REQUIREMENTS FOR PERFORMANCE INSTRUMENTATION OF DCE RPC AND CDS SERVICES

1. INTRODUCTION

Managing applications in a distributed environment is a complex task. The DCE Serviceability SIG has published a requirements document that focuses on isolating, diagnosing and correcting error conditions [Hubb]. The DME Managed Objects SIG has specified the requirements for managed objects [Sear]. However, performance related tasks such as configuration planning, application tuning, bottleneck analysis, and capacity planning require supplementary data. These performance measurement capabilities are a necessary component of any commercially viable computer technology.

This RFC represents work done by the DCE Performance SIG. It complements the DCE and DME management working groups by proposing the requirements and capabilities to monitor the performance of a DCE cell. To accomplish this we discuss the requirements for performance monitoring, the metrics that are of interest to performance analysts, and the instrumentation necessary to collect performance data.

The guiding philosophy is to define a set of *standardized* performance instrumentation that can be consistently collected, reported and interpreted in a heterogeneous environment. Furthermore, the proposed measurement system should be compiled into the core DCE services for use at customer sites. This requires strict limits on acceptable overhead.

A companion RFC, RFC 33.0, discusses a proposed measurement system architecture and design guidelines for the collection, analysis and presentation of performance data [Frie].

1.1. Minimum Content for DCE 1.1.*

This proposal focuses on the DCE RPC and CDS services. These have been determined to be the most important minimal subset for performance monitoring. (The security service is also a critical component, but due to its evolution, we have delayed addressing it.) A future work group proposal will extend performance monitoring capability requirements to all DCE services.

We also propose a generic server performance interface (GSPI) that application developers can use to automatically compile performance instrumentation into their client and server source.

The working group's goal is to establish requirements so that OSF can begin deployment of core instrumentation with the DCE Release 1.1.* and then roll out additional instrumentation in later releases.

The following list summarizes the minimum content for DCE release 1.1.*:

(a) *General.*

(i) Define common data collection interfaces and implementation for RPC and CDS components.

(ii) Define common data collection interfaces for generic server performance interfaces.

(iii) Establish guidelines for data collection interfaces for other DCE components.

(iv) Establish guidelines for the reporting of collected data.

(b) *RPC instrumentation.*

Develop `rpc_msmt` routines that include data on response times, service requirements, queue lengths, bytes transferred, messages rejected (errors), etc.

(c) *CDS instrumentation.*

Develop `cds_msmt` routines that include data on response time, service requirements, queue lengths, type of entry accessed (group, profile), etc.

(d) *Generic server performance interface.*

Develop/extend IDL and run time libraries to support automatic performance instrumentation in application servers. See section 5.7 for details.

(e) *Standard OS measurement interface.*

Develop a consistent interface for collecting key host OS performance metrics (such as node-wide CPU utilization, disk I/O rates, network traffic) so that it can be reported through the same presentation services as the DCE metrics.

(f) *Documentation.*

Create a user guide for collecting and reporting performance data. Create an application developer guide on instrumenting application servers.

1.2. Terminology

To ensure consistent meaning the following terms are defined for use in this document.

(a) *Metrics.*

Metrics specify those measurable quantities that form the criteria to evaluate the performance of a system under study. They can be raw information (such as events) or derived quantities such as statistical measures.

(b) *Instrumentation.*

Instrumentation are specialized software components incorporated into programs to provide mechanisms for measuring events and calculating the relevant performance metrics. The basic measurement techniques are counting, sampling and tracing. The objective of instrumentation is to provide measures of resource utilization (such as CPU, memory, I/O, network, etc.) and processing time (such as service time, queuing time, etc.) that can be analyzed as frequency and time histograms, or delivered as statistical measures.

The type of instrumentation that we propose utilize high-level procedure call statements and macros that can be incorporated into the specific DCE services at strategic locations. This is supplemented with IDL enhancements to capture client and generic server performance data through automatically created performance interfaces.

(c) *Performance monitor.*

A process that provides on-going collection and reporting of performance data for evaluation by system manager, application designers and capacity planners.

(d) *DCE client.*

A software process/thread executing on a particular network node. A DCE client is usually a custom developed application that issues RPC requests to a DCE server but can also be a core DCE service making a request of another DCE server.

(e) *DCE server.*

A software process/thread executing on a particular network node. A DCE server can be a core DCE service (such as the `rpcd`) or a custom developed service. For the purpose of this paper the hardware server will be referred to as a *network node*.

2. MOTIVATION

Performance instrumentation collects and displays performance information to the cell administrator, system administrator, application designer, and the system designer.

Sufficient capabilities must exist on any viable commercial computer system to answer the many performance related questions that arise during operation:

(a) Why does it take so long to complete this operation?

(b) Should I replicate my CDS?

(c) What is the impact on throughput for the various levels of security?

(d) How should I plan for a 2X increase in application workload demand?

(e) How can I optimize the resources of the cell for month's end processing?

Performance instrumentation and monitors are necessary to avoid, identify and resolve performance bottlenecks. Users cannot succeed with the deployment of a new technology without the ability to monitor and manage their new applications. Furthermore, it should be emphasized that:

What cannot be measured cannot be managed nor improved.

The following summarizes the motivation for performance instrumentation:

(a) Provide the data to software designers to understand complex, dynamic system behavior that can be utilized for algorithm design trade-off analysis.

(b) Provide data to vendors and application designers to evaluate design and implementation trade-offs.

(c) Provide system administrators/managers with the data for system sizing and acquisition.

(d) Provide system administrators with the data to determine if response time goals are being met, assess the load of a particular application, balance the load of the current system and plan for future capacity growth.

(e) Provide system administrators with the data to determine evolution of configuration and workload demands for capacity planning.

(f) Provide distributed data to system administrators to eliminate the single point of failure problem associated with centralized services.

(g) Provide data for modeling inputs to characterize the scalability of DCE services and model application and systems.

Without this data it will be difficult to deploy and manage DCE-based applications.

3. SCOPE OF PROPOSAL

Our initial work will focus on identifying required metrics and instrumentation for RPC and CDS. Future activities will expand to include identifying where instrumentation should be placed (this will likely require close support from OSF and technology suppliers) and the design of the actual instrumentation (algorithms and data structures) [Frie].

3.1. Related Activities

As we investigated the need and requirements for DCE performance instrumentation we discovered that there exists several related activities. How we incorporated their requirements are discussed for each:

(a) *Serviceability.* — Work is progressing in the definition of an event tracing mechanism. This is described in the RFC [Hubb]. Our proposal would use this service for an optional most detailed collection level.

(b) *Managed Objects Subgroup.* — This DCE Management SIG group is defining a set of managed objects for the DCE [Sear]. We have reviewed their proposal and most of our metrics have been incorporated into the managed object definitions.

(c) *UI PMWG Measurement Interface.* — Our desire is compatibility through our standard OS interface definition [UI]. This requires additional investigation to determine the degree of leverage that is available for both the DCE and PMWG measurement interfaces.

(d) *SNMP/CMIP and network management.* — We need to investigate SNMP MIBs or OSI CMIPs as a possible data storage/interoperability mechanism.

3.2. Non-Related Activities

The following items are not specifically addressed by this requirements document.

(a) *Trend analysis/capacity planning.*

(b) *Accounting.*

(c) *Fault/error detection.*

4. PERFORMANCE MEASUREMENT SYSTEM REQUIREMENTS

We have identified the following requirements for the successful design of standardized instrumentation and a measurement system.

4.1. REQT: General

The following general requirements should be satisfied by the measurement system.

(a) It is important that from the end users perspective different vendor solutions converge to provide a seamless, single view of the DCE cell. This implies that in a heterogeneous network the DCE measurement system must collect data from a range of vendor systems (nodes) and present it for analysis in a consistent manner.

(b) Define a common set of performance metrics and standardized instrumentation to ensure consistent collection and reporting across heterogeneous platforms.

(c) Define standard APIs to ensure pervasive support in heterogeneous environments.

(d) Metrics are chosen so as to meet the following goals: low variability, nonredundacy, and completeness. Low variability helps to reduce the number of repetitions required to obtain a given level of statistical confidence.

(e) Wherever necessary and efficient, utilize self-describing data to ensure accessibility in heterogeneous environments.

4.2. REQT: Users/Uses of Performance Data

This section describes the users and uses of performance data to ensure that the metrics and architecture are satisfactory. The users and uses of performance data are summarized in the following list(s). (Though presented here in list format, this data can best be conceptualized as a table or matrix, with columns tagged with the five named *user* types and rows tagged with the the five named *use* types.)

The five major users of performance data are: (a) the platform vendor who provides the DCE infrastructure on a particular platform; (b) the third party/in-house application developer; (c) the cell system administrator; (d) the cell capacity planner; (e) the information technology decision maker. For each of these users we have summarized the major attributes of performance data: (i) the user function; (ii) estimated factors; (iii) measurable factors; (iv) actions that can be taken based on the data; (v) the degrees of freedom that the particular user has in using this data to improve the productivity of the computing environment.

(a) *User: Platform Vendor.*
  - (i) *Use: User Function.*
    - [a] Configuration planning.
    - [b] Infrastructure development.
  - (ii) *Use: Estimated Factors.*
    - [a] Service center cost.
    - [b] Routing probability.
    - [c] Population size.
    - [d] Arrival rates.
    - [e] Transaction classes.

- (iii) *Use: Measured Factors.*
    - [a] Response time.
    - [b] Throughput.
    - [c] Utilization.
    - [d] Queueing.
- (iv) *Use: Actions.*
    - [a] Maximize throughput.
    - [b] Minimize response time.
- (v) *Use: Degrees of Freedom.*
    - [a] High.

(b) *User: Application Developer.*

- (i) *Use: User Function.*
    - [a] Application development.
- (ii) *Use: Estimated Factors.*
    - [a] Service center cost.
    - [b] Routing probability.
    - [c] Population size.
    - [d] Arrival rate.
    - [e] Transaction classes.
- (iii) *Use: Measured Factors.*
    - [a] Response time.
    - [b] Throughput.
    - [c] Utilization.
    - [d] Queueing.
- (iv) *Use: Actions.*
    - [a] Maximize throughput.
    - [b] Minimize response time.
- (v) *Use: Degrees of Freedom.*
    - [a] Medium.

(c) *User: System Administrator.*

- (i) *Use: User Function.*
    - [a] Tuning.
    - [b] Problem identification.
    - [c] Load balancing.
- (ii) *Use: Estimated Factors.*
    - [a] Service center cost.
    - [b] Routing probability.
- (iii) *Use: Measured Factors.*
    - [a] Throughput.
    - [b] Utilization.
    - [c] Response time.
    - [d] Population size.
    - [e] Arrival rate.
- (iv) *Use: Actions.*
    - [a] Impact of service center changes and application load.
    - [b] Impact of hardware changes.
- (v) *Use: Degrees of Freedom.*
    - [a] Low.

(d) *User: Capacity Planner.*

- (i) *Use: User Function.*
    - [a] System sizing.
    - [b] Growth.

[c] Requirements for future apps.
(ii) *Use: Estimated Factors.*
[a] Population size.
[b] Service center cost.
[c] Routing probability.
[d] Transaction classes.
(iii) *Use: Measured Factors.*
[a] Throughput.
[b] Utilization.
[c] Response time.
[d] Arrival rate.
(iv) *Use: Actions.*
[a] Predict application resource consumption, hardware requirements.
(v) *Use: Degrees of Freedom.*
[a] Medium.

(e) *User: IT Decision Maker.*
(i) *Use: User Function.*
[a] New applications.
(ii) *Use: Estimated Factors.*
[a] Population.
[b] Transaction classes.
(iii) *Use: Measured Factors.*
[a] Maximum TPS.
[b] K$/TPS.
(iv) *Use: Actions.*
[a] Authority for software and hardware purchase.
(v) *Use: Degrees of Freedom.*
[a] High.

4.3. REQT: Client and Server Perspective

Isolating performance problems in a distributed environment is much more difficult than in a centralized environment. It is important that the measurement system provide data from the client's perspective to approximate the end users perception of system performance. This provides insights into why a client's transaction or operation is not meeting performance expectations.

From the server's perspective requests appear as an incoming stream and replies signify completions (throughput). Monitoring from a server's perspective also provides node utilization.

Having information from both the client and server perspectives can also provide an indication about the performance of the interconnect.

4.4. REQT: User Defined Generic Server Performance Interfaces

To maximize the value of the measurement system it is required that application developers be provided with the capability to easily create generic performance management interfaces for application servers. Predefined performance metrics should be available by including the appropriate header file and custom, extensible application defined metrics should also be supported. These interfaces should be accessible from the standard measurement system and data for these application servers should be combined and displayed through the measurement system presentation services.

4.5. REQT: Collection Intensity Levels

Requirements for different data capture granularities and subsets requires that the maximum amount of data is collected with the minimum amount of overhead. Consequently, we have defined varying data collection intensity levels to provide increasing detail in the collected data.

Under the best scenario there should be no overhead incurred by the measurement system when there are no observations required. This can be approached practically if the design includes a zero intensity level.

Increasing the observation intensity level allows more detailed data to be collected which results in increasing complexity of metrics and collected data. In general, increasing intensity levels provide a superset of statistical data and metrics.

The following list specifies the five data collection intensity levels and their characteristics.

(a) Level 0.
  (i) *Description:*
  Minimum overhead, no data needed.
  (ii) *Statistics added per performance metric:*
  None.

(b) Level 1.
  (i) *Description:*
  Provides simple utilizations, usage counts, error counts, mean times, and mean rates for the critical subset of performance metrics.
  (ii) *Statistics added per performance metric:*
  Counts; simple sums.

(c) Level 2.
  (i) *Description:*
  Includes Level 1 data plus provides 2nd and 3rd moments so that analysis can yield variances, skews, and other indicators for distributional behavior. Increases set of collected metrics.
  (ii) *Statistics added per performance metric:*
  Sum of squares; sum of cubes; minima; maxima.

(d) Level 3.
  (i) *Description:*
  Includes Level 2 data plus provides direct measurement of the distributions of interesting metrics. Bin sizes are automatically generated (default 10) with an option for customizing.
  (ii) *Statistics added per performance metric:*
  Histograms.

(e) Level 4.
  (i) *Description:*
  Includes Level 3 data plus provides event tracing (this is the highest overhead but the most descriptive). Data can be used for debugging, model building, simulation studies and extended statistical breakdown of operations.
  (ii) *Statistics added per performance metric:*
  Events in a time-ordered causal relationship.

Level 4 functions should be provided by hooking into the tracing services defined by the DCE serviceability group [Hubb].

4.6. REQT: Concurrent Measurement

Support concurrent requests of the measurement system with different configurations and sampling intervals.

4.7. REQT: Operational Behavior

Enable/disable this instrumentation on a running system without disrupting an active application environment.

Provide a closed feedback loop so that changes to system configuration parameters are reflected in the data collected by the measurement system.

4.8. REQT: Adherence to Standards

A performance measurement system, although not a system management service in and of itself, is an important aspect of any system management capability. Therefore, the measurement system should wherever possible:

(a) Converge with relevant ISO measurement standards [ISO].

(b) Be compatible with OSF/DME.

4.9. REQT: Resource Consumption

Provide configurable data collection overhead to minimize impact on system resources. Both the measurement interval and the granularity of the collected data should be configurable. The CPU overhead goals of the measurement system are listed below based on the amount of data that has been requested (specific memory, disk and network resource consumption goals have not been identified):

(a) <2% CPU utilization for continuous monitoring of a subset of instrumentation for monitoring normal operation.

(b) <5% CPU utilization for continuous monitoring of the core DCE service's instrumentation during normal node operation.

(c) <10% CPU utilization for continuous monitoring of all DCE server's instrumentation for detailed performance analysis during normal node operation.

(d) <20% CPU utilization for continuous tracing of all DCE server's events during normal node operation.

4.10. REQT: Security Interface

The measurement system should work within the framework of the DCE security system.

4.11. REQT: Validity

Establish a validation suite so that the integrity of the measurement system can be confirmed and maintained for each platform and release.

5. PERFORMANCE METRICS AND INSTRUMENTATION

This section describes and establishes a set of prioritized metrics that will guide the design and placement of performance instrumentation. Performance metrics are provided for a client perspective (end user) and for a server perspective.

This section also describes the statistics that the measurement system must produce.

5.1. Basic Performance Metrics

The following metrics define the quantities and the notation that will be used throughout the remainder of the document. These metrics have been derived from [Lazo] and [Jain]. The notation is derived from [Lazo].

(a) **T** — The length of *time* that observations (measurements) were made.

(b) **A** — The number of request *arrivals* observed.

(c) **C** — The number of request *completions* observed.

(d) λ — The *arrival rate* of requests: $\lambda = \mathbf{A/T}$.

(e) **X** — The *throughput* of completions: $\mathbf{X = C/T}$.

(f) **B** — The length of time that a single resource was *busy*.

(g) **U** — The *utilization* of a resource: $\mathbf{U = B/T = X*S}$.

(h) **S** — The average *service requirement* per request: $\mathbf{S = B/C}$.

(i) **N** — The average *number of requests* in the system: $\mathbf{N = X*R}$.

(j) **R** — The average system *response/residence time* per request.

(k) **Z** — The average user *think time*.

(l) $\mathbf{V_k}$ — The average number of *visits* that a system level request makes to resource **k**.

(m) $\mathbf{D_k}$ — The *service demand* at resource **k**: $\mathbf{D_k = V_k*S_k = B_k/C}$.

(n) $\mathbf{Q_k}$ — The average *queue length* at resource **k**.

(o) $\mathbf{W_k}$ — The average *waiting time* at resource **k**.

(p) $\mathbf{W_k^p}$ — The average *packet waiting time* between the first and last packet at resource **k**.

In general, a metric with an annotation of ''**k**'' is for a particular resource **k**. Non-annotated metrics are for the system as a whole. The above non-annotated metrics can also be defined for a particular resource. For example, $\lambda_{\mathbf{k}}$ is the arrival rate of requests at resource **k**.

5.2. General Client Performance Metrics

The following metrics should be collected or derived from a client perspective:

(a) Response time.
(b) Number of server request completions.
(c) Service demand.
(d) Think time.
(e) Number of active client requests in system (**N**).
(f) Length of measurement interval.

5.3. General Server Performance Metrics

The following metrics should be collected or derived from a server perspective:

(a) Number of arrivals.
(b) Arrival rates.
(c) Number of completions.
(d) Throughput.
(e) Service requirement.
(f) Residence time.
(g) Visit count.
(h) Queue time.
(i) Queue length.
(j) Utilization.
(k) Measures of locking contention.
(l) Length of measurement interval.

The typical server operation completes normally and that is the expected view when we collect performance information. Three categories of metrics are associated with the outcomes of a requested operation. These categories are termed *speed*, *reliability*, and *availability*. This RFC discusses only those metrics related to speed. The Management SIG might wish to pursue the reliability and availability categories.

5.4. Statistics to Collect

The instrumentation must provide analysis software with the data required to compute the following statistical quantities:

(a) Minimum.
(b) Maximum.
(c) Sum.
(d) Mean.
(e) Variance.
(f) 90th Percentile.
(g) Skew.
(h) Histogram.

These quantities are described in detail in [Alle] and [Jain].

5.5. RPC Performance Metrics and Instrumentation

The following sections describe the metrics for the RPC and CDS DCE services. It also provides some guidance to technology providers in how to interpret the metrics so that they can be accurately and consistently implemented.

Specific instrumentation design guidelines can be found in [Frie].

For each DCE core service a prioritized list of performance metrics is provided. The metrics were developed and prioritized by the working group and are subject to change based on general DCE SIG comment. *The DCE Performance SIG expects that all high priority items will be incorporated into the next available DCE release.* Instrumentation for other DCE services, as well as medium and low priority metrics presented here, should be incorporated into a future release based on SIG input.

5.5.1. Specific RPC instrumentation requirements

Based on the above client and server views the following metrics should be collected for each client[1] thread and for each server RPC interface. The threads and interfaces must be uniquely identifiable.

For each metric the minimum, maximum, sum, mean, variance, 90th percentile, skew and histograms should be collected based on the collection intensity level.

5.5.1.1. High priority items

(a) *Response Time per RPC call* from the Client perspective.

This measures the total elapsed time, including server processing time and delay/queueing, for a client routine that invokes a particular DCE server.

Measure the elapsed time per RPC call from the time the client's runtime initiates the call until the last packet has been received and unmarshalled. This should include nested RPC call elapsed times if other DCE servers, such as the security service, must be invoked (the nested RPC call should optionally be broken out). Note that this time will not include client application or user interface response time since they are outside the scope of the DCE services.

For a given sampling interval measure the Elapsed Time from the execution of `rpc_call_start` to the execution of `rpc_call_end` as measured in the client process.

(b) *Response Time per RPC call* from the Server perspective.

This measures the total elapsed time, including server processing time and delay/queueing, required for the server to satisfy a client request.

Measure the elapsed time per RPC call from the time the server runtime receives the call until the last packet has been marshalled and sent. This should include nested RPC call elapsed times if other DCE servers, such as the security service, must be invoked (the nested RPC call times should optionally be broken out).

For a given sampling interval collect the mean elapsed time from the execution of `rpc_call_start` to the execution of `rpc_call_end` as measured in the server process.

(c) *Service requirement at Server per client request* for each unique client.

This measures the service requirement at the server, including operating system and network software processing time, required to satisfy a client's request. This request may consist of multiple RPC packets but only one RPC call. This requires that the host operating system support a performance measurement system and that DCE servers use it to gather CPU service time.

(d) *Server throughput.*

This measures the server's total throughput as determined by the number of successful completions of client requests per unit time.

---

1. This implies that clients can be located by the measurement system. The current DCE technology does not provide a mechanism to locate clients — only servers that register in the CDS.

(e) Visit count of the number of *RPC calls initiated* by the client for each unique client-server pair.

This measures the frequency of client requests and should be available for each server RPC interface.

For a given sampling interval:

(i) Number of times `rpc_call_start` is executed by the client (initiates an RPC call).

(ii) Number of times `rpc_call_end` is executed by the client (client has received results).

(iii) Number of secondary RPC calls, such as to the security server, necessary for the server to complete the requested task.

(f) Count of *total packets sent and received* by the client for each unique client-server pair.

These two metrics measure the number of packets sent by the client and the number of packets received by the client.

For connectionless RPC count the number of times `rpc__dg_xmit_pkt` is called by the client.

For connection oriented RPC count the number of time `rpc_cn_assoc_send_frag` is called by the client.

(g) Count of *total packets sent and received* by the server.

These two metrics measure the number of packets sent by the server and the number of packets received by the server for all clients.

For connectionless RPC count the number of times `rpc__dg_xmit_pkt` is called by the server.

For connection oriented RPC count the number of time `rpc_cn_assoc_send_frag` is called by the server.

(h) Number of *bytes per RPC call sent from the client* to the server for each unique client-server pair.

This metric provides information about the size of the data transferred from the client to the server.

(i) Number of *bytes per RPC call sent from the server* to the client for each unique client-server pair.

This metric provides information about the size of the data transferred from the server to the client.

(j) Count the number of *RPC call errors and failures*.

This information, although not a performance metric, provides insight into the operational environment and whether error conditions may be causing performance problems.

5.5.1.2. Medium priority items

The following metrics were assigned medium priority and are briefly described.

(a) Amount of real and virtual memory allocated for each client application program executable, libraries, RPC run time library, heaps and application stubs.

(b) Amount of real and virtual memory allocated for each server application program executable, libraries, RPC run time library, heaps and application stubs.

5.5.1.3. Low priority items

The following metrics were assigned low priority and are briefly described.

(a) Number of RPC interfaces exported for servers.

(b) Number of RPC retransmissions.

(c) Number of clients active per RPC interface.

(d) Number of RPCs by authentication type.

5.5.2. Example RPC client view

In the following example we illustrate the use of these performance metrics. Relevant performance metrics are indicated at the bottom of the figure (client request arrival rate and response time; service demand on the server).

```
                FIGURE 1. Client View of Performance

+----------+ Request-A         +------------+
| Client_A | ----------------> |            |
|          | <---------------- |            |
+----------+             Reply |            |
                               |            |
                               |  Server_C  |
                               |            |
+----------+ Request-B         |            | Request    +----------+
| Client_B | ----------------> |            | ---------> | Remote   |
|          | <---------------- |            | <--------- | Server_D |
+----------+             Reply +------------+      Reply +----------+

Metrics:
        Arrival Rate           Service Demand
        Response Time
```

Two clients, Client_A and Client_B, issue RPC requests to a server, Server_C. From a client's perspective RPC requests are made to a specific server. After some amount of processing and queueing time the server replies to the request. The simple case involving one client and one server is shown as Request-A in Figure 1 where Client_A requests an operation of Server_C.

The actual communication can be more complicated as shown for Request-B. In this example Client_B makes a request of Server_C who in turn must initiate a client request to a remote Server_D in order to complete Request-B. Note that from the client's perspective there is no change in the performance metrics but the measurement system must be able to include the processing and queueing time of the remote server in the total processing and queuing time for Request-B. The measurement system is further complicated if Server_C is multi-threaded because two different threads may process the initial request from Client_B and the completion from Server_D.

The following table provides an example performance report from the client's view. The performance measurement system should be able to provide this summary. The detailed description of each metric is in section 5.5.1.

TABLE 1. Sample Performance Output — Client View

Metric	Client_A	Client_B
RPC Calls/sec		
Ave	35.4	12.6
Std Dev	14.2	7.5
RPC Response Time (msec)		
Min / Max	02.5 / 175.3	2.4 / 256.3
Ave / Std Dev	14.5 / 22.8	22.5 / 28.6
Packets/sec		
Sent	23.0	78.0
Received	45.0	12.0
KBytes/sec		
Sent	2.5	15.0
Received	35.0	10.0

5.5.3. Example RPC server view

From a server's perspective an incoming stream of RPC requests from a pool of clients must be processed. After some amount of queueing and processing time the server completes the processing and replies to the request. This is shown in Figure 2 as a server with an input queue. The actual communication situation can become more complicated if the requested server must initiate a client request to a remote server in order to complete the incoming request. The measurement system must be able to include the processing and queueing time of the remote server in the total processing and queuing time of the original incoming request. Again, the relevant performance metrics are indicated at the bottom of the figure (message arrival rate and interarrival time; length of queue; number of servers, residence time and utilization).

FIGURE 2. Server View of Performance

```
+----------+                                                    Reply
|          | <------------------------------------------------------+
|          |                                    +------------+      |
|          |                      Queue        /|           /|      |
|  Client  |                  ---------+       +------------+|      |
|  Pool    | Request               | | |       ||    RPC    ||      |
|          | ----------------->    | | | ----> || Server(s) || -----+
|          |                       | | |       |+-----------|+
|          |                  ---------+       |/           |/
+----------+                                   +------------+

Metrics:
        Arrival Rate          Queue Length     Number of Servers
        Interarrival Time                      Residence Time
                                               Utilization of Servers
```

The following table provides an example performance report from the server's view. The performance measurement system should be able to provide this summary. The detailed description of each metric is in section 5.5.1.

TABLE 2. Sample Performance Output — Server View

Metric	Server_C	Server_D
Arrival Rate		
Ave	48.0	8.6
Std Dev	10.6	2.5
Throughput	48.0	8.6
Residence Time (sec)		
Min / Max	1.5 / 246.3	1.5 / 350.3
Ave / Std Dev	8.5 / 18.8	12.5 / 24.8
Utilization (%)	78.0	35.0
Queue Length	1.4	0.5
Packets/sec		
Sent	57.0	22.0
Received	101.0	45.0
KBytes/sec		
Sent	45.0	8.0
Received	17.5	5.0

5.6. CDS Performance Metrics and Instrumentation

The following sections describe the metrics for the CDS services. It also provides some guidance to technology providers in how to interpret the metrics so that they can be accurately and consistently implemented. Specific instrumentation design guidelines can be found in [Frie].

5.6.1. Specific CDS client/clerk performance instrumentation

The CDS performance metrics have been subdivided into client (clerk) and server (cdsd) metrics. This section describes client metrics and the next section describes server metrics.

The following are collected for each CDS client (cdsclerk[2]). Each client must be uniquely identifiable.

For each metric the minimum, maximum, sum, mean, variance, 90th percentile, skew and histograms should be collected based on the collection intensity level.

5.6.1.1. High priority items

For a given sampling interval:

(a) Count the number of *Cache Hits* in the CDS Clerk Cache.

The CDS architecture tries to provide a cache as close to the client as possible. This metric, when combined with the next, provides a means for assessing the efficiency of the CDS caching mechanism.

It appears that the counter `DNS_CLERK_CACHE_HITS_COUNT` in routine `CAADSVCR` collects this metric. If so, all that is required is to hook this up to the standardized collection mechanism.

(b) Count the number of *Cache Misses* in the CDS Clerk Cache.

The CDS architecture tries to provide a cache as close to the client as possible. This metric, when combined with the previous, provides a means for assessing the efficiency of the CDS caching mechanism.

It appears that the counter `DNS_CLERK_CACHE_BYPASS_COUNT` in routine `CAADSVCR` collects this metric. If so, all that is required is to hook this up to the standardized collection mechanism.

2. The cdsclerk is a transitory process that will terminate after a specified interval of time. This will present challenges for instrumentation initialization and storage.

(c) Count the number of *read cache operations* in the CDS Clerk Cache.

This metric, when combined with the next, provides a means for assessing the workload demand on the CDS caching mechanism.

It appears that the counter `DNS_CLERK_READ_OPS_COUNT` in routine `CAADSVCR` collects this metric. If so, all that is required is to hook this up to the standardized collection mechanism.

(d) Count the number of *write cache operations* in the CDS Clerk Cache.

This metric, when combined with the previous, provides a means for assessing the workload demand on the CDS caching mechanism.

It appears that the counter `DNS_CLERK_WRITE_OPS_COUNT` in routine `CAADSVCR` collects this metric. If so, all that is required is to hook this up to the standardized collection mechanism.

(e) Count the number of requests to the CDS clerk.

Summarize total CDS request demand by summing the following:

(i) `DNS_CLERK_READ_OPS_COUNT`.
(ii) `DNS_CLERK_WRITE_OPS_COUNT`.
(iii) `DNS_CLERK_MISC_OPS_COUNT`.

(f) Count the number of equests forwarded to the Global Directory Service.

This metric counts the number of requests that are not local to the cell and hence may experience significant delays due to inter-cell communication.

5.6.1.2. Medium priority items

The following metrics were assigned medium priority and are briefly described.

(a) Count of times request satisfied by a non-local clerk.

This metric counts the number of requests that are not satisfied by the local cdsclerk and hence may experience additional delays.

(b) The age of an entry that is replaced in the cdsclerk cache.

This metric describes the length of time that an entry resides in the cache and can provide insight into cache efficiency.

5.6.1.3. Low priority items

The following metrics were assigned low priority and are briefly described.

(a) The cdsclerk cache size in kilobytes.

(b) The cdsclerk cache size as a percent of main memory.

(c) Count of cache locked, cannot obtain lock events.

(d) Count of cdsclerk and cdsadvertiser errors and failures.

5.6.2. Specific CDS server instrumentation

The following metrics are collected per CDS server (clearinghouse) for a given sampling interval. Each server must be uniquely identifiable.

Unless otherwise stated all CDS server metrics are measured from the RPC NSI interface level and not the CDSPI level.

For each metric the minimum, maximum, sum, mean, variance, 90th percentile, skew and histograms should be collected based on the collection intensity level.

5.6.2.1. High priority items

(a) *Service requirement per client request* from the Server perspective for each unique client.

This measures the service requirement, including operating system and network software processing time, required to satisfy a client's request of a particular CDS server. This request may consist of multiple RPC packets but only one RPC NSI call. This requires that the host operating system support a performance measurement system and that DCE servers use it to gather CPU service time.

(b) *Server throughput.*

This measures the CDS server's total throughput as determined by the number of successful completions of all client requests.

(c) *Response time for read requests* to the CDS server.

This metric describes the service time of the CDS server for RPC binding handle imports or XDS read attributes requests.

(d) *Response time for write/modify requests* to the CDS server.

This metric describes the service time of the CDS server for RPC binding handle exports or XDS modify attributes requests.

(e) Number of *read requests* to the CDS server.

This metric describes the workload demand placed on the CDS server for RPC binding handle imports or XDS read attributes requests.

(f) Number of *write/modify requests* to the CDS server.

This metric describes the workload demand placed on the CDS server for RPC binding handle exports or XDS modify attributes requests.

5.6.2.2. Medium priority items

(a) Count of number of disk I/Os issued by the cdsd logging facility.

This metric describes the number of disk writes that the CDS server performs during a modify attribute request to maintain physical consistency of the name space. A calling process blocks until the I/O completes.

(b) Amount of data logged by the cdsd logging facility.

This metric describes the amount of data written to disk when the CDS server performs a logging I/O.

(c) Count of number of checkpoints.

This metric describes the number of checkpoints that the CDS server performs during its normal operation.

(d) Count of number of skulks.

This metric describes the number of skulks that the CDS server performs during its normal operation.

5.6.2.3. Low priority items

(a) Count of directory reads by type (object, attribute, etc.).

(b) Count of directory writes by type (object, attribute, etc.).

(c) Count of directory entries added.

(d) Count of directory entries deleted.

5.7. Generic Server Performance Interface for Application Server Developers

Application developers who create DCE-based servers need consistent and integrated performance instrumentation. This ‘‘custom’’ performance instrumentation should be controlled, collected, and presented similar to and in conjunction with the DCE core services.

A mechanism to create a generic server performance interface (GSPI) for a server needs to be provided by DCE. This server interface should be accessible by the performance measurement monitor. This mechanism needs to provide the following metrics for the server as a whole and for each server interface:

(a) Response time.
(b) Throughput rates.
(c) Service requirement.
(d) Visit ratio.
(e) Queue time.
(f) Queue length.
(g) Utilization (of server).
(h) Counts of server specific events.
(i) Number of unique clients accessing the server.
(j) Server name and node location.

This performance interface should be easy to compile into a DCE server and customizable if necessary.

Furthermore, this instrumentation should be designed around a framework in which the set of observable metrics is at a minimum statically extensible (compile time) and preferably dynamically loadable. It is important that more metrics can be added to the initial set and that the data gathered by them can be fed into higher level collection and analysis tools without having to rewrite those tools. This can be done in several ways but any extensible scheme probably requires that the metrics are named (ideally with globally unique names and probably assigned a UUID) and associated with a self-describing data structure. This information would not be transmitted with actual data samples but would be used by the data collectors and analyzers to interpret the data and possibly exercise control. Metrics should probably be grouped into families, for example RPC metrics or CPU metrics, and described at that level rather than individually.

The instrumentation framework needs to provide a standardized ‘‘metric description language’’ (perhaps IDL although it may not be a very good match). It would be nice to support the following concepts as part of the description language:

(a) Counters (unidirectional, up/down, re-settable (or not)).

(b) Statistic vector.

(c) Histograms.

(d) Periodic timers.

This makes adding new metrics easier and limits the variation which is likely to occur if application developers work with lower level primitives.

Although each metric family could be its own RPC process and register its own interface it is preferable to have a single data collection process (daemon) on each measured system that gathers the data and presents it through a common interface. This would limit the proliferation of RPC interfaces and improve manageability of the instrumentation system. It would need to support methods for registering, activating, and listing the state of available metrics on an individual system. If the metrics themselves are well-described then it is relatively simple to develop either a DCE-specific collector or to interface to existing monitoring packages.

5.8. Transaction Flow Tracing through Network Resources

More sophisticated performance diagnosis and modeling requires that the resources consumed by a particular transaction (or application) be traceable as various network resources are invoked. The result of this instrumentation is that it should provide a dynamic call graph of all services invoked and the callee of those services. Also, for each call the corresponding service time should be collected.

For example, invoking a server interface may require that the server become a client and invoke other servers. This instrumentation provides tracing of the parent call and all subsequent cascading calls.

5.9. Standard OS Performance Metrics Interface

To maximize the information and effectiveness of presenting performance data for distributed applications it is necessary to integrate critical non-DCE performance data into a cell presentation screen. This non-DCE performance data includes node CPU utilization, disk I/O activity and network activity.

Specifically, each DCE-host operating system should optionally[3] provide a standard interface for the following node-wide metrics:

(a) CPU utilization (system + user).

(b) File Disk I/Os per second.

(c) Paging I/Os per second.

(d) Network I/Os per second.

This interface is described in [Frie].

It can be argued that this interface is an unnecessary requirement and should be left to third party suppliers. This overlooks two problems. First, critical performance metrics can only be captured by the host operating system. Second, third party tools, if not available for every node in a heterogeneous environment are of limited use to a cell administrator. Consequently, it is imperative to *integrate* performance data from various sources into a seamless whole to make performance management possible in a distributed environment.

6. PERFORMANCE MEASUREMENT SYSTEM ARCHITECTURE

This section provides an overview of a performance measurement architecture to support efficient collection of the required performance instrumentation. Complete details are in [Frie]. We will also briefly describe several potential data collection mechanisms.

6.1. Measurement Architecture

Monitoring a distributed system is more difficult than monitoring a centralized system. The data is distributed and must be assembled from various disparate locations before analysis can start.

A modular approach is necessary to design a monitoring system that is flexible, generalized and efficient in implementation. This section describes our proposed approach. Significant parts of this section have been derived from [Jain] and [Ferr].

The layers of the distributed system monitor are shown in the following figure and briefly described below. Note that our proposed architecture is focused on both DCE core services and custom designed servers developed by application writers.

3. If a platform vendor does not support this standard interface then the performance presentation screens should report the data as ‘‘Unavailable’’.

FIGURE 3. Layers in a Distributed Performance Monitoring System

```
+------------------+
| Observation      |      -- DCE Server
+------------------+
         |
+------------------+
| Collection       |
+------------------+
         |
+------------------+
| Analysis         |
+------------------+
         |
+------------------+
| Presentation     |
+------------------+
         |
+------------------+ \
| Interpretation   |  \
+------------------+   \
         |              \
+------------------+      Can also be automated but often done by
| Console/Control  |      humans.  Can also be integrated with a
+------------------+      larger distributed management scheme.
         |              /
+------------------+   /
| Management       |  /
+------------------+ /
```

(a) *Observation*: This layer gathers raw data on individual components of the system. Generally, each component may have an observer designed specifically for it. Thus there may be several observers located on different subsystems.

(b) *Collection*: This layer collects data from various observers. It is possible to have more than one collector on large systems. Collectors may coordinate the activities of observers across the network and sometimes be responsible for store and forward activities for collectors situated on other nodes.

(c) *Analysis*: This layer analyzes the data gathered at various collectors. It may consist of various statistical routines to summarize the data characteristics. Simple analysis such as counting of events is done most efficiently in the observer but more complex statistical calculations are performed here.

(d) *Presentation*: This component of the monitor deals with human user interface. It produces reports, displays, and alarms. The presentation layer may be integrated within another tool that may manage some part of the system.

(e) *Interpretation*: This component refers to the an intelligent entity (human or expert system) that can make meaningful interpretations of the data. This generally requires multiple rules and trend analyses. Simple threshold-based alarms may be considered part of the presenter rather than of the interpreter which usually requires the application of more sophisticated rules.

(f) *Console/Control*: This component provides an interface to control the system parameters and states. Strictly speaking console/control is not a part of the monitor. However, the monitoring and control functions are often used together and it is desirable to allow system control as well as system observations facilities to be used together.

(g) *Management*: The entity that makes the decision to set or change system parameters or configurations based on interpretation of monitored performance is called the manager. A software manager component exists only in monitors with automated monitoring and control facilities.

A monitor may consist of multiple components from each of the layers listed above. There is a many-to-many relationship between successive layers. A single collector may gather data from multiple observers. Many analysis layers may use data gathered by a single collector. Several collectors may be acting with a single observer, etc.

Thus another view of these layers that may exist for a simple distributed system are shown in the following figure. As indicated by the figure, the interaction among layers in the measurement architecture is flexible. Multiple collectors can communicate with a single or multiple observers. Also, the analysis, presentation and higher levels of the architecture can be distributed throughout the cell.

FIGURE 4. DCE Measurement System Architecture

```
+----------------+   +------------+   +------------+   +------------+
| DCE SUBSYSTEM  |   | STUB CODE  |   | STUB CODE  |   | SUBSYSTEM  |
+----------------+   +------------+   +------------+   +------------+
          \         /             \               \           /
  +------------------+   +------------------+   +------------------+
  | Observation      |   | Observation      |   | Observation      |
  +------------------+   +------------------+   +------------------+
                 \          /                            |
             +------------------+               +------------------+
             | Collection       |               | Collection       |
             +------------------+               +------------------+
                 /          \                            |
  +------------------+   +------------------+   +------------------+
  | Analysis         |   | Analysis         |   | Analysis         |
  +------------------+   +------------------+   +------------------+
           |                      |                       |
  +------------------+   +------------------+   +------------------+
  | Presentation     |   | Presentation     |   | Presentation     |
  +------------------+   +------------------+   +------------------+
                                  |                       |
                         +------------------+   +------------------+
                         | Interpretation   |   | Interpretation   |
                         +------------------+   +------------------+
                                  |                       |
                         +------------------+   +------------------+
                         | Console/Control  |   | Console/Control  |
                         +------------------+   +------------------+
                                  |                       |
                         +------------------+   +------------------+
                         | Management       |   | Management       |
                         +------------------+   +------------------+
```

6.2. Measurement Collection Mechanisms

We have identified three instrumentation implementation strategies. The first is to instrument the RPC run time libraries. Second, the automatic IDL definition of performance interfaces for each server in a cell. Third, the performance interfaces to the internal DCE services (such as CDS, DFS, etc.). These are described in detail in [Frie].

7. ACKNOWLEDGEMENTS

This document is the result of many individuals from the DCE Performance SIG who contributed their time and expertise. Principal among these are the individuals listed below.

Judi Bank (IBM), Peter Bates (Bellcore), Sarr Blumson (CITI at University of Michigan), Don Bolinger (OSF), Art Gaylord (Project Pilgrim at University of Massachusetts), Russ Heinselman (DEC), Bruce Keith (DEC), Joe

Martinka (HP), Peter Oleinick (Transarc), Jerome Rolia (Carleton University), Chuck Royalty (Boeing), Barry Searle (IBM), Tracy Sienknecht (HP), Jim Van Sciver (OSF).

APPENDIX A. CURRENT INSTRUMENTATION IN THE DCE SERVICES

Current instrumentation in the RPC and CDS services is described below. The list may not be complete. It should be noted that the current instrumentation was deemed unsatisfactory because it was incomplete and inconsistent. Some of it, however, may be leveraged into the recommended measurement system.

A.1. RPC

This section briefly documents the existing instrumentation that we have found in the DCE 1.0 release of RPC.

The DCE Application Development Reference manual intro(3rpc) page describes the statistics vector data structure. The RPC runtime maintains statistics on a per address space basis.

The following metrics are collected:

(a) `rpc_c_stats_calls_in` — The number of RPCs received by runtime.

(b) `rpc_c_stats_calls_out` — The number of RPCs initiated by runtime.

(c) `rpc_c_stats_pkts_in` — The number of network packets received by the runtime.

(d) `rpc_c_stats_pkts_out` — The number of network packets sent by the runtime.

These values can be obtained by using the `rpc_mgmt_inq_stats` routine.

The data structure and additional information can be found in the Application Reference Guide.

Additionally, we found a set of undocumented logging utilities by examining the RPC 1.0 source.

In the RPC runtime library source the files `rpclog.h` and `rpclog.c` contain reference to an ULTRIX only logging function implemented by way of macros. One of the more interesting ones is the `TIMESTAMP(code)` macro (where the code is one of 256 possibilities delineated in the rpclog.h). This debugging trace capability could have some performance usage.

A cursory investigation revealed that these macros write to a single location in main memory and that specialized hardware is used to store the data to another machine for processing. It also appears that this special hardware works only on Ultrix/VAX and not Ultrix/MIPs. Also, this instrumentation has been done only for the connection RPC. It appears that this mechanism would only be useful on machines with custom built hardware monitors (something that is not likely to be pervasive across DCE platforms).

A.2. CDS

The following counters are kept by routine `clerk_counter`, which updates the counts. They are available through the `show_clerk` command, which can be invoked via an RPC call. The counters are also written to the client caches.

(a) Clerk counters:
   (i) `DNS_CLERK_PROTOCOL_ERROR_COUNT`.
   (ii) `DNS_CLERK_AUTHENTICATION_COUNT`.
   (iii) `DNS_CLERK_READ_OPS_COUNT`.
   (iv) `DNS_CLERK_CACHE_HITS_COUNTs`.
   (v) `DNS_CLERK_CACHE_BYPASS_COUNT`.
   (vi) `DNS_CLERK_WRITE_OPS_COUNT`.
   (vii) `DNS_CLERK_MISC_OPS_COUNT`.

(b) Remote clearinghouse counters:
   (i) `REMOTE_CH_READ_OPS_COUNT`.
   (ii) `REMOTE_CH_WRITE_OPS_COUNT`.

`adver_service_rpc.c` counts the number of clearinghouses in the cache (results accumulated in variable `clh_count`).

CDS also keeps information on the cache size. The size is initialized by routine `casys.c`. If the size is explicitly specified, and is less than the storage configured, the specification is used. Otherwise, it is set to 0.5% of physical memory.

REFERENCES

[Alle] Arnold Allen, *Probability, Statistics, and Queueing Theory — With Computer Science Applications*, Academic Press, Inc., Orlando, FL, 1978.

[Frie] Richard Friedrich, ‘‘DCE Performance Measurement Architecture and Design Guidelines’’, OSF DCE-RFC 33.0, to be published.

[Ferr] Domenico Ferrari, et al, *Measurement and Tuning of Computer Systems*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1983.

[Hubb] Mark Hubbard, ‘‘DCE SIG Serviceability Requirements Document’’, OSF DCE-RFC 11.0, August 1992.

[ISO] ISO/IEC JTC, ‘‘Performance Management Working Document’’ — Seventh Draft, Arles, May 1991.

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

[Lazo] Edward Lazowska, et al, *Quantitative System Performance*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1984.

[Sear] Barry Searle, ‘‘DME/DCE Managed Objects Requirements Document’’, OSF DCE-RFC 38.0, April 1993.

[UI] Unix International — Performance Management Working Group, ‘‘Requirements for a Performance Measurement Data Pool’’, Revision 2.3, May 1993.

AUTHOR’S ADDRESS

Rich Friedrich
Hewlett-Packard Company
19410 Homestead Road
Mailstop 43UA
Cuptertino, CA 95014
USA

Internet email: richf@nsa.hp.com
Telephone: +1-408-447-2425