THE IMPACT OF LAZINESS ON THE PERFORMANCE OF SNAPSHOT ALGORITHMS

Zhijun Liu and Paolo A.G. Sivilotti
Department of Computer Science and Engineering
The Ohio State University
2015 Neil Ave
Columbus, OH 43210-1277, USA
email: {liuzh,paolo}@cse.ohio-state.edu

ABSTRACT
A snapshot algorithm gathers global state in a distributed system. Co-ordinated snapshot algorithms, such as the Chandy-Lamport algorithm, use control messages to ensure consistency of the gathered state. "Lazy snapshots" represent a generalization of Chandy-Lamport in which processes can delay recording local state and can anticipate or delay propagating control messages. In this paper, we present a new taxonomy of the class of lazy snapshot algorithms. This taxonomy precisely characterizes each variant within this class and forms a framework for a systematic investigation of the performance of each algorithmic variant. Through simulation and analysis, we quantify the tradeoff between flexibility and storage cost. We find that eagerly sending markers before recording the local state always reduces the amount of storage required to save global state. Furthermore, this approach allows processes modest flexibility in deciding when to record the local state. Conversely, lazily recording the local state provides significant flexibility but can increase the amount of storage needed. In light of this tradeoff, we present a new hybrid algorithm that blends these strengths.

KEY WORDS
distributed algorithms, snapshots, checkpointing

1 Introduction

The collection of global state is a problem of fundamental importance in asynchronous distributed systems. Since processes do not share state, each is responsible for recording an image of its own, local, state. A collection of local images forms a valid snapshot if and only if it is *consistent*: Every message recorded as having been received is also recorded as having been sent [1]. Snapshots have a variety of applications, including: stable predicate detection [7, 2], distributed debugging [4], and fault tolerance through roll-back recovery [6].

The importance of this problem has led to the development of many snapshot algorithms [3]. The algorithms fall in two broad categories: coordinated and uncoordinated. The former approach is conservative: Every image is part of a valid snapshot. The latter approach is optimistic: Processes decide independently when to take local images and valid snapshots are later formed from the available images. While the former requires fewer images to be recorded, the latter allows the processes more flexibility in scheduling when to take these images.

In earlier work [8], the classic coordinated snapshot algorithm of Chandy and Lamport [1] was generalized to a class of algorithms termed "lazy snapshot algorithms". Laziness permits a process to delay its recording of the local state without violating the conditions needed for guaranteeing that the resulting snapshot is valid. This previous work hypothesizes that laziness, while enjoying all the benefits of coordinated strategies, might also have two additional advantages: (i) lazy snapshots could include fewer messages in transit, and (ii) lazy snapshots could afford greater flexibility to processes in when to record their local state. The former is a benefit in *storage* and the latter is a benefit in *flexibility*.

In this paper, we quantify the extent of these benefits for lazy snapshots. We form a taxonomy of the algorithm variants that satisfy the lazy snapshot generalization. This taxonomy is generated from two degrees of freedom: laziness in recording the local state, and laziness in sending control messages. We systematically investigate the impact of each of these degrees of freedom through a collection of simulation studies.

Our results indicate that laziness does indeed provide significant improvements in flexibility. These improvements, however, may come at the cost of storage. The extent to which laziness increases storage depends on the topology and message traffic of the underlying computation. We describe a new, hybrid algorithm that blends the advantages of both approaches to reduce storage costs while maintaining some flexibility. In this hybrid algorithm, processes dynamically decide how lazy to be, based on locally observed message traffic.

2 Background

2.1 The System Model

A distributed system consists of a finite set of processes P that communicate by message passing. Communication is

Marker-Sending Rule for a Process p. For each outgoing channel C:
p sends one marker along C after p records its state and before p sends further messages along the same channel.

Marker-Receiving Rule for a Process q. On receiving a marker along a channel C:
if q has not recorded its state **then**
  begin q records its state;
    q records the state of C as the empty sequence
  end
else q records the state of C as the sequence of messages received along C after q's state was recorded and before q received the marker along C.

Figure 1. Chandy-Lamport algorithm

point-to-point, asynchronous, FIFO, and failure-free. The variables p and q are understood to range over P. The channel from p to q is denoted C_{pq}.

The execution of a process is a sequence of events. There are three kinds of events: local, send, and receive. A message is said to be *in transit* when it has been sent by the source process but has not been received by the destination process. At any given moment, the state of a channel is the set of messages that are in transit in that channel.

The *global state* of the system is defined as the union of local states of processes and the states of the channels (i.e., the messages in transit). A global state is *consistent* when every message that is recorded as received in the local state of the receiver has also been recorded as sent in the local state of the sender. Consistent global states are meaningful snapshots as they represent a system state that *may* have occurred during the computation.

2.2 The Chandy-Lamport Algorithm

This algorithm is a coordinated snapshot algorithm where markers are used as control messages to trigger the recording of the local state by other processes. Since channels are FIFO, markers serve to separate the messages that were sent before a local snapshot (*recorded* sends) from those that were sent after (*unrecorded* sends). By recording its local state on the receipt of its first marker, a process guarantees that every message it records as received came from a recorded send action, and hence the recorded global state is consistent. The outline of the algorithm is presented in Figure 1. [1]

2.3 Notation

We follow the notation used in the original presentation of lazy snapshots [8]. A channel on which a marker has been received is termed a *dirty channel*. The times at which events related to recording local state occur are given by:

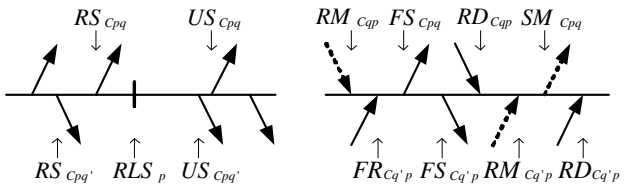

Figure 2. Definitions of Events

RLS_p**:** p records its local state.

$US_{C_{pq}}$**:** p sends a message on channel C_{pq} *after* recording the local state (the first "unrecorded send").

$RS_{C_{pq}}$**:** p sends a message on channel C_{pq} *before* recording the local state (the last "recorded send").

The times at which events related to marker propagation occur are given by:

$RM_{C_{pq}}$**:** q receives a marker on channel C_{pq}.

$SM_{C_{pq}}$**:** p sends a marker on channel C_{pq}.

$RD_{C_{pq}}$**:** q receives a message on a dirty channel C_{pq}.

$FR_{C_{pq}}$**:** q receives the first message on channel C_{pq} *after* having received some marker (the "first receive").

$FS_{C_{pq}}$**:** p sends the first message on channel C_{pq} *after* having received some marker (the "first send").

These terms are illustrated in Figure 2, where the solid vertical line indicates the recording of local state, solid arrows are application messages, and dashed lines are marker messages.

From these definitions, it follows that:

$$\begin{aligned} (\,\mathbf{Max}\, q \,::\, RS_{C_{pq}}\,) &< RLS_p \\ RLS_p &< (\,\mathbf{Min}\, q \,::\, US_{C_{pq}}\,) \\ RM_{C_{pq}} &< RD_{C_{pq}} \\ FR_{C_{pq}} &\leq RD_{C_{pq}} \\ (\,\mathbf{Min}\, q' \,::\, RM_{C_{q'p}}\,) &< \mathbf{min}(FS_{C_{pq}}, FR_{C_{qp}})\ . \end{aligned}$$

With this notation, the Chandy-Lamport algorithm can be characterized by the following two inequalities:

E1. $(\forall p \,::\, RLS_p \leq (\,\mathbf{Min}\, q \,::\, RM_{C_{qp}}\,))$

E2. $(\forall p, q \,::\, RLS_p < SM_{C_{pq}} < US_{C_{pq}}\,)$.

The first inequality controls when the local state is recorded: The local state is recorded no later than the receipt of the first marker. Any process that records its state before receiving the first marker is an *initiator*. The second inequality controls when markers are sent: A marker is sent on a channel after recording the local state and before the first unrecorded send on that channel.

3 Lazy Snapshots

Lazy snapshots [8] are a generalization of the Chandy-Lamport algorithm. They are motivated by the observation that requiring a process to record local state immediately when receiving a marker is too strict: The receiving process only needs to guarantee that it does not record an orphan message. Similarly, requiring a process to send a marker only after recording local state (and before its first unrecorded send) is also too strict: The sending process only needs to guarantee that a marker is sent between the last recorded send and the first unrecorded send.

Thus, the family of lazy snapshot algorithms are characterized by the following pair of inequalities:

L1. $(\forall p :: RLS_p < (\mathbf{Min}\, q :: RD_{C_{qp}}))$

L2. $(\forall p, q :: RS_{C_{pq}} < SM_{C_{pq}} < US_{C_{pq}})$.

Since **E1** and **E2** are strictly stronger than **L1** and **L2**, lazy snapshots define a strictly larger family of algorithms.

4 A Taxonomy of Laziness

The inequalities **L1** and **L2** each control different dimensions of laziness. The first defines the interval in which the local state can be recorded, while the latter defines the interval in which markers can be sent. A particular algorithm is instantiated from this class by deciding when, within these intervals, the events occur.

For recording local state, there are four natural divisions within the interval **L1**. In increasing degree of laziness, a process p can record local state:

I: When the first marker is received.
$RLS_p \leq (\mathbf{Min}\, q :: RM_{C_{qp}})$

II: Before the first send or receive.
$RLS_p < (\mathbf{Min}\, q :: \mathbf{min}\ (FS_{C_{pq}}, FR_{C_{qp}}))$

III: Before the first send or dirty receive.
$RLS_p < (\mathbf{Min}\, q :: \mathbf{min}\ (FS_{C_{pq}}, RD_{C_{qp}}))$

IV: Before the first dirty receive.
$RLS_p < (\mathbf{Min}\, q :: RD_{C_{qp}})$

For propagating markers, there are three natural divisions within the interval **L2**. In increasing degree of laziness, a process p can send a marker:

A: When the first marker is received.
$(\forall q :: SM_{C_{pq}} = (\mathbf{Min}\, q' :: RM_{C_{q'p}}))$
$\wedge\ (\forall q :: RS_{C_{pq}} < SM_{C_{pq}} \leq RLS_p)$

B: When the local state is recorded.
$(\forall q :: SM_{C_{pq}} = RLS_p)$

C: Before the first unrecorded send.
$(\forall q :: RLS_p < SM_{C_{pq}} < US_{C_{pq}})$

A particular snapshot algorithm within this family is instantiated by selecting a degree of laziness for recording local state ($I-IV$) and a degree of laziness for sending markers ($A-C$). Thus, the cross-product of these two dimensions provides a taxonomy for this family of algorithms. This taxonomy is summarized in Figure 3, where each cell illustrates a characteristic execution for a particular variant.

It is important to note that the two dimensions are not truly independent. The first dimension (when to record local state) is defined only in terms of application messages and the reception of markers. The second dimension (when to send a marker), however, is defined relative to the first (i.e., when local state is recorded). An example of this interdependence is the equivalence of AI and BI. In the former, markers are sent when the first marker is received, while in the latter, markers are sent when local state is recorded. For column I, however, these two times are the same. Hence $AI = BI$ (and we refer to this variant as ABI).

Another example of this interdependence is AIV. This algorithm has maximum laziness in recording local state: local state is recorded just before the first dirty receive. On the other hand, this algorithm has minimum laziness in propagating markers: a marker is sent on a channel immediately after receiving the first marker and after the last recorded send. These two requirements are inconsistent in executions for which there is a send after receiving the first marker but before the first dirty receive (see cell AIV in Figure 3.) This algorithm can not be implemented without prescience since deciding whether or not to propagate a marker immediately depends on what future communication actions will occur. Absent this information about the future, an implementation must either anticipate recording local state (behaving like $AIII$) or postponing propagating markers (behaving like BIV).

The proofs of correctness for algorithms within this taxonomy can be found in [5].

5 Evaluation

We evaluate the performance of these algorithms with respect to two metrics: *storage* and *flexibility*.

Storage measures the amount of space required to store a gathered snapshot. We assume that, for a fixed number of processes, the amount of space required for local state is constant. The storage metric, therefore, measures only the space required to measure channel state—that is, the number of messages in transit.

Flexibility measures the amount of independence afforded a process in deciding when to record local state. Informally, the longer a process can defer recording local state, the more flexibility it has. In order to compare flexibility across different topologies and different application message traffic patterns, we normalize the time the process defers recording local state to the average time between send events for that process. Thus, a flexibility mea-

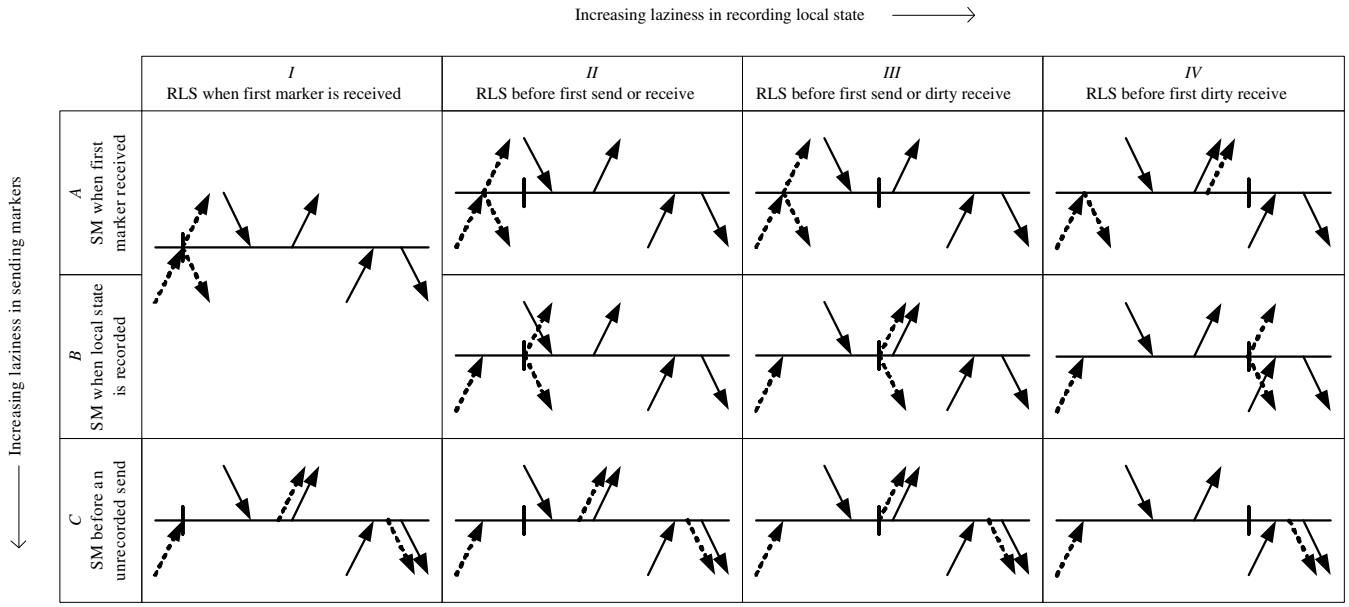

Figure 3. Lazy Snapshot Algorithm Variants

sure of f reflects that processes can expect to perform f send operations between receiving the first marker and being obliged to record local state.

Our evaluation is empirical. We implement and simulate each algorithm variant[1] using the Anylogic simulation package [9]. Each simulation run consists of 30 processes and a randomly generated (connected) topology of FIFO, bidirectional channels. Each process p is associated with a mean send interval, μ_p, and application message traffic is generated on each of p's outgoing channels using a negative exponential distribution, with mean μ_p. Message delay is constant.

We characterize each simulation run with two parameters: connectivity (the total number of channels), and message traffic density (the average number of messages in transit per channel). For any one run, we evaluate all the variants using the same underlying application message traffic pattern.

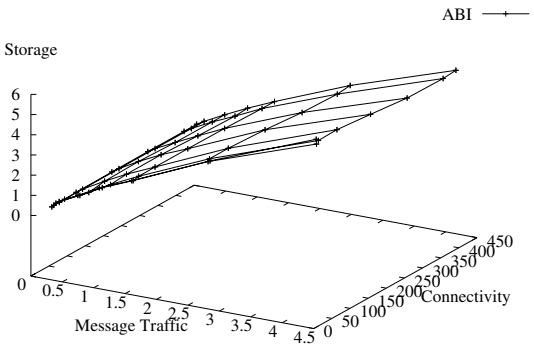

Figure 4. Storage: Algorithm ABI

5.1 Baseline: Chandy-Lamport

As a baseline performance measurement for algorithm variants within our topology, we examine the Chandy-Lamport algorithm with eager propagation of markers (i.e., ABI). Since the flexibility of algorithms in column I is by definition 0, we measure only the storage cost. Figure 4 shows the storage cost (messages recorded as in transit, per channel) as a function of message traffic density and connectivity.

As expected, the storage cost for algorithm ABI increases with increased message traffic density. This cost is constant, however, with respect to connectivity. Since this algorithm has the least laziness in each dimension, it represents a baseline for performance comparisons.

5.2 Laziness in Marker Propagation

Although the storage complexity of ABI is independent of topology, this is not the case for algorithms in which marker propagation is lazy (i.e., rows B and C). For these variants, marker propagation is influenced by the un-

[1] Algorithm AIV is unimplementable without prescience and so was not included in the simulation study. Also, algorithms AI and BI are equivalent, as discussed in Section 4, and so a single implementation was used.

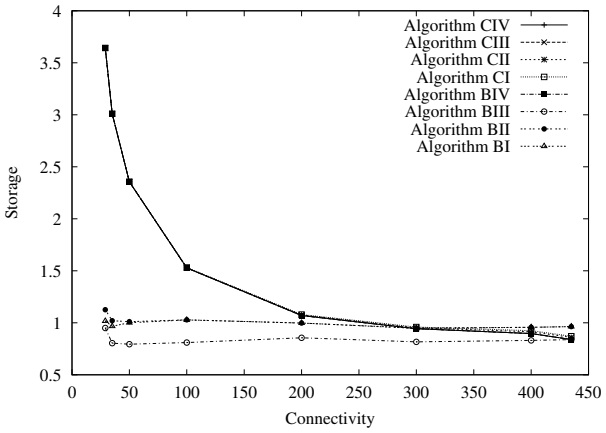

Figure 5. Storage vs Connectivity: Rows B and C

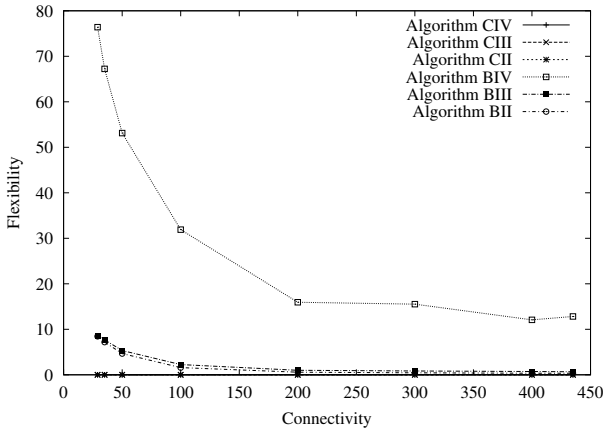

Figure 6. Flexibility vs. Connectivity: Rows B and C

derlying application message pattern and is therefore sensitive to topology. For these two rows, we examine the relationship between topology and performance (i.e., storage and flexibility). We use a fixed density of traffic (0.6 messages per channel).

First, we observe that maximum laziness in marker propagation results in every marker being immediately followed by an (unrecorded) send. Since receiving a dirty message always prompts the recording of local state, all variants in row C have essentially the same pattern of local state recording. Thus, the storage complexity of each variant in this row is the same. Furthermore, we observe that deferring recording local state until the first communication action (i.e., the difference between column I and II) should not affect the number of messages recorded as in transit. Therefore, we expect BI to have the same storage cost as BII. Finally, for algorithms BIV and CIV, the additional delay in propagating markers does not change the time at which processes record local state. Since processes record local state at the same time for these two algorithms, they also record the same messages as being in transit. Hence, we expect BIV to have the same storage cost as CIV (and therefore the same storage cost as all the row C algorithms).

All of these hypotheses are confirmed in Figure 5. Only 3 distinct lines are visible since the storage complexity of BIV is the same as each variant in row C, and BI is the same as BII.

This figure also reveals the cost of propagating markers lazily: algorithms from row C have high storage complexity, in particular for sparse topologies. The reason for this cost is that delaying the propagation of markers increases the time it takes to receive a return marker from a neighbor. In the meanwhile, messages received from that neighbor are likely to be recorded as in transit. This effect is pronounced in sparse topologies where the neighbor is unlikely to receive a marker from any other process.

As for flexibility, it is clear that algorithms in column I have none. In addition, we expect no flexibility for algorithms in row C since algorithms in this category propagate markers only immediately before an unrecorded send, hence prompting an immediate recording of state at the destination. This leaves BII, $BIII$ and BIV as candidates for high flexibility. Since the amount of deferral is greatest in column IV we expect this variant to have the greatest flexibility. Figure 6 confirms that algorithm BIV enjoys high flexibility.

5.3 A Hybrid Algorithm for Improved Performance

From the previous section, we observe that BIV has the greatest flexibility, but the worst storage cost. The reason for this is that delaying recording local state increases both the number of recorded receives *and* the number of recorded sends. While the former can represent an improvement in storage cost (fewer messages in transit), the latter can represent a worsening in storage cost (more messages in transit).

To reap the flexibility advantage of BIV as well as the storage advantage of being eager (BI) we design a hybrid algorithm that blends both. The hybrid algorithm is lazy in recording local state when it expects to be a net *consumer* of messages (i.e., receiving more than it sends), otherwise it records its local state immediately. The judgement of whether it is a net producer or consumer of messages is based on past observations of message frequency so is only an estimate of future behavior.

Recall that process p sends messages on each outgoing channel at an average rate of $1/\mu_p$. Let N_p be the set of p's neighbors and let D_p be the set of neighbors from which p has received a marker. As a result of delaying recording local state, the expected rate of new recorded sends is $\sum_{q \in N_p} 1/\mu_p$. On the other hand, the expected rate of new recorded receives is $\sum_{q \in N_p \setminus D_p} 1/\mu_q$, since

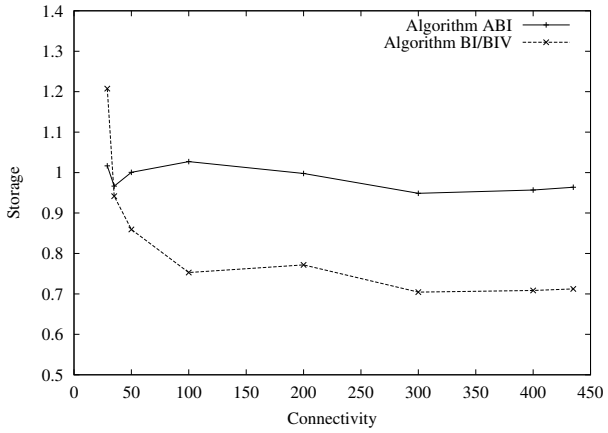

Figure 7. Storage: Algorithm BI/BIV

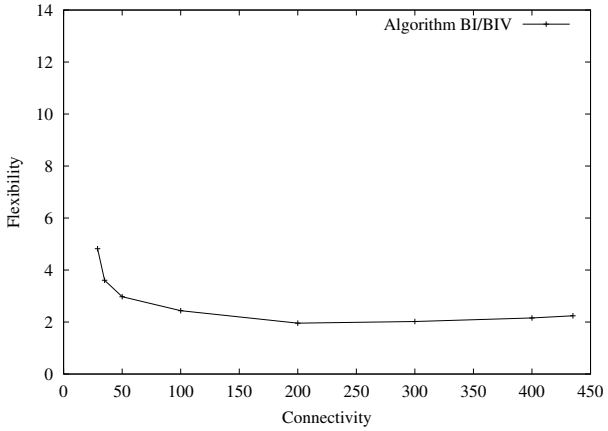

Figure 8. Flexibility: Algorithm BI/BIV

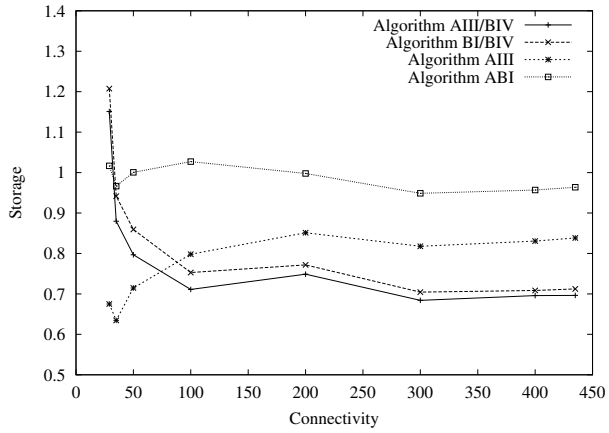

Figure 9. Storage: Row A

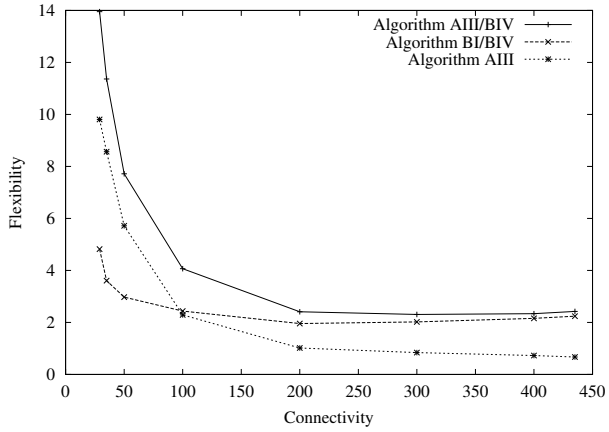

Figure 10. Flexibility: Row A

dirty channels cannot contribute recorded receives. Every time p receives a marker, it re-assesses the likelyhood that it is a net consumer by comparing these two rates. When the rate of sending exceeds the rate of receiving, it records local state.

The performance of this hybrid algorithm (BI/BIV), in comparison to the Chandy-Lamport algorithm (ABI) is shown in Figures 7 and 8. (Recall that the flexibility of ABI is, by definition, 0.) These figures show an improvement in both storage cost and flexibility.

5.4 Eagerness of Marker Propagation

The algorithms discussed above all propagate markers after recording local state. Our taxonomy, however, includes the possibility of eagerly sending out markers *before* recording local state (i.e., row A). Algorithm AI is Chandy-Lamport, while algorithm AIV is unimplementable as discussed above. This leaves algorithms AII and $AIII$ for analysis.

First note that these two algorithms have identical marker propagation patterns. Algorithm $AIII$, however, allows greater delay in recording local state and therefore has greater flexibility than AII . Furthermore, since every unrecorded receive in $AIII$ is also an unrecorded receive in AII , the storage cost of the former is less than or equal to that of the later. Hence, we only evaluate algorithm $AIII$.

Compared with our baseline (ABI), we expect $AIII$ to have trivially more flexibility. We also expect $AIII$ to have better storage cost since this algorithm has the same marker propagation pattern as ABI but delays recording local state past some receives. It does not change the recorded sends, but increases the number of recorded receives. These hypotheses are confirmed in Figures 9 and 10.

Given that $AIII$ dominates ABI [2] we adapted our

[2]Indeed, $AIII$ compares favorably to *all* algorithms in row B , ex-

hybrid algorithm to combine $AIII/BIV$ (rather than ABI/BIV). The adaptive decision-making process of this new hybrid is the same: whenever a marker is received a new estimate is calculated of whether the process will be a net consumer of messages. If it will not be a net consumer, the process broadcasts markers immediately, but delays recording local state until there is an outgoing message for any channel or an incoming message from the dirty channel. As the Figures 9 and 10 show, the improved hybrid algorithm is always better than the old one in both storage and flexibility.

Among all these algorithms, when the connectivity is low, algorithm $AIII$ has the best storage. However, for high connectivity network, hybrid algorithm $AIII/BIV$ offers the best storage. Regardless of connectivity, hybrid algorithm $AIII/BIV$ has the best flexibility.

6 Related work

A survey of rollback-recovery techniques is available in [3]. Techniques can be categorized as uncoordinated checkpointing, coordinated checkpointing, and communication-induced checkpointing. Under this decomposition, the lazy snapshot variants discussed here are all nonblocking coordinated checkpoint protocols. Work on uncoordinated checkpointing has focussed on minimizing the domino effect, while work on coordinated checkpointed has focussed on minimize the communication and control overhead. Lazy snapshots, although clearly a conservative, coordinated approach, have some of the advantages of uncoordinated strategies in so far as they do permit some flexibility as well as reducing the storage costs.

7 Concluding remarks

Lazy snapshots are a generalization of the Chandy-Lamport coordinated snapshots algorithm. This generalization loosens the requirements on when processes must record the local state and when markers must be sent. We have described a taxonomy of lazy snapshot algorithm variants. This taxonomy is organized around two orthogonal dimensions of laziness: when a process records the local state, and when a process sends out markers. Within this framework, we conducted a methodical investigation of the performance implications for each variant.

Our results indicate that eagerly sending markers always reduces the amount of storage required to save channel state as well as affording the process some flexibility in scheduling the potentially expensive task of recording the local state. On the other hand, lazily recording the local state dramatically increases the amount of flexibility. One might hope that this delay would also reduce the number of messages in transit, since it reduces the number unrecorded receives (from which messages in transit are inferred). Unfortunately, while a delay in recording the local state at process p decreases the number of messages in transit *into* p (as unrecorded receives become recorded receives), it also increases the number of messages in transit *out of* p (as unrecorded sends become recorded sends). The net result is that laziness in recording local state provides significant flexibility at the cost of a some increase in storage.

With these relative strengths in mind, we have designed a hybrid algorithm that combines eager and lazy recording of the local state. A process decides dynamically, based on locally observable network traffic, whether delaying its recording of the local state will be beneficial. This hybrid approach blends the strengths of the two base algorithms, yielding benefits in both storage and flexibility.

cept for the flexibility acheived by BIV .

References

[1] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, February 1985.

[2] R. Cooper and K. Marzullo. Consistent detection of global predicates. *Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging, published in ACM SIGPLAN Notices*, 26(12):167–174, 1991.

[3] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, 2002.

[4] E. Fromentin, N. Plouzeau, and M. Raynal. An introduction to the analysis and debug of distributed computations. *1st International Conference on Algorithms and Architectures for Parallel Processing*, pages 545–553, May 1995.

[5] Zhijun Liu and Paolo A. G. Sivilotti. A Taxonomy of Laziness in Snapshot Algorithms. Technical Report OSU-CISRC-9/05-TR59, The Ohio State University, September 2005.

[6] R. Koo and S. Toueg. Checkpointing and rollback-recovery for distributed systems. *IEEE Transactions on Software Engineering*, 13(1):23–31, 1987.

[7] A. D. Kshemkalyani and M. Singhal. Efficient detection and resolution of generalized distributed deadlocks. *IEEE Transactions on Software Engineering*, 20(1):43–54, 1994.

[8] N. Sridhar and P. A. G. Sivilotti. Lazy snapshots. In *Proceedings of the 14th IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 96–101, 2002.

[9] XJ Technologies Company Ltd., http://www.xjtek.com. *AnyLogic V User's Manual*, 2004.