July 1998

UILU-ENG-98-2218
CRHC-98-09

University of Illinois at Urbana-Champaign

Fast Distributed Simulation for Dependability Analysis of a Cache-based RAID System

Yiqing Huang, Zbigniew Kalbarczyk, and Ravi K. Iyer

Coordinated Science Laboratory
1308 West Main Street, Urbana, IL 61801

REPORT DOCUMENTATION PAGE

Form Approved OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE July 1998	3. REPORT TYPE AND DATES COVERED

4. TITLE AND SUBTITLE Fast Distributed Simulation for Dependability Analysis of a Cache-based RAID System	5. FUNDING NUMBERS DABT63-94-C-0045 NASA NAG 1-613 TANDEM
6. AUTHOR(S) Y. Huang, Z. Kalbarczyk, and R. K. Iyer	
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(ES) Coordinated Science Lab University of Illinois 1308 W. Main St. Urbana, IL 61801	8. PERFORMING ORGANIZATION REPORT NUMBER UILU-ENG-98-2218 (CRHC-98-09)
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) NASA Langley Research Center Hampton, VA 23681 Tandem Computers 19333 Vallco Parkway Cupertino, CA 95014 DARPA/ITO 3701 N. Fairfax Dr., Arlington, VA 22203-1714	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12 b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

In this paper, we propose a new speculation-based, distributed simulation method for fast, yet accurate, simulation-based dependability analysis of complex systems in which a detailed functional simulation of a system component is essential to obtain an accurate overall result. Our target example is a networked cluster with compute nodes and one I/O node. Accurate system dependability characterization is achieved via a combination of detailed simulation of I/O subsystem behavior in the presence of faults and more abstract simulation of the compute nodes and the switching network. Dependability measures such as error coverage and error detection latency and performance measures such as delivery time in the presence of faults are obtained. The approach is implemented on a network of workstations, and experimental results show significant improvements over a Time Warp simulator for the same model.

14. SUBJECT TERMS detailed simulation, speculation, error correction and recovery	15. NUMBER IF PAGES 25
	16. PRICE CODE

17. SECURITY CLASSIFICATION OR REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Fast Distributed Simulation for Dependability Analysis of a Cache-based RAID System

Yiqing Huang, Zbigniew Kalbarczyk, and Ravi K. Iyer
Center for Reliable and High Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 W. Main, Urbana, IL 61801 USA
Phone: 217-244-8288 Fax: 217-244-5686
`yhuang|kalbar|iyer@crhc.uiuc.edu`

July 24, 1998

Abstract

Often detailed simulation of a subsystem is important to understand overall system/network reliability. However, detailed simulation is time-consuming, e.g., simulation of the overlapping error recovery mechanisms of the cluster system presented in this paper. We propose a new speculation-based, distributed simulation method which is applicable to situations where a detailed functional simulation of a system component is essential to obtain an accurate overall result. The proposed method is characterized by the following features: 1) The traffic to the subsystem with detailed simulation is infrequent. 2) The speculation can most often successfully predict the outcome of the detailed simulation. Hence, the detailed simulation can be postponed until an idle time slot is found during which the detailed simulation can be scheduled. The dependability analysis presents an ideal application for this approach. Our target example is a networked cluster with compute nodes and one I/O node. Accurate system dependability characterization is realized via a combination of detailed simulation of the I/O subsystem behavior in the presence of faults and more abstract simulation of the compute nodes and the switching network. The speculation, in this case, is based on the observation that errors are most likely detected and corrected in the I/O subsystem. Therefore, if traffic to the detailed error recovery simulation process is relatively infrequent, the detailed simulation can take place during the process idle time slots. In case the detailed simulation indicates a failure, a rollback is necessary. Dependability measures like error coverage, error detection latency and performance measures such as delivery time in the presence of faults are obtained. The approach is implemented on a network of workstations, and experimental results show significant improvements over a Time Warp simulator for the same model.

Keywords: detailed simulation, speculation, error correction and recovery

1 Introduction

As the complexity and size of computer design increases, dependability analysis becomes increasingly difficult and important. Often detailed simulation of a subsystem is important to understand overall system/network reliability, since it captures the changing system dynamics due to variability of the workload or variety of failure scenarios (including single and bursty faults). In addition, application dependent measures like error detection latency can often be obtained accurately via detailed simulation. However, detailed simulation is time-consuming, e.g., simulation of the overlapping error recovery mechanisms of the cluster system presented in this paper. We propose a new speculation-based, distributed simulation method which is applicable to situations where a detailed functional simulation of a system component is essential to obtain an accurate overall result. The proposed method is characterized by the following features:

- The traffic to the subsystem with detailed simulation is infrequent.
- The speculation can most often successfully predict the outcome of the detailed simulation. Hence, the detailed simulation is postponed until an idle time slot is found during which the detailed simulation can be scheduled.

The dependability analysis presents an ideal application for this approach. In our target example of a networked cluster with an I/O node, the traffic to the I/O node is infrequent, the disk error detection/correction must be simulated in detail to obtain accurate dependability measures, and the chance of recovery from errors is high. Accurate system dependability characterization is realized via a combination of detailed simulation of the I/O subsystem behavior in the presence of faults and more abstract simulation of the compute nodes and the switching network. The speculation, in this case, is based on the observation that errors are most likely detected and corrected in the I/O subsystem, [3, 15, 21]. In other words, we can most often assume that the system recovers from errors. Therefore, if traffic to the detailed simulation process is relatively infrequent, the detailed simulation can take place during the process idle time slots. In case the detailed simulation indicates a failure, a rollback is necessary.

The simulated system is a reliable high-performance cluster of compute nodes and an I/O node connected via ServerNet[13], a system area network by Tandem. The I/O node supports a new generation, cache-based RAID storage system which is critical to the availability and performance of the cluster. The RAID system is controlled and managed by the array controller which includes a cache subsystem. The cache subsystem employs several layers of overlapping error detection and recovery mechanisms to provide high coverage against operational errors occurring in the cache and in the disk. To analyze the impact of the storage subsystem on the cluster dependability, we developed a detailed model of the RAID controller cache subsystem to capture interrelationships and interactions among different error detection and recovery mechanisms, including Cyclic Redundancy Checking (CRC), parity checking and Error Detection and Correction (EDAC). This detailed model is incorporated into the overall cluster model and is simulated while faults are injected into the cache subsystem. The compute node model focuses on a high level simulation of its local disk server and packet transmission interfacing the switch.

The primary dependability measures obtained from the cluster system simulation include error coverage of EDAC code and error detection latency distribution of errors introduced to the cache components. The error coverage is high, and consequently it indicates the validity of the speculation. The latency of errors injected into different parts of the cache system varies by as much as 10^6 ms. Efficiency of the overlapping detection/correction mechanisms is demonstrated as well. Fault injection impact on the performance is studied via the performance measures. Performance measures derived include a distribution of 2-way delivery time and the mean of 2-way delivery time (2-way delivery time is the time between a request is generated by a compute node and the corresponding response is received) in the presence of faults injected into the cache subsystem.

The proposed approach is implemented on top of Time Warp [14]. Time-consuming detailed simulation of dependability analysis is "hidden" efficiently via speculation. For our study of the cluster system, we observed as much as 200% performance improvement over Time Warp. In addition, 4.6 times speedup over sequential simulation on six workstations was demonstrated.

The rest of the paper is organized as follows. Related work are briefly reviewed in Section 2. Section 3 describes the architecture of the cluster system. Section 4 presents the simulation model design. Section 5 illustrates the benefits of speculation in the simulation. The speculation-based approach is explained in Section 6 in the context of the simulated architecture. Performance results are shown in Section 7. Section 8 concludes the paper.

2 Related Work

To achieve fast and accurate simulation for dependability analysis of complex systems, several important issues need to be addressed: 1) accurate characterization of hardware and software faults, 2) adequate representation of error detection and recovery, 3) simulation time acceleration. Addressing the first two issues, system behavior in the presence of faults can be evaluated. The third issue reflects the fact that a detailed simulation can easily exceed the execution time tolerable by computing resources. As a result, it is necessary to develop approaches to achieve a significant speedup in simulation time while providing high accuracy results.

Many methodologies have been proposed to support fast simulation for dependability analysis. Hierarchical simulation and hybrid simulation are two ways to speed up simulation. In hierarchical simulation, a system is decomposed into several abstraction levels, and a corresponding simulation model is associated with each level. Submodels are analyzed individually and results from a lower level are used to feed the higher-level simulator. Therefore, the overall system behavior can be analyzed with great details and within a reasonable amount of time. A good example using this technique is a hierarchical approach for dependability analysis of a commercial cached RAID storage architecture presented in [16]. Three levels have been developed to model the RAID cache subsystem, cache operations and error detection/correction mechanisms. For each level of the hierarchy, the impact of faults and errors occurring in the cache and in the disks are modeled.

With hybrid simulation, a combination of techniques are used to simulate systems. In DEPEND

[8], functional simulation can be combined with Monte Carlo simulation [10, 17] or with a Markov or Semi-Markov model to provide simulation speedup. The most prominent example of the use of hybrid techniques in reliability analysis is the HARP tool[2]. HARP decomposes a model into a fault occurrence/repair model (FORM) and one or more fault/error-handling models (FEHM). The FEHMs are simulated with an extended stochastic Petri Net (ESPN) to obtain instantaneous coverage probabilities. These probabilities are then automatically incorporated into the FORM model, represented by a continuous-time markov chain, and solved to obtain system reliability measures.

To speed up simulation involving rare events, techniques such as importance sampling are developed. Importance sampling is a variance reduction technique which has been used successfully to deal with the problems of rare events. A good survey of fast simulation results using importance sampling techniques for dependability models is reported in [12]. The applicability of importance sampling is extended to non-Markovian models with general failure and repair time distributions in [19]. A significant speedup can be achieved using importance sampling. However, identification of a sampling distribution to ensure small variance is difficult and must be resolved for different applications.

In recent years, distributed simulation is used widely for simulating complex systems. With distributed event-driven simulation, a simulated application is modeled by a group of logical processes(LPs) and the LPs are mapped to multiple physical processes to conduct the simulation in parallel. Fujimoto surveyed existing distributed simulation approaches and analyzed the advantages and drawbacks of various techniques [6]. Distributed simulators developed based on conventional protocol such as Time Warp are mostly for performance studies [9, 11, 18, 22]. For example, a distributed simulator wrote in Maisie, a parallel discrete-event simulation language, is developed to simulate a two-level metropolitan area network using wormhole routing [1]. Discrete-byte rather than discrete-packet simulation is performed with acceptable performance provided by the distributed simulator. Many research has been focusing on developing new protocols to improve distributed simulation performance [4, 5, 7].

The simulation technique proposed in this paper employs dynamic speculation-based rescheduling of events at run-time. The method is applicable to distributed simulation, and works when the traffic goes to the logical processes with detailed simulation is relatively infrequent (e.g. simulation of highly dependable systems.). The efficiency in detailed complex system simulation is achieved by overlapping detailed critical subsystem simulation with more abstract simulation of noncritical components.

3 System Architecture

The architecture analyzed in this paper is a reliable high-performance cluster system, as shown in Fig. 1. The system consists of five compute nodes and one I/O node connected via a six-port ServerNet switch. A compute node can access data from its local disk and remote data stored on the I/O node. The I/O node operates as a data server for files shared among the compute nodes in the cluster. As the data on the I/O node is shared among all the compute nodes, the availability of the I/O node is crucial to the overall cluster dependability and performance. In order to provide high performance and high availability, the I/O node supports RAID architecture, which consists of a collection of disks drives

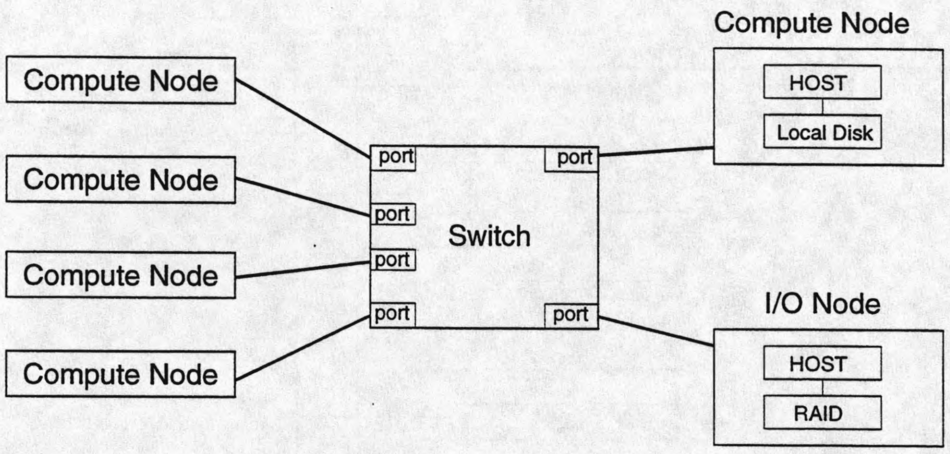

Figure 1: System architecture

storing data, parity and adequate coding information for use in reconstructing the corrupted data. The RAID system is controlled and managed by the array controller (see Fig. 2), which is responsible for data transfer between hosts and disks. The array controller is composed of a set of control units which process user requests received from the channel interfaces and direct these requests to the cache subsystem. The communication, control and data transfer inside the array controller are performed via reliable and high-speed control and data buses.

Each I/O request received by the array controller is processed by the cache subsystem. To illustrate the array controller operation, we will describe the read operation scenario. The cache controller first checks the contents of the cache memory. If the track is already in the cache ("cache hit"), the track is read from the cache memory and the corresponding data is delivered to the channel interfaces. If the track is not stored in the cache ("cache miss"), a request is directed to the disk array and the data is brought to the cache memory. Then, the track is read from the cache memory and the data is sent back to the channel interfaces. More details about the array controller operations can be found in [16].

As shown in Fig. 2, the array controller is composed of a cache subsystem (see the dotted line box in Fig. 2), a channel interface(CI) to the local host and a disk interface to the disk array. Among these components, the cache subsystem is of a particular importance to the correct operation of RAID, because all data transfer operations between channels and the disk array are performed by the cache subsystem. Our study, therefore, focuses on the detailed analysis of the cache subsystem. The cache subsystem is decomposed into two separate parts, the cache memory (CM) and the cache controller. The cache controller provides interfaces to the cache memory and to the channels and the disks. The communication links between the cache controller interfaces and the cache memory, channel/disk are provided by multi-directional buses depicted as Bus 1 and Bus 2.

The cache subsystem employs a combination of parity code, error detection and correction(EDAC) code and Cyclic Redundancy Checking (CRC) code to detect and/or correct data errors, which occur during the cache operation. In the following, we briefly present how these different coding techniques are applied to cope with errors in the data path.

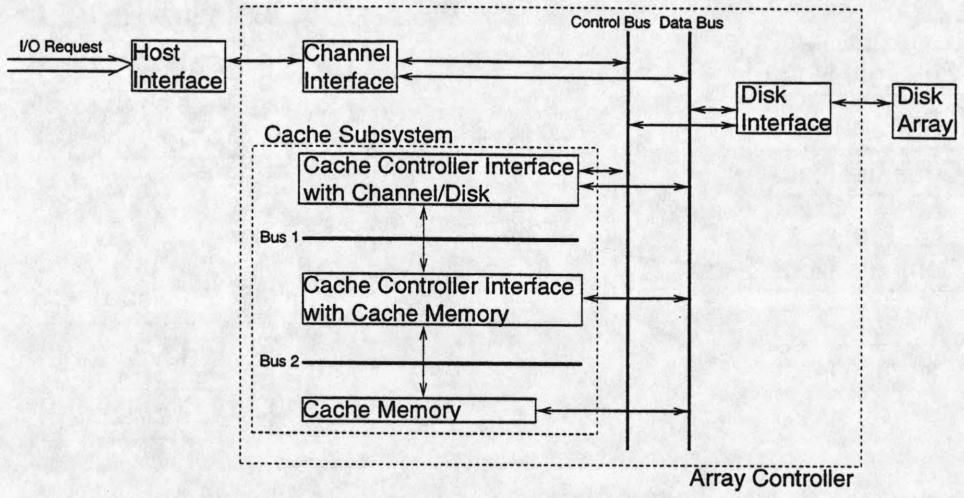

Figure 2: Array controller architecture and external interfaces

Parity is used to cover data transfer, over Bus 1, between the cache controller interfaces. Parity bits are appended for each data word transmitted over the bus. When a parity error is detected, automatic retries are attempted by the cache controller to recover from the error.

EDAC is used to protect data transmitted from the cache controller interfaces to the cache memory and data stored in the cache memory. The EDAC is appended to each data word transfer over Bus 2 and then stored in the cache.

Two CRC are used by the cache subsystem the frontend CRC and the physical sector CRC. The frontend CRC is used to protect the data transfer to the cache subsystem during the write operation. The physical sector CRC protects the data stored on the disk. Detailed information on the employed coding techniques can be found in [16].

4 Simulation Model

This section presents a simulation model of the system described in Section 3. This model is used in our case study to demonstrate capabilities of the speculation-based simulation approach. The simulation focuses on the cluster behavior in the presence of transient faults introduced to the cache subsystem. The behavior of the cache subsystem is modeled in detail to precisely capture and analyze its ability to cope with faults injected into the cache memory, cache controller interfaces and buses. In our simulation study we do not inject faults into the compute nodes and to the network. To account for the network latency in a data transmission through the network, we have conducted a separate cycle-by-cycle of the ServerNet switch. The obtained network latency distribution is used to derive the latency in data transfer from a compute node to the I/O node in the simulated cluster system. The remainder of this section presents a compute node, a network switch model and an I/O node model used in our simulation study.

4.1 Compute Node Model

Each computer node generates I/O requests to its local disk system or to the I/O node. An arbitrary decision (based on the fixed probability) is taken to determine the percentage of requests to the I/O node. The interarrival time of the read/write requests to the local disk or to the I/O node is based on a real access trace which captures the track skew phenomenon [20], i.e., the distribution of the number of accesses per track is not uniform, rather a few tracks are generally more frequently accessed than the rest of the tracks. Each request entry specifies the request type (read/write), the track accessed and the interarrival time. The operations simulated target at the track level.

The I/O requests are buffered in the compute node before being sent out to its local disk or onto the network. The local disk is modeled using a single server. A request sent to the I/O node and its response are delivered through the switch port connected to the compute node. The compute node has two separate incoming and outgoing queues that receive responses and send out requests separately. A request-response pair is modeled for each read/write operation. Each successful write operation is acknowledged by a response returned to the source compute node. At any time, there are at most nine outstanding requests (as specified in the ServerNet design) from a compute node to the I/O node. That is, the compute node waits for a response until it can generate more I/O node requests.

4.2 Switch Model

The switch consists of six independent bidirectional ports and a routing table. Logically, each port can be subdivided into an input port and an output port. Each input port has a FIFO queue that buffers incoming packets before being forwarded to the appropriate target ports. As ServerNet employs worm-hole routing, as soon as the packet arrives at the switch, the switch begins the process of determining the target output port. If two input ports request the output port at the same time, one of them is chosen randomly.

The packet transmitted over the network is simulated at the network physical link level going through the ServerNet switch. The ServerNet operates using worm-hole routing and a packet is transmitted byte by byte for each cycle. We developed a cycle-by-cycle based simulator to obtain the network latency per packet. The average network latency obtained from the cycle-based simulation is 2.29ms and Fig. 3 shows the distribution. The network latency in data transmission from a compute node to the I/O node for our cluster system simulation is sampled from this distribution.

4.3 I/O Node Model

A detailed model is built to characterize the RAID cache architecture, cache operations and error detection/correction mechanisms, as described in Section 3. Requests generated from other compute nodes form the workload to the RAID system. The impact of faults/errors occurring in the cache subsystem is analyzed to assess the overlapping error detection and recovery mechanisms. In the following, a fault/error model as well as an error detection/correction model are described.

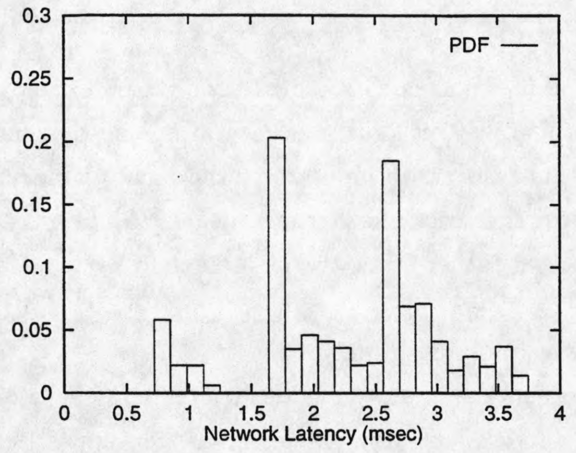

Figure 3: Network latency distribution

4.3.1 Fault/Error Model

In the simulation of the cluster system, we inject transient faults leading to single or multiple bit-errors in the track while the track is stored in the cache memory or during the transfer in the cache subsystem. Faults are injected using a load dependent fault injection strategy. The load dependent fault injection aims to simulate occurrence of faults due to the stress during system operation. The rate of load dependent fault injection is based on the number of accesses to the memory or to the disk and is tuned to allow a single fault injection or consecutive multiple near-coincident fault injections. This enables us to analyze the impact of isolated and bursty fault patterns. For each error injection, the probability leading to single or multiple bit errors is determined by a distribution obtained from a lower level model of track data block transfer inside the cache subsystem [16].

4.3.2 Error Detection/Correction Model

As mentioned in Section 3, the cache subsystem uses a combination of coding techniques to detect/correct errors, which occur in the cache memory or during the data transfer. The behavior of these mechanisms and their interactions are incorporated into our simulation model of the cluster system.

The ability of detecting/correcting the track errors depends on the number of errors which affected the track. To simulate this, the number of errors in each track is recorded and updated during the simulation. Each time a new error is injected into the track, the number of errors is incremented. When a request is sent to the cache controller to read a track, the number of errors affecting the track is checked and compared with the error detection conditions to decide whether errors are detected/corrected. During a write operation, the track errors that have been accumulated during the previous operations are overwritten and the number of errors associated with the track is reset to zero. Several possible scenarios of error behavior are possible as it is illustrated in Fig. 4. This figure shows the data flow of cache operations, error detection boundaries, error propagation and the components targeted by fault injection. Note that errors may escape from some error confinement boundaries and propagate inside the cache subsystem.

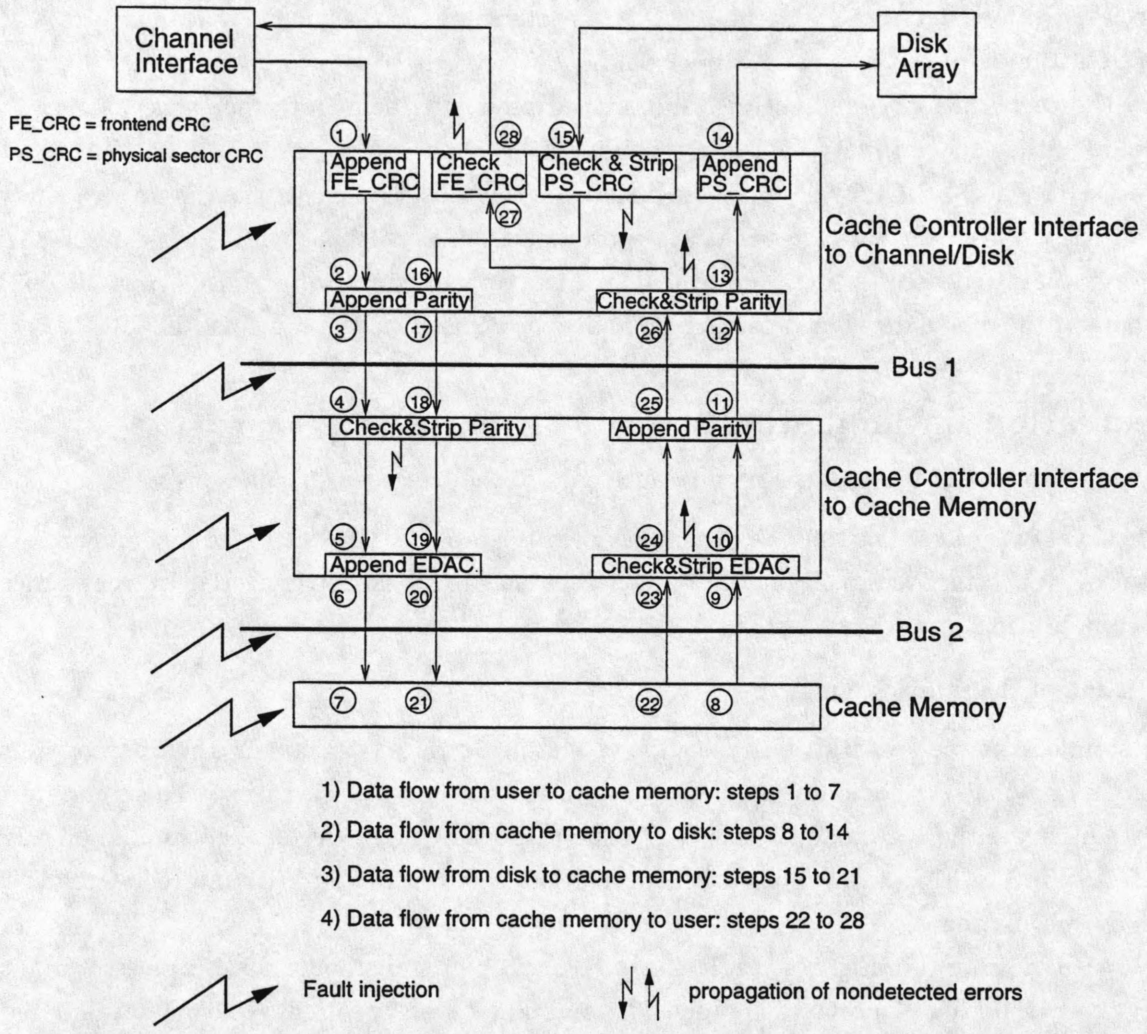
Channel Interface
Disk Array
FE_CRC = frontend CRC
PS_CRC = physical sector CRC
Append FE_CRC
Check FE_CRC
Check & Strip PS_CRC
Append PS_CRC
Cache Controller Interface to Channel/Disk
Append Parity
Check&Strip Parity
Bus 1
Check&Strip Parity
Append Parity
Cache Controller Interface to Cache Memory
Append EDAC
Check&Strip EDAC
Bus 2
Cache Memory
1) Data flow from user to cache memory: steps 1 to 7
2) Data flow from cache memory to disk: steps 8 to 14
3) Data flow from disk to cache memory: steps 15 to 21
4) Data flow from cache memory to user: steps 22 to 28
Fault injection
propagation of nondetected errors

Figure 4: Cache operations, data flow, fault occurrence, error detection boundaries and error propagation

These errors may be finally detected by the frontend CRC or propagate outside the cache subsystem.

Fig. 4 illustrates how different coding techniques are utilized to detect and correct error in different parts of the cache subsystem. The coding steps are labeled from 1 to 28 in Fig. 4, where each label denotes an action of appending, checking or striping a code.

Frontend CRC (FE_CRC) is computed by the channel interface and appended to the data sent to the cache subsystem during a write request. It is checked when the data is read from the cache subsystem. This is shown as the steps 1 and 28. FE_CRC detects errors that may occur at any step of the data transferring to or from the cache subsystem. FE_CRC can detect the case when less than 4 data bytes contain errors.

Parity is used to cover errors occurred during the transfer over Bus 1. Data is appended with parity code before it is sent onto Bus 1 and parity is checked right after the data is taken off Bus 1. This corresponds to the steps 3, 4, 17 and 18 in Fig. 4.

The error detection and correction code (EDAC) protects the cache memory against errors. It is capable of detecting triple bit error and correct double bit error. It is appended to the data before they are sent to the cache memory, and checked and striped after it is read from the cache memory. It is stored as part of the cache memory. This corresponds to the steps 6, 7, 8, 9, 20, 21, 22 and 23 in Fig. 4.

Physical sector CRC(PS_CRC) is appended before the data is written to the disk array, and it is checked and striped from the data after the data is brought from the disk array. The checking is invoked by a read operation that accesses data from the disk array. It is used to protect disk errors. PS_CRC is stored as part of the disk data. The detection capability of PS_CRC is the same as FE_CRC.

5 Speculation in Simulation

Two primary reasons contribute to the performance gain of the proposed speculation-based method:

- Reduction of processor idle time, i.e., the time when the processor is waiting for incoming messages. This provides better overlap of computation on different processes, better overlap of communication and computation.

- Reduction of the rollback number.

We use simple example scenarios based on the model in Section 4 to explain the above points. A compute node and the I/O node are pictured in Fig. 5 for the example scenarios. The compute node sends I/O requests to the I/O node and the I/O node responses the requests. The simulation without speculation is shown in Fig. 5(a), (c) and with speculation is shown in Fig. 5(b) and (d). Only the I/O nodes employ speculation.

A typical scenario shown in Fig. 5(a) explains the reason why the method reduces processor idle time. To simplify the explanation, four instead of nine outstanding requests are used. Nine outstanding request of a compute node is based on the ServerNet design as mentioned in Section 4.1. In Fig. 5(a), bursty requests are sent to the I/O node during the simulation. As shown in Fig. 5(a), if four bursty requests are sent out to the I/O node, due to slow responses arriving at the compute node, the compute node is idle before it can generate more requests for the simulation. If speculation is employed as shown in Fig. 5(b), the responses can be sent back earlier, thereby, simulation on the compute node can proceed without delay.

In Fig. 5(c), without speculation, the response "response 1" of the "remote 1" request is sent back not early enough, and it rolls back the simulation of requests "local 4", "local 5" and "local 6" since its timestamp is 150 and timestamp of request "local 4" is 200. These can be avoided if the response is sent back earlier as shown in Fig. 5(d) using speculation. Therefore, the rollback number is reduced for the simulation.

6 Methodology

In this section, we first describe the speculation-based simulation methodology in general, then the method is illustrated using examples based on the cluster system presented in Section 3, and later we

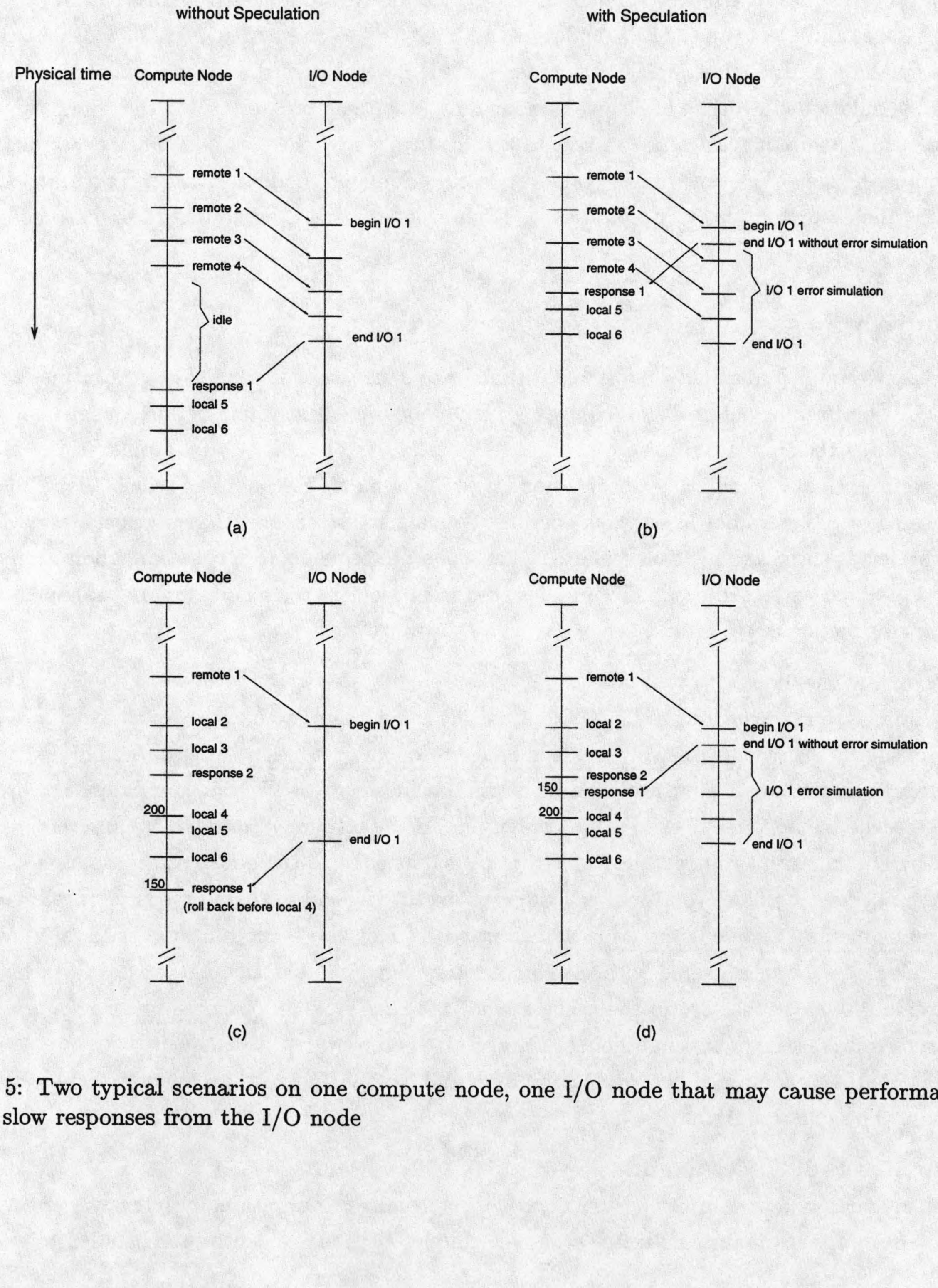

Figure 5: Two typical scenarios on one compute node, one I/O node that may cause performance loss due to slow responses from the I/O node

present the algorithm developed to implement the method.

6.1 Concepts

A *logical process* (LP) is a distributed simulation unit which receives messages from other LPs, processes incoming messages and events on the LPs, and sends out messages to other LPs. *Local virtual time* (LVT) is the local simulation clock that indicates how far the LP has progressed. *Global virtual time* (GVT) is the global simulation clock that indicates how far the system has progressed for the simulation. An event routine consists of an event and constituent actions invoked by the event. *Timestamp* of an event routine denotes the time the event routine takes on in the simulated model. *A sequential event list* of a LP maintains the event routines to be processed. Each event routine is scheduled to execute based on its timestamp.

6.2 Approach

The proposed method is implemented as an algorithm for dynamic rescheduling of event routines at run-time on the LPs performing detailed simulation, e.g., I/O node cache subsystem detailed simulation. The rescheduling allows the simulation to move forward as quickly as possible. For example, in the cluster system simulation, the LP simulating the I/O node uses the dynamic rescheduling which allows the I/O response messages to be sent out as early as possible. The algorithm ensures that the speculation-based simulation preserves correct simulation semantics, i.e., the event sequencing is based on a non-decreasing timestamp associated with each event. Techniques used to implement the algorithm are explained in the remainder of this section.

Event Routine Design

To facilitate run-time rescheduling of event routines via speculation, each event routine includes three sub-event routines: 1) Schedule new events or manage resource contention (*routineI*): This sub-event routine schedules future event routines, simulates the advancement of LVT or simulates the resource sharing, e.g., cache memory read/write ports contention. If the resource requested by an event routine is occupied by another event routine, the event routine is deposited to the queue of the resource to simulate the resource waiting. This routine is scheduled to execute when the shared resource is released. 2) Computation or manipulation of simulation data(*routineII*): This sub-event routine conducts computation and manipulates data associated with the computation. For example, there exists data which stores the information of what tracks are in the cache memory. 3) Schedule an outgoing message(*routineIII*): This sub-event routine sends out an outgoing message.

Each event routine is composed of *routineI*, *routineII* and *routineIII*, and consequently these sub-event routines share the same timestamp.

Dynamic Rescheduling of Events

For detailed simulation of complex system, *routineII* is usually computationally intensive. If the data manipulated by *routineII* doesn't influence event scheduling, the routine can be rescheduled to execute after *routineI* or *routineIII* of subsequent event routines. It is expected that the rescheduling of sub-event routines can avoid the delay of generating messages to other LPs due to time-consuming *routineII*. The execution of rescheduled sub-event routine *routineII* can take place while the LP waits on incoming messages. Therefore, it is generally beneficial if the dynamic rescheduling is chosen for LPs that incoming messages are infrequent.

Speculation facilitates the rescheduling since event scheduling may depend on the data. The speculation for our dependability analysis is based on the observation that errors are most likely detected and corrected in a reliable system. Therefore, before the detailed simulation of the error correction/detection in *routineII* completes, *routineI* of the next event routine can be rescheduled to execute ahead of the current *routineII* since we speculatively assume that potential errors are corrected.

Correction of Simulation Errors Introduced by Speculation

The simulation errors need to be corrected if the actual execution of *routineII* provides results different than the speculation. In the simulation on the I/O node, if it turns out that errors in the track are not corrected, simulation errors occur. The state of the simulation is rolled back right before the request with the *routineII*s that don't match the speculation. In other words, the I/O responses sent out earlier under speculation should be nullified and the I/O requests simulated after the request with simulation errors need to be resimulated. These are implemented based on the rollback mechanism of the optimistic protocol. As errors in the track are not corrected, the I/O request needs to be retried to retrieve correct data after the rollback.

6.3 Examples

A sequence of event routines that simulate service of a cache hit read request is adopted to illustrate the approach presented in the previous section. For the sake of brevity, the operations are explained at disk track level. Three event routines are designed to simulate the steps of servicing the cache hit read request. They are: 1) the track transmission from CM to CCICM. 2) the track transmission from CCICM to CCICI. 3) the track transmission from CCICI to CI.

Based on the design criteria for rescheduling via speculation, each event routine above is designed to include three sub-event routines: 1) *routineI*: releases the data path to possible operations existing in parallel which are waiting for resources like CCICM, schedules the next event routine associated with the same request and advances the simulation time for the transmission, fault injection and error correction. 2) *routineII*: fault injection for this transmission, error detection/correction for data bytes in the track, and update the error statistics for the track. 3) *routineIII*: sends out an I/O response if there is one.

6.3.1 A Read Cache Hit with No Other Parallel Requests

In Fig. 6, the scheduling of event routines for a read cache hit request with no occurrence of other requests in between the start and end of this request is shown. Each rectangular box on the axis indicates an event routine, and the shaded block inside indicates the sub-event routines. The axis indicates LVT. The length of the box indicates the duration of event routine execution time.

Without speculation, the sub-event routines are not rescheduled, as shown in Fig. 6(a). In Fig. 6(b), the event routines are rescheduled based on the speculation that errors are corrected. *routineI* of each event routine is rescheduled as early as possible ahead of all the other sub-event routines for this request. As the event routine at t3 schedules a message sent out to the requested node, the rescheduling makes it possible to send out the response much earlier. Before the message is sent out, the errors in the track are all cleaned based on the speculation that the errors are corrected. Note that timestamp associated with the rescheduled sub-event routines *routineII* is the same as before the rescheduling to guarantee the correctness of timing statistics collected during these sub-event routines.

6.3.2 A Read Cache Hit with Other Parallel Requests

Fig. 7 shows an example if two requests occur in parallel and the event routine execution of these requests are interleaved. In Fig. 7, request(2) arrives before the transfer from CCICI to CI of request(1). Note that sub-event routines *routineII* of parallel requests preserve their scheduling order as the order before rescheduling to guarantee the simulation correctness. Otherwise, if the requests share data and the order is not preserved, statistics collected may not hold. For example, in Fig. 7, requests(1) and request(2) occur in parallel and the disk tracks specified by the requests share data. During the sub-event routine "transfer from CM to CCICM checking EDAC" of request(2) at time t3, faults are injected into the shared disk data area. If sub-event routines of request(2) is not rescheduled, then after the rescheduling of *routineII* of request(1), *routineII* of request(1) at t1 is scheduled after *routineII* at t1 of request(2). Following this scheduling order, the error number in the state data for *routineII* of request(1) at time t1 is different than without rescheduling since *routineII* of request(2) at t3 introduced more errors to the shared data area. Therefore, the error scenario simulated with and without speculation is different. If the data is not shared between the two operations, no dependency exists. More aggressively, the *routineI* at t5 and t6 and *routineIII* at t6 of request(2) in Fig. 7 can be rescheduled ahead of *routineII*s of request(1).

6.4 Implementation

The outline of the algorithm implemented for the I/O node is shown in Fig. 8. The simulation on the I/O node consists of three parts. First, processing an arriving message based on the message type. If the timestamp of the message is smaller than the I/O node LP's LVT, or it is an anti-message, rollback is invoked. In case the message contains an I/O request, corresponding event routines are scheduled to simulate the transmission in the cache subsystem and the disk.

Second, event routines maintained in the event list (EVL) are processed. Each event routine is decomposed into the three sub-event routines. *routineII* are rescheduled and deposited into L_II. The sorted list L_II is created to contain the rescheduled *routineII*. If *routineII* of the processed event routine is rescheduled, *routineI* or *routineIII* are processed.

Third, the rescheduled *routineII* are processed. This takes place if the LP is idle waiting for the next I/O request or $st(EVL)$ is greater than $st(L\_II)$ by a prespecified interval of time, tl. So in case the smallest timestamp of event routines in EVL is greater than the smallest timestamp of event routines in

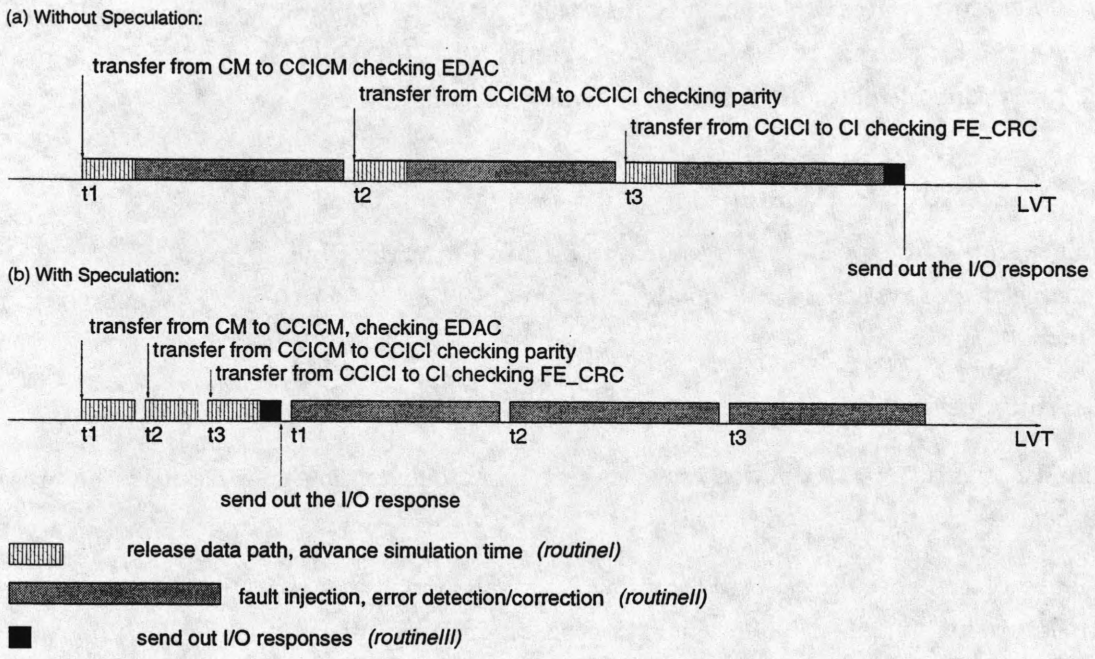
(a) Without Speculation:
transfer from CM to CCICM checking EDAC
transfer from CCICM to CCICI checking parity
transfer from CCICI to CI checking FE_CRC
t1
t2
t3
LVT
send out the I/O response
(b) With Speculation:
transfer from CM to CCICM, checking EDAC
transfer from CCICM to CCICI checking parity
transfer from CCICI to CI checking FE_CRC
t1
t2
t3
t1
t2
t3
LVT
send out the I/O response
release data path, advance simulation time (routineI)
fault injection, error detection/correction (routineII)
send out I/O responses (routineIII)

Figure 6: A read cache hit with no other parallel requests

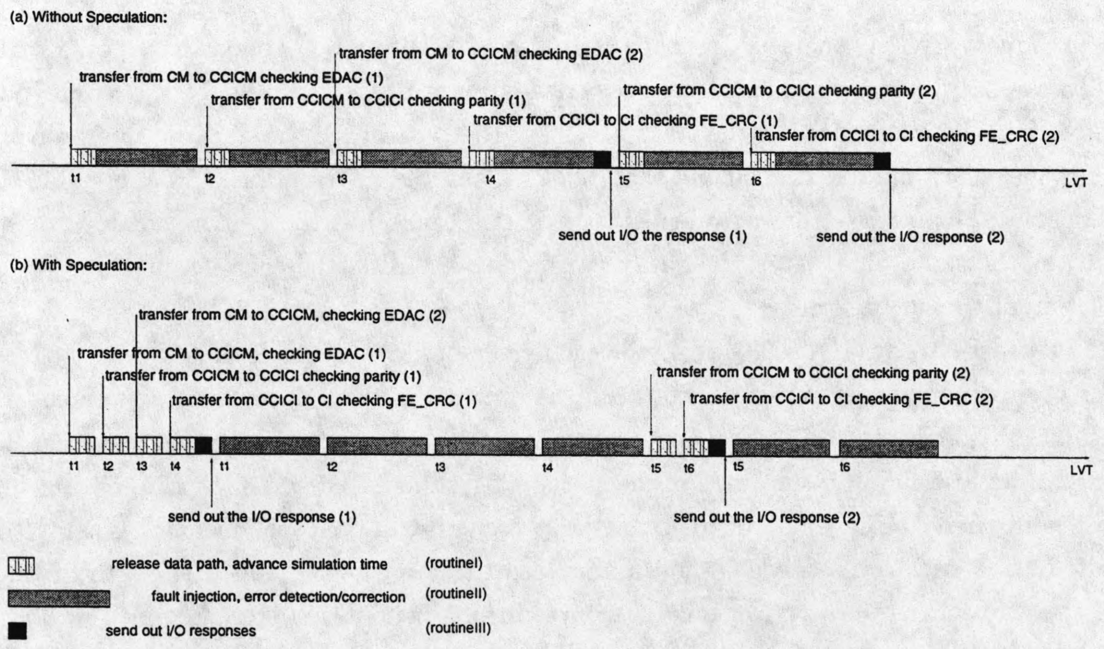
(a) Without Speculation:
transfer from CM to CCICM checking EDAC (2)
transfer from CM to CCICM checking EDAC (1)
transfer from CCICM to CCICI checking parity (1)
transfer from CCICM to CCICI checking parity (2)
transfer from CCICI to CI checking FE_CRC (1)
transfer from CCICI to CI checking FE_CRC (2)
t1
t2
t3
t4
t5
t6
LVT
send out I/O the response (1)
send out the I/O response (2)
(b) With Speculation:
transfer from CM to CCICM, checking EDAC (2)
transfer from CM to CCICM, checking EDAC (1)
transfer from CCICM to CCICI checking parity (1)
transfer from CCICI to CI checking FE_CRC (1)
transfer from CCICM to CCICI checking parity (2)
transfer from CCICI to CI checking FE_CRC (2)
t1
t2
t3
t4
t1
t2
t3
t4
t5
t6
t5
t6
LVT
send out the I/O response (1)
send out the I/O response (2)
release data path, advance simulation time (routineI)
fault injection, error detection/correction (routineII)
send out I/O responses (routineIII)

Figure 7: A read cache hit with other parallel requests

st(L_II) by some predefined value, *tl*, *routineII* are processed. This prevents infinite delay for processing *routineII*. If the execution of *routineII* at time *td* turns out that errors injected into the disk track are not corrected, retry of the request is necessary. As a result, the LP is rolled back to the state just before *td*. After the processing of *routineII*, the LVT is updated for this LP.

7 Experimental Results

In this section, we present experimental results obtained from the distributed speculation-based simulation of the reliable cluster system described in Section 3 and Section 4. The cluster system is characterized by two types of measures:

- dependability measures: error detection coverage and error detection latency.
- performance measures: distribution and mean of 2-way deliver time of a track in the presence of faults injected into the I/O node cache subsystem.

The performance of the speculation-based approach is evaluated by comparing with the performance of sequential simulation and distributed simulation based on Time Warp. Simulation experiments are conducted on Sun Ultra1-170 workstations interconnected by 100Mb/sec fast Ethernet. Message Passing Interface (MPI) is used as the communication layer for supporting communication between workstations running the simulation.

7.1 Experiment Set-up

The input I/O request stream generated from the compute node to the I/O node is based on a real trace under a real workload. The real trace demonstrates track skew which is common for I/O traffic pattern. Each I/O request in the trace specifies the track accessed, the type of request and the interarrival time. Two types of requests are simulated, read and write though, and the distribution is 85% reads, 15% write through operations. For a write through operation, if it is a cache hit, the data is written to the CM and disk array at the same time. After the data is written to the disk, the channel interface is signaled of the completion of the operation.

The accessible tracks are around 200,000 and the cache memory size is around 10,000 tracks. Transient faults leading to track errors are injected into the cache subsystem using load-based injection strategy. The rate of faults injected into the components other than the buses is assumed to be one error per 10^7 bits accessed. Faults injected into the buses of the cache subsystem is 100 times the rate of that injected into the other components. The length of the error burst occurring in the cache memory is sampled from a normal distribution with a mean of 100 and a standard deviation of 10, and the length of the error burst during the transfer in the cache controller is sampled from distribution presented in [16]. The mean length of the error burst measured from the simulation is around 100 bits during transfer over the buses, 800 bits when the track is temporarily stored in the cache controller interface to the cache memory and 1,000 bits when the track is temporarily stored in the cache controller interfaces to the disks.

TS:	Time Stamp
LVT:	Local Virtual Time
EVL:	Event List
er:	Event Routine
L_II:	The list of sub-event routines *routineII* that are rescheduled, the order is preserved as in the original event routines.
tl:	time interval to execute L_II.
td:	timestamp of a *routineII* in which errors are not corrected.
$st(LVT)$:	timestamp of the first event on EVL.
$st(L\_II)$:	timestamp of the first event on L_II.

```
simulate_IO_Node() {

  while (1) {
        /* receive message from another LP */
        if event $e$ is received
              /* Roll back */
              if $TS(e) < LVT$ or $e$ is an anti-message
                    restore to the immediate checkpoint before $TS(e)$ for both $EVL$ and $L\_II$
                    if $e$ is an anti-message
                          cancel the message in the input queue of this LP associated with $e$
                    else if $e$ is an I/O request
                          schedule event routines of this I/O request
              else if $(LVT <= TS(e) < st(EVL))$
                    restore to the immediate checkpoint before $TS(e)$ in $EVL$
              /* Schedule a new I/O request */
              else
                    schedule the start event routine of this I/O request

        /* Process $EVL$ */
        Obtain the next $er$ from $EVL$;
        Decompose $er$ into routineI, routineII, routineIII
        Deposit routineII of $er$ to $L\_II$
        Annotate routineII of $er$ with $TS(er)$
        Execute routineI of $er$
        If routineIII of $er$ exists
              Send out an I/O response message to other LPs
        Send out anti-messages if any

        /* Process $L\_II$ */
        If ((no next $er$ from $EVL$) || (($st(EVL)$ - $st(L\_II)$) > $tl$))
              Execute routineII in $L\_II$
              If errors of a track not corrected in routineII at $td$
                    Rollback before $td$
                    Schedule retry of the $er$
              Update $LVT$
  }
}
```

Figure 8: The outline of the implementation of the speculation-based algorithm on the I/O node

7.2 Dependability Measures

Dependability measures obtained from the simulation include: 1) error coverage of the EDAC code, 2) error detection latency of errors injected into various cache components. The measures are used to assess the error detection/correction mechanisms of the cache subsystem. In addition, to validate the detailed I/O node model, the results are also compared with that presented in [16] where a detailed model of the cache subsystem is developed.

The EDAC detection and correction coverage are reported in Fig. 9. It is observed that coverage factor is very high and tends to stabilize as the simulation time advances.

Fig. 10 shows the error detection latency probability density function for errors injected into bus 1 (B1), errors into bus 2 (B2), errors into cache controller interface to cache memory/disk (CCI), errors into cache memory (CM). Error detection latency is defined to be the time between when an error is first injected to a track and the time when the error is detected, corrected or overwritten. For errors injected into a track at a component, we record the latency associated with the first error injected into the track. In other words, the measured latency corresponds to the maximum latency for errors present in the tracks. The latency allows us to analyze the potential of errors that remain undetected, or to cross the boundaries of detection mechanisms before being detected and efficiency of the overlapping detection/correction mechanisms.

Results shown in Fig. 10 indicate several points. Type B1 errors are mostly covered by parity. This latency is very short due to the immediate parity checking for the tracks transferred over bus 1. Those that escape parity tend to be latent for a long time. Type B2, CCI and CM errors have much longer latency. The reason is that their detection/correction by EDAC or CRC are triggered by read/write operations to the tracks containing these errors. As a result, these latency depends on the distribution of I/O requests to the I/O node. If a track is not frequently accessed, then errors preserved in the track might remain latent for a long period of time.

The obtained results are consistent with those presented in [16]. They also validate the correctness of the detailed model of the I/O node for the simulated cluster system.

7.3 Performance Measures

Measures used to characterize the cluster system performance in the presence of faults include: 1) the 2-way delivery time distribution. 2) the mean time of 2-way delivery as a function of a fault injection rate to the cache memory. 2-way delivery time of a track, is the difference between the time a request is created and the corresponding response is received.

2-way delivery time distribution under faults is measured, as shown in Fig. 11. The distribution is bi-modal. The first mode corresponds to the delivery time of a track if no faults corrupt its data and retry to access the track is not necessary. The second mode corresponds to higher delivery time when retry of track transfer in the cache controller is necessary due to errors detected during the transfer of the track.

To assess the impact of the I/O node errors on the system performance, the mean time of 2-way

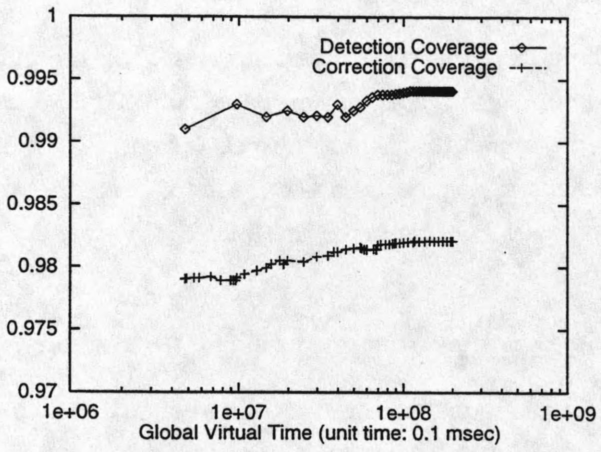

Figure 9: EDAC coverage

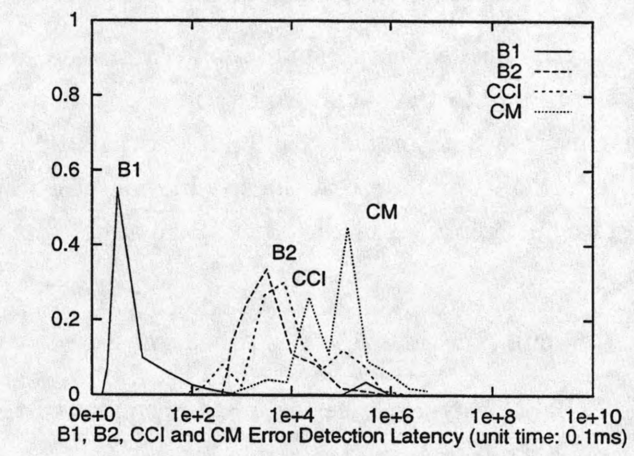

Figure 10: Error detection latency distribution

delivery of a track is measured under various fault injection rate into the cache memory, as shown in Fig. 12. Fault injection rate into the cache memory is specified in the amount of data bits accessed for the occurrence of one error. From Fig. 12, we observe that if more faults are injected into the cache memory, the mean time of 2-way delivery of a track increases. This is due to the retry of the requests.

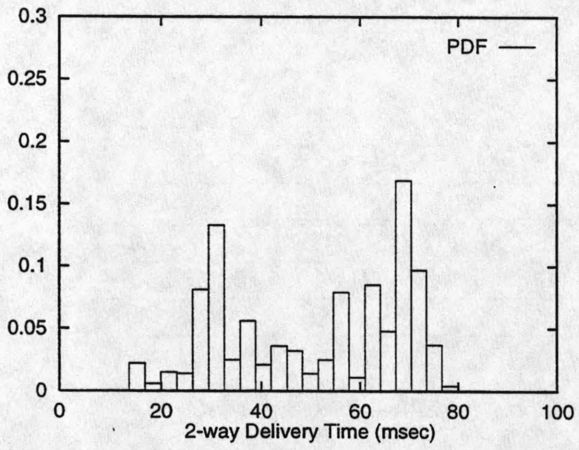

Figure 11: 2-way delivery time distribution

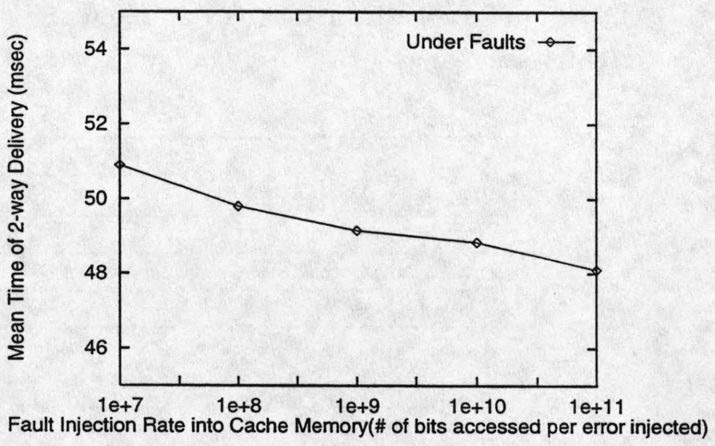

Figure 12: Mean time of 2-way delivery

7.4 Simulation Performance

In this section, we evaluate the speculation-based methodology using the cluster system model and compare its performance with sequential simulation and a traditional optimistic protocol, Time Warp of distributed simulation.

7.4.1 Performance Comparison of Sequential vs. Distributed Simulation

We obtain the sequential simulation performance numbers from our distributed simulator via mapping all the LPs onto the same workstation. For distributed simulation, speculation-based and Time Warp, the cluster system model with 6 nodes is partitioned and mapped to 2, 4 and 6 workstations.

Performance numbers of the speculation-based approach vs. sequential are recorded in Table 1, and performance numbers of the Time Warp approach vs. sequential are reported in Table 2. The two tables present:

- Simulation time: the real execution time to complete the simulation of the simulated system.
- Global Virtual Time (GVT): the simulated time for the model in the distributed simulation environment.
- Speedup: the ratio between the sequential simulation time and the simulation time obtained for a distributed simulation.

The results from Table 1 demonstrate significant speedup of the speculation-based approach over sequential simulation. Our method yields as much as 77%(4.60/6) of ideal speedup (If a model is distributed onto n workstations, the ideal speedup is n) for the 6 workstation case. The variation of the global virtual time is due to the global virtual time computation algorithm. GVT is periodically computed and for each simulation run, the timing for GVT calculation is not exactly the same due to synchronization among the network of workstations.

	Sequential	Speculation-based Approach		
Number of Workstations	1	2	4	6
Simulation Time (sec)	21,652	17,748	7,415	4,707
GVT (unit: 0.1 msec)	199,454,001	199,606,290	199,921,715	199,492,367
Speedup	1	1.22	2.92	4.60

Table 1: Performance of sequential vs. the speculation-based approach

In Table 2, we observe that, compared with the speculation-based approach, Time Warp offers less performance gain over sequential simulation. On 6 workstations, 39% ideal speedup is observed compared with sequential simulation. Two major factors limit Time Warp's performance. One is that the computation intensive events due to detailed simulation of error behavior on the I/O node causes poor overlap of computation and communication. In other words, the progress of local virtual time on the I/O node is slower than that on the compute nodes. Another is due to the large number of rollbacks, which is demonstrated in later sections. With speculation, rollback number is greatly reduced and parallelism in the simulation model is fully exploited.

	Sequential	Time Warp		
Number of Workstations	1	2	4	6
Simulation Time (sec)	21,652	20,621	12,516	9,373
GVT (unit: 0.1 msec)	199,454,001	199,731,324	199,864,005	199,970,300
Speedup	1	1.05	1.73	2.31

Table 2: Performance of sequential vs. Time Warp

7.4.2 Performance Comparison of Speculation vs. Time Warp

In this section, simulation time and cumulative rollback number of the speculation-based approach and Time Warp are compared by varying the simulation parameters of the cluster model. The numbers reported are obtained from experiments mapping the model on a network of six workstations.

Three simulation parameters are varied to examine the performance numbers between the two approaches:

- Request frequency to the I/O node: the probability of generating a request to the I/O node.
- Cache memory size: the number of disk tracks in the cache memory of the I/O node.
- Injection rate to cache memory: the rate is specified in the amount of data bits accessed for the occurrence of one error injected into CM. For example, the rate of 1e+7 means for every 10^7 bits accessed, one error is injected into the cache memory.

Fig. 13 and Fig. 14 shows the simulation time and rollback number varying the request frequency to the I/O node. With speculation, simulation time is reduced by as much as 50% and the speculative simulation performs consistently better than Time Warp. As the request frequency to the I/O node increases, simulation time increases. For Time Warp, detailed simulation of error behavior on the I/O node delays the responses to compute nodes and causes a significant number of rollbacks, as shown in Fig. 14. Using the speculative approach, the rollback number is reduced by as much as 50%. However, when frequency to the I/O node increases, the simulation time of the speculation-based approach increases at a higher rate than Time Warp. The reason is that the rollback cost per I/O request for the speculation-based approach increases as the frequency to the I/O node increases. Also more rescheduled *routineII* are delayed to be processed.

Fig. 15 and Fig. 16 shows the simulation time and rollback number varying the cache memory size on the I/O node. The results shown here exhibit less regularity than those obtained by varying request frequency to the I/O node. This is due to the fact that varying the cache memory size, cache hit ratio is affected. This in turn may lead to variation in patterns of simulation message generation and transmission. The computationally intensive error characterization affects our approach and Time Warp in a different way, and the speculative approach delivers better performance for different cache memory size than the Time Warp algorithm.

Fig. 17 and Fig. 18 shows the simulation time and rollback number varying the injection rate for cache memory. If less faults are injected into the cache memory, the detailed simulation takes less time and the overall simulation takes less time. It can be observed that the speculation-based approach performs consistently better than Time Warp in terms of simulation time and rollback numbers.

8 Discussion and Conclusions

We proposed a speculation-based distributed simulation approach to detailed evaluation of system behavior in the presence of faults. The simulation method is demonstrated and validated in the case study that analyzes dependability and performance of a reliable cluster system. The cluster consists of five compute nodes and one I/O node connected via a ServerNet switch. The I/O node supports a cache-based RAID storage architecture to provide high performance and availability. The RAID system operates under control of the array controller, which employs a number of overlapping error detection/correction mechanisms to improve availability. In order to capture the cluster system behavior and to evaluate the efficiency of the error detection/correction mechanisms, we used detailed modeling of I/O operations in the cache subsystem in the presence of faults.

The speculation-based distributed simulation for system dependability analysis is based on the observation that in a reliable system most of potential errors are detected and corrected. Consequently, the simulation can progress without waiting until the detailed simulation of error behavior completes. Our method introduces speculation into the distributed simulation and proposes a simulation algorithm for speculation-based, dynamic event rescheduling. This method provides:

- an efficient overlapping between simulation of the computation intensive detailed error characterization and the rest of computation conducted at a more abstract level, and the communication among logic processes.
- a reduction of rollback number (a major drawback of an optimistic, distributed simulation).

Our experimental results demonstrate that using speculation-based simulation, valid and accurate dependability and performance measures can be obtained, including error detection coverage, error latency, and performance degradation. Moreover, real input traces can be used to drive the simulation and this offers accuracy, which cannot be achieved using analytical models with simplified distributions of access pattern.

Furthermore, run-time performance results show that the speculation-based simulation contributes to a significant reduction in the simulation time. For example, in the speculation-based simulation of the cluster system, 4.6 times speedup over sequential simulation and 2.0 times speedup over the Time Warp simulation algorithm is achieved in an environment of six workstations. This indicates the strength of the speculation in the distributed simulation.

Efficiency of the speculation-based approach depends on characteristics of the simulated system. The approach will work when the traffic going to the logical processes participated in speculation is relatively

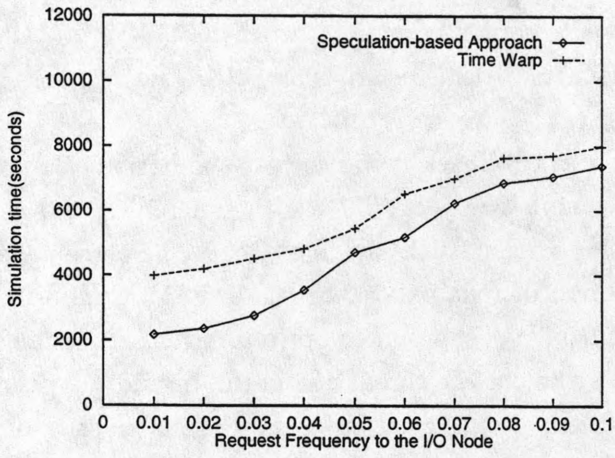

Figure 13: Execution Time

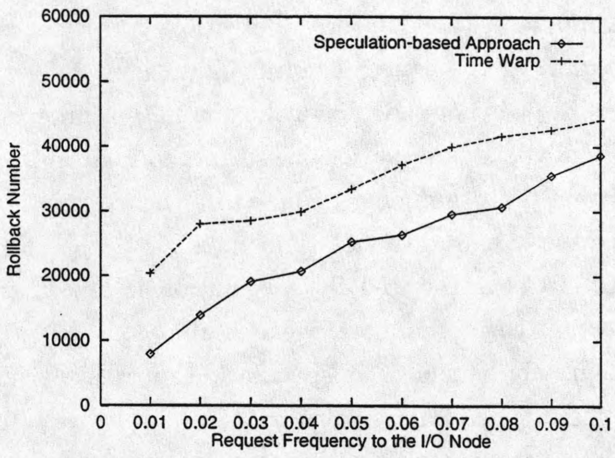

Figure 14: Rollback Number

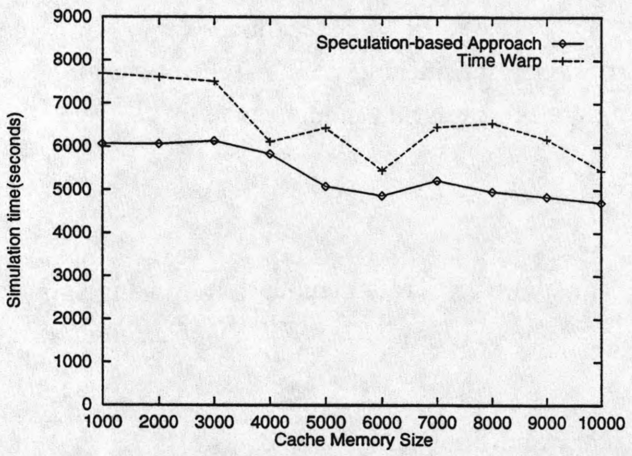

Figure 15: Execution Time

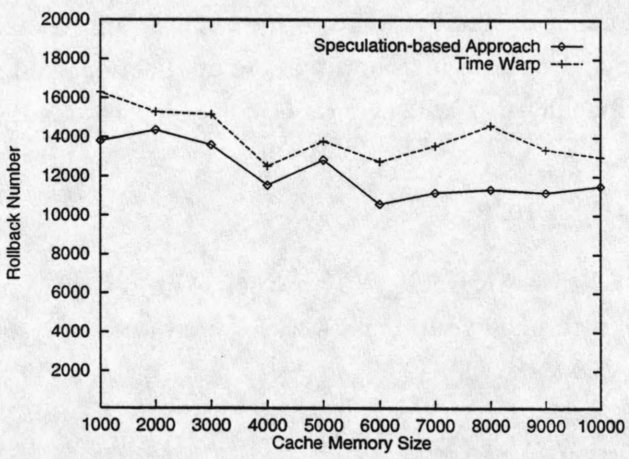

Figure 16: Rollback Number

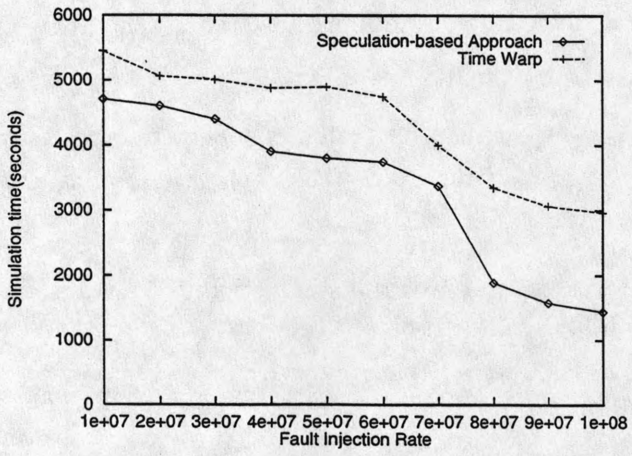

Figure 17: Execution Time

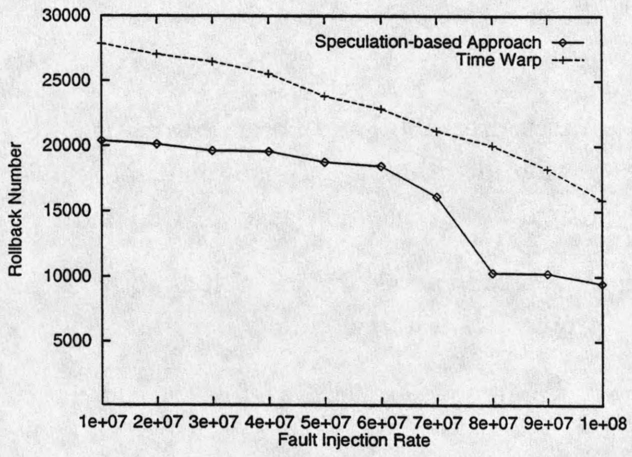

Figure 18: Rollback Number

infrequent. For example, in our detailed simulation of the I/O node, the error simulation is time-consuming, because the error detection/correction is simulated at low-level (i.e., data symbol level). With speculation, the time-consuming simulation for errors does not delay sending back the I/O responses, thereby, does not slow down the simulation progress.

Finally, the speculation-based method can introduce extra rollback if the correct simulation takes a different path than one predicted by the speculation. These instances of rollback are different than rollback due to causality violations intrinsic to the optimistic protocol, such as Time Warp. The percentage of rollbacks introduced by speculation is low in simulation of reliable systems, i.e., usually the error detection/correction coverage is high. Consequently, utilizing speculation to optimize event scheduling is definitely beneficial as speculation most likely matches the correct simulation path. In addition, the proposed speculation algorithm can be selectively implemented on logical processes that are most beneficial to the overall simulation. Despite the implementation overhead introduced, the method brings significant performance improvement to the simulation.

The paper concentrates on the simulation of a cluster system, however, the speculation-based method is by no means limited to a specific application. Therefore, our future work will concentrate on generalizing the approach to incorporate more applications and in improving its efficiency, for instance, developing a more efficient state saving mechanism which handles larger size simulation model.

References

[1] Rajive Bagrodia, Yu an Chen, Mario Gerla, and Bruce Kwan. Parallel simulation of a high-speed wormhole routing network. *Proceeding of the 1996 Workshop on Parallel and Distributed Simulation*, pages 47–56, 1996.

[2] S.J. Bavuso, J.B. Dugan, K.S. Trivedi, Rothman E.M., and Smith W.E. Analysis of typical fault-tolerant architectures using HARP. *IEEE Trans. on Reliability*, pages 176–185, 1987.

[3] Ram Chillarege. Fault and error latency under real workload - an experimental study. *Ph.D Thesis*, Electrical and Computer Engineering Department, University of Illinois at Urbana-Champaign, 1986.

[4] Alessandro Fabbri and Lorenzo Donatiello. SQTW: a mechanism for state-dependent parallel simulation. description and experimental study. *Proceeding of the 1997 Workshop on Parallel and Distributed Simulation*, pages 82–89, 1997.

[5] A. Ferscha and G. Chiola. Self-adaptive logical processes: the probabilistic distributed simulation protocol. *The 27th Annual Simulation Symposium*, pages 78–88, 1995.

[6] R. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, 1990.

[7] R. Fujimoto and David Nicol. State of the art in parallel simulation. *Proceedings of the 1992 Winter Simulation Conference*, pages 246–254, 1992.

[8] K.K. Goswami. Design for dependability: A simulation-based approach. *Ph.D Thesis*, Computer Science Department, University of Illinois at Urbana-Champaign, 1993.

[9] B. Groselfj and C. Tropper. The distributed simulation of clustered processes. *Distributed Computing*, pages 111–121, 1991.

[10] J.M. Hammersley and C.C. Handscomb. Monte Carlo methods. *Methuen and Co.LTD.*, 1964.

[11] D.O. Hamnes and Anand Tripathi. A comparative study of adaptive risk vs. adaptive aggressiveness control in parallel and distributed simulation. *Proceedings of the 29th Annual Simulation Symposium*, pages 90–96, 1996.

[12] P. Heidelberger. Fast simulation of rare events in queuing and reliability models. *ACM Transactions on Modeling and Computer Simulation*, pages 43–85, 1995.

[13] Robert W. Horst. Tnet: A reliable system area network. *IEEE MICRO*, pages 37–45, 1995.

[14] D.R. Jefferson. Virtual time. *ACM Transaction on Programming Language and System*, 7(3):404–425, 1985.

[15] Barry W. Johnson. *Design and Analysis of Fault Tolerant Digital Systems.* Addison Wesley, 1989.

[16] M. Kaaniche, L. L.Romano, Z. Kalbarczyk, R.K. Iyer, and R. Karcich. A hierarchical approach for dependability analysis of a commercial cached raid storage architecture. *The 28th Annual International Symposium on Fault-Tolerant Computing*, 1998.

[17] A.M. Law and W.D. Kelton. *Simulation Modeling and Analysis.* McGraw-Hill, New York, 1991.

[18] David M. Nicol. High performance parallelized discrete event simulation of stochastic queuing networks. *Proceedings of the 1988 Winter Simulation Conference*, pages 306–314, 1988.

[19] Victor F. Nicola, Nakayama M.K., and P. Heidelberger. Fast simulation of highly dependable systems with general failure and repair processes. *IEEE Transactions on Computers*, pages 1440–1452, 1993.

[20] David L. Peterson. New perspectives in DASD subsystem cache performance. *23rd International Conference for the Resource Management and Performance Evaluation of Enterprise Computing Systems*, 1997.

[21] D.P. Siewiorek and R.S. Swarz. *Reliable Computer Systems Design and Evaluation.* Digital Press, 1992.

[22] Simon J.E. Taylor and Thierry Delaitre. Estimating the benefit of the parallelism of discrete event simulation. *Proceedings of the 1995 Winter Simulation Conference*, pages 674–681, 1995.