2011 Eighth IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems

# Large Scale Monitoring and Online Analysis in a Distributed Virtualized Environment

Rajat Mehrotra[*] Abhishek Dubey[†] Sherif Abdelwahed[*] Weston Monceaux[#]

[*] Electrical and Computer Engineering, Mississippi State University, Mississippi State, MS
[†]Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN
[#]US Army Corps of Engineers, ERDC, Vicksburg, MS

## Abstract

*Due to increase in number and complexity of the large scale systems, performance monitoring and multidimensional quality of service (QoS) management has become a difficult and error prone task for system administrators. Recently, the trend has been to use virtualization technology, which facilitates hosting of multiple distributed systems with minimum infrastructure cost via sharing of computational and memory resources among multiple instances, and allows dynamic creation of even bigger clusters. An effective monitoring technique should not only be fine grained with respect to the measured variables, but also should be able to provide a high level overview of the distributed systems to the administrator of all variables that can affect the QoS requirements. At the same time, the technique should not add performance burden to the system. Finally, it should be integrated with a control methodology that manages performance of the enterprise system.*

*In this paper, a systematic distributed event based (DEB) performance monitoring approach is presented for distributed systems by measuring system variables (physical/virtual CPU utilization and memory utilization), application variables (application queue size, queue waiting time, and service time), and performance variables (response time, throughput, and power consumption) accurately with minimum latency at a specified rate. Furthermore, we have shown that proposed monitoring approach can be utilized to provide input to an application monitoring utility to understand the underlying performance model of the system for a successful on-line control of the distributed systems for achieving predefined QoS parameters.*

## 1 Introduction

In general, distributed systems are comprised of several data processing nodes deployed over a network and attached through communication channels for exchange of information and control commands. There is tremendous growth in usage of distributed system paradigm for health-care, e-commerce, transport, manufacturing, social networking, and enterprise applications. With the increase in usage, size, and implementation complexity of the distributed system architecture, efficient and accurate monitoring of these systems to achieve QoS requirements of the underlying application, has become a challenging task for researchers. Furthermore, due to large size and complexity of distributed systems, they generate large number of events with varying level of importance for the administrator. Occasionally, multiple low-level events should be combined to generate high-level overview to understand the abnormal behavior of the system operation. Moreover, presence of multiple processes running on separate nodes, synchronization issues among nodes, communication delay, and bottleneck resource performance make administration and analysis of system event reports extremely difficult that results in to outage or poor performance of these systems. Due to the mission critical nature of the underlying application, distributed systems require an effective monitoring framework as well as real time on-line analysis of the monitoring data to make appropriate configuration or resource provision changes in the system to maintain the QoS requirement. Additionally, distributed systems require an effective monitoring technique, which can work in a distributed manner similar to the system in observation, and reports each event to the system administrator with maximum accuracy and minimum latency. Distributed event based systems (DEBS) can be used for monitoring and management of distributed systems for faults and system health notifications through system alarms. Moreover, the DEBS based monitoring is able to provide the notion of system health in terms of fault/status signals, which will help to take the appropriate control actions to maintain the system in safe boundary of operation.

**Contribution:** In this paper, we propose an effective

978-0-7695-4380-2/11 $26.00 © 2011 IEEE
DOI 10.1109/EASe.2011.17

event based monitoring framework for distributed systems hosted in virtualized environment and present the experimental steps with results for monitoring the underlying application as well as complete system. The proposed monitoring framework is based upon the appropriate and accurate combination of each state-of-the-art monitoring utility used for each component of the distributed system. According to the results, the proposed event based framework has minimal overhead / latency with high accuracy for the system monitoring and modeling the system performance. Additionally, the monitoring information generated by the proposed approach can also be used to control the distributed systems in autonomic manner to achieve / maintain the multi-dimensional QoS requirements of the deployed system. Discussion about the controller is not included in this paper, but is available as technical report in [7].

**Outline:** This paper is organized as follows. Previous efforts for distributed event based monitoring are outlined in section 2 and some preliminary concepts are presented in section 3. Our experiment setup is described in section 4, while the proposed approach is introduced in section 5. The benchmarking and experimental results of the monitoring framework are described in section 6 and benefits of the approach are listed in section 7. Finally, conclusions are presented in section 8.

## 2 Related Work

In [6] an event based monitoring approach is presented for distributed systems with help of "GEM", a declarative and interpreted language, which can specify the operation for event monitors. Each event monitor can be controlled interactively with help of GEM scripts, which contain the actions to be taken in case of particular trigger. Additionally, GEM trigger scripts can be changed and loaded dynamically for the dynamically changing pattern of the events in the system.

Another novel approach "NetLogger" is described in [10] for distributed system monitoring, which consists of time stamp based event logs, visualization tools, and real time system state logs. This approach combines network, host, and application level monitoring to provide complete aggregate view of the system. It is extremely lightweight and can be used for pinpointing the component responsible for failure or fault in the distributed system.

An event based programming model is introduced in [9] for flexible monitoring of distributed information systems (DIS), where an administrator can specify the monitoring of critical properties for successful functioning of a DIS. This approach translates specified events in to event predicates, which is detected by the distributed algorithms for monitoring of DIS. Any deviation of critical properties from the desired behavior is considered as malfunctioning of the DIS under given operating and environmental conditions.

The primary goal of this paper is to present an event based distributed monitoring framework for distributed systems through monitoring all of the different computing nodes and their processes with minimum jitter in recorded events, and provide an accurate high level overview of the current state of the system with reference to QoS parameters. In this approach, we have tried to combine the framework described in [8, 3] with added monitoring utilities for virtualized environment. Additionally, we have shown results of combining the proposed framework with a predictive controller to maintain the distributed system performance.

Timing jitter is one of the major difficulties for accurate monitoring of a computing system for periodic tasks when the computing system is too busy in running other applications instead of the monitoring sensors. [3] presents a feedback based approach for compensating timing jitter without any change in operating system kernel. This approach has low overhead with platform independent nature and maintains total bounded jitter for the running scheduler or monitor. Our current work uses this approach to reduce the distributed jitter of sensors across all machines.

## 3 Preliminaries

Our work uses **Data distribution services (DDS)** middleware standard. Specifically, we use the Opensplice Community edition [1]. It is a middleware standard for publish-subscribe communication model that overcomes the typical shortcomings of traditional client-server model. It features extensive QoS configurations. In publish-subscribe model, message sender and message receiver are decoupled. Messages are distributed by publisher without knowing who will receive the messages. Each message is associated with a special data type called topic. A subscriber registers to one or more topics of its interest. DDS guarantees that the subscriber will receive messages only from the topics that it subscribed to. In DDS, a host is allowed to act as a publisher for some topics and simultaneously act as subscriber for others.

An initial design and integration of a hierarchical event based distributed system monitoring framework with a scientific workflow management system is presented in [8]. This approach shows an improvement in monitoring real time high performance computing systems by applying an effective work flow management through tracking the progress of scientific job execution and maintaining the timed state machine for those jobs.

[1]http://www.opensplice.org/cgi-bin/twiki/view/Community/WebHome

| Name | Cores | Description | RAM | DVFS | VMs |
|---|---|---|---|---|---|
| Nop01 | 8 | 2 Quad core 1.9GHz AMD Opteron 2347 HE | 8GB | No | Nop04,Nop07 (Development Machines) |
| Nop02 | 4 | 2.0 GHz Intel Xeon E5405 processor | 4GB | No | Nop05,Nop08 (Client Machines) |
| Nop03 | 8 | 2 Quad core 1.9GHz AMD Opteron 2350 | 8GB | Yes | Nop06,Nop09 (Application server) |
| Nop10 | 8 | 2 Quad core 1.9GHz AMD Opteron 2350 | 8GB | Yes | Nop11,Nop12 (Database Server) |

**Table 1. Physical machine configuration.**

## 4 Computing System Setup

Our set up consists of four physical computing and virtual computing nodes. Table 1 summarizes configuration of physical machines and the virtual machines (VM) running on all physical machines used in the experiments presented in this paper. *Nop03* and *Nop10* both have Dynamic Voltage and Frequency Scaling (DVFS) capability that allows administrator to tune the complete physical node or its individual core for desired performance level. We have used Xen Hypervisor (`http://www.xen.org/`) and Linux version 2.6.18-92.el5xen to create and manage physical resources (CPU and RAM) for cluster of Virtual Machines (VMs) on these physical servers. This table also shows the roles played by these machines in an experimental enterprise system. Client machines are used to generate request load. Application servers run the open source version of IBM's J2EE middleware, *Web Sphere Application Server Community Edition* (WASCE) and Database machines run MySQL.

We use *Daytrader*, an open source benchmark application developed to compare and measure the performance of a distributed multi-tier J2EE web servers in industry, as our web application. It drives a trade scenario based application that allows users to monitor their stock portfolio, inquire about stock quotes, buy or sell stock shares, as well as measure the response time for benchmarking.

Out of the box, the *Daytrader* application puts most of the load on the database server. We modified the main trade scenario servelet to allow shifting the processing load of a request from the database node to the computing node. This was done to emulate business enterprise loads in highly dynamic environment. For the basic service, we distributed the *Daytrader* application across multiple instances of WASCE, deployed over virtual machines. *Daytrader* instances belonging to same cluster share common instance of database. Finally, a modified *Daytrader* client is used to generate workload requests based on a given throughput profile (specified as a lookup table). This table contains the sampling period and number of requests in that period.

In addition to *Daytrader* client, we use the Httperf [1] benchmarking application tool for measuring the web server performance. This tool supports both http 1.0/1.1 and SSL protocols. It provides flexibility to generate various workload patterns (poisson, deterministic, and uniform) with numerous command line options for benchmarking. At the end of experiment, it prints out the detailed performance statistics of the experiment. Due to its availability as open source, we have modified it to print the performance measurements of our interest periodically while running the experiment. Furthermore, to support the high workload between the client - web server interface, changes were made to remove the file socket limitation in the Linux OS installed over client and webserver machines.

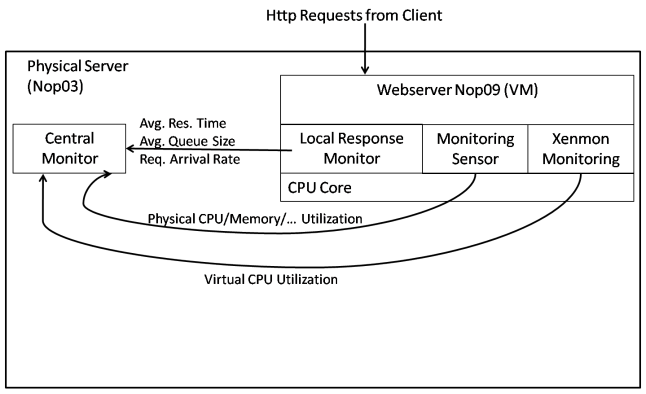

**Figure 1. Main components of the Monitoring Framework**

## 5 Proposed Approach

For efficient management of the distributed systems in a highly dynamic and unpredictable environment, an effective monitoring framework is needed, which can provide the exact notion of system state that can help to take actions to meet the customer and market needs. In other words, The monitoring framework should make the distributed system capable of adapting to the changes in the environment and interaction with its own components in autonomic manner. These systems should follow service oriented architecture (SOA) foot prints and should be ready to integrate with new services in the enterprise domain.

Different components of the monitoring framework are described in following subsections.

## 5.1 Monitoring Utilities

We have used multiple utilities to synchronize all computing nodes and monitor system resources i.e. physical/virtual CPU, memory utilization, disk utilization, and application performance in terms of average queue size, average response time, and arrival workload for a sample period. Description of these utilities is given in following subsections.

### 5.1.1 Synchronization Scripts

The primary concern while running a distributed computing infrastructure is to maintain the time synchronization among all the elements. In general, when a computing node restarts, it gets synchronized with the pre-specified time server. However, the computing node gets out of synchronization with the time due to difference between the local clock and the pre-specified time server clock. Initially the difference can be quite small, but with the time, this difference gets accumulated and results in to range of seconds or sometime minutes too. To remove this time discrepancy, we synchronize all computing nodes (client, application, and database) with help of specially developed scripts, which use Network Time Protocol (NTP version 3). Additionally, we schedule these synchronization utilities to run after a certain time period to maintain the synchronization among all computing nodes.

### 5.1.2 Monitoring Sensors

Specially written monitoring sensors are used on each node to report the physical CPU and memory utilization in every sample period and at the end of the experiment. These sensors are based upon the feedback controller approach to eliminate the problems due to timing jitter in computing systems for periodic tasks [3]. These sensors are deployed on each computing node. Currently, we are using multiple sensors to monitor CPU utilization, memory utilization, and heartbeat as listed in table 2. These sensors report different events of utilization periodically through as DDS message to the central monitor.

### 5.1.3 Xenmon Monitoring

We are using Xen hypervisor built in the physical server to support virtualization technology paradigm. Xen provides a virtual machine monitoring tool named *Xenmon* [4] for measurement of performance metrics related to virtual CPU utilization at a virtual machine in previous execution period. These performance metrics consist of following key parameters: CPU usage indicates percentage of time when a particular physical CPU is used by a particular domain, blocked time indicates the percentage of time when a domain was blocked for some I/O events, and waiting time indicates the waiting time for domain to get the CPU for scheduling. During our experiment, we primarily used the virtual CPU utilization metric to measure the current state metrics of the system. This xenmon monitoring utility reports virtual CPU utilization through DDS approach to the central monitor.

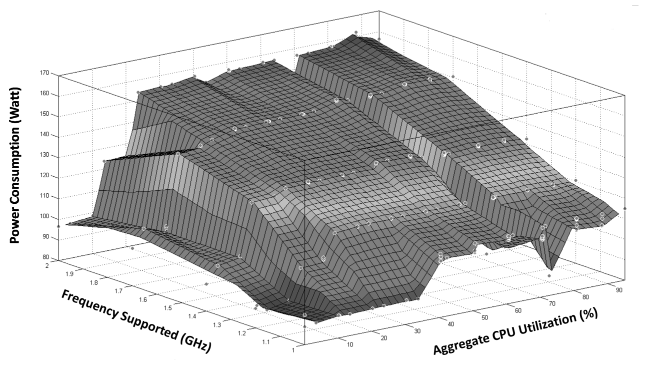

**Figure 2. Power consumption on Nop03 vs CPU frequency and Aggregate CPU core utilization.**

### 5.1.4 Power Consumption Monitoring

During our experiment, for off-line analysis, we measured the real time power consumption with help of a real time watt meter. For real time prediction of the power consumption in the computing node, we have created an aggregated interpolated map of power consumption through off-line analysis of the system. The mutual relationship among physical CPU core utilization, CPU frequency, and power consumption of the physical server is identified. Fig. 2 shows the power consumed on one of the physical server *Nop03* with respect to the aggregate CPU core usage and CPU frequency. An extensive experiment was performed over physical server *Nop03* with help of a specially written script, which exhausted a physical CPU core through floating point operations in increments of 10% utilization independent of the current CPU frequency. With multiple instances of this utility, all eight physical CPU core of the *Nop03* server were loaded in incremental manner for different discrete values of CPU frequencies. CPU frequency of all physical cores (1 to 8) was kept same during each step. Based on this experiment, we created a regression model for power consumption at physical machine with respect to CPU core frequency and aggregate CPU utilization. After analysis of the results (and reconfirmation with several other experiments across other nodes), it was observed that power

| Sensor Name | Period | Description |
|---|---|---|
| CPU Utilization | 30 seconds | Aggregate utilization of all CPU cores on the machines. |
| Swap Utilization | 30 seconds | Swap space usage on the machines. |
| Ram Utilization | 30 seconds | Memory usage on the machines. |
| Hard Disk Utilization | 30 seconds | Disk usage on the machine. |
| CPU Utilization Per Process | 30 seconds | CPU usage of each process on the machines. |
| Swap Utilization Per Process | 30 seconds | Swap space usage of each process on the machines. |
| Ram Utilization Per Process | 30 seconds | Memory usage of each process on the machines. |
| Network Connection | 30 seconds | Number of TCP connections on the machines. |
| Heartbeat | 300 seconds | Periodic liveness messages. |

**Table 2. Monitoring Sensors**

consumption model of a physical machine is non-linear because power consumption in these machines depends not only upon the CPU core frequency and utilization, but also depends non-linearly on other power consuming devices e.g. hard drive, CPU cooling fan etc. As a result, a look-up table with near neighbor interpolation was found to be the best fit for aggregating the power consumption model of the physical machine.

#### 5.1.5 Application Monitoring

Modification in web server code allows us to monitor the web server performance in terms of active java threads, average queue size, incident request rate, response time at application tier, and database tier separately in each sample interval. A specially written java program is developed to collect the the performance data from the web server over the sample interval and provide as input to the **Central Monitor** utility, which can process the data further and create aggregate view of the web server.

### 5.2 System Modeling Utilities

For identification of an accurate system model, extensive experiments have been performed and results have been analyzed with help of an exponential kalman filter [5] implementation. We have implemented an *exponential Kalman filter* to predict the computational nature of the incident requests over web server by predicting the **service time** $S$ and **delay** $D$ of a request by observing the current average response time of the incident request and request arrival rate on the web server. This filter uses an $M/G/1/\infty$ processor sharing (PS) approximate queuing model as the system state equation and considers variation in $S$ and $D$ at previous approximation to estimate the $S$ and $D$ at next sample time. This filter is exponential because it operates on the exponential transformation of the system state variables. This transformation allows us to enforce the $\geq 0$ constraints on the state variables.

An exponential Kalman filter (KF) is used to estimate the system state as mentioned earlier. It is important to note that we can approximate the system as a M/G/1/$\infty$ PS queue if the system has no bottleneck. In the presence of bottleneck, the system utilization (not necessarily CPU) will approach unity. At that time, the system will change to the limited processor sharing (LPS) queue model. However, it is difficult to build a tractable model for LPS queuing systems. Hence, we just identify the operating regions where the system changes the mode between two queuing models and analyze the system in the Infinite PS queue region only.

Written in the term of exponentially transformed variables, $[x1 \in \mathbb{R}; x2 \in \mathbb{R}]$ s.t. $S = exp(x1)$ and $D = exp(x2)$, the equations 1 and 2 define the state update dynamics and observation for a given timed index of observation, $k$. $N$ and $V$ are Gaussian process and measurement noises with mean zero and covariances $Q$ and $R$ respectively. Note that this transformation ensures $S, D \in \mathbb{R}^+$. One can verify that these equations described the behavior of a M/G/1 PS queue. Here, predicted bottleneck utilization is given by $\hat{\rho}_k = \lambda_k * exp(x1_k)$. Additionally, the Kalman filter does not update its state when the predicted bottleneck resource utilization becomes more than 1. The kalman filter works on-line by monitoring the performance data of the web server and estimates the service time $S$ of the incident requests.

$$\begin{pmatrix} exp(\hat{x1}_k) \\ exp(\hat{x2}_k) \end{pmatrix} = \begin{pmatrix} exp(x1_{k-1}) \\ exp(x2_{k-1}) \end{pmatrix} + N(0, Q), \text{ and} \quad (1)$$

$$T = \frac{exp(x1_k)}{(1 - \lambda_k * exp(x1_k))} + exp(x2_k) + V(0, R) \quad (2)$$

### 5.3 Data Processing Utilities

We have developed a $JAVA^{TM}$, based Data Processing utility to process the performance data received from the web server, xenmon monitoring data for virtual CPU utilization, sensor monitoring data for physical CPU and memory

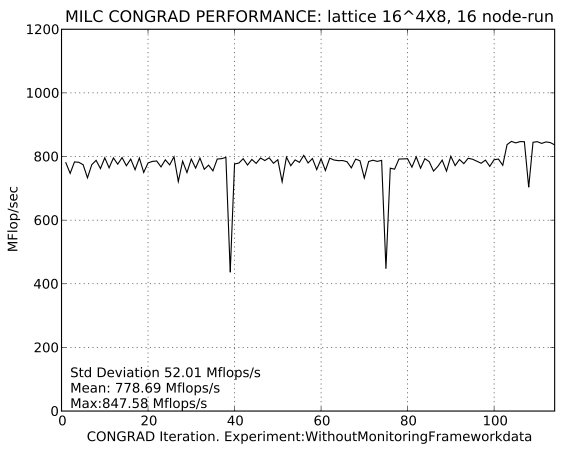

**Figure 3. Baseline performance of a MILC run on 16 nodes. The Y-axis is MFLops/sec. The X-axis shows the iterations.**

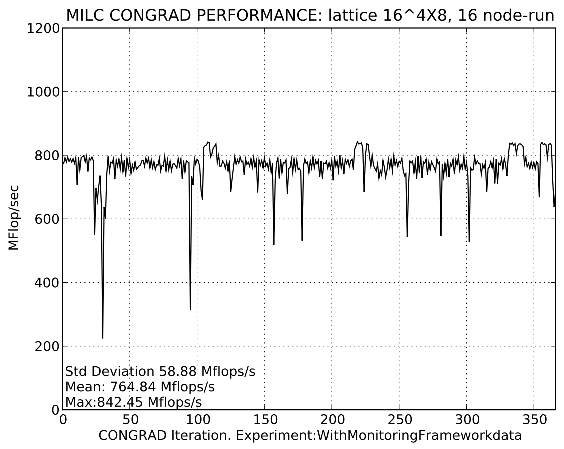

**Figure 4. Effect of our Monitoring framework on the same MILC configuration. This experiment was run longer than the baseline test shown above.**

utilization, and estimated service time from kalman filter. With help of data processing utilities, different monitoring data are co-related to visualize aggregate system overview for performance variables and appropriate data is transfered to the online monitoring framework for estimating the appropriate control actions to apply in the system configuration for maintaining the QoS requirement of the system.

## 6 Experimental Results

This section describes the results of three experiments designed to illustrate the use of monitoring data. The first experiment was performed on 16 physical nodes to measure the performance penalty imposed by the increasing CPU, memory and network utilization. The second experiment was done to use the monitoring data and characterize the web server behavior under varying workload with help of all sets of monitoring data. All monitoring data was analyzed online.

### 6.1 Experiment 1: Performance Impact

A programming model commonly followed in parallel computing jobs is Multiple Instruction Stream and Multiple Data Stream (MIMD). In this model, different nodes execute in parallel on different data streams. However, the computation results from various nodes have to synchronize together in order to solve for boundary conditions. For example, a high energy physics application called MILC[2](MIMD Lattice computation collaboration) has to do a global synchronization every 45 milliseconds. This application is extremely sensitive to extra load on the CPU and change in network traffic. We used this application in this experiment to test the performance penalty of running our sensors. All nodes (total 16) used in this experiment had single CPU. The goal was to study the worst case performance impact. Results for this experiment is shown in Fig: 3 and Fig: 4. It can be seen that the mean performance of the application only dropped by 14 MFlops/sec. The standard deviation increased by 6 MFLops. Fig: 5 shows the average delay between publication of monitoring data and its assimilation by the control system for tracking the system state. The delay was less then 0.28 seconds. This delay includes any time synchronization drift between the worker node and the central monitor.

[2]`http://physics.indiana.edu/~sg/milc.html`

### 6.2 Experiment 2: Online Data Use by a Predictive Controller

This experiment is performed to use the monitoring data gathered by monitoring utilities described in section 5.1 in real time.

**Experiment settings:** This experiment was configured with *Nop09* (physical machine *Nop03*) as the virtual machine running the first tier of *Daytrader* application. With help of Xen, virtual CPU of *Nop09* was pinned to a single physical core and 50% of the physical core was assigned to *Nop09* as maximum available computational resource. Physical memory was also limited to 1000MB for *Nop09*. *Nop11* was configured as database using similar CPU and memory related operating settings over physi-

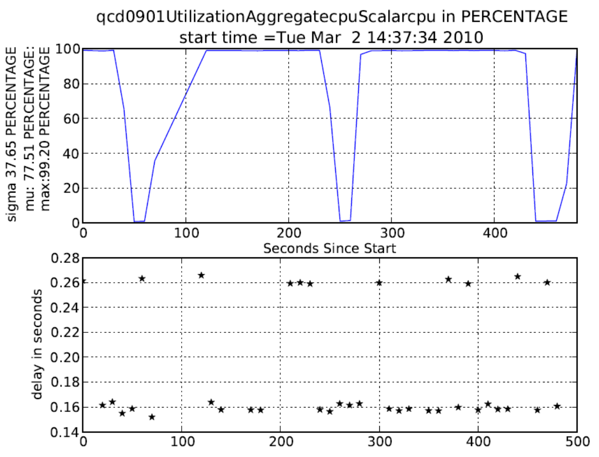

**Figure 5. CPU Utilization and Delay in event report With DDS**

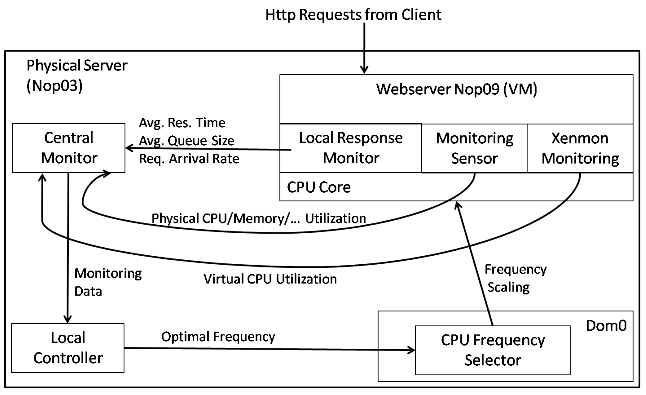

**Figure 6. Integration of Monitoring Framework with a Predictive Controller**

cal server *Nop10*. To simulate real time load scenario, all CPU cores of physical server *Nop03* (except the CPU core hosting Nop09) were loaded approximately 50% with help of utility scripts described in Section 5.1.4. MAX JAVA threads, a parameter that sets the maximum concurrency limit was configured as 600 in the webserver application.

The architecture is shown in Fig. 6. The **Local Response Monitor** monitors the webserver performance on the VM (Nop09) hosting web server. It collects, processes, and reports performance data after every SAMPLE_TIME (in this case it was set to 30 seconds) to the **Central Monitor** running on host machine (*Nop03*). These performance data includes average service time at webserver (computation time at application tier as well as query time over database tier), average queue size (average resident request into the system) of the system during the time interval of SAM-

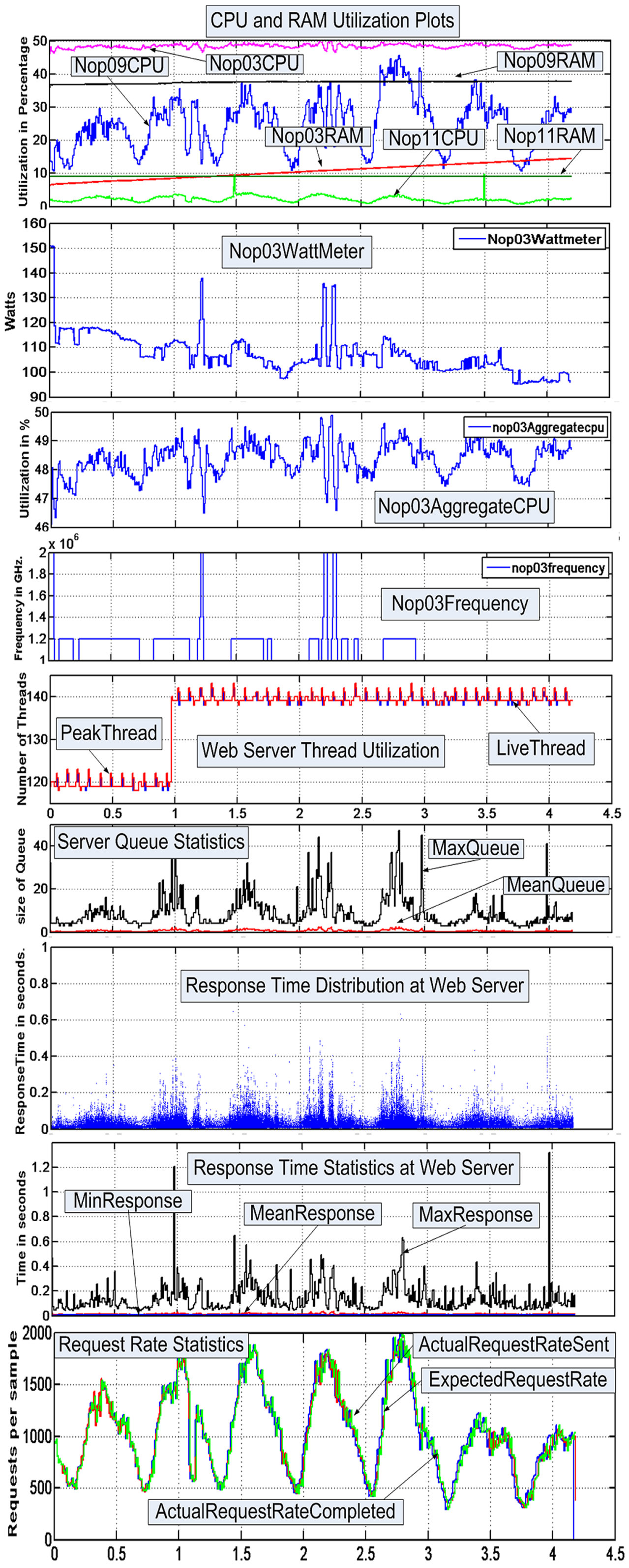

**Figure 7. Web server behavior: Sampling period=30 seconds.**

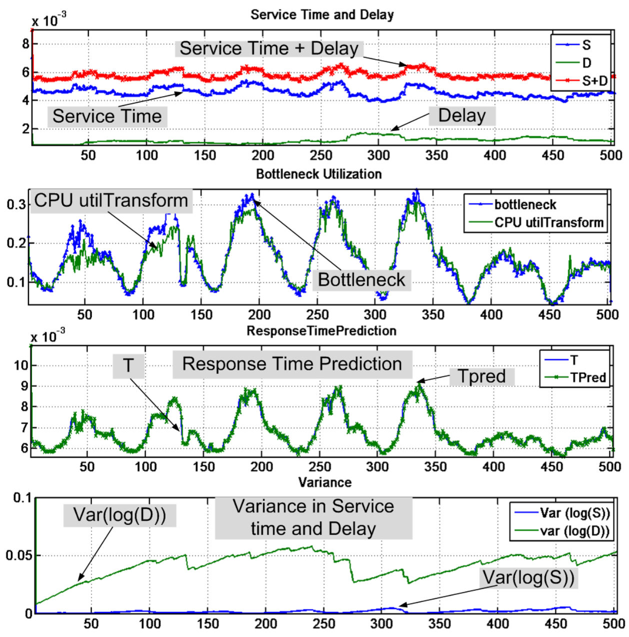

**Figure 8. Online exponential Kalman filter output corresponding to the experiment from Section 6.2 figure 7. Service time and delay are in milli second range. Response time is specified in seconds.**

PLE_TIME, and request arrival rate. The average queue size of the system is measured based on the total resident request in the system at previous sample, (plus) total incident request into the system, and (minus) total completed request from the system in the current sample duration.

We used the exponential Kalman filter described earlier to track the system state online. The two main parameters received from the filter are the current service time and predicted response time. The power model described in Section 5.1.4 was used to estimate the system (physical node of webserver) power consumption (graph not shown).

We are developing an online controller that will use the analyzed data to control the total power consumption by setting the optimal CPU core frequency while maintaining a set response time. Performance of the online controller directly depends upon the accuracy of the different monitoring utilities: Kalman filter estimation for the parameters of the webserver, application monitoring utility, monitoring sensors, xenmon monitoring for virtual CPU utilization, and the power consumption model of the physical system.

Figure 7 shows the generated client request rate, (ActualRequestRateSent), and corresponding server throughput (ActualRequestRateCompleted) for the experiment. Time series "ExpectedRequestRate" shows the request profile supplied to the client, which can be different from the "ActualRequestRateSent" in case of client side connection limitations. This request trace is based on the user request traces from the 1998 World Cup Soccer(WCS-98) web site [2].

The aggregate CPU utilization of webserver and database tier are shown in Fig. 7. Observed response time at webserver is also shown in the same figure.Correlation among request rate, CPU utilization, and response time is clearly visible. Finally, plots of the estimates from the online Kalman filter are shown in Fig 8. It is evident that the observed response time tracks the predicted response time closely. Furthermore, it can be seen the Kalman filter tracks the utilization of bottleneck resource well. In some regions the utilization of bottleneck resource is greater than CPU utilization indicating that a different computation resource is the bottleneck.

## 7 Benefits of the Approach

Experiment 1 is instrumental in showing that our monitoring framework does not impact the system detrimentally.

Experiment-2 describes how the proposed event based monitoring framework can be utilized effectively to analyze and then potentially control a distributed computing system for QoS maintenance with almost negligible overhead but with significantly higher performance.

The proposed monitoring framework is used to monitor the web server characteristic in the real time dynamic environment. The proposed approach utilizes the benefits of DDS approach mentioned in section 3 as well as the accuracy of different monitoring utilities: xenmon, application monitoring, kalman filter, and monitoring sensors. The major benefit of the proposed monitoring framework is that the monitoring utilities can be used for online and autonomic control of the web server behavior to maintain the system close to QoS requirements without many changes to the system. Direct observation of the figure 8 shows that system modeling utility (Kalman filter) tracks the system response behavior and bottleneck resource utilization with high accuracy.

## 8 Conclusion

We have presented a distributed event based monitoring framework for a distributed enterprise system hosted in a virtualized environment, to report different event data logs periodically with maximum accuracy in reference to the time stamp and corresponding data values with minimal overhead. Additionally, with the help of reported event data logs, we are able to capture the performance behavior of the distributed system accurately. In future we will show that this monitoring framework can be used with an intelligent predictive controller to maintain the system in closed boundary of QoS even in case of highly dynamic and unpredictable environment. Furthermore, the proposed technique is highly scalable in terms of accommodating the monitored parameters and event logs for system monitoring in online manner.

## 9 Acknowledgments

This work is supported in part by the NSF I/UCRC CGI Program grant number IIP-1034897 and and The Engineer Research and Development Center (ERDC) at Vicksburg, MS. Part of this work was conducted at Vanderbilt University in collaboration with Fermi National Accelerator Laboratory supported by DoE SciDAC-II program under the contract No. DOE DE-FC02-06 ER41442.

## References

[1] httperf documentation. Technical report, HP, 2007.

[2] M. Arlitt and T. Jin. Workload characterization of the 1998 world cup web site. Technical Report HPL-99-35R1, Hewlett-Packard Labs, September 1999.

[3] A. Dubey et al. Compensating for timing jitter in computing systems with general-purpose operating systems. In *ISROC*, Tokyo, Japan, 2009.

[4] R. C. L. Gupta, Diwaker; Gardner. Xenmon: Qos monitoring and performance profiling tool. Technical report, HP Labs, 2005.

[5] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME Journal of Basic Engineering*, (82 (Series D)):35–45, 1960.

[6] M. Mansouri-Samani and M. Sloman. Gem: a generalized event monitoring language for distributed systems. *Distributed Systems Engineering*, 4(2):96–108.

[7] R. Mehrotra, A. Dubey, S. Abdelwahed, and A. Tantawi. Model identification for performance management of distributed enterprise systems. (ISIS-10-104), 2010.

[8] P. Pan, A. Dubey, and L. Piccoli. Dynamic workflow management and monitoring using dds. In *Proceedings of the 2010 Seventh IEEE International Conference and Workshops on Engineering of Autonomic and Autonomous Systems*, EASE '10, pages 20–29, Washington, DC, USA, 2010. IEEE Computer Society.

[9] K. Ravindran and J. Wu. Event-based programming models for monitoring of distributed information systems. In *DS-RT '05: Proceedings of the 9th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pages 236–248, Washington, DC, USA, 2005. IEEE Computer Society.

[10] B. Tierney and D. Gunter. Netlogger: A toolkit for distributed system performance tuning and debugging. 1997.