27

Integration of Performance Measurement and Modeling for Open Distributed Processing

Richard Friedrich, Joseph Martinka, Tracy Sienknecht, Steve Saunders

Hewlett-Packard Company, HP Laboratories, Palo Alto, California 94304 USA
{richf, martinka, tracy, saunders}@hpl.hp.com

Successful deployment of open distributed processing requires integrated performance management facilities. This paper describes measurement and modeling technologies that provide quality of service (QoS) measures and projections for distributed applications. The vital role of performance instrumentation and modeling is applied to the Reference Model for Open Distributed Processing. We discuss an architecture and prototype for an efficient measurement infrastructure for heterogeneous distributed environments. We present an application model useful for application design, deployment and capacity planning. We demonstrate that integrated measurement and modeling yields the QoS measures that guide application deployment and increase management capability.

Keyword Codes: C.4, I.6.3, C.2.4
Keywords: Performance of systems; Simulation; Distributed systems

1 INTRODUCTION

Open Distributed Processing (ODP) offers advantages in performance, availability and resource sharing. However, managing applications in a distributed environment is a complex task and the lack of integrated performance management facilities is an impediment to large-scale deployment. The performance management tasks of application design, deployment, bottleneck analysis, and capacity planning require the collection, modeling and analysis of workload data. Previous techniques used to design and manage high performance, monolithic applications are inadequate for ODP. A systematic approach based on performance engineering is required, supplemented by stronger support of performance metrics in the Reference Model for ODP (RM-ODP) [11].

This paper describes the architecture of an efficient, scalable Distributed Measurement System (DMS). The DMS is a software-based measurement infrastructure that defines standard performance metrics, instrumentation and interfaces. DMS provides correlated performance metrics across objects and their channels, integrates disparate measurement interfaces from a node's nucleus object (operating system) and channels (networking), and efficiently transports collected data. The architecture is realized in an object-oriented prototype based on the OSF Distributed Computing Environment (DCE).

Strong interdependence exists between measurement and modeling. The DMS architecture was designed in concert with distributed application modeling requirements. We demonstrate the benefit of this integrated methodology on the QoS specification and measurement of a distributed application. The models ease application deployment by estimating expected resource demands and QoS for various designs and network topologies. DMS enables performance management by measuring application QoS and reporting exceptions.

Section 2 motivates our research and summarizes related research. We describe how

K. Raymond et al. (eds.), *Open Distributed Processing*

performance management maps to the ODP framework in section 3. The DMS architecture and prototype are discussed in section 4. Derived metrics are provided to performance models of two distributed applications in section 5. Conclusions and future work are summarized in the last section.

2 MOTIVATION

The complexity of distributed applications bewilder application designers and system managers. We illustrate this complexity in Figure 1 with the RPC-flow diagram of a simple query transaction of a client-server application using a distributed transaction processing monitor. Each arrow indicates an RPC request/response. The thick arrows indicate the logical RPC operation from the application developer's perspective and the thin arrows represent supporting RPCs necessary for explicit binding from a transaction monitor. Note that RPCs are nested such that the primary RPC will not return until the secondary RPC is complete (e.g., RPC 13 and its nested RPC 14), thus further complicating analysis. Arrows that cross a dotted line indicate that a network communication occurs with the potential of adding tens or hundreds of milliseconds to the transaction's latency. *How is the user's QoS estimated or measured in this complex and dynamic environment?*

Our research focuses on measurement and modeling solutions that decrease the risk of ODP application deployment. The application of these techniques in a user environment results in *performance management*. Specifically, our objective is to ensure realizable application deployment by:

- estimating the cost of an object's invocation as a function of resource consumption, target hardware capacity, and channel latency and contention, prior to deployment;
- creating an efficient, pervasive measurement infrastructure that collects, transports, correlates and analyzes the performance metrics of monitored applications;
- providing effective performance management that supplements the ODP architecture with abstractions of performance metrics and measurement interfaces, and integrates measurements and models to address system management *what-if* questions before initiating an expensive, risky course of action.

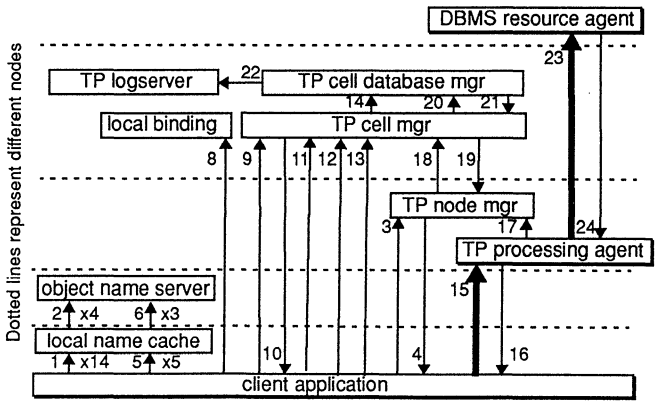

Figure 1 RPC flow for a distributed TP monitor database query.

2.1 Related research

Our work extends previous research in measurement and modeling and integrates them to provide distributed application performance management. Several trends exist in the performance measurement of distributed systems. A hardware monitor approach [20] has the advantage of minimal perturbation effects on the system under test. However, it cannot support application measurements in an enterprise environment and deployment cost is excessive. A second approach is a software monitor [17] that collects application and distribution infrastructure metrics, but requires careful design to minimize perturbation of the system under test. A variation is hybrid software and hardware monitors [5][12]. These suffer from the enterprise deployment issues of hardware monitors. A growing trend is to use the Internet SNMP protocols [3] for more than network management. Software subsystems from operating systems to databases are implementing SNMP access for distributed management. We believe that SNMP's polling nature makes it unscalable for large distributed application environments and that its trap function can flood the network with data. Our research uses a software monitor to provide an efficient, scalable infrastructure for operational enterprise environments.

Researchers and practitioners have recognized the critical need for model and instrumentation integration [19]. There has been work in developing requirements and standards for performance measurement and management [1][3][8] but others note the current paucity of instrumentation and tools in distributed client/server systems [4][16][21]. With the increasing role that middleware components play in ODP applications, Software Performance Engineering (SPE) methods [19] are dependent on pervasive performance instrumentation. Advocates of performance modeling early in the design cycle support the notion of decompositional techniques [21]. Impressive efforts by ESPRIT to create this performance design environment have been mounted for ANSAware systems [10]. Franken and Haverkort [7] describe a performance model-based *Performability Manager* that uses SPN techniques to analyze QoS in a distributed environment, guaranteeing user-requested QoS and reliability. They demonstrate its use in an ANSAware-based environment for performability management, but recognize the complexity involved in the mapping of SPN components to alternative configurations.

3 PERFORMANCE MANAGEMENT IN ODP

We map the aspects of performance engineering as applied to the RM-ODP framework in Table 1. ODP transparencies are subject to application constraints specified in the enterprise viewpoint. Transparencies mask the difficult programming decisions about distribution semantics yet their performance determines if the application can achieve the enterprise QoS requirements.

We believe that ODP objects that manage the enterprise viewpoint's QoS requirements are essential to knit together the dynamic needs of an application across the RM-ODP viewpoints, and negotiate the channel-object relationships as necessary. This need is highlighted and argued forcefully for multimedia applications by Campbell and Fédaoui [2][6]. Indeed, we believe that the QoS architecture described in [2] is extensible to on-line transaction processing (OLTP). Their focus is on the multimedia *streams* interactions between objects in the computational viewpoint of the RM-ODP and specifies a QoS manager object using its own channel. Our emphasis is similar but applied to the *operational* interactions between objects using a client-server model. Our research on performance methodologies has focused on OLTP and its version of QoS, the service level agreement.

Table 1 Performance Engineering Requirements for the RM-ODP.

RM-ODP Viewpoint	Applicable Aspects of Performance Engineering
Enterprise	Establish end-to-end user QoS requirements. These specifications guide design decisions and dynamic binding and are compared to measurements or performance models.
Information	The information schemas of this viewpoint define performance metrics that describe application behavior. This schema describes logical groupings of metrics that allows user-level QoS analysis and isolation.
Computational	An object's Interface Description Language (IDL) serves to map object functions to other objects, independent from distributional concerns. Since objects interact through their IDL, this enables performance management objects to collect and aggregate performance data measured at an object's interface. The *environment contract* for each object is specified here which includes QoS constraints. These object contracts must satisfy mappings to the engineering viewpoint.
Engineering	This viewpoint is expressed in terms of nodes and channel objects that are mapped from the computational viewpoint using transparencies. These mappings require QoS agreements comprised of performance measures that are dynamic and change in real-time. A measurement architecture is needed to implement transparencies efficiently and provide performance knowledge of an application's distributed domain.
Technology	The distribution infrastructure must have a low overhead, pervasive performance instrumentation facility that is complete and provides metrics to monitor and manage QoS.

4 DISTRIBUTED MEASUREMENT SYSTEM

This section discusses the DMS architecture and prototype implementation.

4.1 DMS architecture

DMS is a software architecture that facilitates the routine measurement of performance metrics on distributed objects. Furthermore, it provides a measurement infrastructure that collects and transports data independent of the underlying distribution mechanism.

DMS provides information for application design, deployment, QoS monitoring, load balancing and capacity planning. DMS specifies a common set of performance metrics and instrumentation to ensure consistent collection and reporting across heterogeneous nodes [8]. It defines standard application programming interfaces (APIs) to ensure pervasive support for performance metric measurement. The DMS objects shown in Figure 2 are capable of monitoring distributed technologies such as DCE and CORBA. DMS also supports integrating performance measurement interfaces from external sources, such as the host operating system and networking, into a single unified measurement infrastructure. This results in a seamless, integrated view of the behavior of a distributed application in a heterogeneous environment.

Instrumentation is specialized software incorporated into programs or libraries to calculate performance metrics. DMS *sensor* objects are instances of a general performance metric for a specific function. The sensors calculate and buffer primitive statistical quantities such as counts, summations, or interval times, but defer the computation of more complex statistics such as moments to higher-level DMS objects. Individual sensors are uniquely named within the enterprise with a string name or an *object identifier* (OID).

A sensor's collection granularity, and thus overhead, is controlled by specifying an *information level*. The information level controls the statistical detail of the collected data. The lowest information level corresponds to instantiated but inactive sensors and incurs nearly zero

overhead. The *threshold* information level is the lowest overhead setting for active sensors and is used to monitor QoS. This level reports data only when user specified threshold values are exceeded. Higher levels provide moments and histograms for analyzing distributions.

Three categories of *standard sensors* are defined: *timers* provide interval times, *counters* record the number of occurrences or state of an event, and *composers* return an arbitrary structure to higher level DMS objects.

Application objects may define *custom sensors* that supplement the standard sensors by recording application specific behavior. These sensors support extensible measurement of business or organizational units of work that are not available in the distribution infrastructure.

An *observer* object resides within each instrumented process and provides a sensor control point. 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 exports a sensor access and control interface named the *Performance Measurement Interface* (PMI). The observer supports sensor registration and unregistration and transmits intervalized data generated by sensors within the local process address space to the collector object. Multiple sensors are transferred simultaneously to minimize the overhead.

A *collector* is a network-node level object that controls sensors via the PMI and performs local node data management. There is one collector per node. It provides network transparent sensor access and control to higher levels of the architecture using the *Collector Measurement Interface* (CMI). The collectors accumulate sensor data from all observers on the node using the *Collector Data Interface* (CDI). The observer summarizes sensor data and periodically "pushes" it to the collector using the CDI. The CDI eliminates the need for polling of sensor data by providing an asynchronous data transport channel. The collector provides a *sensor registry* that contains the state of all registered sensors on the node.

An *analyzer* object analyzes the data gathered by collectors within a specific domain. It applies statistical routines to compute the distributional characteristics of the collected data,

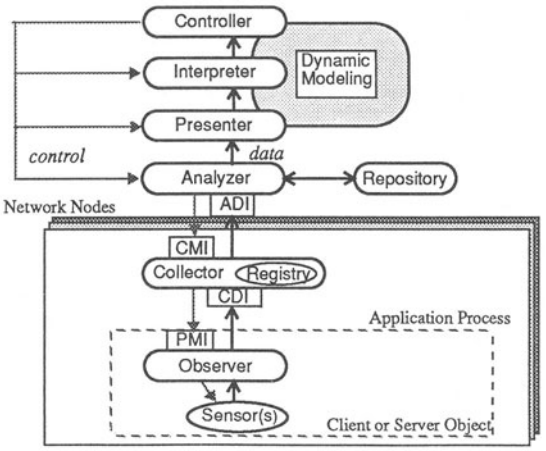

Figure 2 Distributed Measurement System Architecture.

correlates data from application elements residing in different processes and on different nodes, and prepares data for expert system or human analysis. Simple operations, such as counting events, are most efficiently provided by the sensor, but an analyzer performs complex statistical calculations such as computing moments. One analyzer can request subsets of sensor data from multiple collectors; multiple analyzers can access a specific collector. The analyzer provides data to a presentation service for visualization or an interpretation service for autonomous agent operation. An analyzer provides the basis for dynamic end-to-end QoS negotiation and monitoring.

The CMI is a network communication interface exported by a collector. An analyzer uses this interface to communicate with collectors anywhere in the network to request sensor data and specify sensor configurations. Multiple sensor values are batched by the collector and returned in bulk using the *Analyzer Data Interface* (ADI). This minimizes the number of network packets between the collector and analyzer. The collector periodically "pushes" data to the ADI which eliminates the overhead of an analyzer polling for its next update. This technique improves scalability and reduces overhead. The collector must use self-describing data techniques to facilitate interpretation of data formats since collector and analyzer may reside on heterogeneous nodes.

Additional DMS objects complement the measurement infrastructure objects. The *presenter* object supports the human user interface and interactively produces reports, displays, and alarms. It presents a logically integrated view that resembles that of an application executing on a single, centralized node. Visualization techniques are necessary to provide efficient on-line analysis of the collected data. The *interpreter* object is an intelligent entity, human or expert system, that evaluates and identifies complex relationships between the data and the environment and then draws meaningful interpretations. It relies on *dynamic models* to estimate and compare measured data with QoS requirements. The *controller* object provides control of the application and system parameters and states. The controller decides to set or change system parameters or configurations based on the interpretation of monitored performance data. Together with the measurement system and dynamic models, the controller provides the closed feedback loop necessary for providing *self-adapting* systems that manage themselves with less human intervention.

Collecting performance data must not significantly impact the applications and systems under measurement. We have used optimization techniques to minimize network bandwidth utilization and to improve scalability. We used thresholds that report data only when QoS levels are violated, sensors that report summarized data periodically but do not report unchanged data, and bulk transport of aggregated sensor data at the observer and collector level.

4.2 DMS prototype implementation

The DMS prototype provided a research tool to evaluate functional partitioning of the architecture. It implemented the sensor, observer, collector, and presenter objects, and was based on OSF DCE [15] because of availability and commercial interest. The use of DCE as the prototype's distribution infrastructure impacts only the implementation of sensors and observers but not the interface.

Sensors were developed and placed within the DCE runtime library (RTL). A copy of the RTL is linked with all clients and servers. Thus standard sensors are available without modifying client and server application source. The observer was implemented within the RTL. The collector was implemented as a daemon process that communicated with all the observers on the node via IPC using the PMI and CDI interfaces. The analyzer was

implemented as a daemon and it communicated with all the collectors in the network via RPC.

Figure 3 illustrates the prototype run time environment. The RTL supports a pool of application call threads, represented by vertical arrows, that execute within a single application address space. Curved arrows represent data transfer paths. The RTL and application RPC requests are executed on an available thread. Sensors are reentrant and use locks to access the global sensor data since they execute in a threaded environment. The analyzer periodically retrieves the data temporarily held for it from each collector. DCE RPC marshalling/ unmarshalling handles data translation.

We integrated DMS into a network management infrastructure provided by current SNMP-based tools. These tools provide a possible foundation for QoS monitoring and we wanted to demonstrate coexistence. In Figure 4 we display the *Performance Management Information Screen* (PMIS) that includes DMS integration into the Hewlett-Packard OpenView network management framework [14].

Effectively displaying an application's performance is necessary after collecting, correlating and analyzing its data. The presentation of performance management information is shown in Figure 4 for the application *PhoneDB*. The row of icons at the top of this Motif-based screen represents the integrated view of the application's performance metrics. The data available to this display is independent of the location of the application's objects. Selecting one of these icons displays a graph of the corresponding performance metrics.

DMS objects measure, map, and isolate any violations of the application's specified QoS. During normal operation, a particular ODP node or channel becomes visible to the administrator only if a QoS service level exception occurs. The inset of Figure 4 graphs a client and server object's response time. It illustrates that client perceived performance has a server latency that is only one component of its response time (solid line in graph). Channel latency and binding, stub, and protocol objects that participate in the distribution transparencies contribute an additional response time component (difference between dashed and solid lines). In this example, network loading markedly degrades the client's response time but does not affect server response.

The lower portion of the PMIS screen shows two nodes, *boom* and *winch*, with their corresponding performance metrics and node-centric tools. Other node based tools, such as SNMP or PerfView (a Hewlett-Packard performance monitor), are used as necessary for further analysis.

We learned valuable architectural and implementation lessons from the prototype. First,

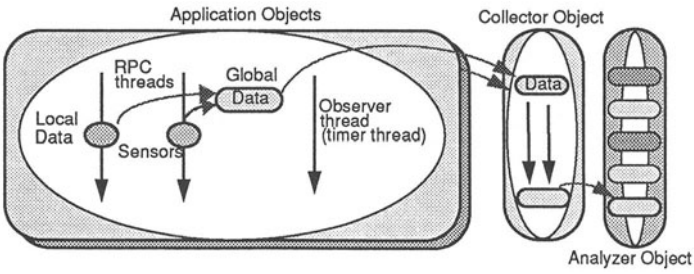

Figure 3 DMS implementation run time environment.

measurement infrastructure activities that have bursty behavior, especially sensor registration and data transportation, require bulk functions in the CDI and ADI interfaces to minimize the use of expensive communication mechanisms. Second, obtaining timestamps is an expensive operation if using standard library functions such as *gettimeofday*. We were motivated to create a custom, low overhead timer mechanism, portable to other operating systems. Finally, the viability of the DMS architecture depends on overhead incurred and channel bandwidth utilized. Our prototype's primary focus was on flexibility and adaptability, yet the overhead of the prototype was negligible (on the order of a few percent).

5 MODELING AND MEASUREMENT INTEGRATION

This section demonstrates the benefit of modeling distributed applications with data provided by DMS measurements.

Modeling eases the design, deployment, and management of ODP applications. Models provide the basis for evaluating the partitioning of application functionality, upgrading performance, developing new designs, and planning capacity. Capacity planning, for example, requires modeling of various application environments and network topologies since few large distributed environments are built solely to benchmark application design alternatives. Furthermore, the dynamic nature of ODP applications makes transparency mapping of location, migration, resource, and transaction semantics impossible to predict and manage without modeling techniques. Models answer *what-if* questions regarding load balancing, capacity planning, or QoS violation causes.

DMS plays a crucial role in modeling by providing data for workload characterization, and

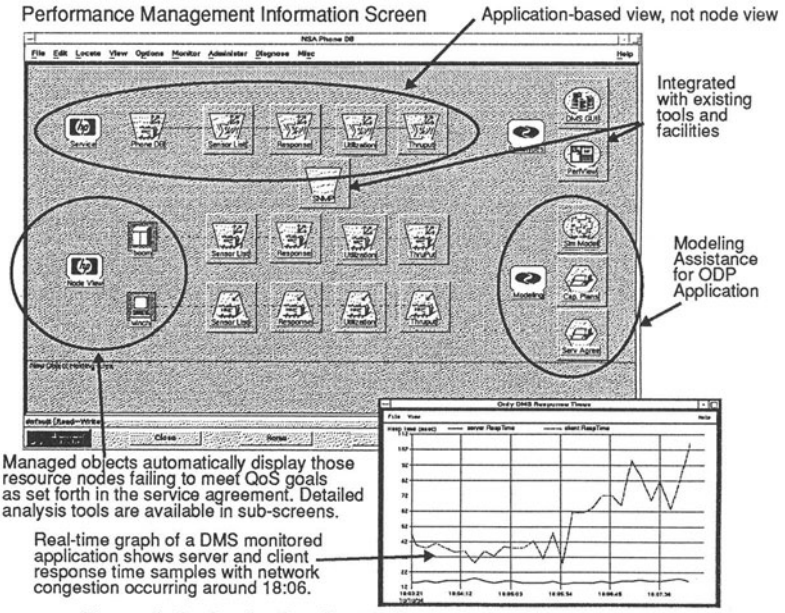

Figure 4 An Application-Centric Performance Management View.

model construction, parameterization, and validation. Models have little value without effective parameterization based on instrumentation. As further illustration of the synergy between measurement and modeling, the placement of DMS sensors was driven by modeling requirements.

5.1 Model description

Based on the complexity of distributed systems and the uncertainty in treating them analytically we approached this research using a general discrete simulation engine. We used SES/Workbench from Scientific and Engineering Software, Inc. [18]. Our prototype model was designed to meet the functional modeling requirements of client/server applications in an OLTP environment [13]. The model was designed to satisfy requirements for flexible topology specifications, nested and asynchronous RPC's, statistical model termination, and a network abstraction with realistic latencies. The simulation model is based on a resource-centered event paradigm that lends significant flexibility to the specification of the model. A simple ASCII configuration file specifies a large variety of distributed applications and topologies. This file input includes number and type of compute nodes, application transactions, users, networks, routers and routing technology. Nested and asynchronous RPCs were accommodated.

5.2 Benefits of integrated measurement and modeling

We illustrate the benefit of modeling and measurement by describing their use for two distributed applications.

PhoneDB is a client-server application that provides a database of telephone numbers for a user community. The *PhoneDB* server supports several remote functions that includes adding database entries, deleting entries, and searching for an entry using either a regular expression or a binary search algorithm.

We used DMS to measure application transactions using no-load single class techniques. This provided an estimate of the service time for both classes of transactions: regular expression and binary search (77 msec and 14.5 msec respectively). The network delay was measured at 1 msec. The client contribution to the service time was 3 msec. We assumed one packet exchange per RPC based on our knowledge of the application. The CPU times were converted to instruction pathlength using an assumed MIPs rate for the nodes on which this test was run.

Clients in the sample application were simulated to initiate transactions with a uniform inter-arrival time distribution between 0.25 and 0.75 seconds with a mean of 0.5 seconds, yielding an average arrival rate of 2 TPS. We modeled this application for a combined transaction rate load that varied from 4 transactions to 36 transactions per second. The model simulated the contention at the server for the two transaction types and generated expected mean service times for each transaction type. Up to the knee of the response time curve (around 10 TPS) the model results agreed to within 10% of the actual system measured with DMS. In Figure 5 we plot the estimate of two workload classes' average transaction response time and DMS measurements as a function of increasing load.

Using this validated model provides the benefit of estimating QoS applications and ODP topologies while in the design phase. For example, we used the model to predict the impact on QoS for a deployment environment that placed the PhoneDB server object on a node 1000 miles from the client object [13]. We also used it to demonstrate the improvement in QoS provided by upgrading the Phone DB server node to a 2-way multiprocessor. Furthermore, the model improves performance management effectiveness by estimating QoS impacts for load balancing and capacity planning before a system manager decides on a course of action. An extension to the model and the addition of a QoS manager mechanism in the distribution

infrastructure would provide dynamic QoS negotiations among objects, and quantitative evaluation before binding client and server objects.

In the second use of this model, we applied it to an OLTP application built on a DCE-based middleware product that provides transactional-RPC semantics. A distributed order processing application, *Telshop,* was measured using event-tracing and then modeled. In Figure 6 we plot the modeled and measured results of a transaction consisting of five read queries to an inventory database for a range of workload intensity. The agreement with the measured results was within the normal modeling goals of 15% accuracy.

In summary, the integration of measurement and modeling results in two major benefits. First, the validated models provide insight into application resource consumption and expected QoS behavior prior to full-scale deployment. Second, measurements supply the data necessary to comprehend dynamic application behavior, manage QoS, and parameterize models.

6 CONCLUSIONS AND CONTRIBUTIONS

This paper highlighted the benefits of integrating modeling and measurement for application design, deployment and management in ODP.

RM-ODP provides a framework that requires extension to address the performance engineering problems of building and managing distributed applications. We described a distributed measurement architecture that provides an integrated view of application resource consumption across heterogeneous nodes and network channels that node-based tools cannot. DMS provides an extensible architecture for integrating disparate performance measurement interfaces from operating system and networking software with the distribution infrastructure. It provides efficient mechanisms for controlling, collecting and transporting performance data. We have used techniques to minimize the amount of processing and network bandwidth required.

Distributed client/server application models are crucial to ensure efficient design, deployment, and scalability. RM-ODP transparencies need modeling to help guide the binding, replication, location, and migration activities specified by the application's environmental contracts. The necessary modeling technologies are not generally available. We developed a simulation model to address these requirements and validated it with the DMS prototype.

Many problems remain in providing performance management of large-scale ODP

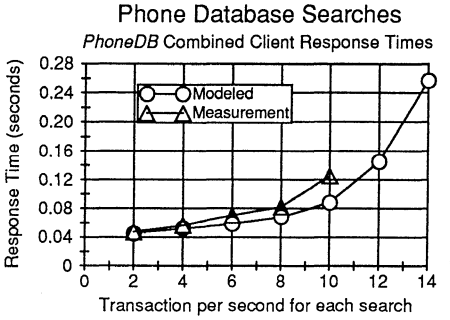

Figure 5 PhoneDB application model validation.

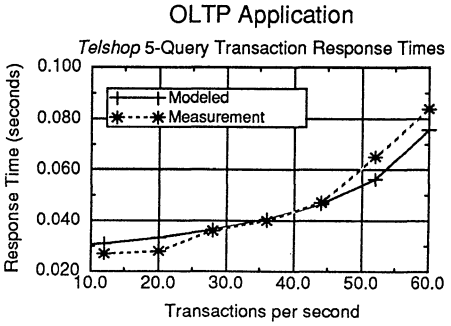

Figure 6 OLTP application model validation.

applications. A few of the most important include:

- Include standardized performance instrumentation in the RM-ODP and implement the standard interfaces proposed by the OSF DCE community [9].
- Provide a methodology to decompose the activities of middleware dependent applications so that a compound object's behavior is more easily characterized by simpler objects, preferably through the RM-ODP language specifications.
- Extend the models to include QoS negotiation for dynamic control of object selection based on service requested, service pricing, and current service QoS levels.

Acknowledgments

We wish to acknowledge the consistent support of HP's management. We are grateful to the referees who provided constructive feedback. We are indebted to the HP-Chelmsford development laboratory for providing access to the OSF DCE RTL source and development environment. We also thank Muthusamy Chelliah and Aravindan Ranganathan for contributions to the design and development of the DMS prototype.

References

1 ANSA, *ANSA Architecture Report: Monitoring in Distributed Systems*, Report Number AR.010.00, Feb 1993.

2 A. Campbell, et al, *Resource Management in Multimedia Communication Stacks*, Telecommunications, 1993, IEEE Conference Publication 371, pp 287-295.

3 Jeffery Case, et al, *A Simple Network Management Protocol (SNMP)*, Internet RFC 1157, May 1990.

4 Peter Dauphin, et al, *ZM4/Simple: A General Approach to Performance Measurement and Evaluation of Distributed Systems,* Readings in Distributed Computing Systems, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp 288-309.

5 Oliver Endris, et al, *NETMON-II a Monitoring Tool for Distributed and Multiprocessor Systems*, Performance Evaluation 12 (1991) 191-202, North Holland.

6 Linda Fédaoui, Wassim Tawbi, Eric Horlait, *Distributed Multimedia Systems Quality of Service in ODP Framework of Abstraction: A First Study*, 2nd International Conference on Open Distributed Processing '93, Elsevier Science B.V. (North Holland), pp 265-273.

7 Leonard Franken and Boudewign Haverkort, *The Performability Manager*, IEEE Network, Jan/Feb 1994, pp 24-32.

8 Richard Friedrich, *The Requirements for the Performance Instrumentation of the DCE RPC and CDS Services*, OSF DCE RFC 32.0, June 1993.

9 Richard Friedrich, Steve Saunders and Dave Bachmann, *Standardized Performance Instrumentation and Interface Specification for Monitoring DCE Based Applications*, OSF DCE RFC 33.0, November 1994.

10 Peter Hughes and Dominique Potier, *The Integrated Modelling Support Environment*, Esprit 89 Conference Proceedings, Document IMSE R-1.2-4, STC plc and Thomson CSF, 1989.

11 ISO/IEC JTC1/SC21/WG7 N885, *Reference Model for Open Distributed Processing Part 1*, November 1993.

12 Frank Lange, et al, *JEWEL: Design and Implementation of a Distributed Measurement System*, IEEE Transactions on Parallel and Distributed Systems, Volume 3 Number 6, November 1992, pp 657-671.

13 Joseph Martinka, *Requirements for Client/Server Application Performance Modeling- An Implementation using Discrete Event Simulation*, submitted for publication.

14 *HP Open View Distributed Management Platform - Administrator's Reference*, Manual J2322-90003, Hewlett-Packard, Dec 1992.

15 Open Software Foundation, *Introduction to OSF DCE*, Cambridge, MA, USA, 1992.

16 Jerome A. Rolia, *Distributed Application Performance Metrics and Management*, Proceedings from 2nd International Conference on Open Distributed Processing '93, Elsevier Science B.V. (North Holland), pp 235-246.
17 Hemant Rotithor, *Embedded Instrumentation for Evaluating Task Sharing Performance in a Distributed Computing System*, IEEE Transactions on Instrumentation and Measurement, Volume 41 Number 2, April 1992, pp 316-321.
18 *SES/workbench User's Manual*, Scientific and Engineering Software, Inc., Austin, Texas, 1992.
19 Connie Smith, *Software Performance Engineering*, in the Tutorial Proceedings of PERFORMANCE '93, Rome, Italy, September 1993.
20 Ram Sudama, et al, *Experiences of Designing a Sophisticated Network Monitor*, Software Practices and Experience, Volume 20 (6), June 1990, pp 555-570.
21 Vidar Vetland, *Measurement-Based Composite Computational Work Modelling of Software*, Ph.D. Thesis, Norwegian Institute of Technology, University of Trondhem, 1993.