Implementing Performance Flight Recorders in a Distributed Computing Environment with A+UMA

Neil J. Gunther
Computer Dynamics Consulting, Mountain View, CA 94040
gunther137@aol.com, gunther@oes.amdahl.com

Leon M. Traister
Amdahl Corporation, Sunnyvale, CA 95054
lmtra@oes.amdahl.com

1. Introduction

Over the last five years, and most recently under the sponsorship of the Computer Measurement Group (CMG), the Performance Measurement Working Group (PMWG) -- a group comprised of representatives from companies like: Amdahl, AT&T/NCR, BGS, Hitachi, HP, IBM, Instrumental, OSF, Sequent and many others -- has been designing a framework for the capture and transport of distributed performance data called the *Universal Measurement Architecture* or UMA (pronounced "you-mah") for short [PMWG]. This level of architectural sophistication is required to address the difficulties of measuring performance across many software layers in the many geographical locations of a modern distributed computing environment.

The UMA reference model defines four layers and two interfaces as shown in Figure 1. These layers and interfaces are briefly described from the bottom up, starting with the Data Capture Layer.

- **Data Capture Layer**: The Data Capture Layer is responsible for collecting raw data. Its architecture together with the Data Capture Interface (DCI) allow data from multiple sources to be collected by a single consumer above the DCI, and this in turn improves the synchronization of the data collection.
- **Data Capture Interface**: The Data Capture Interface is the interface between the Measurement Control Layer and the Data Capture Layer. It provides the means for dynamically extending data collection to new providers such as databases without affecting existing programs.
- **Measurement Control Layer**: The Measurement Control Layer schedules and synchronizes data collection through the Data Capture Interface.

- **Data Services Layer**: The Data Services Layer accepts measurement requests from Measurement Application Programs (MAPs) through the Measurement Layer Interface (MLI), and distributes data to the destination requested by the MAP. A destination may include, the MAP itself, a private file or the UMA Data Storage (UMADS), which will be described later.

Note that the interface between the Data Services Layer and the Measurement Control Layer is not formally specified. These two layers, though functionally distinct, and which constitute a logical service layer for the MLI, may be combined in some implementations.

Agents at the Data Capture Interface

Agents may be created directly at the DCI in cases where neither historical data nor access to distributed data are needed by the agent.

Measurement Layer Interface

The Measurement Layer Interface (MLI) is the interface between the Measurement Application Layer and the Data Services Layer. It provides the medium for all interactions between a MAP and UMA, thus isolating the application for the implementation details of the rest of UMA. Separate instances of the Measurement Layer Interface exist as a library linked to each active MAP.

The Measurement Layer Interface allows transparent communication across networks, therefore a MAP running on one system can request and examine data from another system. Together with the Data Services Layer, it provides an infrastructure for the distribution of data over large numbers of heterogeneous sites and multiple platforms.

Measurement Application Layer

The Measurement Application Layer consists of the various Measurement Application Programs (MAPs) that provide services for technical support of management goals. These MAPs may consist of performance monitors, capacity planning tools, tuning advisors, and so on.

Amdahl Corporation has developed an implementation of the MLI and agent portions of the UMA Reference Model (labeled A+UMA® [AUMA]) and Hatachi Corporation has developed a working DCI portion. This work is a necessary part of the X/Open branding process. In addition, Amdahl has developed a set of MAP-level products to support enterprise performance management. This paper presents an overview of the UMA architecture and a status report on A+UMA engineering developments.

2. Open System Measurement Problems

The commercialization of POSIX-based computing is continuing at a rapid pace adding capabilities not just expected, but desperately needed by operators in commercial datacenters. One such feature is performance management. Those familiar with mainframe data processing environments are used to having sophisticated tools available

to determine resource utilization, predict system capacities and growth paths, and even to compare CPU models for making procurement decisions.

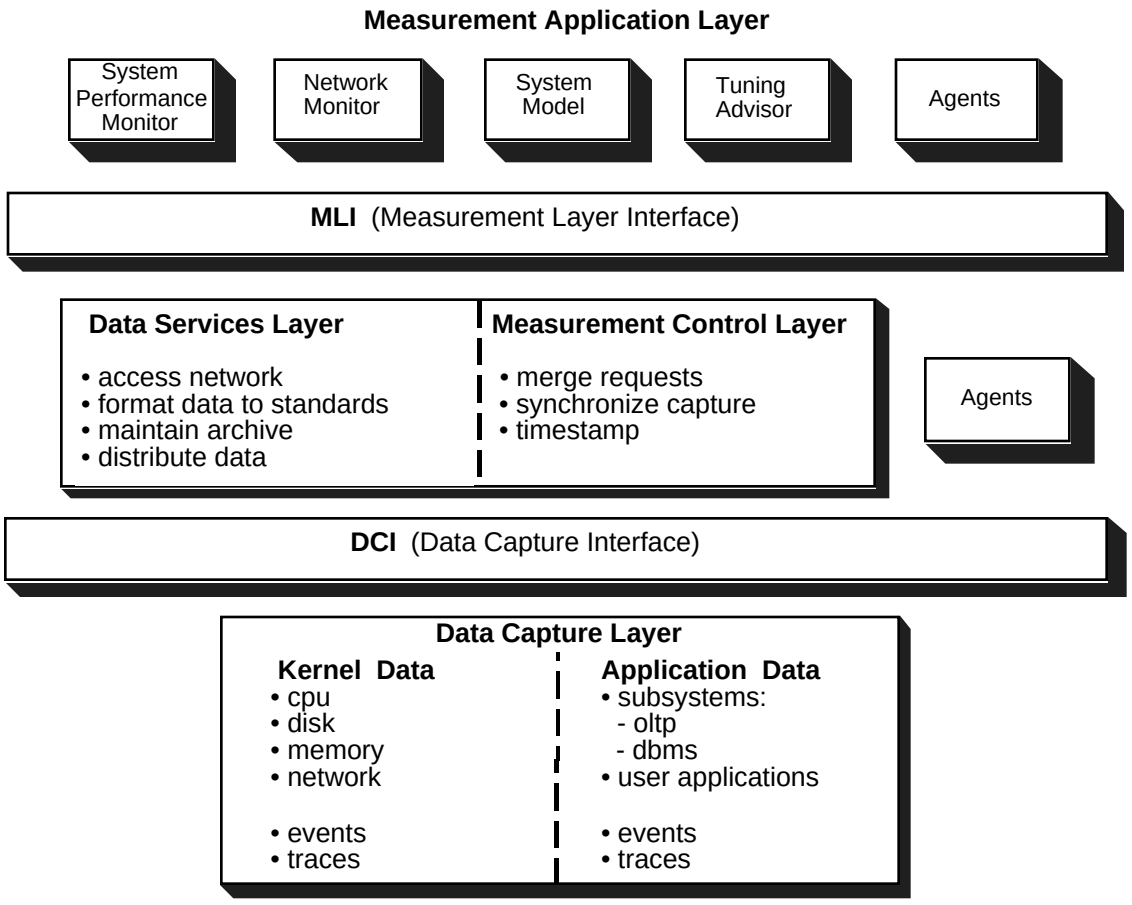

Figure 1. The UMA Reference Model

Although the open system concept is creating a revolution in applications development and system migration paths, certain capabilities-such as performance management-have not been standardized. Currently, no UNIX system vendor provides enough performance management functionality, and certainly no two vendors provide equivalent functionality.

Key areas being considered by the CMG/PMWG include performance data availability and interfaces for its collection. Until the data and interfaces are standardized, each computer vendor, performance software vendor, or large end user is faced with the task of kernel modification to collect the necessary data, development of a proprietary kernel interface to move the data to user-space, and development of "roll-your-own" performance monitoring and management software system. Until such interfaces are standardized, no compatible performance management tools can be built because of the

cost of their migration between operating system versions or POSIX-based system implementations.

As open systems become the platforms of choice for larger, faster, and more complex computer systems, there is an increased need to effectively manage these systems. But there exists little software to support performance management of these complex systems. For example, administrators of standard UNIX systems must rely on the system activity reporter (SAR) data to manage their systems. For reasons described below, this data does not meet all the needs of system administrators. Operators responsible for the performance of large applications, such as those involving database applications, must rely on process accounting data to measure the activity of their application, but when the application is distributed, this per platform performance data is difficult to correlate.

There are several reasons for the lack of performance management software. One reason is that many of the desired metrics are not available. Another reason is the fear that release-to-release kernel changes will make it necessary to frequently modify performance-related applications. This discourages developers from using any but the most basic metrics or developing any but the most basic applications, particularly in cases where the application must execute on platforms supplied by different vendors. There are, furthermore, no well-defined interfaces for obtaining even the existing performance data from the kernel, and the current access methods, are restrictive and expensive.

3. Current UNIX Performance Measurement Architecture

Considering UNIX as a more mature open systems example, the */dev/kmem* interface has historically been the primary interface used by UNIX System performance measurement utilities for extracting data from the kernel. If a program is aware of the name of a particular data structure, it can find the virtual address of that data structure by looking at the symbol table of the UNIX bootable object file. It can then open */dev/kmem* to seek to and read the value of that data structure. The advantage of this approach is its generality: if the address of a data structure can be found, its value can be read. But its generality is also a disadvantage. Since almost any data structure can be used to provide performance data, the tendency is to do so without regard to whether it is supported. This makes it very difficult to maintain a performance application across releases when data structures change. For example, programs such as *ps* and *sadc* have been notoriously difficult to maintain from release to release.

There is also the issue of processing cost. The retrieval of each virtually contiguous piece of information requires a seek system call and a read system call of */dev/kmem*. If there are many such pieces, the CPU costs of gathering the information can be very high. And since each piece requires a separate seek and read, it is very hard to guarantee that the data obtained is consistent.

Then there is the issue of access permissions. For security reasons, */dev/kmem* is not set to be readable by ordinary users. Thus programs such as *ps* and *sadc* must be run as

setuid or *setgid* programs. Ordinary programs must invoke either *ps* or *sadc* and read data either through pipes or files. This adds to the cost of accessing this information.

Then there is the issue of binary compatibility. In order to reduce the number of seeks and reads necessary to obtain the data, many metrics are combined into a single data structure (e.g. *sysinfo* in UNIX). The result is that programs must be aware of the layout and contents of the data structure. If the data structure layout or content change significantly between releases, binary compatibility cannot be maintained; the programs must be recompiled with new headers that reflect the new data structure layout and contents.

Another issue to consider is that of data synchronization. Using a variety of user space collectors to gather data can result in skewed collection times for various data items. This is illustrated in Figure 1. Here the collection times from *sar* and *stats* do not reliably correspond (due to scheduling delays for each process) with the result that the usefulness of the data is impaired. A common source of user level collection would reduce such time skews.

Finally, we must consider commercial distributed computing environments. In the past, performance analysis activities of a single platform at a time were meaningful because most, if not all, of the processing of a user interaction took place on a single platform. In the emerging open systems environment, however, this is no longer the case. Figure 2 illustrates the situation where a user interaction is serviced by processing on a number of platforms and in addition, these platforms may be supplied by a variety of vendors. In this case, the response time experienced by the user is dependent on the delays on the individual service platforms and on the delays of various network components. To carry out an analysis of response time requires that data be captured and tagged with identification at least at a transaction level and that there be a mechanism that can gather this data from distributed systems where it is captured.

To help address the above data collection issues and limitations, the CMG/PMWG has developed the following three specifications for the Universal Measurement Architecture (UMA):

- UMA Performance Measurement Data Pool Specification,
- UMA Data Capture Interface Specification (DCI),
- UMA Measurement Layer Interface Specification (MLI).

The rest of this paper reports on a working implementation of A+UMA and how it is used to render performance flight recorders for large-scale commercial distributed computing environments.

4. Timing Chains and Flight Recorders

The view depicted in Figure 2 is not far removed from that of computer performance models. More specifically, a performance model analysis requires time-correlated data for local performance measures of:

- CPU user and system service times (not just percentages)
- I/O subsystem service times
- Network latencies
- DBMS latch-wait statistics
- DBMS process service times

per business transaction type.

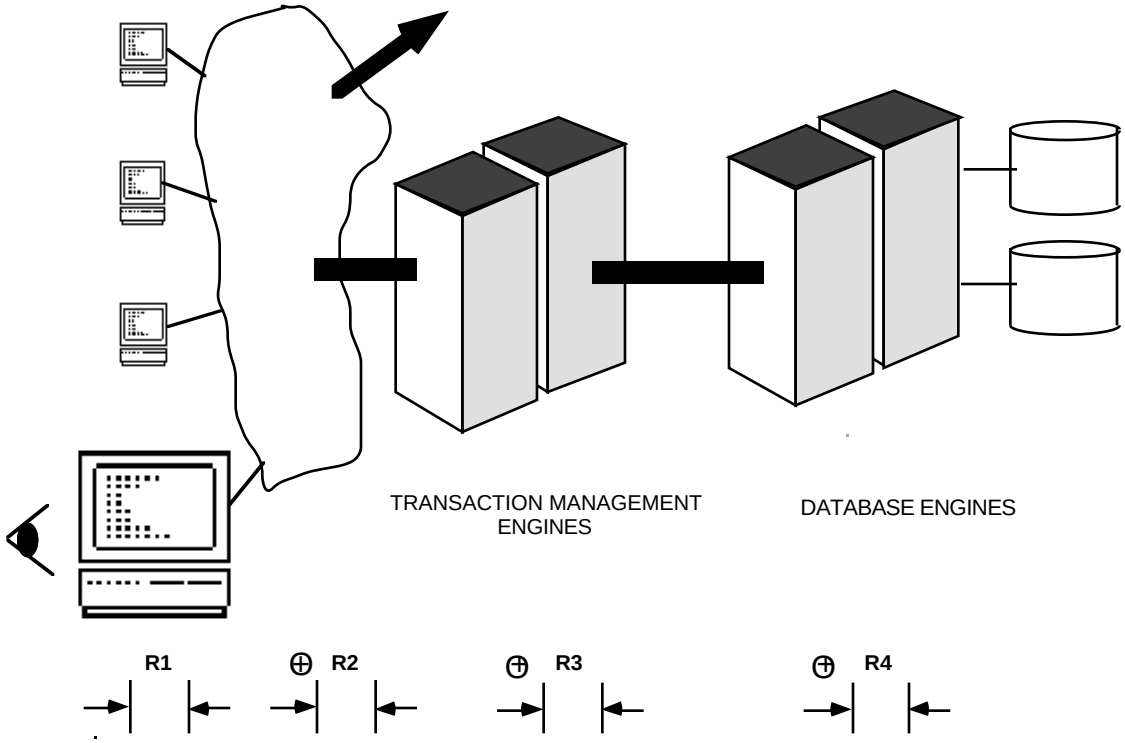

Figure 2. Components of a Distributed Transaction

A performance model aims to reconcile the total business transaction round-trip response time with the presence of local system bottlenecks. In general, this requires knowledge of

delays at each contiguous software component that handles a business transaction during its "flight" through the system. Without synchronized instrumentation, all we have is a series of “black boxes”. The UMA architecture, on the other hand, offers the possibility of implementing the type of black boxes used in aircraft i.e., the "flight recorder". Just like the FAA, a common recording format and common playback tools become a necessity. One of us (NJG), has proposed the same solution to address some of the credibility issues currently faced by benchmarking organizations [GCMG] such as: TPC [GTPC] and SPEC [GSPEC].

On average, the sum of all the local delays (residence times) should add up to the measured response time for that transaction type, within some prescribed tolerance. The performance modeler sees the flight of the transaction as a unit of work consuming resources at a series of queueing centers (Figure 2) representing the various software components. The number of queues is determined by the location of measurement probes.

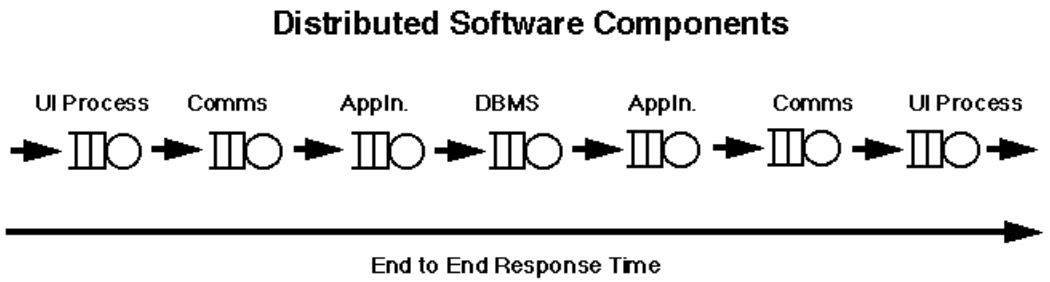

Figure 3.

More formally, the mean residence times (wait + service) must sum to the average end-to-end response time. Bottleneck detection requires that the utilization of computational resources also be measured while the transaction is in residence at each center.

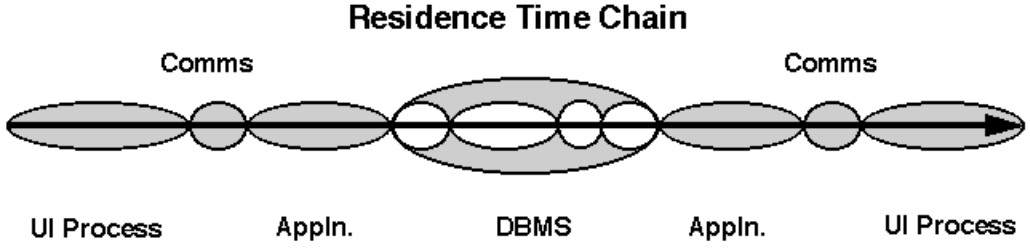

Figure 4.

From the user standpoint, on the other hand, sees the flight of the transaction as passing through a linked chain of components (Figure 4). The length of the chain corresponds to the system response time, RT, for that transaction. The number of links in the chain, once again, is given by the probe points. The size of each link corresponds to the residence time at each of the queueing centers in Figure 3. Since the residence times are different for each component, not all links have the same size but there cannot be any missing links! A little more formally:

$$RT = \sum_{links} size\ (link)\ .$$

The distribution of link sizes, shown in Figure 4 as shaded links, is just one possibility. Additional probe points could be inserted, for example, within the DBMS software such that link would be replaced by a sub-chain of (unshaded) smaller links.

For probes, most Unix systems have SAR data (at a minimum), MVS systems have RMF and SMF performance data, and DBMSs can report performance statistics. The problem remains, however, that such performance data is not only incomplete, it has neither a common format nor a common repository and carries a high collection overhead (especially in the case of database management systems). Some vendors offer more sophisticated performance tools than the typical UNIX suite but these also tend to be point solutions that carry a lot of proprietary baggage.

Moreover, every significant software component would report performance data differently, in different files, in different places and anyone who wished to review those data would not only need to have the corresponding tools, but also the ability to assemble such discontiguous data into the correct time-ordered sequence.

5. Unification Through UMA-fication

Figure 5 gives a schematic impression of how UMA ties together distributed data collection along the timing chain (oriented vertically on the left side of the Figure), with storage in a location-transparent database that can then be read by the appropriate Measurement Application Program (MAP) which is typically a GUI-based analysis tool of the type shown in Figure 6 below. Application probes are realized via a procedure call, *umaPostData*(), in the current A+UMA implementation. At present this is done in lieu of having an integrated DCI layer in A+UMA.

One of the most important strengths of UMA is that control of both data collection and data filtering are supported via a set of APIs [AUMA] now accepted as base documents

by X/Open. In this sense, UMA is an Open Architecture that is vendor-independent. Vendor-specific data collection is handled via these APIs. The UMA architecture incorporates the notion of a time-indexed database called UMADS. A common set of software probes can write performance data into UMADS and a common MAP interface can read historical performance data from the same UMADS database. The data format seen by MAPs is specified by the UMA standard but the actual field names for UMA classes is vendor or application specific. In this way UMA is able to provide a flight recording and playback mechanism to reveal how particular business transactions are performing. This is precisely the kind of integrated information that is required for both bottleneck analysis and capacity forecasting.

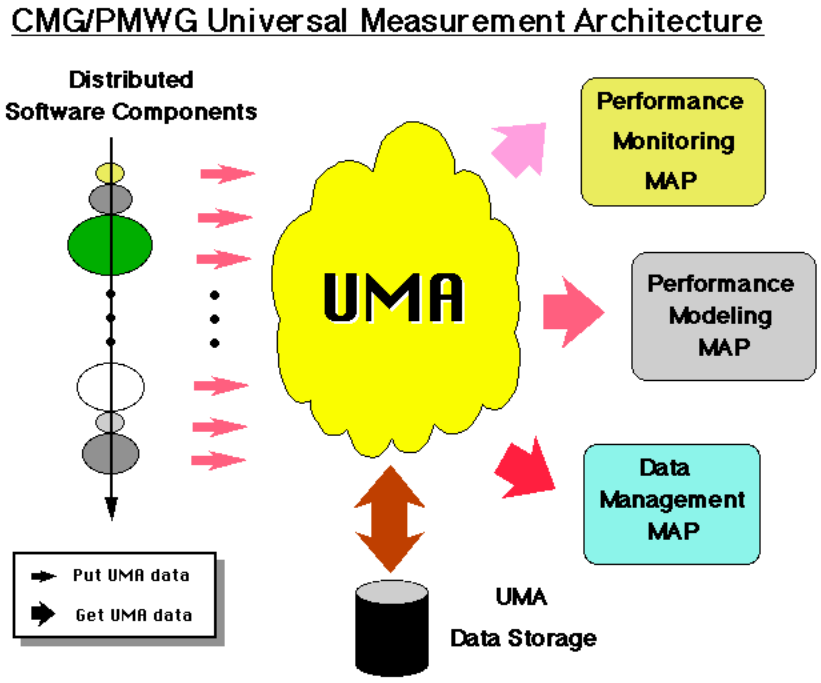

Figure 5.

Table 1 contains time-indexed UMADS data, in ASCII format, to show how it is organized into class and subclass data structures for the case of some selected Unix kernel performance metrics. Of course, a human would prefer to "replay" the transaction flights graphically using specially developed MAPs of the type shown in Figure 6. In this MAP

there is the capability of sweeping back and forth through the UMADS historical records. The UMADS contains performance data from the DBMS and the business application; not just the operating system.

CLASS: Processor
SUBCLASS: Global Measured Processor Times - Basic Segment

Timestamp	procr_id	usr_tm/s (usec)	sys_tm/s (usec)	intr_tm/s (usec)	wait_tm/s (usec)	idle_tm/s (usec)
03/08/94 11:51:00	0	22540.77	130540.45	0.00	84282.87	762661.94
03/08/94 11:51:00	1	39397.34	180326.14	0.00	69386.36	710916.19
03/08/94 11:52:00	0	8165.23	266453.06	0.00	26162.06	699043.56
03/08/94 11:52:00	1	26495.33	293948.22	0.00	21829.49	657550.88

CLASS: Processor
SUBCLASS: Global Processor Counters - Basic Segment

Timestamp	procr_id	syscalls/s	hard_intr/s	soft_intr/s	voln_swtch/s
03/08/94 11:51:00	0	192.27	0.00	192.27	139.63
03/08/94 11:51:00	1	217.98	0.00	217.98	140.47
03/08/94 11:52:00	0	53.07	0.00	53.07	143.40
03/08/94 11:52:00	1	106.42	0.00	106.42	148.52

CLASS: Processor
SUBCLASS: Global Processor Counters - Extended Segment

Timestamp	procr_id	ext_intr/s	prog_intr/s	io_intr/s
03/08/94 11:51:00	0	0.00	0.00	120.86
03/08/94 11:51:00	1	0.00	0.00	276.98
03/08/94 11:52:00	0	0.00	0.00	122.47
03/08/94 11:52:00	1	0.00	0.00	309.87

CLASS: Memory
SUBCLASS: Global Real Memory Counters - Basic Segment

Timestamp	user (kb)	free (kb)	bcache (kb)	system (kb)	usr_priv (kb)	usr_share (kb)
03/08/94 11:51:00	0	2917	0	0	0	0
03/08/94 11:52:00	0	2908	0	0	0	0

CLASS: Memory
SUBCLASS: Global Paging Counters - Basic Segment

Timestamp	procr_id	pg_flt/s	page_in_pg/s	page_out_pg/s	page_in_op
03/08/94 11:51:00	0	9.88	2.76	1.78	0
03/08/94 11:51:00	1	7.23	1.88	4.45	0
03/08/94 11:52:00	0	0.37	0.23	0.13	0

Table 1.

Although the format in Table 1 resembles SAR output, any similarity is purely superficial. Many other system performance metrics are attached to each UMADS interval record. In fact, a page three foot wide would be required to display all the fields belonging to this example.

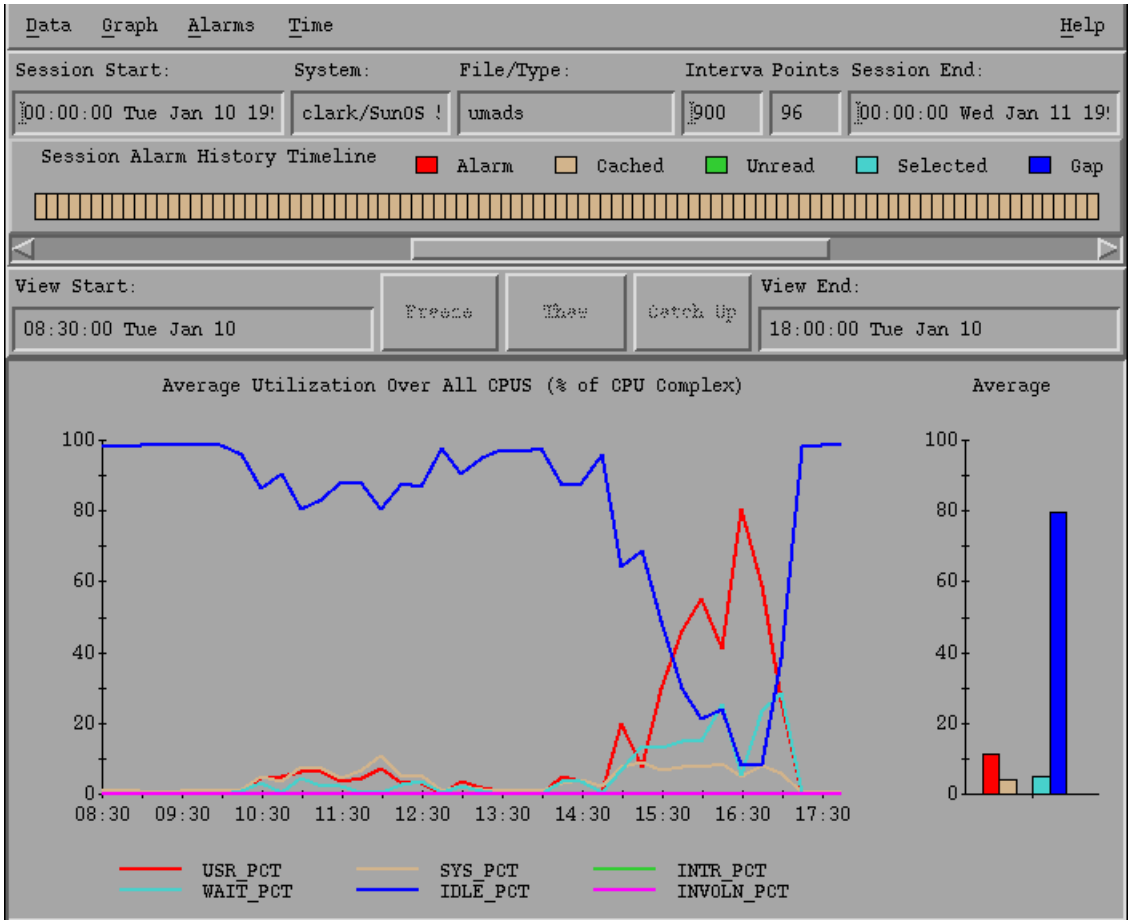

Figure 6.

6. Summary

We have presented an overview of the Universal Measurement Architecture (UMA) that is being standardized by X/Open. Amdahl was a founding member of the PMWG and has developed a non-DCI subset of the UMA specification (A+UMA) as a suite of performance management products. These A+UMA products serve the dual purpose of

offering proof-of-concept for the X/Open standardization process and addresses the needs of managing performance in a commercial DataCenter setting.

A+UMA products have now been deployed at several large commercial accounts. Examples of how A+UMA flight recorders are helping to solve performance management problems may be the subject of a future report.

7. References

[AUMA] The interested reader can find more information about the UMA specification and A+UMA performance management methodology at Amdahl's web site URL: http://www.ccc.amdahl.com/doc/products/oes/pm.oes/perfhome.html.

[GCMG] N.J. Gunther, "The Answer is Still 42 But What's the Question?: The Paradox of Open Systems' Benchmarks," CMG '94 Proceedings, Orlando, Florida, vol. 2, p.732, December 1994.

[GTPC] N.J. Gunther, "Thinking Inside the Box and the Next Step in TPC Benchmarking -- A Personal View," *TPC Quarterly Report*, January 1995.

[GSPEC] N.J. Gunther, "Flight Recorders, Timing Chains, and Directions for SPEC System Benchmarks," *SPEC Newsletter*, vol. 7, #1, p.7, March 1995.

[PMWG] Some of the material in sections 1-3, of this paper, appears elsewhere in PMWG documents contributed by Ram Chelluri, and other PMWG members.