2, \cdots, I_{k+1}$ 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 need 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 the synchronous algorithm 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.

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 5. Elapsed times of concurrent execution at the same site in SA (in milliseconds).

number of processes of each instance: 5
All the five processes are on the same site.
single checkpoint/rollback delay: 251 ms

| | | process of checkpoint instances | process of rollback instances |
|---|---|---|---|
| single instance | $elap_c$ | 583 | 324 |
| | $elap_p$ | 338 | 303 |
| two concurrent instances | $elap_{cp}$ | 665 | 426 |
| | $elap_{pp}$ | 428 | 419 |
| three concurrent instances | $elap_{cpp}$ | 701 | 540 |
| | $elap_{ppp}$ | 463 | 544 |

Table 6. Elapsed times of concurrent execution at different site in SA(in milliseconds).

number of processes of each instance: 5
Each process is on a different site.
single checkpoint/rollback delay: 251 ms

| | | process of checkpoint instances | process of rollback instances |
|---|---|---|---|
| single instance | $elap_c$ | 559 | 301 |
| | $elap_p$ | 322 | 303 |
| two concurrent instances | $elap_{cp}$ | 588 | 354 |
| | $elap_{pp}$ | 360 | 359 |
| three concurrent instances | $elap_{cpp}$ | 617 | 400 |
| | $elap_{ppp}$ | 389 | 405 |

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.

$$elap_{cp} \approx elap_c + elap_p - d$$
$$elap_{pp} \approx elap_p + elap_p - d$$
$$elap_{cpp} \approx elap_{cp} + elap_p - d$$
$$elap_{ppp} \approx elap_{pp} + elap_p - d$$
$d = 251$ ms, which is the delay of taking a single checkpoint or rollback.

Data on the left side of $\approx$ are measured experimentally. The expressions on the right side represent expected values. When a process executes checkpoint operations for two concurrent checkpoint instances, the process makes a single checkpoint instead of two. Therefore, $elap_{cp}$ and $elap_{pp}$ can be expected to be $d$ milliseconds shorter than if the process makes two checkpoints. When a process executes operations for two concurrent rollback instances, the process rolls back once 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 experimental data are even smaller than expected, because processes tend to utilize cpu idle time more efficiently in concurrent processing.

## 5. Overhead of the Algorithm

To evaluate the overhead of these algorithms, we need to measure both cpu time used and message traffic induced by the algorithms. The additional message traffic also degrades the performance of a system since computers have to spend time in message handling routines.

### 5.1. The Synchronous Algorithm

Processes synchronize their checkpoint operations and rollback operations by sending messages. In our algorithm, 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. Message overhead leads to cpu overhead in processing the messages. We study the worst case when the maximum number

of synchronization messages are sent among the processes.

Table 7 shows the maximum number of synchronization messages a process sends and receives 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.

number of processes in the instance: 5
There are one coordinator and four participants.

| | number of messages sent | number of messages received | total |
|---|---|---|---|
| coordinator | 13 | 26 | 39 |
| participant | 8 | 5 | 13 |
| total | $13 + 4 \times 8 = 45$ | $26 + 4 \times 5 = 46$ | |

We have measured the cpu time a process spends during the execution of a single instance and that of concurrent instances. Each instance is executed by the same five processes.

Table 8 shows the cpu costs when all five processes are on a single site. Table 9 shows the cpu costs when each process is on a different site.

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

number of processes of each instance: 5
All the five processes are on the same site.

| | | process of checkpoint instances | process of rollback instances |
|---|---|---|---|
| single instance | $CPU_c$ | 47.0 | 48.8 |
| | $CPU_p$ | 18.3 | 18.6 |
| two concurrent instances | $CPU_{cp}$ | 67.1 | 63.1 |
| | $CPU_{pp}$ | 36.3 | 38.6 |
| three concurrent instances | $CPU_{cpp}$ | 80.9 | 79.9 |
| | $CPU_{ppp}$ | 55.5 | 57.9 |

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

number of processes of each instance: 5
Each process is on a different site.

| | | process of checkpoint instances | process of rollback instances |
|---|---|---|---|
| single instance | $CPU_c$ | 102.0 | 99.1 |
| | $CPU_p$ | 41.6 | 41.0 |
| two concurrent instances | $CPU_{cp}$ | 135.1 | 136.7 |
| | $CPU_{pp}$ | 87.7 | 86.3 |
| three concurrent instances | $CPU_{cpp}$ | 173.0 | 174.9 |
| | $CPU_{ppp}$ | 138.8 | 135.6 |

We have the following observations from these data:

- 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.
- When each process is on a different site, the cpu cost is about 2.2 times that when all five processes are on a single site. This is because remote communication is more expensive than local communication.
- cpu cost contains two components: a) *communication cost*, and b) *computation cost* in executing the algorithm. Communication cost is about 45% of the total cpu cost in a single site environment, and increases to 75% when each process is on a different site. This result is obtained as follows. In our experiment, it costs 1.2 ms cpu time to deliver a message from one process to another through a message queue in Sun UNIX. From Table 7, there are 45 send/receive pairs. Each pair takes 1.2 ms. Therefore, the communication cost is

$$45 \times 1.2 = 54.0 \text{ ms.}$$

From Table 8, the total cpu cost of all five processes in executing a single instance is

$$CPU_c + 4 \times CPU_p = 47.0 + 4 \times 18.3 = 120.2 \text{ ms.}$$

Hence, the computation cost in executing the algorithm is

$$\begin{aligned} &\text{total CPU cost} - \text{communication cost} \\ &= 120.2 - 54.0 = 66.2 \text{ ms.} \end{aligned}$$

The communication cost is 45% of the total cpu cost when all processes are on a single site. When each process is on a different site, the computation cost in executing the algorithm is about the same. But the communication cost will be higher. It can be computed from Table 9 as follows.

$$\begin{aligned} &\text{total CPU cost} - \text{computation cost} \\ &= CPU_c + 4 \times CPU_p - 66.2 \\ &= 102.0 + 4 \times 41.6 - 66.2 \\ &= 202.2 \text{ ms.} \end{aligned}$$

It is about 75% of the total cpu cost.

### 5.2. The Independent Checkpointing Algorithm

In the independent checkpointing algorithm, checkpointing actions incur no message overhead but the rollback recovery has message overhead of $O(2n)$, where $n$ is the number of processes in the instance. The checkpointing instances also incur no extra cpu overhead. The cpu overhead for rollback recovery is an important parameter to measure.

Table 10 shows the cpu time of the coordinator in rollback recovery. In Table 10 we notice that the cpu time increases as the message size increases. The increasing rates of the cpu time is much smaller than that of the message size. The message size depends on the number of checkpoints maintained by each process. From this results, discarding old checkpoints is not very critical in the sense that a larger message size does not significantly incur more cpu overhead. Hence, processes can discard old checkpoints when the processes run out of the storage.

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

number of processes of the rollback instance: 2, 4, 6, 8, 10
All processes are on the same site.
synchronization message sizes: 160, 360, 640, 1000 bytes

| | 2 | 4 | 6 | 8 | 10 |
|---|---|---|---|---|---|
| 160 | 20 | 35 | 50 | 100 | 132 |
| 360 | 17 | 38 | 78 | 130 | 210 |
| 640 | 20 | 40 | 108 | 193 | 278 |
| 1000 | 30 | 90 | 160 | 250 | 376 |

In Table 10, we also observe that the increasing rate of cpu time is much larger than that of the number of processes. However, 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

Since every checkpointing and rollback instance in the synchronous algorithm requires a coordination among processes, it is important to evaluate the synchronous algorithm in different environments that equipped with different communication

facilities. In the environment where processes are distributed in different sites, communication cost is usually more expensive than if the processes are all on the same site. On the other hand, time sharing delay 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 study the execution time of the synchronous algorithm in two different cases: 1) Processes are all on a single site ( experiment was conducted on the mini-RAID system ), and 2) processes are distributed over multiple sites ( experiment was conducted on the RAID system ). We choose 1, 2, 4 as the multiprogramming levels. That means, each site has one process, each site has two processes, and one site has all the four processes respectively. For the rollback recovery, we expect 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.

Tables 11 and 12 show the experiment results of the synchronous algorithm at different multiprogramming levels. Table 11 shows the elapsed times of the coordinator and that of a participant. Table 12 shows the increment of the elapsed times as the multiprogramming level increases.

multiprogramming levels: 1, 2, 4 (# of processes per site)
number of processes in the instance: 4
single checkpoint/rollback delay: 251 ms

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

| $x_i$, $i = 1,2,4$ | | 1 | 2 | 4 |
|---|---|---|---|---|
| checkpoint instance | coordinator | 542 | 587 | 645 |
| | participant | 308 | 344 | 395 |
| rollback instance | coordinator | 299 | 327 | 374 |
| | participant | 291 | 302 | 328 |

Table 12. Increment of elapsed times

| $\Delta x_i = x_i - x_{i/2}$ | | 1 | 2 | 4 |
|---|---|---|---|---|
| checkpoint instance | coordinator | - | +45 | +58 |
| | participant | - | +36 | +51 |
| rollback instance | coordinator | - | +28 | +47 |
| | participant | - | +11 | +26 |

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 replys 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 replys to the coordinator and propagates the checkpoint request only after it has made a checkpoint. From Table 11 and Table 12, 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 elaped time of the coordinator increases faster than that of a participant.

# 7. Comparison of the SA and ICA approaches and Concluding Remarks

The synchronous and independent checkpointing algorithms are similar in that the execution time elapsed in rollback recovery is positively correlated with the number of processes in a system. However, the synchronous algorithm differs from the independent checkpointing algorithm in the following respects: (1) In the synchronous algorithm, it takes less time for processes to roll back when recovering from system failure than it does in the independent checkpointing algorithm, at the cost of large checkpointing overhead. Section 3.1 and 3.2 contain the performance data. (2) The number of checkpoints taken by processes is irrelevent to the performance of rollback recovery in synchronous algorithm, while it slightly affects the performance of recovery in independent checkpointing algorithm as discussed in Section 5.2. (3) The message exchange pattern among processes is irrelevent to the length of rollback distance in the synchronous algorithm, while in the independent checkpointing algorithm, the message exchange pattern has direct impact on the rollback distance. (4) The synchronous algorithm introduces much more message traffic in the system than the independent checkpointing algorithm. (The message overhead is shown in Section 5.1 and 5.2.)

As far as the performance of each algorithm is concerned, it takes more time in synchronizing each checkpointing instance than it does in rollback recovery for the synchronous algorithm as shown in Section 3.1. Therefore, the cost of synchronization would be better paid off if the system has a high rate of failure.

The independent checkpointing algorithm, on the other hand, causes no synchronization overhead in checkpointing actions but takes more time in rollback recovery when system failure occurs. The required rollback distance in the independent checkpointing algorithm is dependent on the message exchange pattern as mentioned above, but is independent of the number of engaged processes. Let $x$ be the probability that each process in the system sends at least one message to every other process during each checkpoint interval. As the value of $x$ increases, the average rollback distance taken by all processes in a recovery increases. The assumption made in the experiment (described in Section 3.2) is a uniform message exchange pattern over all pairs of processes in the system. When a large value is assigned to $x$, it acccounts for a more pessimistic situation than ordinary cases. Generally speaking, 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 the independent checkpointing algorithm. 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

[1] G. Barigazzi and L. Strigini, "Application-transparent setting of recovery points," in *Proc. 13th IEEE Symp. Fault-Tolerant Computing*, Milano, Italy, June 1983.

[2] B. Bhargava and S. Lian, "Independent checkpointing and concurrent rollback recovery for distributed systems - An optimistic approach," in *Proc. 7th Symp. on Reliability in Distributed Systems,* Columbus, OH, Oct. 1988.

[3] B. Bhargava, T. Mueller, and J. Riedl, "Experimental analysis of layered Ethernet software," in *Proc. Fall Joint Computer Conference,* Dallas, Texas, Oct. 1987.

[4] B. Bhargava, P. Noll, and D. Sabo, "An experimental analysis of replicated copy control during site failure and recovery," in *Proc. 4th International Conference on Data Engineering,* Los Angeles, CA, Feb. 1988.

[5] B. Bhargava and J. Riedl, "The RAID distributed database system," *IEEE Trans. Software Eng. SE,* June 1989, 726-736.

[6] A. Ciuffoletti, "Error recovery in systems of communicating processes", *Proc. 7th International Conference on Software Engineering*, March 1984.

[7] K.H. Kim, "Approaches to mechanization of the conversation scheme based on monitor," *IEEE Trans. Software Eng., SE-8*, May 1982, 189-197.

[8] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems," *IEEE Trans. Software Eng. SE-13*, 1(Jan. 1987), 23-31.

[9] P. Leu and B. Bhargava, "Concurrent robust checkpointing and recovery in distributed systems," in *Proc. 4th IEEE Int. Conf. Data Engineering*, Los Angeles, CA, Feb. 1988.

[10] J. E. Moss, 'Checkpoint and restart in distributed transaction systems," in *Proc. 3rd IEEE Symp. on Reliability in Distributed Software and Database Syst.*, July 1983.

[11] R. E. Strom and S. Yemini, "Optimistic Recovery in Distributed Systems," *ACM Trans. on Computer Systems 3,* 3(Aug. 1985), 204-226.

[12] Y. Tamir and C. H. Séquin, "Error recovery in multicomputers using global checkpoints," in *Proc. 13th IEEE Int. Conf. Parallel Processing*, Aug. 1984.

[13] W. G. Wood, "A decentralized recovery control protocol", *Proc. 11th IEEE Symp. on Fault-Tolerant Computing*, June 1981.