Modeling and Evaluating Design Alternatives for an On-Line Instrumentation System: A Case Study

Abdul Waheed, *Member*, *IEEE*, Diane T. Rover, *Member*, *IEEE*, and Jeffrey K. Hollingsworth, *Member*, *IEEE*

Abstract—This paper demonstrates the use of a model-based evaluation approach for instrumentation systems (ISs). The overall objective of this study is to provide early feedback to tool developers regarding IS overhead and performance; such feedback helps developers make appropriate design decisions about alternative system configurations and task scheduling policies. We consider three types of system architectures: network of workstations (NOW), symmetric multiprocessors (SMP), and massively parallel processing (MPP) systems. We develop a Resource OCCupancy (ROCC) model for an on-line IS for an existing tool and parameterize it for an IBM SP-2 platform. This model is simulated to answer several 'what if' questions regarding two policies to schedule instrumentation data forwarding: collect-and-forward (CF) and batch-and-forward (BF). In addition, this study investigates two alternatives for forwarding the instrumentation data: direct and binary tree forwarding for an MPP system. Simulation results indicate that the BF policy can significantly reduce the overhead and that the tree forwarding configuration exhibits desirable scalability characteristics for MPP systems. Initial measurement-based testing results indicate more than 60 percent reduction in the direct IS overhead when the BF policy was added to Paradyn parallel performance measurement tool.

Index Terms—Instrumentation system, resource occupancy model, workload characterization, parallel tools, parallel and distributed system, monitoring, intrusion.

1 Introduction

Application-level software instrumentation systems (ISs) collect runtime information from parallel and distributed systems. This information is collected to serve various purposes, for example, evaluation of program execution on high performance computing and communication (HPCC) systems [24], monitoring of distributed real-time control systems [3], [11], resource management for real-time systems [21], and administration of enterprise-wide transaction processing systems [1]. In each of these application domains, different demands may be placed on an IS and it should be designed accordingly. In this paper, we present a case study of IS design. We use specific measurements from Paradyn, an example of an on-line data gathering system that is found in performance analysis and program steering environments [23]. We apply a structured approach that is based on modeling and simulating the IS to answer several "what-if" questions regarding possible configurations and scheduling policies to collect and manage runtime data [28]. The Paradyn IS is enhanced based on the initial feedback provided by the modeling and simulation process. Measurement-based testing validates the simulation-based results and shows more than 60 percent reduction in the data collection overheads for two applications from the NAS benchmark suite executed on an IBM SP-2 system.

A rigorous system development process typically involves evaluation and testing prior to system production or usage. In IS development, formal evaluation of options for configuring modules, scheduling tasks, and instituting policies should occur early. Testing then validates these evaluation results and qualifies other functional and nonfunctional properties. Finally, the IS is deployed on real applications. Evaluation requires a model for the IS and adequate characterization of the workload that drives the model. We have developed a Resource OCCupancy (ROCC) modeling methodology that can account for interactions among different types of processes, such as application and IS, and their contention for shared system resources. This methodology is based on a coarse-grained workload characterization that does not require extensive and expensive measurements.

One may ask if such rigor is needed in IS development. The IS represents enabling technology of growing importance for effectively using parallel and distributed systems. However, users typically see a tool and not the IS and therefore may be unaware of IS overheads. Unfortunately, the IS can perturb the behavior of the application [20], degrading the performance of an instrumented application program from 10 percent to more than 50 percent according to various measurement-based studies [9], [22]. Perturbation can result from contention for system resources among application and instrumentation processes. With increasing sophistication of system software technologies (such as mul-

- *A. Waheed is with MRJ Technology Solutions, NASA Ames Research Center, Moffett Field, CA 94035. E-mail: waheed@nas.nasa.gov.*
- *D.T. Rover is with the Department of Electrical Engineering, Michigan State University, East Lansing, MI 48824. E-mail: rover@egr.msu.edu.*
- *J.K. Hollingsworth is with the Department of Computer Science, University of Maryland, College Park, MD 20742. E-mail: hollings@cs.umd.edu.*

Manuscript received 26 Dec. 1996; revised 19 Dec. 1997.
Recommended for acceptance by D. Eager.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 104740.

tithreading), an IS process is expected to manage and regulate its use of shared system resources [25]. Toward this end, tool developers have implemented adaptive IS management approaches; for instance, Paradyn's dynamic cost model [13] and Pablo's user-specified (static) tracing levels [24]. With these advancements come increased complexity and more design decisions. Modeling and early evaluation facilitates dealing with these design decisions.

A Resource OCCupancy (ROCC) model for an on-line IS is developed and parameterized in Section 2 using Paradyn as an example performance measurement tool. Initial 'back-of-the-envelope' calculations are obtained through application of operations analysis and presented in Appendix B. We simulate the ROCC model to answer a number of interesting "what-if" questions regarding the performance of the IS in Section 3. Modifications to the Paradyn IS design are tested in Section 4 to assess their impact on IS performance. We conclude with a discussion of the contributions of this work to the area of parallel tool development.

2 A Model for Paradyn IS

In this section, we introduce Paradyn and present a model for its IS. Paradyn is a tool for measuring the performance of large-scale parallel programs. Its goal is to provide detailed, flexible performance information without incurring the space and time overheads typically associated with trace-based tools [23]. The Paradyn parallel performance measurement tool runs on TMC CM-5, IBM SP-2, and clusters of Unix workstations. Our model of the Paradyn IS is sufficiently general to be useful for any of these platforms and is representative of many on-line measurement systems. However, in this paper we use parameters based on the IBM SP-2 unless otherwise noted. The tool consists of the main Paradyn process, one or more Paradyn daemons, and external visualization processes.

The main Paradyn process is the central part of the tool, which is implemented as a multithreaded process. It includes the Performance Consultant, Data Manager, and User Interface Manager. The Data Manager handles requests from other threads for data collection, delivers performance data from the Paradyn daemon(s), and distributes performance metrics. The User Interface Manager provides visual access to the system's main controls and performance data. The Performance Consultant controls the automated search for performance problems, requesting and receiving performance data from the Data Manager.

Paradyn daemons are responsible for inserting the requested instrumentation into the executing processes being monitored. The Paradyn IS supports the W3 search model implemented by the Performance Consultant for on-the-fly bottleneck searching by periodically providing instrumentation data to the main Paradyn process [12]. Required instrumentation data samples are collected from the application processes executing on each node of the system. These samples are collected by the local Paradyn daemon (Pd) through Unix pipes, which forwards them to the main process.

2.1 Modeling Objectives and Metrics

The Paradyn IS can be represented by a queuing network model, as shown in Fig. 1. It consists of several sets of identical subnetworks representing a local Paradyn daemon and application processes. We assume that the subnetworks at every node in the concurrent system show identical behavior in terms of sharing local resources during the execution of an SPMD program. Fig. 1 highlights the performance data collection and forwarding activities of a Paradyn daemon on a node. These IS activities are central to Paradyn's support for on-line analysis of performance bottlenecks in long-running application programs. However, they may adversely affect application program performance, since they compete with application processes for shared system resources. The objectives of our modeling include: comparing alternatives for IS management policies and configurations; evaluating IS overheads due to resource sharing; identifying any IS-induced performance bottlenecks; and determining desirable operating conditions for the IS.

2.1.1 Scheduling Policies for Data Forwarding

Two possible options for a Paradyn daemon to schedule data collection and data forwarding at a node are collect-and-forward (CF) and batch-and-forward (BF). As illustrated in Fig. 2, under the CF scheduling policy, the Paradyn daemon collects a sample from an instrumented application process and immediately forwards it to the main process. Under the BF policy, the Pd collects a sample from the application process and stores it in a buffer until a batch of an appropriate number of samples is accumulated and then forwarded to the main Paradyn process.

2.1.2 IS Configurations for Data Forwarding

In the case of using the Paradyn IS on a Massively Parallel Processing (MPP) system, we consider two options for forwarding the instrumentation data from the Paradyn daemon to the main Paradyn process: direct forwarding and binary tree forwarding. Under the configuration for direct forwarding, a Paradyn daemon directly forwards one or more samples to the main Paradyn process. With the binary tree forwarding scheme, the system nodes are logically arranged as a binary tree; every Paradyn daemon running on a nonleaf node receives, processes, and merges the samples or batches from the Paradyn daemons running on its two children nodes. Fig. 3 illustrates the two configurations.

2.1.3 Metrics

Two performance metrics are of interest for this study: average direct overhead due to IS modules and monitoring latency of data forwarding. Average direct overhead represents the occupancy time of a shared system resource by the IS modules, which is averaged over all the system nodes. A lower value of the direct overhead is desirable. Direct overhead quantifies the contention between application and IS processes for the shared resources on a particular node of the system. Monitoring latency has been defined by Schwan et al. as the amount of time between the generation of instrumentation data and its receipt at a logically central collection facility (in our case, the main Paradyn process) [9]. Monitoring latency impacts the main Paradyn process, since a steady flow of data samples from individual system nodes is needed to allow the bottleneck searching algorithm to work properly. In order to quantify the IS intrusion to the

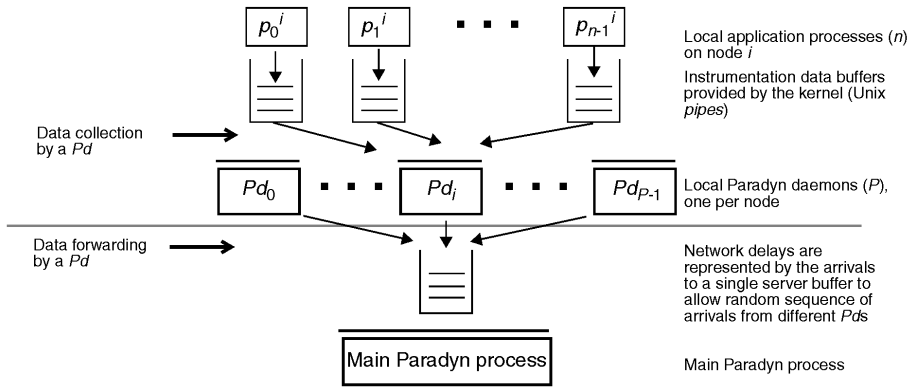


Fig. 1. A model for the Paradyn instrumentation system. The distributed system consists of P nodes and each node may have up to n instrumented application processes.

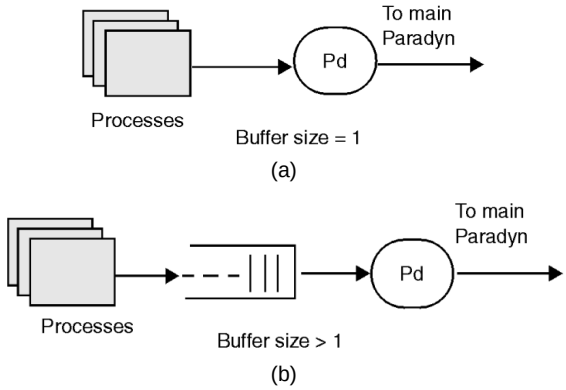


Fig. 2. Two policies for scheduling data collection and forwarding: (a) collect-and-forward (CF) policy, and (b) batch-and-forward (BF) policy.

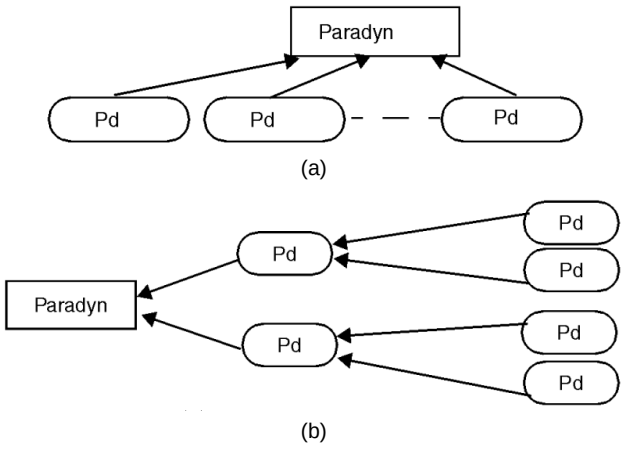


Fig. 3. Two configuration for data forwarding for an MPP implementation of the Paradyn IS: (a) direct forwarding, and (b) binary tree forwarding.

application, we calculate the application CPU utilization per node, with and without instrumentation. Simulation-based experiments presented in Section 3 calculate these metrics to help answer a number of "what-if" questions.

2.2 Resource Occupancy Model

This section introduces the Resource OCCupancy (ROCC) model and its application to isolating the overhead due to nondeterministic sharing of resources between the Paradyn IS and application processes [29]. The ROCC model, founded on well-known computer system modeling techniques, consists of three components: system resources, requests, and management policies. Resources are shared among (instrumented) application processes, other user and system processes, and IS processes; for example, CPU, network, and I/O devices. Requests are demands from application, other user, and IS processes to occupy system resources during the execution of an instrumented application program. A request to occupy a resource specifies the amount of time needed for a single (coarse-grain) computation, communication, or I/O step of a process. Management policies involve scheduling of system resources to fulfill the occupancy requirements of the processes. We identify a series of coarse-grained states to characterize each process, their dependences on the states of other processes, and occupancy requirements corresponding to each state. Fig. 4 depicts the ROCC model with local and global levels of detail. It includes two types of resources of interest for the Paradyn IS: CPU and a network. Each CPU is shared by three types of processes: application, IS, and other user processes.

Due to the interactions among different types of processes at the same node and IS processes at multiple nodes, it is impractical to solve the ROCC model analytically. Therefore, simulation is used. The execution of the ROCC model for the Paradyn IS relies on a workload characterization of the target system, which in turn, relies on measurement-based information from the specific system [5], [14].

2.3 Workload Characterization

The workload characterization for this study has two objectives:

1) to determine representative behavior of each process of interest (i.e., application, IS, and other user/system processes) at a system node (see Section 2.3.1); and
2) to fit appropriate theoretical probability distributions to the lengths of resource occupancy requests corresponding to the states of each of these processes (see Section 2.3.2).

2.3.1 Process Model

After a Unix process is admitted, it can be in one of the following states: Ready, Running, Communication, or Blocked (for I/O). The process can be preempted by the operating system to ensure fair scheduling of multiple processes sharing

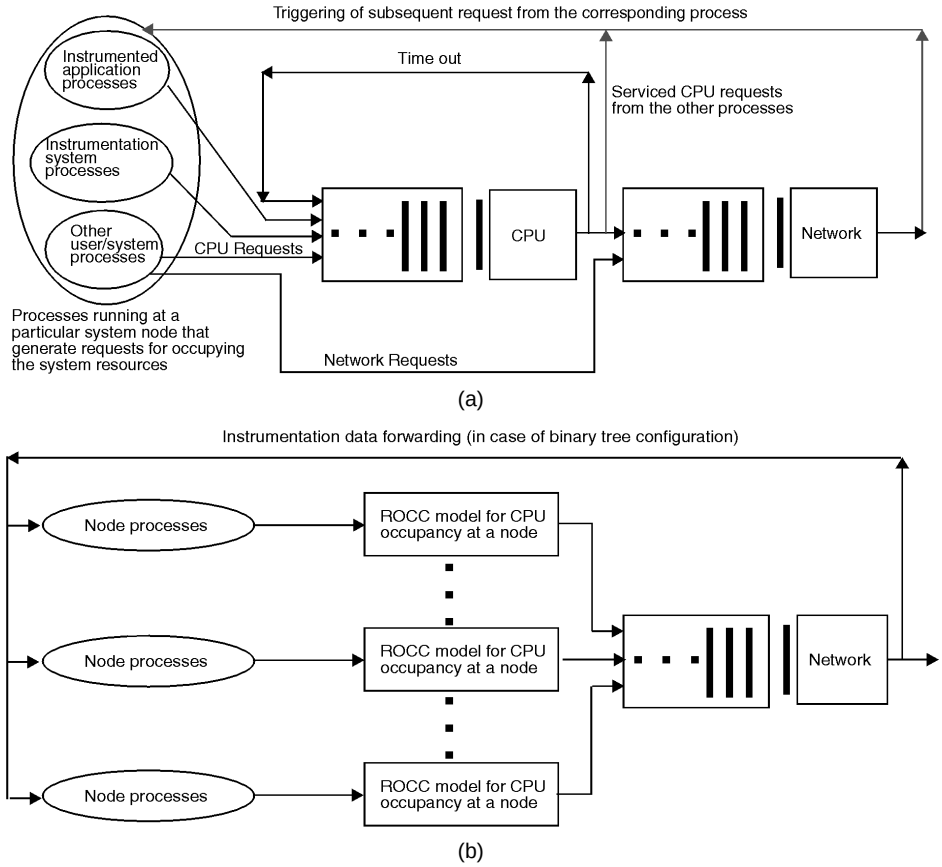


Fig. 4. The resource occupancy model for the Paradyn IS with: (a) local (ROCC model for a paraticular system node, and (b) global (ROCC model for the entire system) levels of detail.

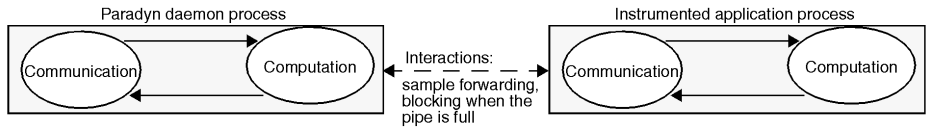


Fig. 5. A process model based on alternating computation and communication states of two types of interacting workloads.

the CPU. After specified intervals of time (in case of sampling) or after occurrence of an event of interest (in case of tracing), such as spawning a new process, instrumentation data are collected from the process and forwarded over the network to the main Paradyn process via a Paradyn daemon.

In order to reduce the number of states in the process behavior model and hence the level of complexity, we group several states into a representative state. The simplified model, shown in Fig. 5, considers only two states of process activity: Computation and Communication. This simplification facilitates obtaining measurements without any special operating system instrumentation. This characterization also considers the interactions among states across different processes. For instance, an instrumented application process interacts with the local Paradyn daemon process either by forwarding a sample to it or by blocking (via the operating system) if the pipe is full. The Computation and Communication states require the use of the CPU and network resources, respectively. The *Computation* state is associated with the Running state of the Unix process model. Similarly, the Communication state is associated with data collection, network file service (NFS), and communication activities with other system nodes. Measurements regarding these two states of the simplified model are conveniently obtained by tracing the application programs. The model provides sufficient information to characterize the workload when applied in conjunction with the resource occupancy model.

2.3.2 Distribution of Resource Occupancy Requests

In order to drive simulation-based experiments, computational workload is often represented in one of three forms:

1) actual source code;
2) detailed traces of system-level activity for trace-driven simulations; and
3) probability density functions (pdfs) with specific parameters. These methodologies are appropriate to address different requirements of workload accuracy, level of detail, and granularity.

Actual application programs are often used for realistic simulation of computer architectures at early design stages [18], called direct-execution simulation. This workload characterization requires the consideration of low-level architec-

TABLE 1
SUMMARY OF STATISTICS OBTAINED FROM MEASUREMENTS OF *NAS* BENCHMARK *PVMBT* ON AN SP-2

	CPU Occupancy (msec)				Network Occupancy (msec)			
Process Type	Mean	St. Dev.	min	max	Mean	St. Dev.	min	max
Application process	2,213	3,034	9	10,718	223	95	48	5,241
Paradyn daemon	267	197	11	6,923	71	109	31	816
PVM daemon	294	206	9	1,662	58	59	36	5,169
Other processes	367	819	8	9,746	92	80	8	198
Main paradyn process	3,208	3,287	11	10,661	214	451	46	4,776

ture and operating system details. Since an IS measures an entire application, direct-execution simulation yields unnecessary fidelity and overhead. Therefore, an IS modeling and evaluation effort may become prohibitively complex and time-consuming using direct-execution simulation. Fine-grained information represented by traces of low-level system activities is typically used for analyzing memory access behavior and memory management policies [7]. A trace represents an execution of a specific application on a specific system and cannot be generalized for a class of applications on a variety of platforms. Using a stochastic model offers greater flexibility by depicting the general behavior of a class of applications in a way that is representative of a range of applications on different platforms [19]. Using this methodology, different applications and platforms are represented by varying key parameters of the system resources and distributions of workloads that utilize those resources. We selected this methodology for IS evaluation due to its generality and extensibility to address different types of workloads and platforms [31].

Trace data generated by the IBM SP-2's AIX operating system tracing facility are the basis for the workload characterization. We used the trace data obtained by executing PVM implementation of the NAS benchmarks BT, LU, FT, CG, and MG on the SP-2 system [27]. Only BT represents a full scientific application while others are kernel benchmarks. Therefore, we focus on the detailed measurements obtained by executing BT before considering other benchmarks. Table 1 presents a summary of the statistics for CPU and network occupancy by various processes for BT (*pvmbt*).

We apply standard distribution fitting techniques to determine theoretical probability density functions that match the lengths of resource occupancy requests corresponding to the states of the processes during the execution of BT [19]. Fig. 6 shows the histograms and probability density functions for the lengths of CPU and network occupancy requests (in (a) and (b), respectively) by the application (NAS benchmark) process. We randomly selected fifty samples of CPU and network occupancy requests using an algorithm provided in [16]. Selecting a relatively small and equally representative sample size is desirable to determine meaningful goodness of fit statistics [19]. We use the Kolmogorov-Smirnov (K-S) goodness of fit test statistic to quantitatively measure the differences between a theoretical pdf and an observed pdf. The K-S test is based on a statistic, called the K-S statistic, which is equal to the absolute value of the largest difference between an observed and a theoretical pdf. Thus, a smaller value of the K-S statistic for a particular distribution indicates a good fit. See [19] for further details about K-S goodness of fit test.

For CPU requests (Fig. 6a), despite the differences, the lognormal pdf is the best match. This is confirmed by the minimum value of the K-S statistic for the lognormal pdf compared to the exponential and Weibull pdfs. For network requests by application processes (Fig. 6b), an exponential distribution yields the best fit.

In order to verify the above characterization of application's CPU requests, we instrumented and analyzed other NAS benchmarks. The histograms and theoretical pdfs of the lengths of CPU requests from PVM implementations of NAS benchmarks LU, FT, CG, and MG are presented in Fig. 7. All of these benchmarks are characterized by:

1) a very large number of requests that require relatively short lengths of CPU time; and
2) a significant number (which is particularly noticeable for FT and MG benchmarks in Fig. 7b and 7d, respectively) of requests that require the full CPU quantum.

This characterization supports our choice of using lognormal distribution to represent the lengths of CPU occupancy requests generated by compute-intensive scientific workloads. Specific features of applications, such as compute-intensive vs. communication-intensive, can be controlled by selecting appropriate parameter values for the pdfs.

2.4 Model Parameterization and Validation

The workload characterization presented in the preceding section yields suitable ranges of parameters for the ROCC model for the Paradyn IS, as listed in Table 5 in Appendix A. In order to validate the ROCC simulation model, we parameterized it using measurements obtained by executing pvmbt on an IBM SP-2 system. Table 2 compares the CPU time for the NAS benchmark and Paradyn daemon during the execution of the program using measurement and simulation. It is clear that the simulation model-based results follow the measurement-based results for pvmbt. Therefore, using the selected range of parameters, the model can be simulated to answer "what if" questions for different workload features and platform combinations, which we consider in Section 3.

TABLE 2
COMPARISON OF MEASUREMENTS OF NAS BENCHMARK *PVMBT* ON AN SP-2 WITH THE SIMULATION RESULTS OF THE SAME CASE

Basis of Experiment	Application CPU Time (sec)	Pd CPU Time (sec)
Measurement	85.71	0.74
Simulation	87.96	0.59

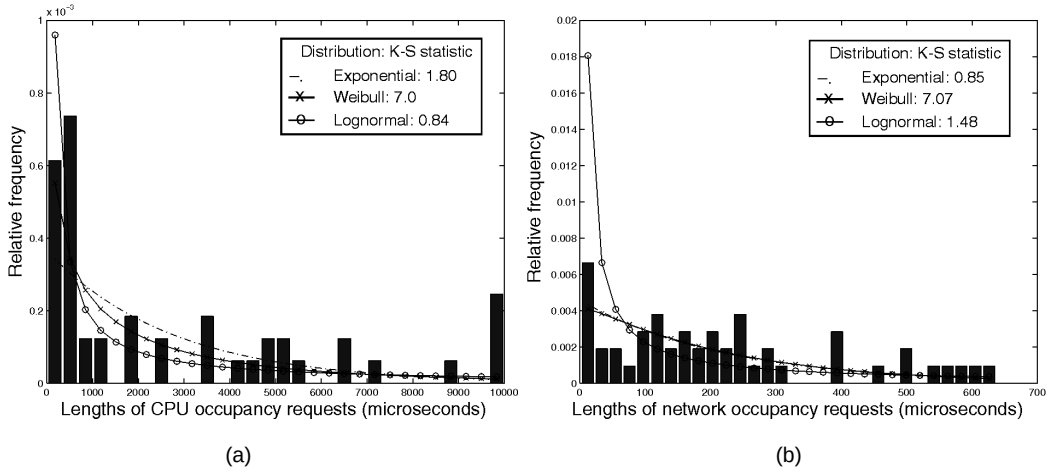


Fig. 6. Histograms and theoretical pdfs of the lengths of: (a) CPU and (b) network occupancy requests from the application process. K-S statistics represent the differences from theoretical distributions.

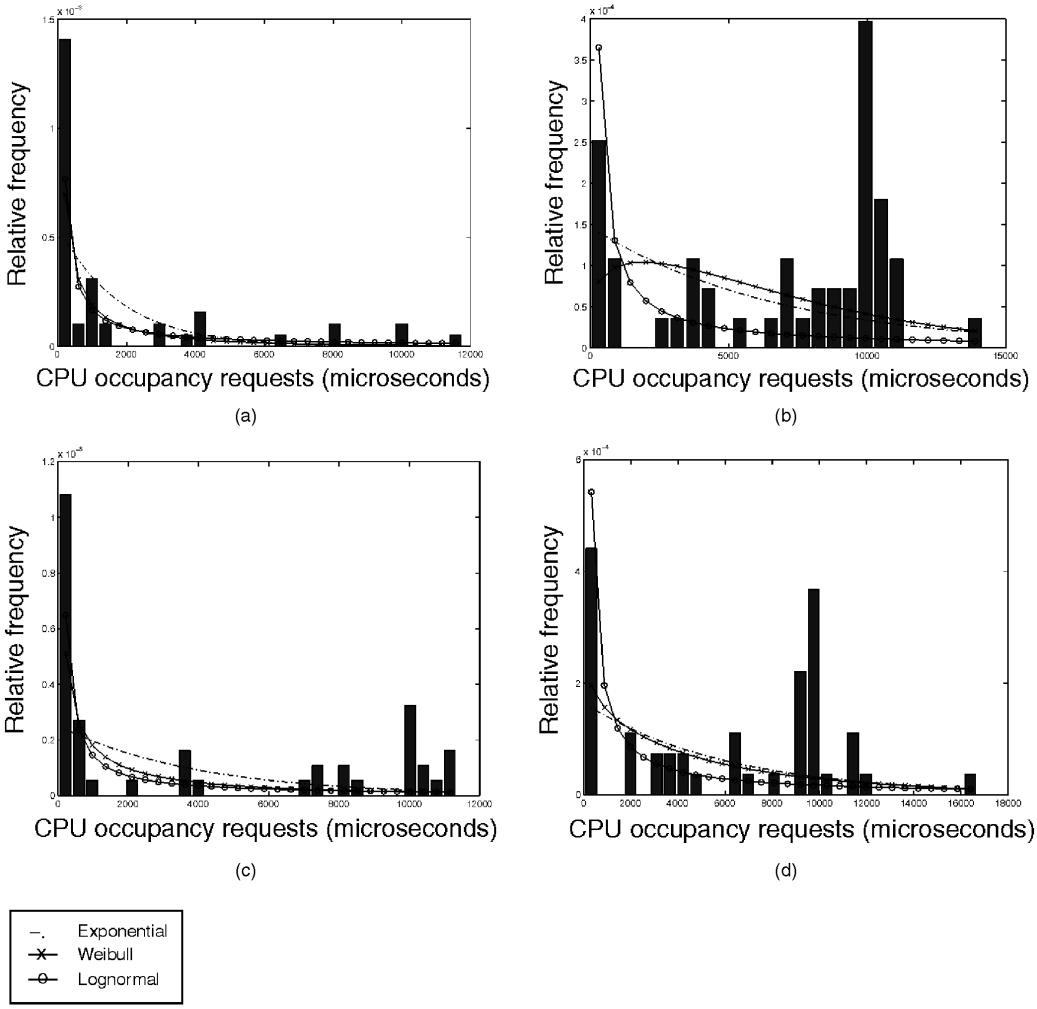


Fig. 7. Histograms and corresponding theoretical pdfs for lengths of: CPU occupancy requests from NAS benchmarks. (a) LU; (b) FT; (c) CG; and (d) MG.

3 Simulation-Based Experiments

We have completed both analytic and simulation studies of the ROCC model. A summary of the analytic study is given in Appendix B. In this section, we compare possible configurations and management policies for the Paradyn IS using simulation-based evaluation of the ROCC model. Simulation-based evaluation is more accurate than analytical approaches because we account for the interdependences between the

application and IS workloads and details of system functionality. We keep this evaluation process focused by posing specific "what-if"questions that are of interest to the developers and users of the IS. This focus is further refined by using the technique of principal component analysis (PCA [15]) to determine the system parameters and their combinations that can significantly affect the selected IS performance metrics.

3.1 Experimental Setup

Simulation-based experiments are based on three types of system architectures: NOW, SMP, and MPP. We make minor modifications in the ROCC model to accommodate the specific characteristics of each of these configurations. In the case of a NOW system, each node has one CPU and the nodes are interconnected via a switch-based or a shared network. For an SMP, multiple CPUs are connected through a bus. An MPP system is similar to the NOW system but has a multistage switched network and typically consists of a larger number of nodes. We parameterized the ROCC model for an IBM SP-2 system, which is closer to the NOW configuration. Nevertheless we use the modified ROCC models to extend the scope of the Paradyn IS evaluation to the SMP and MPP systems. We emphasize that the simulation experiments do not represent any specific application that was used for generic characterization of the workload. Suitable parameters are selected for workload components and system resources to investigate each "what-if" question of interest.

In answering various "what-if" questions regarding the Paradyn IS management and configuration, our simulation experiments are designed to analyze the effects of six parameters (factors):

1) *number of concurrent system nodes*: the number of NOW, SMP, or MPP system nodes that execute the instrumented application as well as the IS processes;
2) *sampling period*: length of wall-clock time between two successive collections of performance data samples from an instrumented application process;
3) *number of local application processes*: number of application processes running on one node of the parallel / distributed system;
4) *forwarding policy*: the policy implemented by the Paradyn daemon at each node to forward instrumentation data samples to the main Paradyn process;
5) *application type*: compute- or communication-intensive (determined by the network occupancy requirement and frequency of synchronization barrier operations); and
6) *network configuration*: direct or binary tree (logical) configuration of the nodes to forward the instrumentation data from a Paradyn daemon to the main Paradyn process.

For each system architecture type, we use a subset of these factors for simulation-based experiments. We use a $2^k r$ factorial design technique for the simulation-based experiments, where k is the number of factors of interest for a given case, r is the number of repetitions of each experiment, and each factor can assume one of two possible values [15]. For these experiments, we select k = 4 factors and r = 50 repetitions, and the mean values of the performance metrics of interest are derived within 90 percent confidence intervals from a sample of fifty values. This approach helps reduce the variance (or "noise") in the results; thus any differences among the performance metrics under varying IS configurations and management policies are clarified.

3.2 Principal Component Analysis

For each of the three system architecture types, we supplement the $2^k r$ factorial experiment design technique with principal component analysis (PCA) to assess the sensitivity of the performance metrics to selected model parameters (factors) [15]. With multiple factors, we cannot assume that each acts independently on the system under test (i.e., the IS). PCA helps determine the relative importance of individual factors, as well as their interdependences. Instead of evaluating the metrics for all possible combinations of the factors for each "what-if" question, we use a subset of combinations that are deemed important by the PCA.

We apply $2^k r$ factorial experiment design by selecting two values at the opposite ends of the range of possible values for each factor. For the NOW architecture, we assume that the system nodes are connected through a shared network (Ethernet). Each node runs an application process and a Paradyn daemon. One of the nodes also executes the main Paradyn process. Paradyn daemons on individual nodes directly forward the instrumentation data to the main process. We select a batch size of 32 samples for the BF policy (based on analysis beyond the scope of this paper; see [30]). For compute-intensive applications, the mean network occupancy requirement is arbitrarily set at 200 msec; and for communication-intensive applications, 2,000 msec. Four factors of interest in this case are: number of nodes, sampling period, forwarding policy, and application type. Applying the $2^k r$ factorial design technique, we conduct 16 simulation experiments, obtaining the results shown in Table 3. Four factors produce 15 combinations that affect a metric: four for individual factors; six for the combinations of two factors; four for the combinations of three factors; and one for the combination of four factors.

The bar graphs in Fig. 8 present the results of the principal component analysis. Clearly, the sampling period (labeled as B) is the single most important factor that affects the direct overhead of the Paradyn daemon, followed by the data forwarding policy (C), and the combination of the two (BC). The data forwarding policy (C) and number of nodes (A) are the most important factors affecting monitoring latency. Thus, a further investigation of the IS behavior with respect to the sampling period (B), the number of nodes (A), and the data forwarding policy (C) is justified.

PCA for the SMP and MPP architectures is conducted in a similar manner but with slightly different sets of factors. The final results of this analysis are depicted in Fig. 9. Fig. 9a shows the results of the PCA for an SMP architecture. The number of nodes (labeled as A) is the most important factor that affects the direct overhead of the Paradyn IS (i.e., daemon and the main process), followed by the forwarding policy (C) and the sampling period (B). The data forwarding policy (C), the number of nodes (A), and the combinations of the two (AC) are the most important factors affecting monitoring latency. Fig. 9b shows the results of PCA for an

TABLE 3
RESULTS OF SIMULATION EXPERIMENTS FOR THE NOW ARCHITECTURE

Parameters			Compute-Intensive Application		Communication-Intensive Application	
No. of Nodes	Sampling Period (msec)	Forwarding Policy	Pd CPU Time per Node (sec)	Monitoring Latency per Received Sample (msec)	Pd CPU Time per Node (sec)	Monitoring Latency per Received Sample (msec)
2	5	CF	5.33	3.52	5.34	2.83
2	50	CF	0.54	0.07	0.54	0.07
32	5	CF	5.34	0.07	5.34	2.92
32	50	CF	0.53	5.42	0.53	4.63
2	5	BF	2.48	0.08	2.48	0.08
2	50	BF	0.21	0.07	0.21	0.07
32	5	BF	1.78	0.68	1.78	0.70
32	50	BF	0.22	0.94	0.20	0.88

Factor	Label	Values
Number of nodes	A	2 or 32
Sampling period	B	5 or 50
Forwarding policy	C	CF or BF
Application type	D	Compute- or communication-intensive

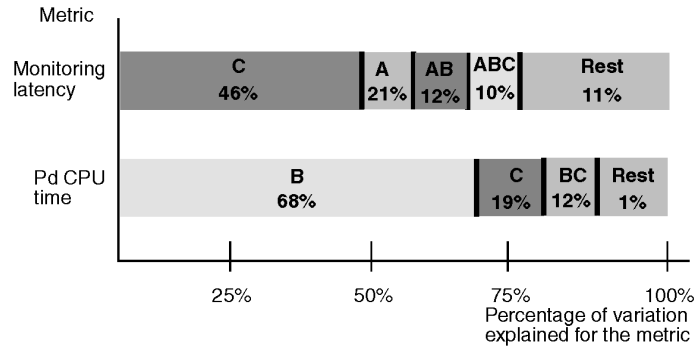
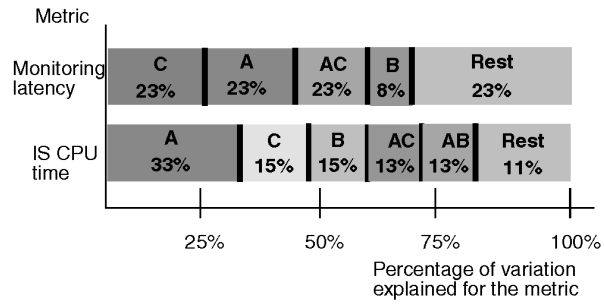


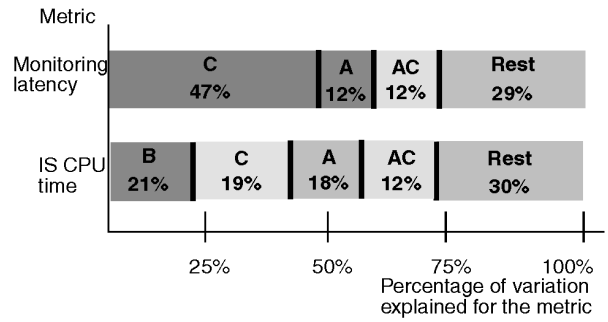
Fig. 8. Results of principal component analysis of four factors and their combinations for the NOW system.

Factor	Label	Values
Number of nodes	A	2 or 32
Sampling period	B	5 or 50
Forwarding policy	C	CF or BF
Application type	D	Compute- or communication-intensive



(a)

Factor	Label	Values
Number of nodes	A	2 or 32
Sampling period	B	5 or 50
Forwarding policy	C	CF or BF
Network configuration	D	Direct or binary tree



(b)

Fig. 9. Results of PCA for (a) SMP (with direct forwarding configurations), and (b) MPP (with compute intensive applications) architectures for four factors and their combinations.

MPP architecture. The sampling period (B), the forwarding policy (C), and number of system nodes (A) are equally important factors affecting the direct overhead of the Paradyn IS, followed by the forwarding policy (C). The forwarding policy (C) and the number of nodes (A) are the most important factors affecting monitoring latency.

In summary, PCA directs us to focus on the following features, in order of importance: B, C, A for NOW; A, C for SMP; and C, A, B for MPP architecture. PCA indicates that monitoring latency is most affected by forwarding policy and number of nodes; and IS overhead by sampling period and forwarding policy as well as number of nodes in SMP and MPP cases.

3.3 Investigation of "what-if" Questions

In this section, we present simulation-based results that answer four specific "what-if" questions using the ROCC model. These questions are explored in order on the NOW, SMP, and MPP architectures. These questions are related to the forwarding policy, use of multiple Paradyn daemons, and logical network configurations for data forwarding.

3.3.1 NOW Architecture: What are the Effects of Forwarding Policy with Varying the Number of Nodes and Sampling Periods?

The PCA in Section 3.2 shows that the choice of data forwarding policy significantly impacts the IS overhead. In this section, we compare the CF and BF policies by varying the number of system nodes and sampling periods. The simulation results, depicted in Fig. 10, lead to following observations:

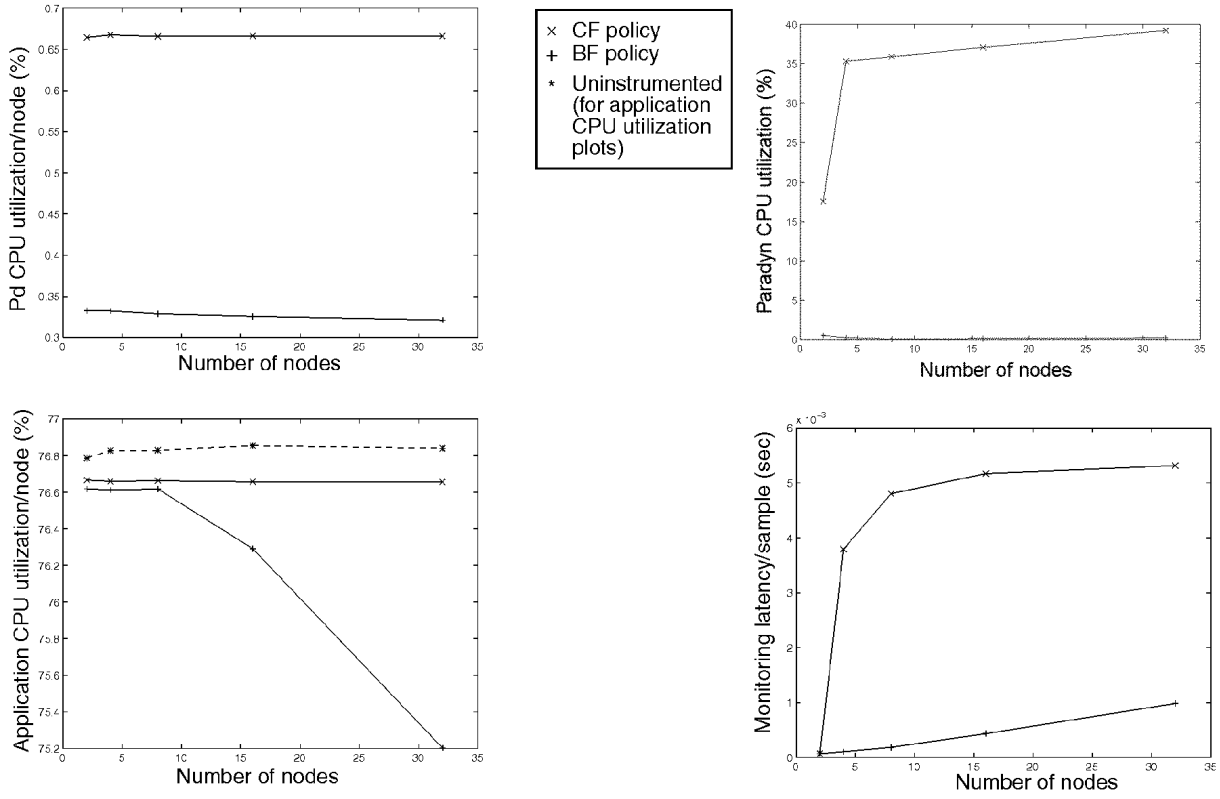
- Although the direct overhead of Paradyn daemon CPU utilization does not vary with the number of system nodes due to its localized nature, Fig. 10a shows that the BF policy incurs lower overhead. The CPU overhead by the main Paradyn process under the CF policy increases with the number of nodes due to more data samples forwarded to it. However, this overhead is significantly lower under the BF policy since the same samples are aggregated into fewer packets. Monitoring latency is also lower under the BF policy because on the aggregate more data can be transferred in a shorter time.
- Fig. 10b shows that the monitoring latency is not significantly affected by variations in the sampling period. The direct IS overhead and intrusion to the application decrease with increasing sampling period. As sampling period increases, the application CPU utilization approaches the uninstrumented level.
- The application CPU utilization significantly decreases at sampling periods less than 4 msec (see the lower left plot of (b)). Therefore, neither CF nor BF policy can support more than 250 samples per second.
- The behavior depicted by the simulation results in Fig. 10 is generally consistent with the analytic results (see Fig. 17 in Appendix B). Differences are due to the approximate nature of analytic calculations that do not accurately consider interdependences and resource contentions among the workloads. These differences are elaborated in Section 3.3.2. Simulation of the ROCC model accurately accounts for the resource contentions according to the scheduling policies used by the operating system.

These results indicate that the BF data forwarding policy outperforms the CF policy with respect to both direct overhead and monitoring latency. This is also true for the SMP and MPP architecture (see [30]). Therefore, we consider only the BF policy in the following sections.

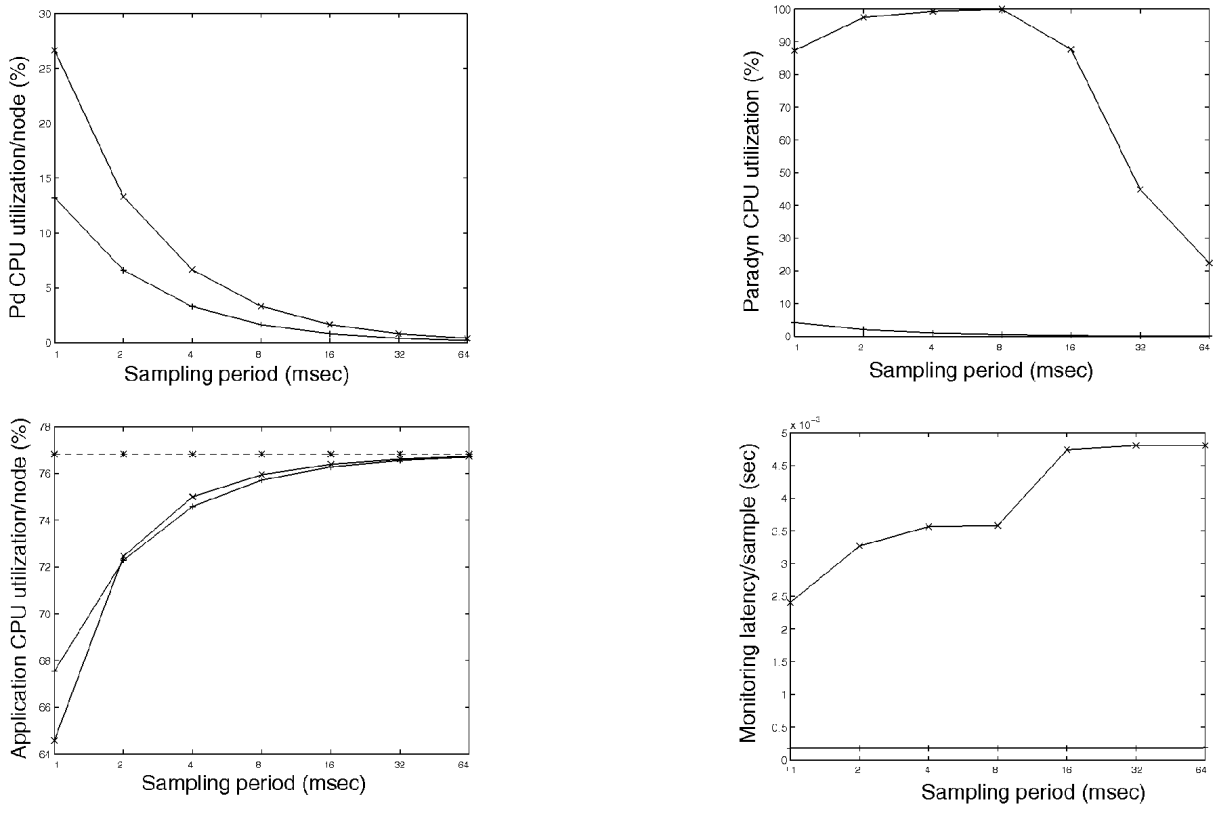
3.3.2 SMP Architecture: What is the Effect of Multiple Paradyn Daemons on the Monitoring Latency?

Simulation results in Fig. 10 show that the monitoring latency increases with the number of nodes. In order to maintain a lower monitoring latency, we investigate the potential effects of using multiple Paradyn daemons (up to four) on an SMP. An important factor with respect to the use of multiple daemons is the sampling period. Fig. 11 evaluates the use of multiple Paradyn daemons in terms of direct Paradyn daemon and main process (IS) overhead, monitoring latency, and intrusion to the application processes under the BF policy. We conclude the following from Fig. 11:

1) The number of Paradyn daemons does not have any intrusive impact on the application except at sampling periods of less than 10 msec. At shorter sampling periods, the application CPU time significantly decreases, particularly for one Paradyn daemon. This is not a consequence of high CPU utilization by Paradyn daemons at lower sampling periods (see the left plot). Rather, at a lower sampling period, the pipe that holds data samples for a Paradyn daemon fills to its capacity more often. When the pipe is full, the application process that generates a sample is blocked until the daemon is able to forward outstanding data samples. The effect of this blocking is reduced if the number of Paradyn daemons is increased for smaller sampling periods.
2) The monitoring latency increases with the number of Paradyn daemons. This small increase is a consequence of additional CPU contention due to multiple Paradyn daemon processes.
3) It is interesting to note that the behavior of the monitoring latency shown in Fig. 11 is opposite to that predicted by analytical calculations in Fig. 18 of Appendix B. Analytical calculation of monitoring latency for the SMP architecture (presented in Table 6, in Appendix B) is a function of only the arrival rate (l). Arrival rate is inversely proportional to the sampling period and directly proportional to the number of Paradyn daemons. Since the ratio of the number of Paradyn daemons to the sampling period decreases with increases in sampling period, the arrival rate and hence analytical monitoring latency also decrease. However, the analytical model does not account for the fact that a longer sampling period means longer periods of time between successive samples being forwarded from a node, which translates to longer latency in the end. On the other hand, simulation of the ROCC model accurately accounts for the time between successive samples in calculating monitoring latency. It also accounts for CPU and bus contention of multiple daemons.



(a)



(b)

Fig. 10. Effects of varying number of nodes and sampling periods on the metrics with respect to the CF and BF data forwarding policies (contention-free network). (a) sampling period = 40 msec; (b) number of nodes = 8.

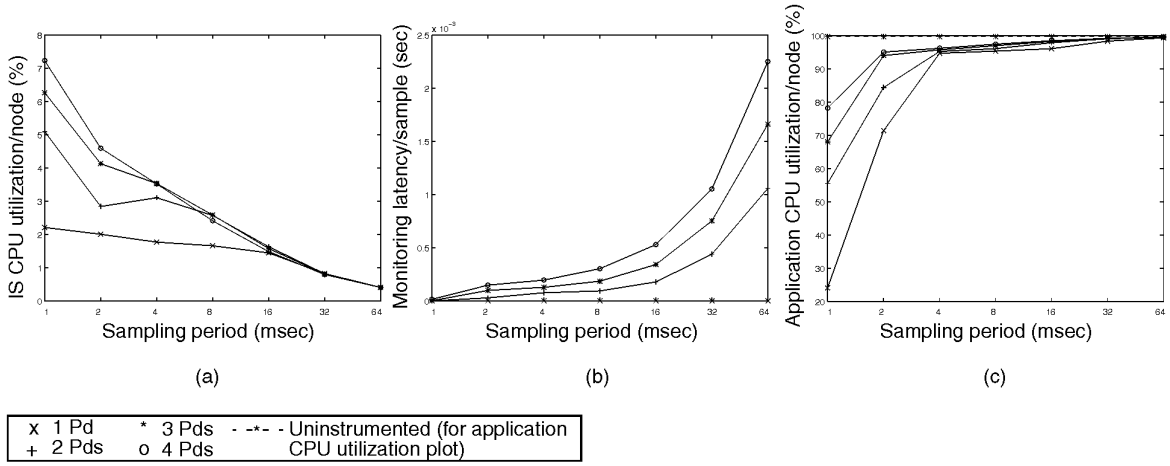


Fig. 11. Effects of multiple Paradyn daemons on two metrics (number of nodes = 16, application processes = 32, BF policy, duration of simulation = 100 sec, logarithmic horizontal scale).

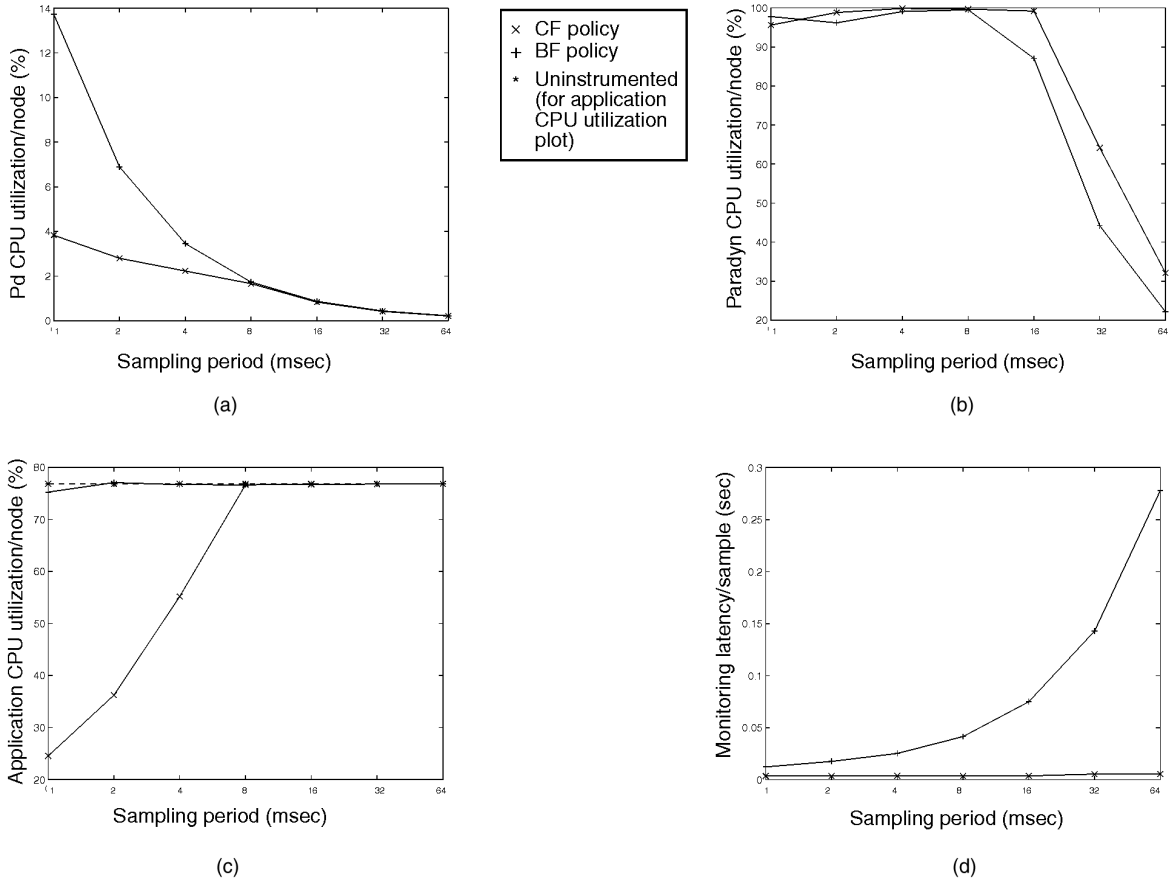


Fig. 12. Effects of varying sampling periods with respect to direct or tree forwarding on the IS performance metrics (number of nodes = 256, BF policy, logarithmic horizontal scale).

These results show that the use of multiple Paradyn daemons per node may not result in improved monitoring latency on an SMP. In fact, it may increase the latency due to additional resource contention.

3.3.3 MPP Architecture: What is the Effect of Direct vs. Tree Forwarding on Scalability?

A typical MPP system may consist of hundreds of nodes. Our objective is to study the scalability of data collection when hundreds of nodes forward instrumentation data samples through their local Paradyn daemons. In this case, a single data collection and reduction node that hosts the main Paradyn process is likely to become a bottleneck. We proposed the use of a binary tree configuration in Section 2.1 (Fig. 3) for intermediate reduction and forwarding of instrumentation data samples. In this section, we compare the scalability of the Paradyn IS under direct and tree configurations.

Principal component analysis in Section 3.2 indicates that the effect of varying the sampling period on the direct IS overhead should be significant. Fig. 12 represents the effects of varying sampling periods under the direct and binary tree data forwarding configurations. The results are again shown under the BF policy only. Analyzing direct forwarding vs. tree forwarding, we make the following comparisons:

- Per node Paradyn daemon CPU overhead is higher under the binary tree configuration at shorter sampling periods due to the increased volume of samples being generated. CPU utilization of the main Paradyn process reaches nearly 100 percent because it is swamped by sample arrivals from 256 nodes. With direct forwarding, a swamped main Paradyn process blocks all the Paradyn daemons that try to forward further samples to it. Blocking results in lower Paradyn daemon CPU utilization even though samples are pending. Since most of the data reduction and merging are handled by intermediate Paradyn daemons, blocking is less likely under tree forwarding because the main Paradyn process has less work to do.
- The same phenomena that lead to the performance of the IS processes impacts the application processes as well. When a Paradyn daemon blocks, waiting to forward additional samples, it forces the application process generating samples to block. Thus the application CPU utilization at a node is reduced to 25 percent instead of an uninstrumented 78 percent. Tree forwarding greatly reduces this intrusion to the application processes.
- Although the CPU utilization values for the main Paradyn process are almost the same under the direct and tree forwarding cases, the number of samples collected under the tree forwarding is larger.
- Monitoring latency is higher for the tree configuration because a set of samples originating from the leaf nodes undergoes a logarithmic number of forwarding operations instead of one for direct forwarding. Additionally, the monitoring latency increases with sampling period for the tree configuration because intermediate Paradyn daemons do not merge and forward the "enroute" samples asynchronously; these samples are forwarded after the expiration of the current sampling period. We do not process the enroute samples asynchronously because doing so significantly reduces the CPU time available for the local application process.

Simulation results presented in this section suggest that the use of binary tree forwarding is beneficial to improve the scalability of the Paradyn IS as the number of system nodes increases to several hundreds. For 256 nodes and sampling periods less than 8 msec (i.e., more than 25 samples per second), the Paradyn IS should switch from direct to tree forwarding.

3.3.4 MPP Architecture: What is the Effect of Varying the Frequency of Barrier Operations in a Program on IS Overhead and Intrusion?

Barrier synchronization is frequently used to explicitly implement a lock-step execution of parts of a program on an MPP system. Since barrier synchronization causes global coordination among application processes, it is of interest to consider the impact of on-line data collection on programs with different rates of barrier synchronization. In particular, we want to verify that the additional Paradyn daemon overhead for tree forwarding does not unduly perturb application execution time. Fig. 13 presents the results of our investigation of the effect of varying the frequency of barrier synchronization operations on the Paradyn IS.

- CPU overhead of both Paradyn daemons and the Paradyn main process decreases at higher barrier frequencies (and lower barrier periods as shown in the figure). While an application process waits to exit from the barrier, the Paradyn daemon does not compete for CPU time with the application process. Note that the CPU overhead for Paradyn daemon is only a fraction of a percent for the entire range of barrier period.
- Tree forwarding does not result in a lower application CPU utilization compared to the direct forwarding at any barrier period value. Thus, tree forwarding does not cause any additional intrusion to the application.
- The penalty of having instrumentation in the application varies from 10 percent 35 percent for a barrier period range of 1 to 100 msec. It appears that any delay in dispatching an instrumented application process on a node significantly reduces the amount of useful work done by that process, especially when coupled with the synchronization operations. This behavior identifies a potential bottleneck in the Paradyn IS for an MPP system.
- The monitoring latency is unaffected due to barrier operations but exhibits differences due to the direct or tree configurations.

These results indicate that barrier synchronization operations result in greater intrusion, independent of the choice of data forwarding configuration. As a result, we are able to use tree forwarding without introducing additional application perturbation.

3.4 Summary of Results and Initial Feedback to the Developers

The investigation of the "what-if" questions presented in the preceding sections evaluated the Paradyn IS with several low-level details. However, such low-level details are typically less beneficial for tool developers or users. In order to provide them with useful feedback, we summarize the simulation-based evaluation results in this section.

Simulation-based evaluation results can be divided into two categories: results directly relevant to the actual implementation of the Paradyn IS on an IBM SP-2 (NOW architecture) platform; and results projecting the performance of the Paradyn IS to the SMP and MPP architectures under different operating conditions. The first category of results is useful for improving the IS; and the second category, for porting the IS to other platforms without compromising scalability or performance. We presented several conclusions from the individual "what-if" simulation-based analyses in the preceding section. The important results are summarized as follows:

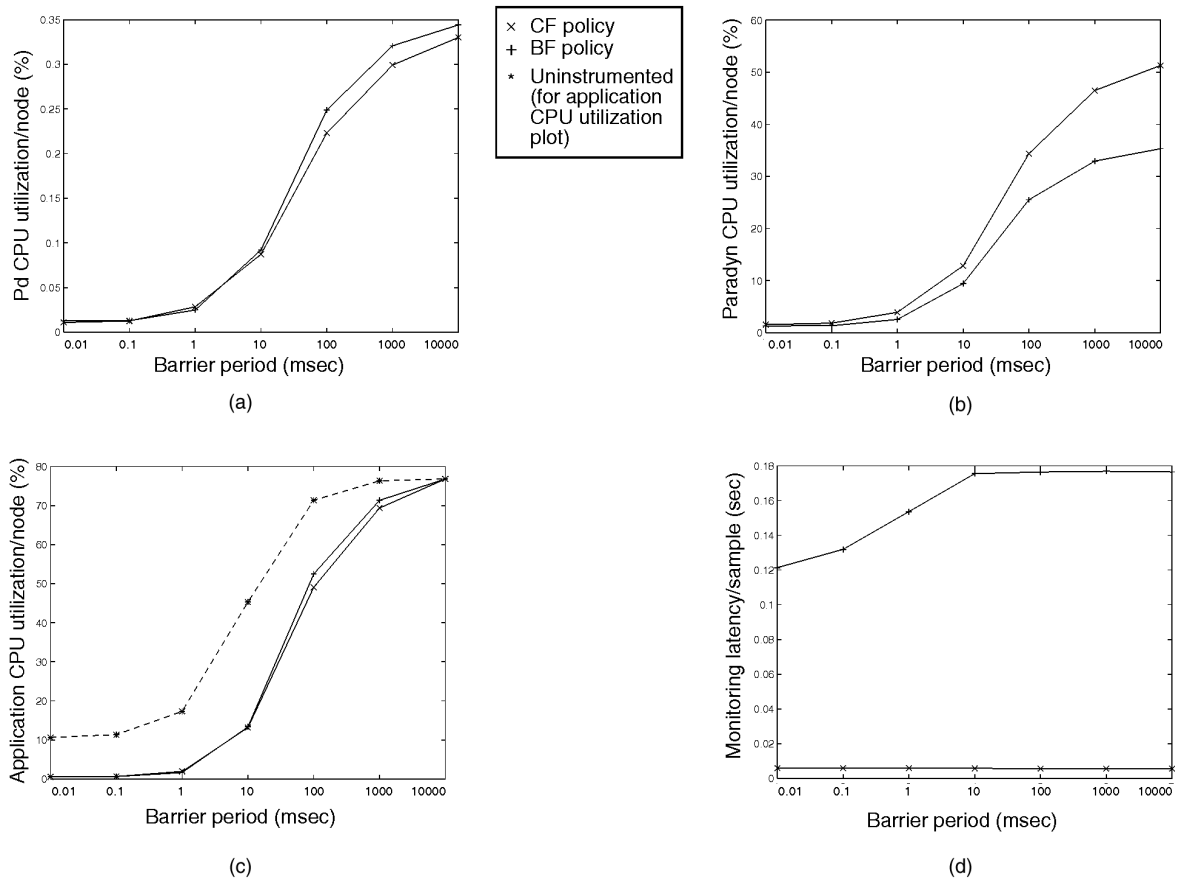


Fig. 13. Effects of varying frequency of barrier operations. (a) (number of nodes = 256; (b) sampling period = 40 msec; (c) BF policy; (e) logarithmic scales for barrier periods).

1) the BF policy should be implemented as a default policy to schedule data forwarding operations because it outperforms the CF policy;
2) in the case of an SMP, use of multiple daemons per node represents a trade-off between more samples received by the main process and additional contention for system resources;
3) binary tree forwarding should be used on an MPP system due to its superior scalability characteristics compared to direct forwarding; and
4) specific application characteristics, such as frequency of barrier operations on an MPP system, may affect IS performance, which may in turn impact the instrumented application

This feedback was well-received by the Paradyn IS developers and the BF policy was implemented in addition to the CF policy for the IBM SP-2 platform. Thus, we can experimentally validate these simulation results via testing of the actual IS.

4 Experimental Validation

We used measurement-based experiments of the IS to validate our simulations. Several factors limit the number of experiments we ran. Due to the volume of data generated when measuring applications, we restrict our experimental runs to a modest number of nodes and execution time. Due to the time required to implement different policies, we restrict our experiments to the BF and CF policies.

4.1 Experimental Setup

Fig. 14 depicts the experimental setup for measuring the Paradyn IS performance on an IBM SP-2 system. We initially use the NAS benchmark pvmbt as the application process; and we use the AIX tracing facility on one of the SP-2 nodes executing the application process. The main Paradyn process executes on a separate node, which is also traced. Therefore, one experiment with a particular sampling period and data forwarding policy results in two AIX trace files. These trace files are then processed to determine execution statistics relevant to the test.

We conduct a set of four experiments based on two factors, sampling period and forwarding policy, each having two possible values. As in the simulation, the forwarding policy options are CF and BF. The sampling period is assigned a relatively low value (10 msec) or a higher value (30 msec). Experiments using Paradyn on SMP and MPP architectures are left to future work with Paradyn. Consistent with the simulation, network occupancy is not considered (which means that communication events are not traced); this also reduces the disk space needed for AIX traces.

4.2 Evaluation

Fig. 15 summarizes the Paradyn IS testing results related to the CPU overhead of the Paradyn daemon (a) and the main Paradyn process (b). The CPU utilization of the Paradyn

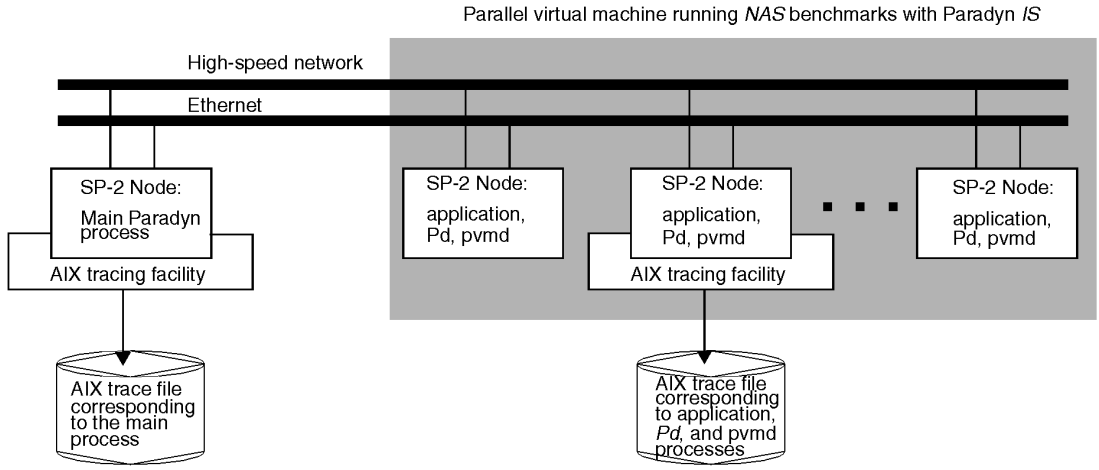


Fig. 14. Measurement-based experiment setup for the Paradyn IS on an SP-2.

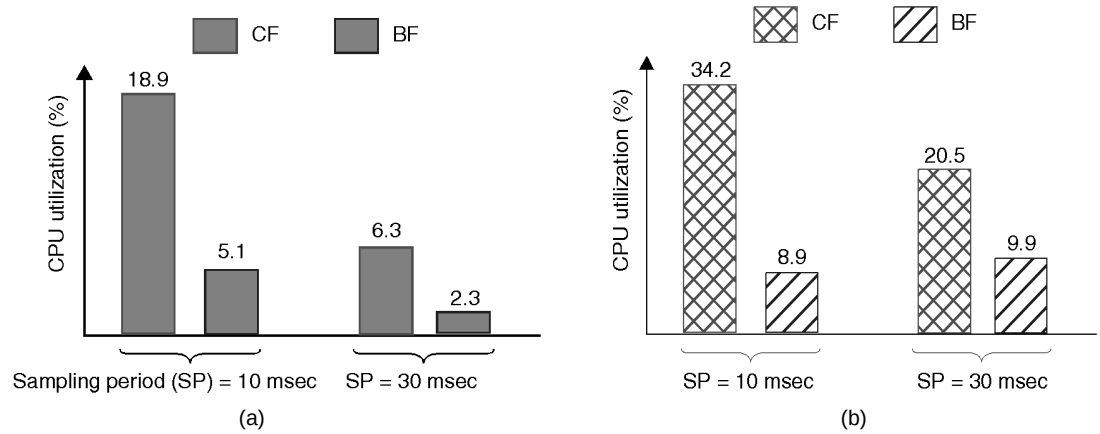


Fig. 15. Comparison of CPU overhead measurements under the CF and BF policies using two sampling period values for: (a) Paradyn daemon and (b) main Paradyn process.

daemon under the BF policy is about one-third of its value under the CF policy. This indicates a more than 60 percent reduction in overhead when Paradyn daemons send batches of samples rather than making system calls to send each sample individually. Similar analysis of the trace data obtained from the node running the main Paradyn process indicates that the overhead is reduced by almost 80 percent under the BF policy.

We conduct another set of measurement experiments to isolate the effect of a particular application on the Paradyn IS overheads. To do this, we experiment with two forwarding policies, CF and BF, and two NAS benchmark programs, pvmbt and pvmis. Benchmark pvmbt solves three sets of uncoupled systems of equations, first in the x, then in the y, and finally in the z direction. The systems are block tridiagonal with 5 × 5 blocks. Benchmark pvmis is an integer sort kernel. All experiments use a sampling period of 10 msec. The results are summarized in Fig. 16. The key observation is that the reduction in IS overheads under the BF policy is not significantly affected by the choice of application program.

5 Discussion

In this paper, we presented a case study of applying a structured modeling, evaluation, and testing approach to the Paradyn instrumentation system. We investigated various performance issues using a model of the Paradyn IS and provided feedback to the developers. Specifically, a simulation-based study indicated the potential advantages of: a proposed batch-and-forward policy over the collect-and-forward policy; and a tree forwarding configuration over the direct forwarding for MPP systems. The effects of implementing the BF policy were tested by using measurement-based experiments. Testing results indicate that use of the BF policy reduces the CPU overhead of the Paradyn daemon and main Paradyn process by about 60 percent. More significantly, this study has demonstrated the successful role of modeling and simulation to design efficient instrumentation systems through appropriate feedback at an early development stage.

The purpose of the initial feedback provided by the modeling and simulation-based study is to answer generic, performance-related "what if" questions. It is both advisable and practical to relax the accuracy requirements at this stage. Achieving a high degree of accuracy is costly due to the complexity of an instrumentation system. One lesson that we learned by modeling the Paradyn IS is that an approximate simulation model, following the gross behavior of the actual instrumentation system, is sufficient to provide useful feedback. At an early stage of modeling the Paradyn IS, we arbitrarily parameterized the model based on information pro-

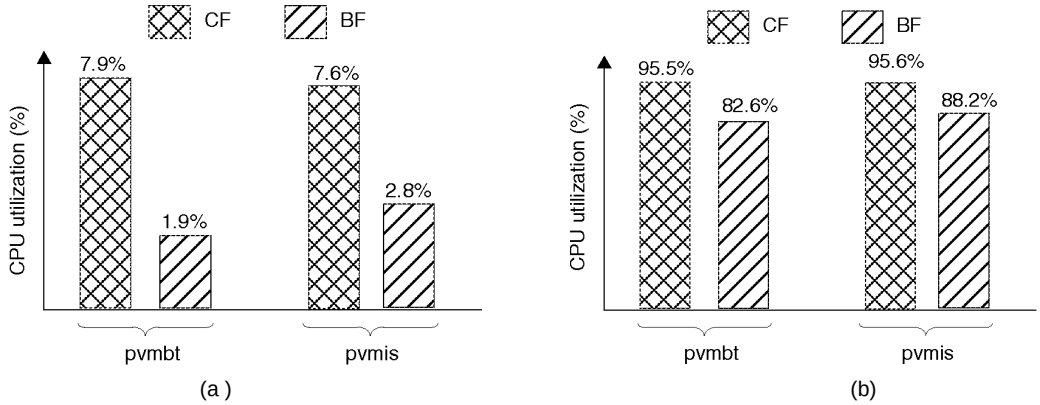


Fig. 16. Paradyn IS testing results related to (a) Paradyn daemon process, and (b) main Paradyn process.

vided by the developers [29]. The case study presented in this paper uses a more detailed workload characterization based on measurement data. Although we enhanced the scope of the "what-if" questions in this study, e.g., to include the SMP and MPP architectures and factors such as forwarding policy and length of instrumentation period, this more detailed study does not contradict the earlier study that used an approximate model [29]. Unfortunately, approximate modeling results are open to speculation without extensive workload study based on actual data.

6 Related Work

We conclude by considering related work. This paper focused on the Paradyn tool. However, a number of tools exist that provide a range of functionality and rely on instrumentation system services. Table 4 is a representative listing of tools, their functionality, and IS services.

While we have emphasized the use of IS modeling for tool developers, users can also take advantage of it. With an appropriate model of the IS, users can specify tolerable limits of IS overheads relative to the requirements of their applications. These limits can be used to adapt IS behavior in order to regulate overheads. Some initial work has already been done in this direction for Paradyn [13]. Previous work related to IS modeling and overhead analysis has focused on analyzing the intrusion due to instrumenting parallel programs [20], [32].

Several other researchers have given special attention to the monitoring overheads of their tools. Miller et al. present measurements of overheads of the IPS-2 tool and compare them with the overheads of a functionally similar tool, *gprof* [22]. Gu et al. use synthetic workloads to exercise specific features of the IS of the Falcon steering tool and measure the IS performance [9].

This study of Paradyn's IS follows previous work by Waheed and Rover to view the IS as enabling technology, or middleware [2], and to establish an approach for characterizing, evaluating, and understanding IS operation, including its overheads [28]. This approach emphasizes a separa-

TABLE 4
IS Services Used by Tools to Support a Range of Functions

Functionality	Representative Tools	Description of Key Services
Performance evaluation	ParAide [26]	ParAide is the integrated tool environment for the Intel Paragon. Commands are sent to the distributed monitoring system, called Tools Application Monitor (TAM). TAM consists of a network of TAM processes arranged as a broadcast spanning tree with one TAM process (part of the IS) at each node.
Debugging	VIZIR [10]	This debugger consists of an integrated set of commercial sequential debuggers. Its IS synchronizes and controls the activities of individual debuggers that run the concurrent processes. The IS also collects data from these processes to run multiple visualizations.
Performance modeling and prediction	AIMS, Lost cycles analysis toolkit [4], [33]	These tools integrate monitoring and statistical modeling techniques. Measurements are used to parameterize the model, which is subsequently used for predicting performance. The IS performs the basic data collection tasks.
Correctness checking	SPI [3]	Scalable Parallel Instrumentation (SPI) is Honewell's real-time IS for testing and correctness checking on heterogeneous computing systems. SPI supports a user-defined, application-specific instrumentation development environment, which is based on an event-action model and event specification language.
Adaptive real-time control	DIRECT/ JEWEL [8], [17]	Runtime information collected by the off-the-shelf instrumentation system JEWEL is fed to a dynamic scheduler. The scheduler uses this information to adaptively control the real-time system to be responsive to the variation of important system variables.
Dynamic resource scheduling	RMON [21]	RMON monitors the resource usage for distributed multimedia systems running RT-Mach. Information collected by the instrumentation system is used for adaptively managing the system resources through real-time features of the operating system.
Visualizing corporate data	AT&T visualization systems [6]	Visualization tools use monitored data to locate long-distance calling frauds through unusual calling patterns, to find communities of interest in local calling, to retain customers, and to compare databases consisting of textual information.

tion of the high-level tool requirement and usability issues from the low-level design and test issues. We applied this two-level approach for modeling and evaluating the Paradyn IS.

APPENDIX A—PARAMETERIZATION

Model parameterization accomplishes two objectives: 1) investigating different types of workloads, such as compute-intensive and communication-intensive; and 2) evaluating the impact of different types of platforms, such as MPPs, SMPs, and NOWs. Using a carefully selected set of parameters, an IS can be evaluated for specific workload and platform combinations. Table 5 lists the range of parameters that are used in this study.

APPENDIX B—SUMMARY OF ANALYTICAL CALCULATIONS

Operations analysis is applied to the ROCC model of the Paradyn IS. The ROCC model may be considered an open queuing network for the Paradyn daemon's workload because its requests leave the system when a sample is received by the main Paradyn process. Thus, the total number of Paradyn daemon requests in the system can vary with time. Alternatively, the ROCC model may be considered a closed queueing network for the application workload. An application process generates a request and waits for its completion before initiating a new occupancy request for the same or a different resource. Thus, the total number of application requests at a given time is always constant. This scenario is typical of a closed queueing network with a batch workload [15]. Therefore, the overall ROCC model for the Paradyn IS is a mixed queueing network with two workloads, assumed to be independent for analytic calculations. We list the analytical results for Network of Workstations (NOW), Symmetric MultiProcessors (SMP), and Massively Parallel Processing (MPP) architectures in Table 6 in Appendix A.

B.1 The NOW Architecture

Fig. 17 plots the results of analytical calculations of the metrics of interest with respect to the number of system nodes and sampling rate. The results indicate that the Paradyn daemon CPU overhead does not change with respect to the number of system nodes. However, under the BF forwarding policy, the overhead is significantly lower. This difference between the CF and BF cases is due to the dependence of arrival rate l on the batch size, as given in Table 6. The batch sizes are 1 and 32, respectively, for the CF and BF policies. The larger batch size for the BF policy results in lower overhead. Analytical results with respect to variable number of nodes and sampling periods predict that the BF policy is more desirable as it yields lower CPU overhead and monitoring latency.

B.2 The SMP Architecture

Fig. 18 plots the results of analytical calculations under the BF policy with respect to sampling periods and multiple Paradyn daemons. Equations corresponding to the SMP case in Table 6 indicate that IS (i.e., Pd and Paradyn) CPU utilization and monitoring latency metrics depend on the number of Paradyn daemon processes because the arrival rate 1 is proportional to the number of Paradyn daemons. Therefore, the analytical results predict that the use of multiple daemons may result in a higher monitoring latency and CPU overhead compared to the single daemon case, but the effects appear to be negligible, especially at larger sampling periods.

TABLE 5
SUMMARY OF PARAMETERS USED IN SIMULATION OF THE ROCC MODEL

Parameter Type	Parameter	Range of Values
Configuration	Number of application processes per node	1—32 (typical 1)
	Number of Pd processes per node	1—4 (typical 1)
	Number of CPUs per node	1
	Number of nodes	1—256 (typical 8)
	CPU scheduling quantum (μsec)	10,000
Application process	Length of CPU occupancy request	Lognormal (2213, 3034)
	Length of network occupancy request	Exponential (223)
Paradyn daemon	Length of CPU request	Exponential (267)
	Length of network request	Exponential (71)
	Interarrival time	5,000—50,000 (typical 40,000)
PVM daemon	Length of CPU request	Lognormal (294,206)
	Length of network request	Exponential (58)
	Interarrival time	Exponential (6485)
Other processes	Length of CPU request	Lognormal (367,819)
	Length of network request	Exponential (92)
	Interarrival time of CPU request	Exponential (31485)
	Interarrival time of network requests	Exponential (5598903)

All time parameters are in microseconds. The range of interarrival times for the Paradyn daemon corresponds to varying the rate of sampling (and forwarding) performance data by the application process. Note that exponential (m) means an exponential random variable with mean interarrival time of m msec, and lognormal(a, b) means a lognormal random variable with mean a (also in microseconds) and variance b. These parameters were calculated using maximum likelihood estimators given by Law and Kelton [19].

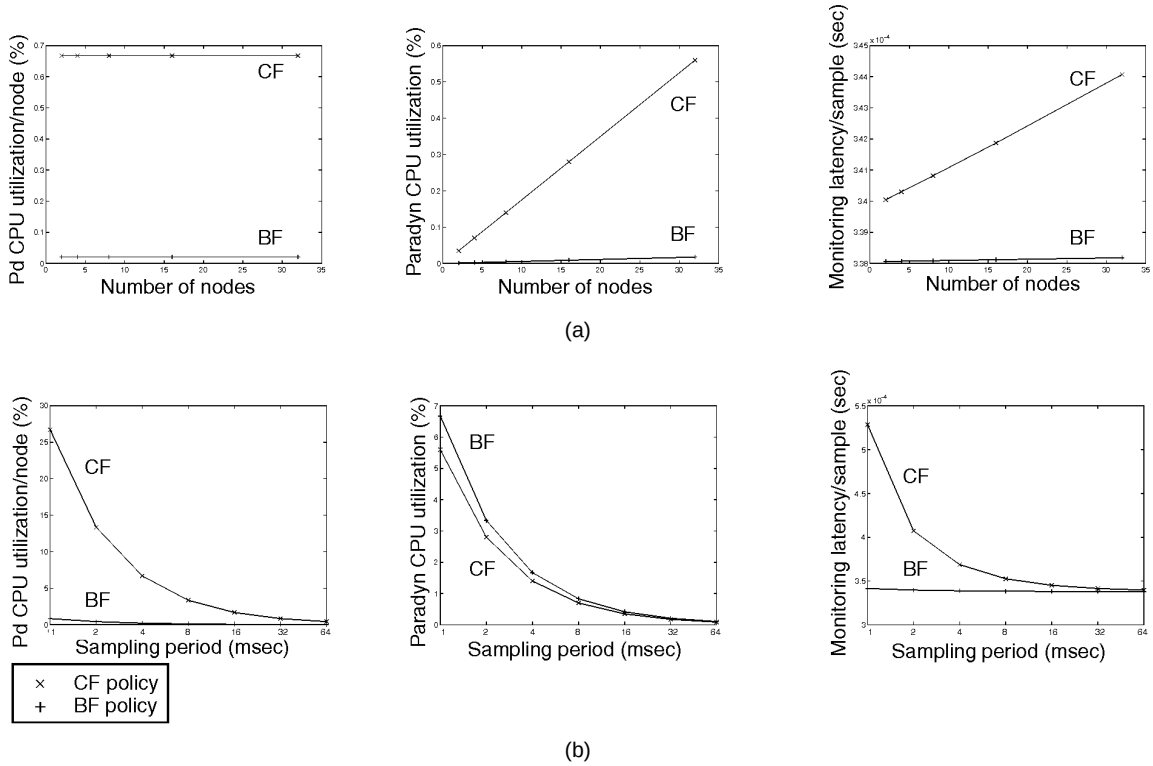


Fig. 17. Analytic calculations of the effects of varying number of nodes and sampling periods on metrics with respect to CF and BF data forwarding policies (logarithmic horizontal scale in (b)).

TABLE 6
SUMMARY OF ANALYTICAL RESULTS FOR THE ROCC MODEL OF PARADYN IS

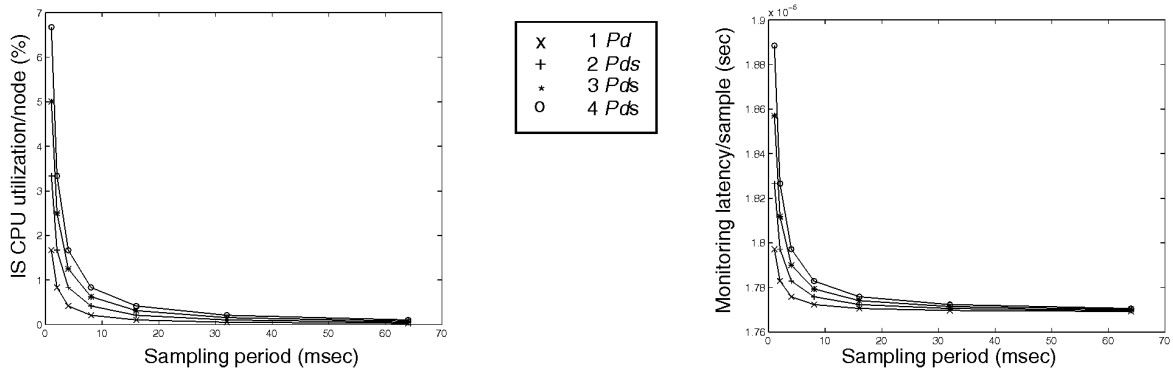


Fig.18. Analytical calculations of the effects of multiple Paradyn daemons on two metrics (number of nodes = 16, number of application processes = 32, BF policy). IS CPU utilization represents the combined CPU utilization due to Paradyn daemons and the main Paradyn process.

B.3 The MPP Architecture

Fig. 19 presents the analytical results under the BF policy with respect to the number of nodes in the MPP system. The graph in the middle indicates that tree forwarding has a clear advantage over direct forwarding in terms of lower CPU overhead for the main Paradyn process. Analytical results in Table 6 in Appendix B show that under tree forwarding the CPU overhead due to Paradyn remains unchanged as long as the arrival rate at a node (i.e., 1) is constant. Conversely, this overhead increases linearly with the number of nodes under direct forwarding. The monitoring latency is higher for tree forwarding due to additional arrivals at nonleaf nodes corresponding to the "enroute" samples. The differences in Paradyn daemon CPU overhead between the two forwarding policies are insignificant (hundredths of a percent).

ACKNOWLEDGMENTS

We thank Bart Miller of the University of Wisconsin, who helped initiate this collaborative work using the Paradyn as a case study for IS modeling, evaluation, and testing. We also thank Tia Newhall for implementing the batch-and-forward policy in Paradyn. Abdul Waheed and Diane T. Rover were supported in part by DARPA contract DABT 63-95-C-0072 and by National Science Foundation grant ASC-9624149. Jeffrey K. Hollingsworth was supported in part by Department of Energy grant DE-FG02-93ER25176 and NIST CRA award 70-NANB-5H0055.

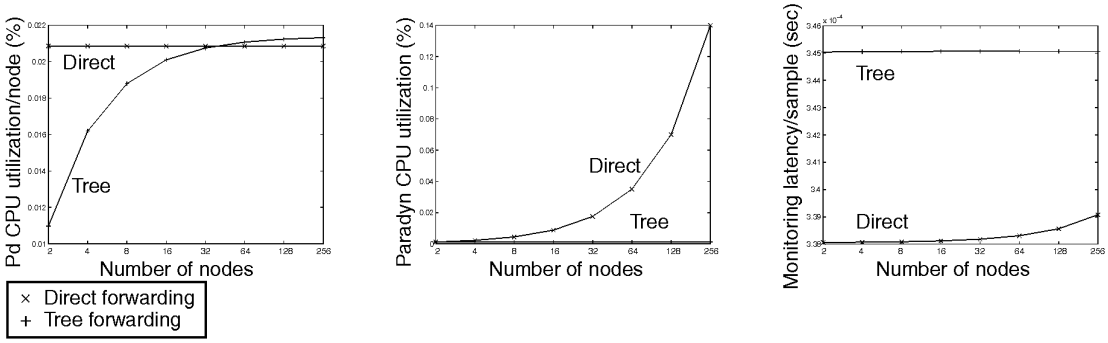


Fig. 19. Analytical calculations of the effects of varying number of nodes with respect to direct and tree forwarding policies. (sampling period = 40 msec, BF policy, logarithmic horizontal scale).

TABLE 6
SUMMARY OF ANALYTICAL RESULTS FOR THE ROCC MODEL OF PARADYN IS

System Architecture	Performance Metric	Analytic Results
NOW	Arrival rate of Pd requests	$\lambda = \frac{1}{\text{Sampling period}} \times \frac{1}{\text{Batch size}} \times \#\text{ of application processes per node}$
	Pd CPU utilization	$\mu_{Pd,CPU}(\lambda) = \lambda D_{Pd,CPU}$
	Pd network utilization	$\mu_{Pd,Network}(\lambda) = n\lambda D_{Pd,Network}$
	Monitoring latency	$R(\lambda) = \frac{D_{Pd,CPU}}{1-\mu_{Pd,CPU}(\lambda)} + \frac{D_{Pd,Network}}{1-\mu_{Pd,Network}(\lambda)}$
	Paradyn CPU utilization	$\mu_{Paradyn,CPU}(\lambda) = n\lambda D_{Paradyn,CPU}$
SMP	Arrival rate of Pd requests	$\lambda = \frac{1}{\text{Sampling period}} \times \frac{1}{\text{Batch size}} \times \#\text{ of application processes per node} \times \#\text{ of Pds}$
	Pd CPU utilization	$\mu_{Pd,CPU}(\lambda) = \lambda \frac{D_{Pd,CPU}}{n}$
	Pd bus utilization	$\mu_{Pd,Bus}(\lambda) = \lambda D_{Pd,Bus}$
	Monitoring latency	$R(\lambda) = \frac{D_{Pd,CPU}/n}{1-\mu_{Pd,CPU}(\lambda)} + \frac{D_{Pd,Bus}}{1-\mu_{Pd,Bus}(\lambda)}$
	Paradyn CPU utilization	$\mu_{Paradyn,CPU}(\lambda) = \lambda \frac{D_{Paradyn,CPU}}{n}$
	IS CPU utilization	$\mu_{IS,CPU}(\lambda) = \frac{(\#\text{ of Pds} \cdot \mu_{Pd,CPU}(\lambda)) + \mu_{Paradyn,CPU}(\lambda)}{\#\text{ of Pds} + 1}$
MPP with binary tree forwarding (for direct forwarding, MPP results are same as in NOW case)	Arrival rate of Pd requests	$\lambda = \frac{1}{\text{Sampling period}} \times \frac{1}{\text{Batch size}} \times \#\text{ of application processes per node}$
	Pd CPU utilization	$\mu_{Pd,CPU}(\lambda) = \frac{\frac{n}{2}\lambda D_{Pd,CPU} + \left(\frac{n}{2}-1\right)(\lambda D_{Pd,CPU} + 2\lambda D_{Pdm,CPU}) + \lambda D_{Pdm,C}}{n}$
	Pd network utilization	$\mu_{Pd,Net}(\lambda) = \frac{\frac{n}{2}\lambda D_{Pd,Net} + \left(\frac{n}{2}-1\right)(\lambda D_{Pd,CPU} + 2\lambda D_{Pd,Net}) + \lambda D_{Pd,Netk}}{n}$
	Monitoring latency	$R(\lambda) = \frac{D_{Pd,CPU} + D_{Pdm,CPU}}{1-\mu_{Pd,CPU}(\lambda)} + \frac{D_{Pd,Network}}{1-\mu_{Pd,Network}(\lambda)}$
	Paradyn CPU utilization	$\mu_{Paradyn,CPU}(\lambda) = 2\lambda D_{Paradyn,CPU}$

REFERENCES

[1] D.G. Belanger, Y.-F. Chen, N.R. Fildes, B. Krishnamurthy, P.H. Rank Jr., K.-P. Vo, and T.E. Walker, "Architecture Styles and Services: An Experiment Involving Signal Operations Platforms-Provisioning Operations Systems," *AT&T Technical J.*, pp. 54-60, Jan./Feb. 1996.

[2] P.A. Bernstein, "Middleware: A Model for Distributed System Services," *Comm. ACM*, vol. 39, no. 2, , pp. 86–98, Feb. 1996.

[3] D. Bhatt, R. Jha, T. Steeves, R. Bhatt, and D. Wills, "SPI: An Instrumentation Development Environment for Parallel/ Distributed Systems," *Proc. Int'l Parallel Processing Symp.*, Apr. 1995.

[4] M.E. Crovella and T.J. LeBlanc, "Parallel Performance Prediction Using Lost Cycles Analysis," *Proc. Supercomputing '94*, pp. 600–609, Washington, D.C., Nov. 1994.

[5] R.T. Dimpsey and R.K. Iyer, "A Measurement-Based Model to Predict the Performance Impact of System Modifications: A Case Study," *IEEE Trans. Parallel and Distributed Systems*, vol. 6, no. 1, pp. 28–40, Jan. 1995.

[6] S.G. Eick, and D.E. Fyock, "Visualizing Corporate Data," *AT&T Technical J.*, pp. 74–85, Jan./Feb. 1996.

[7] J.D. Gee, M.D. Hill, D.N. Pnevmatikatos, and A.J. Smith, "Cache Performance of SPEC92 Benchmark Suite," *IEEE Micro*, Aug. 1993.

[8] M.J. Gergeleit and H. Streich, "DIRECT: Towards a Distributed Object–Oriented Real-Time Control System," technical report, 1996.

[9] W. Gu, G. Eisenhauer, E. Kraemer, K. Schwan, J. Stasko, and J. Vetter, "Falcon: On-Line Monitoring and Steering of Large-Scale Parallel Programs," Technical Report GIT-CC-94–21, 1994.

[10] M.C. Hao, A.H. Karp, A. Waheed, and M. Jazayeri, "VIZIR: An Integrated Environment for Distributed Program Visualization," *Proc. Int'l Workshop Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '95)*, pp. 288-292, Tools Fair, Durham, North Carolina, Jan. 1995.

[11] R. Harrison, L. Zitzman, and G. Yoritomo, "High Performance Distributed Computing Program (HiPer-D)—Engineering Testbed One (T1) Report," technical report, Naval Surface Warfare Center, Dahlgren, Va., Nov. 1995.

[12] J.K. Hollingsworth, B.P. Miller, and J. Cargille, "Dynamic Program Instrumentation for Scalable Performance Tools," *Proc. Scalable High-Performance Computing Conf.*, pp. 841–850, Knoxville, Tenn., 1994.

[13] J.K. Hollingsworth and B.P. Miller, "An Adaptive Cost Model for Parallel Program Instrumentation," *Proc. EuroPar '96*, vol. 1, pp. 88–98, Lyon, France, Aug. 1996.

[14] H.D. Hughes, "Generating a Drive Workload from Clustered Data," *Computer Performance*, vol. 5, no. 1, pp. 31–37, Mar. 1984.

[15] R. Jain, *The Art of Computer Systems Performance Analysis—Techniques for Experimental Design, Measurement, Simulation, and Modeling*. New York: John Wiley & Sons, 1991.

[16] D. Knuth, *The Art of Computer Programming*. Addison-Wesley, 1981.

[17] F. Lange, R. Kroger, and M. Gergeleit, "JEWEL: Design and Implementation of a Distributed Measurement System," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, no. 6, pp. 657–671, Nov. 1992.

[18] J.R. Larus, "The SPIM Simulator for the MPIS R2000/R3000," *Computer Organization and Design—The Hardware/Software Interface* D.A. Patterson and J.L. Hennessy, eds., Morgan Kaufmann, 1994.

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

[20] A.D. Malony, D.A. Reed, and H.A.G. Wijshoff, "Performance Measurement Intrusion and Perturbation Analysis," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, no. 4, pp. 433–450, July 1992.

[21] C.W. Mercer and R. Rajkumar, "Interactive Interface and RT-Mach Support for Monitoring and Controlling Resource Management," *Proc. Real-Time Technology and Applications Symp.*, pp. 134–139, Chicago, May, 1995.

[22] B.P. Miller et al., "IPS-2: The Second Generation of a Parallel Program Measurement System," *IEEE Trans. Parallel and Distributed Systems*, vol. 1, no. 2, pp. 206–217, Apr. 1990.

[23] B.P. Miller, J.M. Cargille, R.B. Irvin, K. Kunchithapadam, M.D. Callaghan, J.K. Hollingsworth, K.L. Karavanic, and T. Newhall, "The Paradyn Parallel Performance Measurement Tool," *Computer*, vol. 28, no. 11, pp. 37–46, Nov. 1995.

[24] D.A. Reed, R.A. Aydt, T.M. Madhyastha, R.J. Noe, K.A. Shields, B.W. Schwartz, "The Pablo Performance Analysis Environment," Dept. of Computer Science., Univ. of Illinois, 1992.

[25] D.A. Reed, "Building Successful Performance Tools," presented in ARPA PI meeting, July 1995.

[26] B. Ries, R. Anderson, D. Breazeal, K. Callaghan, E. Richards, and W. Smith, "The Paragon Performance Monitoring Environment," *Proc. Supercomputing '93*, Portland, Ore., pp. 850–859, Nov. 1993.

[27] S. Saini and D. Bailey, "NAS Parallel Benchmark Results," Report NAS-95-021, NASA Ames Research Center, Dec. 1995.

[28] A. Waheed and D.T. Rover, "A Structured Approach to Instrumentation System Development and Evaluation," *Proc. Supercomputing '95*, San Diego, Calif., Dec. 1995.

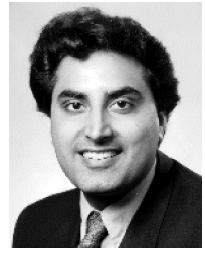
[29] A. Waheed, H.D. Hughes, and D.T. Rover, "A Resource Occupancy Model for Evaluating Instrumentation System Overheads," *Proc. 20th Ann. Int'l Conf. Computer Measurement Group (CMG '95)*, pp. 1,212–1,223, Nashville, Tenn., Dec. 1995.

[30] A. Waheed, D.T. Rover, and J. Hollingsworth, "Modeling, Evaluation, and Testing of Paradyn Instrumentation System," *Proc. Supercomputing '96*, Pittsburgh, Pa., Nov. 1996.

[31] A. Waheed, D.T. Rover, M.W. Mutka, H. Smith, and A. Bakic, "Modeling, Evaluation, and Adaptive Control of an Instrumentation System," *Proc. Real-Time Technology and Applications Symp. (RTAS '97)*, Montreal, June 1997.

[32] J.C. Yan and S. Listgarten, "Intrusion Compensation for Performance Evaluation of Parallel Programs on a Multicomputer," *Proc. Sixth Int'l Conf. Parallel and Distributed Systems*, Louisville, Ky., Oct. 1993.

[33] J.C. Yan, S.R. Sarukkai, and P. Mehra, "Performance Measurement, Visualization and Modeling of Parallel and Distributed Programs Using the AIMS Toolkit," *Software Practice and Experience*, vol. 25, no. 4, pp. 429–461, Apr. 1995.



Abdul Waheed received the BSc degree (with honors) in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, in 1991. He received the MS degree in 1993 and the PhD degree in 1997, both in electrical engineering from Michigan State University. Dr. Waheed is a research staff member at the NASA Ames Research Center. In 1991, he worked as a field service engineer in the Medical Engineering Division at Siemens in Lahore, Pakistan. He held a summer position in the Concurrent Computing Division at Hewlett-Packard Research Laboratories, Palo Alto, California, in 1994. His research interests include high-performance parallel and distributed systems, distributed data collection systems, performance evaluation tools, computer system modeling, and distributed real-time and embedded systems. Dr. Waheed is a member of the IEEE, the IEEE Computer Society, and the ACM.

Diane T. Rover received the BS degree in computer science in 1984, the MS degree in computer engineering in 1986, and the PhD degree in computer engineering in 1989, all from Iowa State University. Dr. Rover is currently an associate professor in the Department of Electrical Engineering and director of the Computer Engineering Program at Michigan State University. Under a Department of Energy postdoctoral fellowship from 1989–1991, she was a research staff member in the Scalable Computing Laboratory at Ames Laboratory. She has held summer positions with McDonnell Douglas and the IBM Thomas J. Watson Research Center. Her research interests include integrated program development and performance environments for parallel and distributed systems, instrumentation systems, performance visualization, embedded real-time system analysis, and reconfigurable hardware. She received an R&D 100 Award in 1991 for the development of the Slalom benchmark; a MasPar Challenge Award in 1996; an MSU College of Engineering Withrow Teaching Excellence Award in 1994; a National Science Foundation Career Award in 1996; an MSU Lilly Teaching Fellowship for 1996-1997; and the MSU Teacher-Scholar Award in 1998. Dr. Rover is a member of the IEEE, the IEEE Computer Society, the ACM, and the ASEE.

Jeffrey K. Hollingsworth graduated with a BS degree in electrical engineering from the University of California at Berkeley in 1988. He received his MS and PhD degrees in computer science from the University of Wisconsin in 1994 and 1990, respectively. He. is an assistant professor in the Computer Science Department at the University of Maryland, College Park, and is affiliated with the Department of Electrical Engineering and the University of Maryland Institute for Advanced Computer Studies. His research interests include performance measurement tools for parallel computing, enabling infrastructure for high performance distributed computing, and computer networks. He received a National Science Foundation Career Award in 1997. Dr. Hollingsworth is a member of the IEEE, the IEEE Computer Society, and the ACM.