*Proceedings of the 1998 Winter Simulation Conference*
*D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds.*

# A SPECULATION-BASED APPROACH FOR PERFORMANCE AND DEPENDABILITY ANALYSIS: A CASE STUDY

Yiqing Huang
Zbigniew T. Kalbarczyk
Ravishankar K. Iyer

Center for Reliable and High Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 W. Main Urbana, IL 61801, U.S.A.

## ABSTRACT

In this paper, we propose two speculation-based methods for fast and accurate simulation-based performance and dependability analysis of complex systems, incorporating detailed simulation of system components. The first approach applies to performance analysis and the second to dependability analysis. Our target example is a networked cluster with compute nodes and one I/O node. Detailed simulation of the cache subsystem of the I/O node is conducted, and more abstract simulation of the compute nodes and the switching network is performed. Performance measures obtained include cache miss ratio and cache subsystem access time. Dependability measures obtained include error coverage of EDAC code and error detection latency distribution of errors introduced to the cache components. The two methods are implemented on a network of workstations.

## 1 INTRODUCTION

In simulating complex systems, a detailed simulation of a subsystem is often needed to characterize the overall system/network performance and dependability. The detailed simulation allows to capture the system dynamics resulting from variability in the workload or the effect of different failure scenarios. However, detailed simulation is time-consuming. In this paper, we propose new speculation-based, distributed simulation methods to accelerate both performance and dependability simulation. Our approach uses optimistic simulation as the underlying mechanism. However, detailed component simulations slow down the overall execution. Using speculation, we can accelerate the overall simulation while obtaining application-specific measures from the detailed simulation. An example of such an application is the reliable, high-performance cluster system studied in this paper with a detailed simulation of the cache subsystem on the I/O node of the cluster.

The targeted cluster consists of compute nodes and an I/O node connected via ServerNet (Horst et al. 1995), a system area network developed by Tandem Computers. The I/O node supports a cache-based RAID storage architecture that is critical to the performance and dependability of the cluster. The RAID system is controlled and managed by an array controller that includes a cache subsystem. To analyze the impact of the storage subsystem on the cluster's performance and dependability, we developed detailed models for the data transfer inside the RAID cache subsystem and for the error detection and recovery mechanisms in the cache subsystem. The detailed models are incorporated into the cluster model as part of the overall simulation.

**Speculation for performance** is used to simulate the data transfer inside the cache subsystem in detail. For each I/O request arriving at the I/O node, the request type, i.e., read or write, and whether it is a cache hit or a cache miss are speculated. An execution path can be constructed by speculating the request to be a read hit, a read miss, a write hit, or a write miss. Speculative simulation of the request can follow any of the four paths. The simulation for each path is completed except for those segments that require the unspeculated information from incoming events such as the request arrival time. Upon receiving the incoming request, its request type and cache hit/miss information are used to select the matched path from the four speculated paths. Then, the simulation segments that are not completed during speculative simulation are

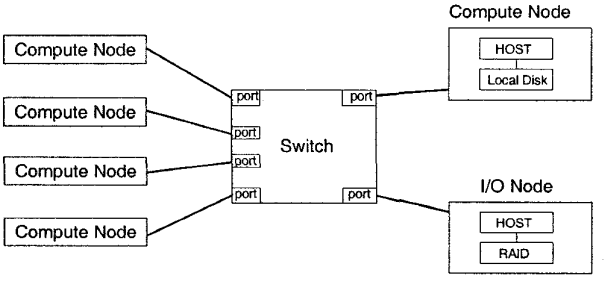


Figure 1: System Architecture

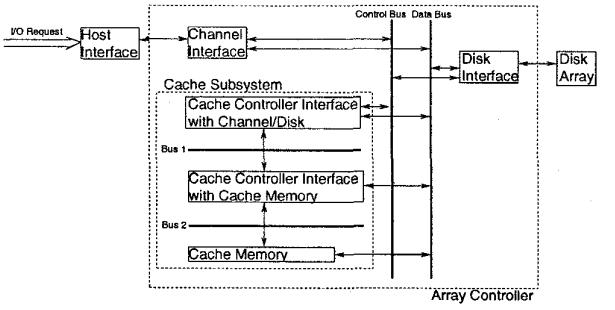


Figure 2: Array Controller Architecture and External Interfaces

finished using the unspeculated information from incoming events. If the request traffic to the I/O node is infrequent, the speculation can be conducted whenever the I/O node is idle waiting for the next incoming request.

**Speculation for dependability** is used to simulate the cache subsystem error recovery mechanisms in detail. The speculation can most often successfully predict the outcome of the error recovery detailed simulation based on the observation that errors are most likely detected and corrected in the I/O subsystem. In other words, we can most often assume that the system recovers from errors. Hence, the detailed simulation is postponed until an idle time slot is found during which the detailed simulation can be scheduled. In case the detailed simulation indicates a failure, a rollback is necessary.

In recent years, distributed simulation has been used widely for simulating complex systems (Ferscha and Chiola 1995, Fujimoto and Nicol 1992, Hamnes and Tripathi 1996, and Jefferson 1985). Studies concerning cache system in an I/O subsystem have focused on cache replacement policies and cache performance varying cache parameters such as cache size, cache block size (Karedla et al. 1994, Menon and Hartung 1988, and Smith 1985). In this paper, we propose speculation-based methods for simulating a cached RAID system, which have not been reported before in the literature.

The rest of the paper is organized as follows. Section 2 describes the architecture of the cluster system. In Section 3, speculation for performance is explained in detail, and speculation for dependability is summarized in the context of the simulated architecture. Performance results are shown in Section 4. Section 5 concludes the paper.

## 2 SYSTEM ARCHITECTURE

The architecture analyzed in this paper is a reliable, high-performance cluster system, shown in Fig. 1. The system consists of five compute nodes and one I/O node connected via a 6-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, it is crucial to the overall cluster performance. The I/O node supports RAID architecture, which consists of a collection of disk drives 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.

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 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 composed of two separate parts: the cache memory (CM) and the cache controller. The cache controller provides interfaces to the cache memory (CCICM) and to the channel/disk (CCICI). The communication link between the cache controller interface and the cache memory is a multidirectional bus depicted as Bus 2, and the link between the cache controller interface and the channel/disk is a multidirectional bus depicted as Bus 1.

Two replacement algorithms for the cache subsystem are evaluated: random replacement policy (RR) and least-recently-used replacement (LRU). Replacement strategies are adopted to discard tracks in the cache memory to make room for the new incoming tracks from the channel. RR randomly selects a track in the cache to discard. It does not take into account spatial and temporal locality in the data trace. Therefore, as the results show later, it behaves poorly. LRU selects the track in the cache used least in the recent past to be replaced. Implementing LRU requires recording track usage.

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 that occur during a cache operation. 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 interface 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 kinds of 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.

## 3 METHODOLOGY

In this section, we describe the speculation-based simulation methodologies: speculation for performance and speculation for dependability. The methods are illustrated using examples based on the cluster system presented in Section 2.

### 3.1 Speculation for Performance

Speculation for performance consists of three major steps, which are explained in the following. A simulation snapshot of a process from wall-clock time $t_i$ to $t_j$ is shown in Fig. 3. Event $e_1$ is scheduled to execute at time

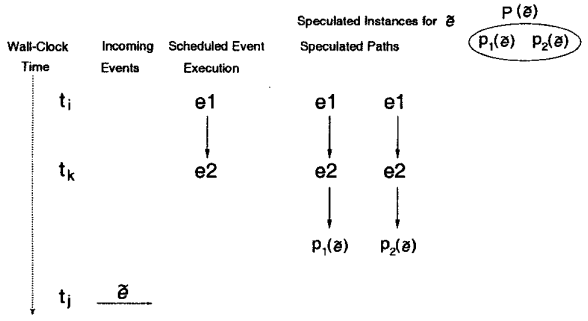


Figure 3: A Simulation Period on an LP Using Speculation for Performance

$t_i$. Event $e_1$ schedules an internal event $e_2$ to execute at time $t_k$. No events are scheduled after $e_2$. The process waits until $t_j$ when an incoming event $\tilde{e}$ arrives. Event $\tilde{e}$ is scheduled to execute at time $t_j$.

Now let us take a look at how the approach works. First, the process speculates information of the next arriving external event that allows the detailed simulation to proceed. To simplify the discussion, for the external event $\tilde{e}$ in Fig. 3, we have only two instances with different speculative information, $p_1(\tilde{e})$ and $p_2(\tilde{e})$. Notation $p(\tilde{e})$ represents a speculated instance of incoming event $\tilde{e}$. $p_1(\tilde{e})$ and $p_2(\tilde{e})$ determines two execution paths.

Second, if the processor is idle waiting for the next incoming event, based on the speculated instances of the next event, execution triggered by the next incoming event takes place for detailed simulation. In this case, the execution following the paths determined by $p_1(\tilde{e})$ and $p_2(\tilde{e})$ are both executed. The simulation for each path is completed except for those segments that require the unspeculated information of the incoming event.

Finally, after the arrival of $\tilde{e}$, it can be decided whether any of the two speculative execution matches the execution determined by the real event. If there exists a matched execution, it is selected as the correct execution. Then, incomplete simulation segments are finished using the unspeculated information from incoming event $\tilde{e}$.

#### 3.1.1 An Example

In this section, a sequence of event routines that simulate service of a speculated read request in the cluster system is adopted to illustrate the idea presented in the previous section. We follow the three steps given above. First the next incoming event is speculated, i.e., the next incoming I/O request. The request includes arrival time, the requested track id, the type read/write, and the information whether it is a cache hit/miss. The request type and whether the request is a cache hit/miss are speculated. The arrival time and the track id are not speculated because their speculation set would become too large.

Second, the speculated request can be a read hit, a read miss, a write hit, or a write miss. In the following, we assume for the purpose of the discussion that the speculated request is a read request. If the request is speculated to be a cache hit, three event routines are designed to simulate the steps involved in the cache hit: (a) the transmission of a disk track from CM to CCICM, (b) the transmission of a disk track from CCICM to CCICI, and (c) the transmission of a disk track from CCICI to CI. During step (a), the track is decomposed into a set of data blocks that are transferred to CCICM. In the execution without speculation, updating the simulated access time of the referenced track in the cache memory is necessary. However, without speculation of the track id, speculative execution is not able to perform the update. Thus, later, when the real request arrives, the update will be conducted. During steps (b), the track is decomposed into a set of

data blocks that are transferred to CCICI. During the data transmission, transfer using controller interface buffers are simulated and simulation time is advanced. During step (c), simulation time is advanced.

If the request is speculated to be a cache miss, an additional event routine is necessary to simulate the transmission of a disk track from the disk array to the cache memory before simulating (a), (b), and (c) above. For the cache miss, step (a) as presented above involves: i) selecting the evicted track used least, e.g., with the earliest reference time, ii) updating the data structure for sorting the reference time of tracks, and iii) setting the current time for the replacing track. Step (b) and (c) are the same as in the cache hit case.

Note that the speculation takes place whenever the I/O node waits for the next incoming I/O request. The speculative execution does not need the knowledge of the timestamp and the track id if the speculated request does not occur in parallel with other requests. If request traffic to the I/O node is infrequent, this can be usually satisfied.

Third, after the arrival of the request, the request is used to select the matched execution among the speculative execution paths. In this example, after the arrival, it can be decided whether it is a read request, whether it is a cache hit or miss, and whether the request occurs in parallel with other requests. Timestamp and track id of the request is used to complete the detailed simulation. An I/O response is sent out.

### 3.1.2 Implementation

The pseudocode of the algorithm of speculation for performance is listed in Fig. 4. The implementation is based on Time Warp (Jefferson 1985). First, definition and notation are introduced.

A simulation model SM is defined as $SM$ = ($C$, $LP$, $SF$). Each element is defined in the following:

**Channel Set** $C$: $C$ = { $c(i,j)$: $c(i,j)$ is a channel from logical process $i$ to logical process $j$ }. If process $i$ schedules events to process $j$ via message passing, a channel exists between processes $i$ and $j$. This channel is an input channel for process $i$ and an output channel for process $j$. $C$ contains all the channels in the model.

**Logical Process Set** $LP$: $LP$ consists of all the logical processes. Logical process $i$ ($LP_i$) is a set represented by $LP_i$ = ($LVT$, $IC$, $OC$, $IMQ$, $OMQ$, $SQ$, $EVL$) where:
1) $LVT$ (Local Virtual Time) is the local simulation clock that indicates how far the simulation of a process has progressed. The $LVT$ of process i is denoted as $LVT_i$.
2) $IC$ (Input Channel) = { $c(i,j)$: input channel from process i to process j }
3) $OC$ (Output Channel) = { $c(i,j)$ : output channel from process i to process j }
4) $IMQ$ (Input Message Queue) is a queue which stores the received incoming events at time $LVT$. Events are sorted in the increasing order of event timestamps.
5) $OMQ$ (Output Message Queue) stores the events on the logical process to be sent out to other processes at time $LVT$. Events are sorted in the increasing order of event timestamps.
6) $SQ$ (State Queue) stores the states at time $LVT$.
7) $EVL$ (Event List) maintains unprocessed event routines. Events are sorted in the nondecreasing order of timestamps. $TS(e)$ denotes the timestamp of event $e$.

**Simulation Function** $SF$ defines how the simulation is performed for each process. Under conventional distributed simulation algorithms, events are processed sequentially based on their timestamps on each logical process. With speculation for performance, events can be speculatively executed.

The speculative execution of incoming events takes place whenever event list $EVL$ is empty, i.e., the processor finishs the processing of existing events and waits for the incoming event. Steps 1 and 2 (see Section 3.1) of the approach is implemented from line 6 to 12. Step 3 is implemented from line 22 to 25.

## 3.2 Speculation for Dependability

Speculation for dependability is implemented using dynamic rescheduling of event routines at run-time on the LPs performing detailed simulation, e.g., the I/O node cache subsystem detailed simulation. The techniques are explained in the following.

**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. 2) Computation or manipulation of simulation data (*routineII*): This sub-event routine conducts computation and manipulates data associated with the computation, for example, data concerning 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 systems, *routineII* is usually computationally intensive. If the data manipulated by *routineII* does not 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

```
simulate_LP_i() {
1     Schedule initial events to EVL_i
2     while (1) {
3         if (next event available)
4             process next event from EVL_i
5         else /* EVL is empty */
6             if (speculative execution of the next event does
              not exist) {
7                 generate P(ē)
8                 for (each p(ē) ∈ P(ē)) {
9                     generate a new event list EVL'
                      containing p(ē)
10                    execute p(ē) and associated events
11                }
12            }
13        if (ē is received) {
14            if (ē is an anti-message) or (TS(ē) < LVT_i) {
15                restore to the immediate checkpoint
                  before TS(ē)
16                if (ē is an anti-message)
17                    cancel the messages in IMQ
                      associated with ē
18                else if (ē is an I/O request)
19                    schedule events associated with this
                      I/O request
20            }
21            else if (TS(ē) >= LVT_i) {
22                if (match (p(ē), ē) is true) {
23                    finish execution triggered by ē
                      not in the speculative execution
24                    adopt the matched EVL'
25                }
26                else {
27                    schedule ē
28                    cancel speculative execution of ē
29                    }
30            }
31        }
32        Send out an I/O response or anti-messages if any
      }
}
```

Figure 4: Speculation for Performance Algorithm

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 dynamic rescheduling is chosen for LPs with infrequent incoming messages. 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 to be detected and corrected in a reliable system.

**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 to right before the request with the *routineII*s that don't match the speculation. These are implemented based on the rollback mechanism of the optimistic protocol. Because errors in the track are not corrected, the I/O request needs to be retried to retrieve correct data after the rollback.

### 3.2.1 An Example

A sequence of event routines that simulate service of a cache hit read request is adopted to illustrate the speculation for dependability. Three event routines are designed to simulate the steps involved in servicing the cache hit read request: 1) the track transmission from CM to CCICM, 2) the track transmission from CCICM to CCICI, and 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 that 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* performs fault injection for this transmission, error detection/correction for data bytes in the track, and updating of the error statistics for the track. 3) *routineIII* sends out an I/O response if there is one.

In Fig. 5, 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.

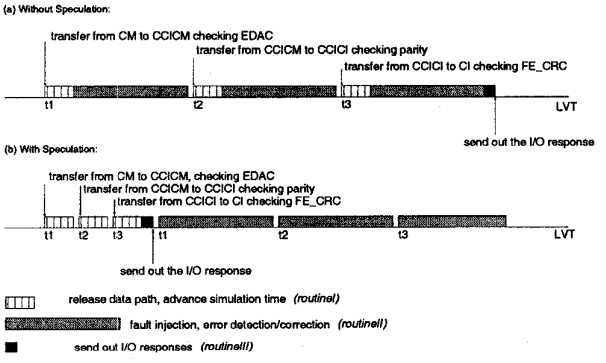


Figure 5: A Read Cache Hit with No Other Parallel Requests

Without speculation, the sub-event routines are not rescheduled, as shown in Fig. 5(a). In Fig. 5(b), the event routines are rescheduled based on the speculation that errors are corrected. *routine1* 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.

## 4 EXPERIMENTAL RESULTS

In this section, we present experimental results obtained from the distributed speculation-based simulation of the reliable cluster system described in Section 2. The cluster system is characterized by performance and dependability measures. Run-time performance of the speculation-based simulation employed to assess the cluster system performance is evaluated by comparing with sequential simulation and Time Warp. Simulation experiments are conducted on Sun Ultra1-170 workstations interconnected by 100 Mb/sec fast Ethernet. Message Passing Interface (MPI) is used as the communication layer for supporting communication between workstations running the simulation.

### 4.1 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. 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 200,000 and the cache memory size is 10,000 tracks.

### 4.2 Performance Measures of Cache Subsystem

Performance measures are obtained using speculation for performance. They include cache miss ratio and cache subsystem access time. Cache miss ratio is important for measuring cache performance. Two cache replacement algorithms are studied using our simulation, random replacement (RR) and least recently used replacement (LRU). The miss ratio is shown in Fig. 6. LRU performs better than RR for the cache size simulated. The reason is that it exploited locality of the data in the trace better than RR.

Cache subsystem access time is another performance measure obtained from the simulation. From Fig. 7, we can see that the access time decreases as the cache size increases. This is due to the fact that cache miss ratio decreases as the cache size increases. The access time using LRU strategy is always lower than RR strategy due to the same reason.

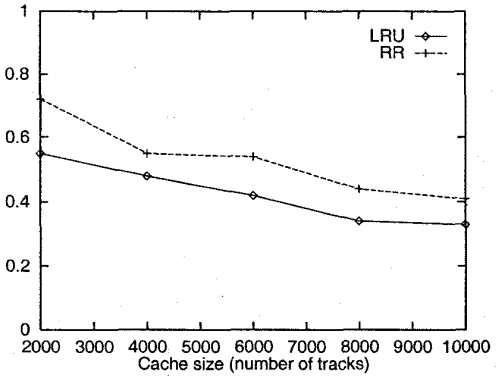


Figure 6: Cache Miss Ratio

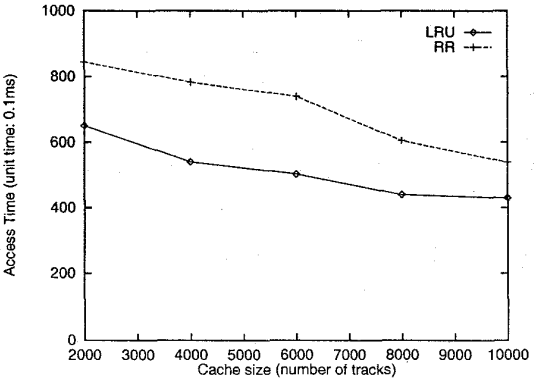


Figure 7: Cache Subsystem Access Time

### 4.3 Run-time Performance of the Speculation-based Simulation

In this subsection, we evaluate the speculation for performance approach using the cluster system model, and compare its run-time performance with sequential simulation and a traditional optimistic protocol, Time Warp of 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 for performance and Time Warp, the cluster system model with 6 nodes is partitioned and mapped to 2, 4 and 6 workstations.

Run-time performance numbers of the speculation for performance approach, Time Warp and sequential

Table 1: Performance of Sequential, Speculation for Performance, and Time Warp

| | Sequential | Speculation for Performance | | | Time Warp | | |
|---|---|---|---|---|---|---|---|
| Number of Workstations | 1 | 2 | 4 | 6 | 2 | 4 | 6 |
| Simulation Time (sec) | 24,271 | 17,688 | 10,284 | 5,891 | 19,573 | 14,710 | 8,956 |
| Speedup | 1 | 1.37 | 2.36 | 4.12 | 1.24 | 1.65 | 2.71 |

simulation are recorded in Table 1. The table presents: 1) Simulation time: the real execution time to complete the simulation of the simulated system, and 2) Speedup: the ratio between the sequential simulation time and the simulation time obtained from a distributed simulation.

The results from Table 1 demonstrate significant speedup of speculation for performance over sequential simulation. Our method yields as much as 69%(4.12/6) of ideal speedup (If a model is distributed onto n workstations, the ideal speedup is n) for the 6 workstation case. From Table 1, we observe that, compared with the speculation for performance approach, Time Warp offers less performance gain over sequential simulation. On 6 workstations, 45%(2.71/6) ideal speedup is observed compared with sequential simulation. The reason that limits Time Warp's performance is that the computation intensive events due to detailed simulation of cache subsystem on the I/O node needs to execute after the arrival of I/O requests, thus, leads to poor overlap of computation and communication.

### 4.4 Dependability Measures of Cache Subsystem

Dependability measures are obtained using speculative for dependability. They include: 1) error coverage of the EDAC code, reported in Fig. 8 and 2) error detection latency of errors injected into various cache components, reported in Fig. 9. The measures are used to assess the error detection/correction mechanisms of the cache subsystem.

Fig. 9 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), and 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. Results shown in Fig. 9 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.

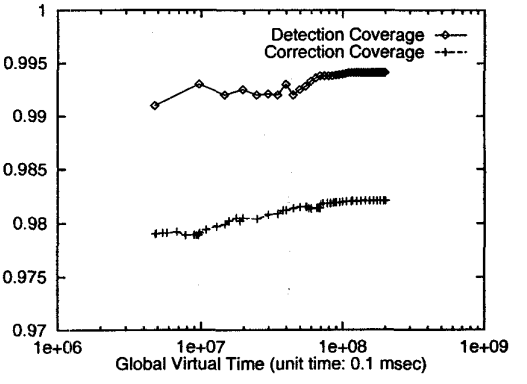


Figure 8: EDAC Coverage

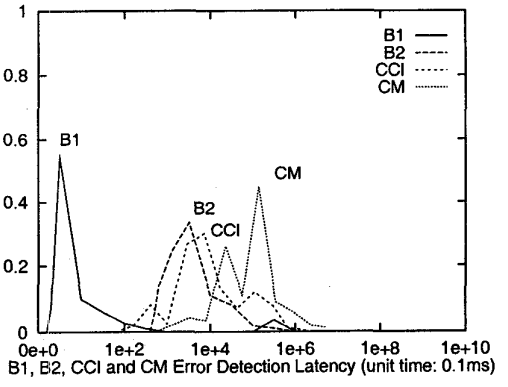


Figure 9: Error Detection Latency Distribution

## 5 CONCLUSIONS

We proposed new speculation-based, distributed simulation approaches for detailed evaluation of system behavior. The simulation methods are demonstrated and validated in the case study that analyzes the performance and dependability

of a cached RAID cluster system. Our experimental results demonstrate that using speculation-based simulation, valid and accurate performance and dependability measures can be obtained efficiently. Furthermore, run-time performance results show that the speculation for performance approach contributes to a significant reduction in the simulation time.

## ACKNOWLEDGMENTS

This research was supported in part by the U.S. Defense Advanced Research Projects Agency (DARPA) under contract DABT63-94-C-0045. The content of this paper does not necessarily reflect the position or policy of the US government, and no official endorsement should be inferred.

## REFERENCES

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

Fujimoto, R., and D. Nicol. 1992. State of the art in parallel simulation. *Proceedings of the 1992 Winter Simulation Conference*, 246–254.

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

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

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

Karedla, P., J. S. Love, and B. G. Wherry. 1994. Caching strategies to improve disk system. *IEEE Computer*, 38–46.

Menon, J., and M. Hartung. 1988. The IBM 3990 disk cache. *COMPCON'88*, 146-151.

Smith, A. J. 1985. Disk cache miss ratio analysis and design considerations. *ACM Transactions on Computer Systems*, Vol. 3, 161–203.

## AUTHOR BIOGRAPHIES

**YIQING HUANG** is a Ph.D candidate in the Computer Science Department and conducts research at the Center for Reliable and High Performance Computing at the University of Illinois at Urbana-Champaign. She received a B.S. degree in Computer Science from Tsinghua University, China and an M.S. degree in Computer Science from State University of New York at Stony Brook.

**ZBIGNIEW T. KALBARCZYK** is currently Visiting Research Assistant Professor at the Center for Reliable and High-Performance Computing in the University of Illinois at Urbana-Champaign. He holds an MS in Mechanical Engineering from the Technical University of Warsaw, an MS in Electronic Engineering and Ph.D. in Computer Science from the Technical University of Sofia, Bulgaria. His research interests include design, implementation, and validation of dependable computing systems including embedded systems and software-based, fault-tolerant distributed systems.

**RAVISHANKAR K. IYER** holds a joint appointment as Professor in the Departments of Electrical and Computer Engineering, Computer Science, and the Coordinated Science Laboratory at the University of Illinois at Urbana-Champaign. He is also Co-Director of the Center for Reliable and High-Performance Computing. Professor Iyer's research interests are in the area of reliable computing, measurement and evaluation, and automated design.

Prof. Iyer is an IEEE Computer Society Distinguished Visitor, an Associate Fellow of the American Institute for Aeronautics and Astronautics (AIAA), and a Fellow of the IEEE. In 1991, he received the Senior Humboldt Foundation Award for excellence in research and teaching. In 1993, he received the AIAA Information Systems Award and Medal for "fundamental and pioneering contributions towards the design, evaluation, and validation of dependable aerospace computing systems."