HEWLETT PACKARD

Performance Evaluation of a Distributed Application Performance Monitor

Richard J. Friedrich, Jerome A. Rolia*
Broadband Information Systems Laboratory
HPL-95-137
December, 1995

performance models, distributed applications, client-server, performance monitors

The Distributed Measurement System (DMS) is a software-based measurement infrastructure for monitoring the performance of distributed application systems. In this paper we evaluate DMS in two configurations: a monitor for quality of service and a collector for model building parameters. Three distributed application workload types are defined and a model for DMS is given. The model parameters for DMS are based on measurement data from an implementation of DMS for the Open Software Foundation's Distributed Computing Environment. We use the model with our workloads to consider the impact of DMS on processor and network utilization and on workload responsiveness. We show how the various factors that control DMS affect its overhead. Lastly, the scalability of DMS is considered for large distributed environments. Our results indicate that DMS is well suited for monitoring QoS and supporting workload characterization for model building.

Internal Accession Date Only
*Carleton University, Ottawa, Ontario, Canada
To be published in and presented at *IFIP/IEEE International Conference on Distributed Platforms*, Dresden, Germany, February 1996

Performance evaluation of a distributed application performance monitor

R. J. Friedrich[a] and J. A. Rolia[b]
[a]Hewlett-Packard Laboratories, Hewlett-Packard Company,
Palo Alto, California, USA 94304; richf@hpl.hp.com
[b]Department of Systems and Computer Engineering, Carleton University,
Ottawa, Canada K1S 5B6; jar@sce.carleton.ca

Abstract

The Distributed Measurement System (DMS) is a software-based measurement infrastructure for monitoring the performance of distributed application systems. In this paper we evaluate DMS in two configurations: a monitor for quality of service and a collector for model building parameters. Three distributed application workload types are defined and a model for DMS is given. The model parameters for DMS are based on measurement data from an implementation of DMS for the Open Software Foundation's Distributed Computing Environment. We use the model with our workloads to consider the impact of DMS on processor and network utilization and on workload responsiveness. We show how the various factors that control DMS affect its overhead. Lastly, the scalability of DMS is considered for large distributed environments. Our results indicate that DMS is well suited for monitoring QoS and supporting workload characterization for model building.

Keywords

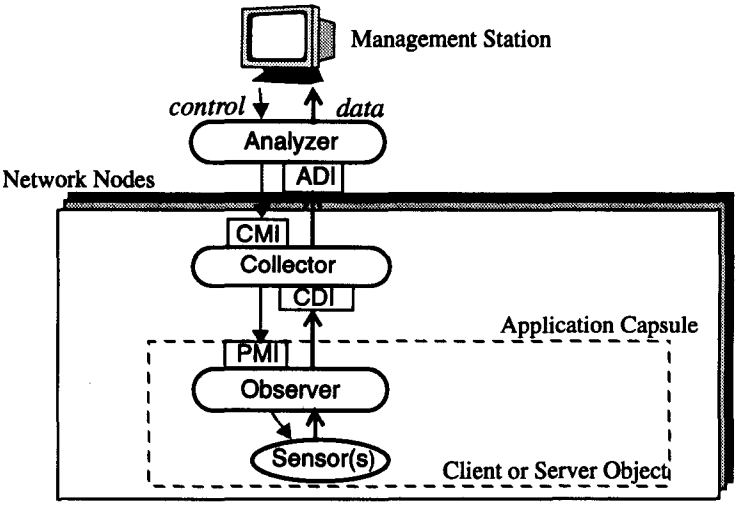
Performance models, distributed applications, client-server, performance monitors

1 INTRODUCTION

Collecting, analyzing and modeling application workload data is necessary for predicting and managing the dynamic behavior of distributed applications. Researchers at Hewlett-Packard developed the Distributed Measurement System (DMS) [6], a software-based distributed application performance monitor, to address these measurement needs. Using the terminology of the Reference Model for Open Distributed Processing (RM-ODP) [1], DMS provides correlated performance metrics across objects (application components) and their channels (network communication), integrates disparate performance measurement interfaces from a node's nucleus object (operating system), and efficiently transports collected data from network nodes to management stations. The data collected by DMS is useful to application designers, model developers, quality of service (QoS) monitors and distributed application managers.

DMS sensors (instrumentation) are designed and placed to facilitate data collection for application workload characterization. This data is used to build and validate predictive performance models of distributed applications. For example, the data can be used to capture the parameters for Layered Queueing Models (LQM) [3][11] of distributed applications. These models are similar to Queueing Network Models [9], which are used to estimate a workload's contention for processors and input-output subsystems, but also capture software interactions between capsules (operating system processes) that can affect application throughput and

Figure 1 The DMS architecture. DMS objects are represented as oval figures and interfaces are shown as rectangles. The control and data paths are shown by arrows.



An *Observer* object within each instrumented capsule implements a sensor access and control interface named the *Performance Measurement Interface* (PMI). It minimizes in-line overhead by allowing the sensors to defer some computation and off-loads sensors of the need to manage and transmit data. The observer transmits intervalized sensor data from the local capsule's address space into the collector object. Multiple nonzero sensor values are transferred at the same time to minimize IPC cost.

The *Collector* is a node level object that controls sensors and performs node-level sensor data management. It provides transparent network access and control of sensors for higher levels of the DMS architecture using the *Collector Measurement Interface* (CMI). The collectors obtain sensor data from all observers on the node using the *Collector Data Interface* (CDI). The observer periodically "pushes" sensor data to the collector using the CDI. The CDI eliminates the need for polling of sensors and provides an asynchronous data transport channel for acquiring sensor data.

An *Analyzer* object analyzes the data gathered by collector objects. It computes the higher moments of the collected data, correlates data from application elements residing on different nodes, and prepares data for expert system or human analysis. The collector periodically pushes sensor data collected from the observers on a node to the analyzer via the *Analyzer Data Interface* (ADI). The ADI is required to support a remote procedure call (RPC) interface since it most likely resides on a node elsewhere in the network.

We have used optimization techniques to minimize the amount of CPU and network utilization required and to improve scalability. Specifically, the DMS architecture uses:

- sensors that periodically report summarized data,
- thresholds so that only exceptions are reported in a normal executing environment,
- and non-polling, bulk transfer algorithms for network requests to minimize bandwidth use.

Other performance monitors focus on distributed debugging with either expensive custom hardware monitors or highly intrusive software-based event tracing [12]. Although distributed debugging is a very important feature in distributed systems management these approaches have too high an overhead for the continuous monitoring of distributed applications in operational environments.

2.2 DMS performance model

A prototype of the DMS architecture was implemented on DCE. It supports the sensor, observer, collector, and analyzer objects. The use of DCE as the prototype's distribution infrastructure impacts only the placement of sensors and the implementation of observers. All other DMS objects are independent of the distribution infrastructure.

The communication mechanism used to exchange data between observers and the collector, RPC versus shared memory IPC, also has an impact on DMS overhead.

Table 1: DMS functions and costs on an HP9000/720 running HP-UX 9.02

Function/event	*Figure 2 label*	*Cost (instructions)*
Total sensor collection	a	280
Update global sensor data store (DS)	b	232
Observer get sensor data from DS	c	232
Observer report data (via RPC)	d	21,264
Observer report data (via shared memory)	e	2,264
Collector receive data (via RPC)	f	430,280
Collector receive data (via shared memory)	g	46,280
Collector reports data (via RPC)	h	51,200

Our experimental design considers four factors:

1. strategies for applying DMS (which determines the number of active sensors per object member, their information level, and the frequency of reporting data);
2. the number of object member functions monitored per server capsule;
3. the communication mechanism between observer and collector;
4. and the workload type.

We now discuss the levels of the first three factors and their corresponding resource costs. The workload types are discussed in section 3.

First we assume that there are no active sensors. This gives us our baseline performance without monitoring. In the second case we examine the cost of monitoring application QoS and in the third case we examine the cost of collecting the data necessary for workload characterization and model building. These cases are considered for both RPC and shared memory communication between observer and collector.

In the QoS monitoring case, 2 sensors are active for each application capsule object member function. These two sensors provide data for response time and throughput. Our motivation for collecting these values is from QoS requirements for transaction processing systems. Such requirements are often expressed in the form "*under a load of 30 transactions per second, 90% of the transactions must complete in less than 5 seconds.*" The response time (timer) sensor collects percentile data using the P^2 algorithm [7] and the throughput (counter) sensor collects counts, sums and sum of squares (used by the analyzer to compute means and standard deviations.) In this example we define the information level of the response time sensor as the *percentile information level* and the throughput sensor as the *basic information level.*

In the model building case we consider detailed measurements with 5 sensors active for all capsule member functions. These 5 sensors provide data for response time, throughput, network bytes transmitted and received, and queueing time (for server objects that do not have an available thread to handle an incoming service request). All 5 of these sensors are set to the basic information level. In addition, once per reporting interval, the observer collects nucleus (operating system) metrics for capsule (process) CPU resource consumption and disk I/O counts using an OS sensor. The sensor cost and data size values are summarized in Table 2.

Note that when DMS is used for continuous application QoS monitoring thresholds are likely to be employed to reduce the amount of data collected and analyzed. Sensors with thresholds report data only when a user configured requirement is exceeded; for example, when 10% of the client response times exceed 5 seconds. The use of thresholds significantly reduces the amount of data transmitted between observer and collector, and collector and analyzer. This limits the collection and communication overhead to only those member functions behaving in an abnormal manner. At first we present results which assume that reporting always takes place. This gives us the worst case behavior for DMS. Later we describe the sensitivity of resource

In actuality these workloads are based on experiences with commercial applications. The light workload reflects an on-line transaction processing application consisting of simple queries of medium size (100 MB) databases with minimal CPU and disk resource demand but with a frequent issue rate (similar to TPC-A). An example of this workload is a banking application that provides customers with account information and financial transactions such as account debits and credits. Typically a large number of clients access a given server, 100 in this case.

The medium workload is similar to the light but with increased query complexity. Consequently, the CPU and disk resource utilization is larger than in the light case and the user think time is larger. An example of this class of applications is an inventory control program for a manufacturing company where parts are entered and deleted from inventory based on incoming customer orders and outgoing shipments. The number of clients per server is still 100.

The heavy workload is based on a decision support application where a user issues complex queries against a large (greater than 1 GB) database. The CPU and disk resource utilization is much larger than for the previous two workloads. This workload has a much smaller number of clients per server due to the client's workload intensity. This workload also has a much larger think time reflecting user analysis of the results. Examples of this workload include retail companies that use data warehouses containing historical data for trend analysis.

In our study the applications all have the same software capsule architecture as illustrated in Figure 2. Clients reside on dedicated nodes and make use of a server node. The server node has ten managed capsules (server processes) and a collector. The processing power (MIP rates) of the server nodes has been chosen so that the processor utilization is 60% for the baseline case without monitoring.

4 DMS PERFORMANCE EVALUATION

We now evaluate the performance impact of DMS when monitoring the three workload types. We create baseline models for the three workloads that have no overhead due to DMS. A second set of models reflects the use of DMS to monitor application QoS. The third set estimates DMS performance to capture data needed to build and validate predictive distributed application performance models.

In the models the number of member functions is essentially the number of different services a server provides. It also controls the amount of monitoring data that is captured. The number is set as either 1, 25 or 50. The second and third levels seem sufficiently large to characterize real applications and were chosen to stress DMS.

The utilization law $U = X D$ is used to compute DMS processor and network utilization. The processor utilization is the sum of the utilizations due to collector, observer, and sensor overheads. Models for the demands of these three components were discussed in section 2.2. The throughputs of the collector and observer instrumentation are based on their reporting frequency. The throughputs of the sensor instrumentation are determined by the number of clients N, their number of visits V to the server per think period, and their think times Z, related by the formula $X = NV/Z$. The values for the workload's N, V, and Z are given in Table 3.

Our model is a closed model so the actual throughput is $X = N/(Z + R)$ where R is the (unknown) response time of a customer, so the sensor throughput and overhead estimates are high. However, as we shall show, the sensors contribute very little to the overall overhead of DMS so this pessimistic approximation does not significantly affect the results.

Network utilization depends on the number of different sensors that report data, the amount of data reported for each sensor, and the reporting frequencies of the observers. Network utilization by the workload itself is not modeled.

We are also interested in understanding how DMS affects client response times. To study this we use the method of layers (MOL) [11]. It is a mean value analysis technique that takes into account software interactions in distributed application systems.

4.1 CPU utilization

QoS monitoring impact on CPU utilization

We consider three scenarios for monitoring quality of service labeled QoS_A, QoS_B and QoS_C and summarized in Table 4. In the QoS_A and QoS_B cases capsule object member function response times and throughputs are measured by DMS. Response times are recorded at the percentile information level and throughputs at the basic information level. In the QoS_A case all of the sensor data is reported by the observers. In the QoS_B and QoS_C cases all of the data is collected but threshold values are set such that only 10% of the data is reported by an observer. Note that we consider 10% reporting a high (pessimistic) value.

Table 4: Model Factors for QoS Scenarios

QoS scenario	*Number of sensors per member function*	*Additional member-member sensors*	*Sensors above thresh-old (%)*	*Capsule object member functions*	*Observer-Collector IPC method*	*Observer reporting period (sec)*
QoS_A	2	0	100%	1, 25, 50	RPC, Shared Memory	5
QoS_B	2	0	10%	1, 25, 50	RPC, Shared Memory	5
QoS_C	2	10	10%	1, 25, 50	RPC, Shared Memory	5

Table 5: Model Factors for Model Building Scenarios

Model building scenario	*Number of sensors per member function*	*Additional member-member sensors*	*Sensors above thresh-old (%)*	*Capsule object member functions*	*Observer-Collector IPC method*	*Observer reporting period (sec)*
$MdlBld_D$	5 + 2 OS	0	100%	1, 25, 50	RPC, Shared Memory	30
$MdlBld_E$	5 + 2 OS	10	100%	1, 25, 50	RPC, Shared Memory	30
$MdlBld_F$	5 + 2 OS	10	100%	1, 25, 50	RPC, Shared Memory	150

The QoS_C case considers a complex QoS management scenario. We assume that server member functions require service from other service providers (for example a security server or database system). In this case a member function's QoS depends on the QoS provided by its nested service providers. In this scenario's model each member function maintains timer sensors to record percentile level information for ten of its nested service providers. We label these *member-member sensors.*

Table 4 gives a summary of the factor levels for our three scenarios. Figure 3 shows the impact on CPU utilization for the three workload types. Results for the two alternatives for observer-collector communication, RPC and shared memory, are given. From the figure we see that DMS has its largest CPU impact on the light workload. Our name for the workload was chosen to suggest that a small number of instructions are required for each client visit to the server. However the Light client service demand of the server is small, so the relative sensor overhead is highest. Note that the overhead for the medium and heavy workloads is less than 0.5%. Note that the use of shared memory for communication decreases processor utilization between 0.1% and 0.6% depending on the case. Figure 4 illustrates the CPU utilization of the various QoS monitoring configurations in Table 4 for the Light workload only.

Model building measurement impact on CPU utilization

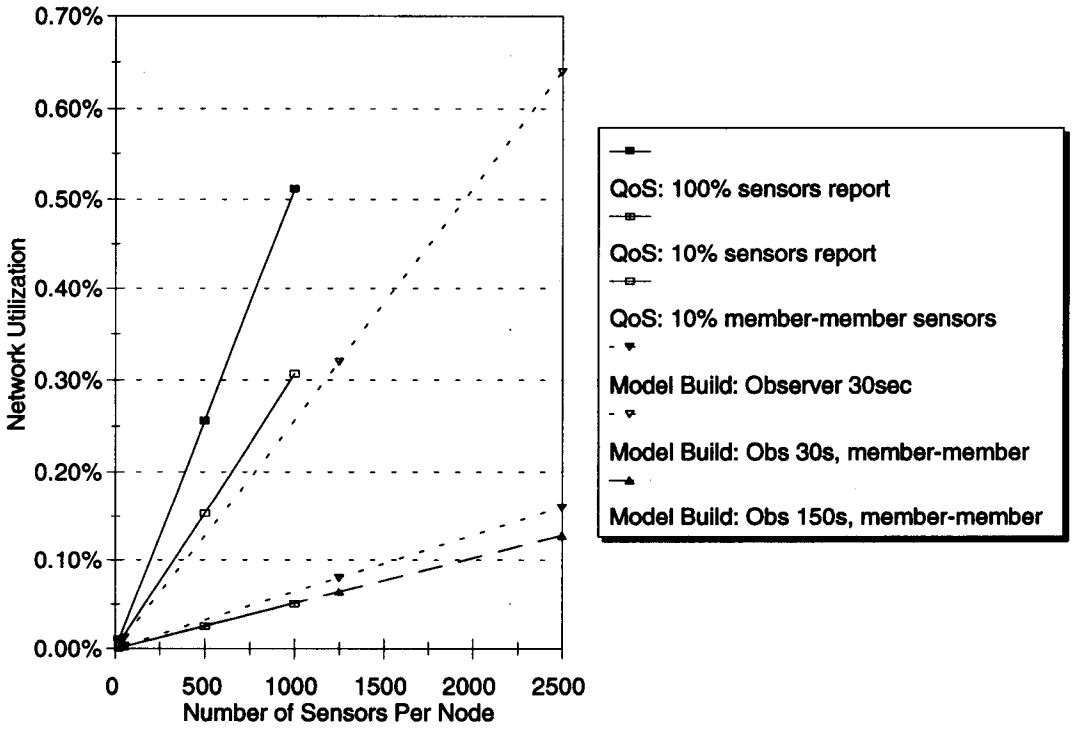
Model building captures the resource demands needed to build predictive performance models of distributed applications [4] [10]. We consider three scenarios for model building labeled $MdlBld_D$, $MdlBld_E$ and $MdlBld_F$ and summarized in Table 5. The $MdlBld_D$ and $MdlBld_E$ cases have 30 second observer periods while the $MdlBld_F$ case has 150 second. The $MdlBld_E$ and $MdlBld_F$ cases have additional instrumentation so that each member monitors its interactions with 10 of its nested service providers using member-member sensors set at the basic information level.

Figure 5 graphs the impact of DMS on CPU utilization for the three workload types. Results for the two alternatives for observer-collector communication, RPC and shared memory, are given. DMS has its largest CPU impact on the light workload but the impact is less than 0.60%. Note that the use of shared memory for communication decreases utilization less than 0.10%. Figure 6 graphs the impact on utilization for the Light workload for the RPC case for the three different model building configurations described in Table 5.

4.2 Network utilization

Figure 7 illustrates the network utilization per collector for the three QoS configuration cases and the three Model Building cases. As expected QoS monitoring without effective threshold values (100% of sensors reporting) has the highest network utilization while the more realistic QoS_B case with 10% reporting is 10 times better -- only 0.05% utilization for 1000 sensors. For the model building cases, the $MdlBld_E$ case of 30 second observer period and member-member measurement has the highest network utilization as a function of number of sensors. The more realistic $MdlBld_F$ case of 150 second observer period and member-member measurement has the lowest network utilization of all the QoS and model building configurations -- only 0.12% for 2500 sensors.

Figure 7 DMS network utilization per collector for three QoS and three Model Building scenarios described in the text. The observer-collector IPC method is set to RPC and the network utilization is based on a 10 Mbit/sec LAN. Only results from the Light workload are shown.



4.3 Client response times

A Layered Queueing Model (LQM) is created to study the impact of DMS overhead on client responsiveness for the light workload. The Method of Layers is used to estimate client response times. The accuracy of this analytic technique with respect to simulation is favorably demonstrated in [11].

There are several ways in which DMS contributes to response time delays. Greater contention for the processor will have an impact as will software contention for the capsule global data store. The method of layers takes both processor and software contention into account when estimating the response times. The impact of network overhead on client

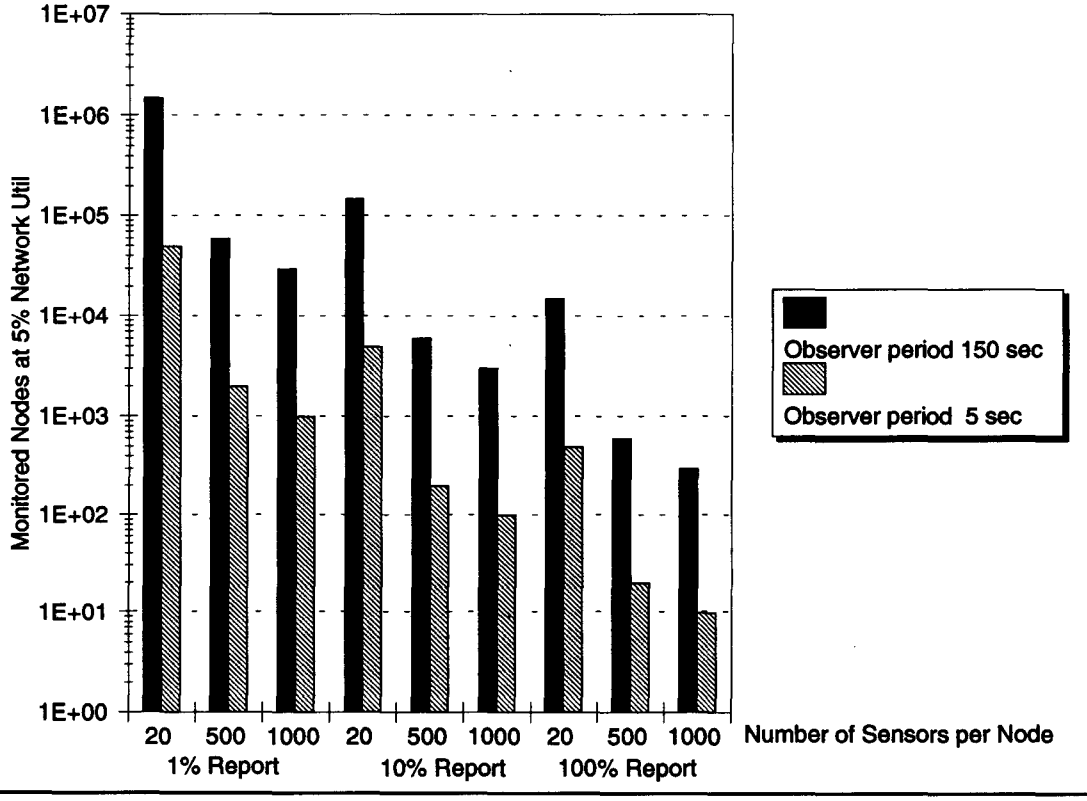
However, in all cases there was a less than 1% increase in response times viewed by end users at the client (note that Figure 8 graphs only the server response components). This response time includes the 0.3 seconds of processing time on a client's own node.

4.4 Scalability

Monitoring performance in large, heterogeneous environments requires a scalable measurement infrastructure. Several techniques were utilized in the design of the DMS to improve its scalability. Figure 9 predicts the scalability of DMS for QoS monitoring cases for a range of sensor reporting percentages of 1%, 10% and 100%, and for observer periods of 5 and 150 seconds. For good application performance we have constrained the amount of DMS network utilization to 5% of a 10 Mbit/sec LAN. We have assumed that the application network requirements are met by this LAN technology and do not consider them further.

Scalability in this figure is the number of distinct nodes supporting a given number of sensors. The left-most bar indicates that DMS can support 1.46 million nodes where each node has 20 active sensors with a threshold level set such that no more than 1% of them report per 150 second observer period. The next bar indicates that 14.6 thousand nodes can be supported where each node has 20 active sensors with a threshold level set such that no more than 1% of them report per 5 second observer period. The trade-off for scalability is the number of nodes supported versus the latency in receiving sensor data. As expected the number of nodes that can be supported is inversely proportional to the number of sensors reporting data.

Figure 9 DMS node scalability for 18 QoS monitoring cases. The observer reporting period is set to 5 or 150 seconds, the observer-collector IPC method is set to RPC, and the threshold percent is set such that 1%, 10%, or 100% of all sensors report per observer period. The results are plotted for 1, 25, and 50 object member functions which results in 20, 500 and 1000 active sensors per node. The number of nodes is constrained by a 5% network utilization for a 10 Mbit/sec LAN. The scale is logarithmic.



5 SUMMARY AND CONCLUSIONS

In this paper we present a predictive model for the overhead for the distributed application