Contents

Chapter 1

A Power-Aware Modeling and Autonomic Management Framework for Distributed Computing Systems

Rajat Mehrotra

Electrical and Computer Engineering, Mississippi State University, Miss. State, MS

Abhishek Dubey

Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN

Sherif Abdelwahed

Electrical and Computer Engineering, Mississippi State University, Miss. State, MS

Asser N. Tantawi

IBM Thomas J. Watson Research Center, Yorktown Heights, NY

1.1 Introduction

According to two related recent studies [1, 2], energy consumption cost contributes to more than 12% of overall cost of data center and is the fastest growing component of the operating cost. Same studies also point out that data centers consume only 15% energy in actual processing, while rest of the energy is used by the cooling equipment. Cooling infrastructure adds a significant overhead to the operating cost in terms of power consumption as well as system management. In addition, according to [2], the data center industry accounts for the 2% of global CO_2 emission which is at the same level as the emission introduced by the aviation industry. Therefore, it is clear that improving power consumption will have a significant influence on the cost effectiveness, reliability, and environmental impact of current and future distributed systems. Consequently, extensive research effort has been recently directed towards developing power efficient computing systems, also referred as "Power-Aware" systems.

Power awareness requires identifying the main factors contributing to power consumption as well as the mechanisms that can be used to control (reduce) this consumption, as well as the effect of using these control mechanisms on other quality of service (QoS) aspects of the system [10]. Adjustments in power consumption need to be made within the tolerance limit of system QoS. Typically, power management requires expert administrator knowledge to identify workload pattern, system behavior, capacity planning, and resource allocation. However, with increasing size and complexity of computing systems, effective administration is not only tedious but also error-prone and in many cases infeasible. Autonomic computing [32] is a new strategy aiming at replacing manual management with a more systematic approach based on well-founded approaches in AI and systems theory. Such approaches rely on a model that defines the relationship among the system performance, various measurements, and operating parameters of the system. An effective system model is therefore essential to achieve the power-awareness in computing systems infrastructure.

In this chapter, a model-based power management framework is presented for distributed multi-tier systems. This framework implements a predictive control approach to minimize power consumption in a representative system while keeping the QoS parameters at the desired level. The proposed approach starts by experimentally identifying the various system parameters that impact system performance. The dependency relationships among identified parameters are determined and then used to develop a mathematical model structure of the system. Offline regression techniques are used to estimate the parameters of the power consumption model while an online Bayesian method (exponential extended Kalman Filter) is used to estimate the state of the system modeled as an equivalent limited processor sharing queue system. Experiments show that the developed model captures the system behavior

accurately in varying environmental conditions with small error variance. Finally, we apply a predictive control approach to optimize power consumption while maintaining a desired level of response time at a negligible overhead for a representative multi-tier system. Some preliminary research and results have been previously published in [43] and [20].

This chapter is organized as follows. Preliminary concepts related to the proposed approach and related research work is presented in Section 1.2. System setup is discussed in 1.4 and identified system parameters are presented in section 1.5. The system modeling approach is outlined in Section 1.6. The predictive controller is presented in Section 1.7 and a case study related to managing power consumption and response time using the developed system model and controller is described in section 1.7.1.

1.2 Background and Related Research

1.2.1 Power consumption in computing systems

Most modern electronic components are built using the CMOS (Complementary Metal Oxide Semiconductor) technology. Advances made in the last decade have led to increased clock rates and narrower feature length of the CMOS transistor. This in turn has allowed chip developers to stack more transistors on the die, increasing the available computational power. However, these advancements have come at the cost of increased power consumption.

At the level of a transistor, the power consumption can be attributed to three factors. These factors are applicable to all electronic systems of the computer, including CPU, memory and even the hard drive. In the hard drive, there are some other mechanical factors that lead to increased power consumption. We will discuss them later in this section.

1. Switching (Dynamic) Power Consumption: Working principle of a CMOS FET (Field Effect Transistor) is based upon modulation of the electric charge stored by the capacitance between the gate and the body of the transistor. This capacitor actually charges and discharges during one cycle i.e. turning the switch on and then off. Effectively, this causes a drain on power, which goes towards charging the capacitor. This power loss is also called the switching or the dynamic loss.

2. Leakage (Static) Current Power Consumption : This is due to leakage current flowing through the transistor while being in OFF state. Previously static power consumption was negligible due to low number of transistors per inch and high resistance of wires used on chip. Currently, power loss due to leakage current is 40% of the total power budget. Lowering the voltage across the chip increases the leakage current by making

transistors too leaky, which in turn increases the power consumption of the microprocessor [48]. Additionally, high operating temperature of microprocessor increases the leakage current power consumption significantly.

3. Short Circuit Power Consumption: Small amount of power consumption is present in CMOS due to short circuit current on short circuit path between supply rails and ground.

Dynamic power loss has been the main component of the total power loss in past. However, lately percentage of static power loss is increasing as feature sizes have been decreasing.

1.2.2 Power consumption modeling

A non intrusive but accurate real time power consumption model is presented in [21], which generates power model with help of AC power measurements and user level utilization metrics. It provides power consumption model with high accuracy for both temporal and overall power consumption in the servers. A microprocessor level power consumption estimation technique is described in [26] that first examines the hardware performance counters and then uses relevant counters to estimate the power consumption through sampling based approaches. Another approach to model hard disk power consumption is shown in [54] that extracts performance information from the hard disk itself and predicts the power model. Additionally, it shows that the modeling of idle periods is an important step in predicting the power consumption model of a hard disk. Another approach for power consumption modeling in embedded multimedia application is presented in [24] that takes various image, speech, and video coding algorithms into account with supplied frequency and voltage to predict the power consumption behavior. A highly scalable power modeling approach is described in [29] for high performance computing systems by linearly extrapolating the power consumed by a single node to complete large scale system using various electrical equipment. A micro architecture level temperature and voltage aware performance and leakage power modeling is introduced in [36] that shows variation of leakage current and energy consumption with varying temperature. An approach for accurately estimating power consumption in embedded systems is presented in [45] while running a software application and considering pipeline stall, inter-instructions effect, and cache misses. A power modeling approach for smart phones is described in [34] that models the power consumption with help of in-built voltage sensors inside the battery and its discharge behavior. Another approach of CMOS power short circuit dissipation is presented in [11] that helps to model the short circuit power dissipation for the configurations when it represents the significant amount of power consumption.

1.2.3 Power management techniques

CPU power management

Main focus of the academia and industry has been target the power consumption of microprocessors. Various methods have been proposed to control the power consumption in microprocessors through logical and system level techniques.

1. Dynamic voltage and frequency scaling (DVFS): In this method, the voltage across the chip and clock frequency of the transistor is varied (increased or decreased) to lower the power consumption and maintain the processing speed at same time. This method is helpful in preventing the computer systems from an overheating that can result into system crash. However, the applied voltage should be kept at the level suggested from the manufacturer to keep the system stable for safe operation. DVFS reduces processor idle time by lowering the voltage or frequency, while continue to process assigned task in permissible amount of time with minimum possible power consumption. This reduces the dynamic power loss.

2. Dynamic power switching (DPS): In contrast to DVFS, DPS tries to maximize the system idle time that in turn forces processor to make transition to idle or low power mode for reducing power consumption. The only concern is to keep track of the wakeup latency for the processor. It tries to finish the assigned tasks as quickly as possible so that rest of the time can be considered as idle time of the processor. It only reduces leakage current power consumption while increases the dynamic power consumption due to excessive mode switching of the processor.

3. Standby leakage management (SLM): This technique is close to the strategy used in DPS by keeping the system in low power mode. However, this strategy comes in to the effect when there is no application running in the system and the system just needs to take care of its responsiveness towards user related wake up events (e.g. GUI interaction, key press, or mouse click).

RAM power management

In general, CPU is considered as the dominant component for power consumption in a computing system. However, recent research [18, 42] shows that random access memory can also be a significant contributor to system power consumption. Therefore, it should be a target for managing the power consumption specially in case of small size computers. Currently, memory chips with multiple power modes (e.g. active, standby, nap, and power down) are available in the market and can be used for designing an efficient memory power management technique. The primary idea behind multiple modes of memory operation is that different amount of power is consumed inside a memory in different states. Memory can execute a transaction only in active

state but can store the data in all of the states. The only concern while utilizing multiple modes is to consider the latency in terms of time and power consumption while switching the modes. Similar to CPU power management, there are primarily two approaches for memory power management through utilizing mode: static and dynamic. In case of static method, memory is kept at a low power mode for all the time duration of system operation, while for dynamic method memory is placed in a low power mode when its idle time is more than the threshold time. Another approach of memory power management is described in [60] that combines multiple hardware components on a single chip to create smaller and power efficient components.

A memory power consumption management technique is described in [16] that concentrates on DRAM modules for energy optimizations by putting them in low power operating modes during idle periods without paying very high penalties. Another approach for power consumption memory management is shown in [18] which changes the power state of memory with the change in load on the memory subsystem modules. A feedback control theory based approach for managing memory power consumption is presented in [42] that puts memory chips in low power mode while maintaining the desired latency and improves memory power efficiency by 19.3%. An Operating system scheduler based approach is presented in [31] where OS directs the memory module to be in low power mode by keeping track of all the processes running on the system. A comprehensive approach of DRAM power management is shown in [25] that combines the benefit of low power mode of modern DRAMs, history based adaptive memory schedulers, and proposes an approach of delaying the issuance of memory commands. A faster and accurate framework for analyzing and optimizing microprocessor power dissipation at the architectural level is described in [12] that validates the power requirement in design stage of the circuit.

Another application level power management technique is described in [51] that solves the cost-aware application placement problem and to design an algorithm that minimizes the cost and migration costs while meeting the performance requirements. Another two step approach of power aware processor scheduling is presented in [55] that first performs load balancing among the multiple processors and then applies DVFS to control the speed of the processors to minimize the power consumption. A proactive thermal management approach in data centers is described in [35] which optimizes the air compressor duty cycle and fan speed to prevent heat imbalance in cooling the data center while minimizing the cooling cost. Additionally, it reduces the risk of damage to the data center due to excessive heating in the data center. An approach of saving power consumption in servers is introduced in [49] in which NAND flash based disk caches is extended to replace PCRAM.

Miscellaneous power consumption

According to [21], miscellaneous components are responsible for large fraction (30% - 40%) of power consumption inside a computing system that consists of disk, I/O peripherals, network, and power supplies. The primary contribu-

tors in power consumption are disk and power supplies. Device vendors have started implementing their power management protocols to ensure that these devices can run in a low performance mode. For example, hard disks typically have a timer that measures the time of inactivity and spins the drive down, if possible, to save power.

1.2.4 Control based management of computing systems

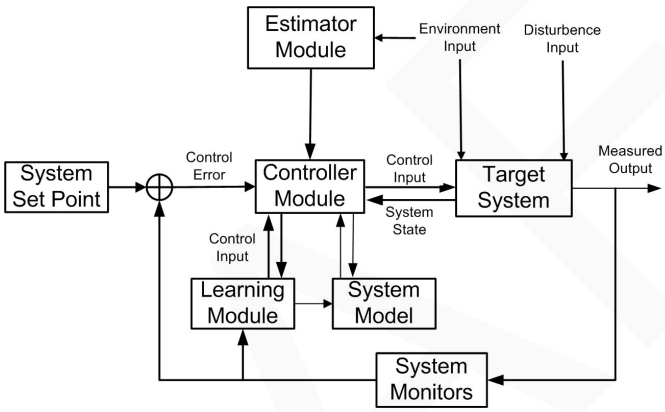


FIGURE 1.1: Elements of a general control system

Control theory offers a promising methodology to automate key system management tasks in distributed computing systems (DCS) with some important advantages over heuristic or rule-based approaches. It allows us to systematically solve a general class of dynamic resource provisioning problems using the same basic control concepts, and to verify the feasibility of a control scheme before deployment on the actual system. Control theory provides a systematic and well-founded way to design an automated and efficient management technique through continuous observation of system state as well as variation in the environment input, and apply control inputs so as to maintain the system within a desired stable state region corresponding to the desired QoS requirements.

Research in real-time computing has focused on using classical feedback control as the theoretical basis for performance management of single-CPU applications [23]. These reactive techniques observe the current application state and take corrective action to achieve a specified performance metric, and have been successfully applied to problems such as task scheduling [14], CPU provisioning [39, 13, 37], bandwidth allocation and QoS adaptation in web servers [8], multi-tier websites [38], load balancing in e-mail and file servers [47, 38], and CPU power management [40, 50]. In more complex control problems, a pre-specified plan, the feedback map, is inflexible and does not adapt well to constantly changing operating conditions. Moreover, classical control

is not a suitable option for managing computing systems exhibiting hybrid and non-linear behavior, and where control options must be chosen from a finite set. Therefore, researchers (including the authors) have studied the use of more advanced state-space methods adapted from model predictive control [41] and limited lookahead supervisory control [15] to manage such applications [7, 6, 30]. These methods offers a natural framework to accommodate the above-described system characteristics, and take into account multi-objective non-linear cost functions, finite control input sets and dynamic operating constraints while optimizing application performance. The autonomic approach proposed in [30] describes a hierarchical control based framework to manage the high level goals for a distributed computing system by continuous observation of the underlying system performance. Additionally, the proposed approach is scalable and highly adaptive even in case of time varying dynamic workload patterns.

A typical control system consists of the components shown in Fig. 1.1. *System Set Point* is the desired state of the system in consideration that a system tries to achieve during its operation. *Control Error* is the difference between the desired system set point and the measured output during system operation. *Control Inputs* are the set of system parameters, which are applied to the system dynamically for changing the performance level of the system. The *Controller Module* takes observation of the measured output and provides the optimal combination of different control inputs to achieve the desired set point. *Estimator Module* estimates the unknown parameters for the system based upon the previous history using statistical methods. *Disturbance input* can be considered as the environment input that affects the system performance. *Target system* is the system in consideration, while *System Model* is the mathematical model of the system, which defines the relation between its input and output variables. A *Learning Module* takes measured output through the monitor and extracts information based on the statistical methods. *System State* typically define the relationship between control or input variables and performance parameters of the system.

1.2.5 Queuing models for multi-tier systems

Multi-tier enterprise systems are composed of various components that typically include web (http) servers, application servers, and database servers, subjected to a stream of web requests. Each component performs its function by servicing requests from upper tiers and may submit sub-requests to lower tiers. In order to avoid overloading, system administrators usually limit the number of concurrent requests served at each tier that is called "concurrency limit" for that particular tier [17].

In general, a web request has to wait in a queue for computational resources before it can enter a tier. For example, if the number of maximum threads allowed in the application (app) server is capped to limit concurrency, a new request will wait until an existing running request releases a thread. Clearly,

the total service time of the enterprise system is directly affected by the queuing policy at each tier. Therefore, an approximate queuing model can be used to capture the behavior of such systems. Thereafter, it can be used to measure the average number of web requests in the queue and the average time spent there. During the progression of this work, we have considered four different queuing models. For a detailed description, readers are referred to [33]. These are:

M/M/1: This is the most basic queuing model where both service time and inter-arrival time are exponentially distributed.

M/G/1 FCFS: The M/G/1 queue assumes that the inter-arrival time is exponentially distributed, but the service time has a general distribution. This is a more realistic model of web service behavior since web requests exhibit a wide variance in their service requirements. The scheduling discipline is *first come first served (FCFS)*. Thus, this model assumes that only one request is serviced at a time, hence restricting the concurrency level to one.

M/G/1 PS: This is an M/G/1 queue with *processor sharing (PS)* scheduling. In such a model, as a web request arrives, it is able to share the server concurrently with all preceding, existing requests. The computational resource is shared equally by all the requests in a round-robin manner, though with an extremely short slice. When not using the server, a request waits in the queue for its turn to get a slice of the server. In other words, the concurrency level is basically unlimited. The mean analysis of this system is rather simple. In fact, the closed form expression for the mean response time is similar to that of the M/M/1 queue. The only catch is that this model can only be used to study a web server realistically if we can ensure that the total number of requests in the system do not increase above the maximum concurrency limit. *Thus, if the bottleneck resource utilization is light to moderate, we can use this queue to model web servers.*

M/G/1 LPS(k): This is an M/G/1 queue with *Limited Processor Sharing(LPS)* scheduling discipline. The parameter k models the concurrency level. In such a model, the first k requests in the queue share the server and the rest of the requests wait in the queue. As requests complete their service and depart from the system, awaiting requests are admitted to be among the k concurrently sharing the server in a processor sharing manner. When $k = 1$ the queue becomes a FCFS queue, and when $k = \infty$ it becomes a PS queue. The LPS(k) is a realistic queuing model for systems that have a limit on the maximum number of concurrently executing jobs. This limit is typically enforced on all web servers and databases. Even in an operating system, the maximum number of processes that can execute concurrently is capped. (e.g. 32000 for Linux kernel 2.6). The analysis of the LPS queue is rather difficult [57, 56] and online prediction for response time and other variables can become intractable.

1.2.6 Kalman filters

Kalman filter [27] is an *optimal recursive data processing algorithm*, which estimates the future states of linear stochastic process in presence of measurement noises. This filter is optimal in the sense that it minimizes the mean of squared error between the predicted and actual value of the state. It is typically used in a predict-and-update loop where knowledge of the system, measurement device dynamics, statistical description of system noise, and the current state of the system is used to predict the next state estimate. Then the available measurements and statistical description of measurement noise is used to update the state estimate.

Two assumption are made before applying Kalman filter for state estimation; first, the system in consideration is described by linear model, or if the system is non-linear, the system model is linearized at the current state (extended Kalman Filter), and second, the measurement and system noise are Gaussian and white respectively. *Whiteness* indicates that noise is not correlated with time and it has equal impact on all operating modes. Due to simple approach with optimal results, Kalman filter has been applied in wide areas of engineering application including motion tracking, radar vision, and navigation systems.

Woodside et al. [52] applied extended Kalman (EK) filtering techniques to the estimation of parameters of a simple closed queueing network model. The population size was approximated by a continuous variable in order to fit the EK mathematical framework. A demonstration of the use of the EK filter to track parameters such as think time and processing time parameters of a time-varying layered queueing system is provided in [59]. The EK filter uses observations of average response time as well as utilization of three resources: a web processor, a database processor, and a disk. In [53], the technique of using an EK filter in conjunction with layered queueing models is employed to control the number of allocated servers so as to maintain the average response time within a given range.

The work on applying EK filtering to estimating performance model parameters is extended in [58] where an estimation methodology is sought. Models such as separable closed queueing network models, open queueing network models, and layered models are considered. Parameters may have lower and upper bounds. Tracking periodic deterministic as well as random perturbation of one or more parameters is demonstrated experimentally.

Recently, Kalman filter has been applied to provision CPU resources in case of virtual machines hosting server applications [28]. In [28], the feedback controllers based on Kalman filter continuously detect CPU utilization and update the allocation correspondingly for estimated future workloads. Overall, an average of 3% performance improvement in highly dynamic workload conditions over a three-tier Rubis benchmark web site deployed on a virtual Xen cluster was observed .

In this chapter, we describe our implementation of an *exponential Kalman*

filter that we used 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 $M/G/1$ 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 ≥ 0 constraint on the state variables. Such a constraint are not possible in typical Kalman filter implementations. Further details are provided in Section 1.6.3.

1.3 Our Approach

A great amount of work has been done in the past for enabling power awareness within computing systems as described earlier at both hardware and software level. However, it has been difficult to implement those in real systems either due to their implementation complexity or inefficiency to capture the system dynamics completely with multi-dimensional QoS objectives in fluctuating environment. Our current work addresses both the modeling and management problems. First we identify the system dynamics accurately with respect to controllable parameters in off-line as well as on-line manner. Second we utilize the system model within a *model predictive controller* to achieve the desired objectives. The developed predictive controller can accommodate multi-dimensional QoS objectives easily just by learning the variation of that objective with respect to system state, control inputs to the system, and environmental changes. The novelty of our approach lies in the collection of state-of-the-art techniques to monitor and model the system performance with scalable nature of the predictive controller with respect to new QoS objectives for performance optimization.

1.4 Case Study: A Multi-Tier Enterprise System

Multi-tier enterprise systems are composed of various components that typically include web (http) servers, application servers, and database servers. Each component performs its function with respect to web requests and forwards the result to the next component (tier). Generally, system administrators limit the number of concurrent requests served at each tier that is called

TABLE 1.1: Physical machine configuration

Name	Cores	Description	RAM	DVFS	VMs
Nop01	8	2 Quad core 1.9GHz AMD Opteron 2347 HE	8GB	No	Nop04,Nop07 (Monitoring Server)
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 2350z	8GB	Yes	Nop11,Nop12 (Database Server)

"concurrency limit" for that particular tier [17]. In order to experiment and validate our work we used the following system setup.

System Setup: Our system consists of four physical nodes: *Nop01*, *Nop02*, *Nop03* and *Nop10*. Names *Nop04* to *Nop09* are reserved for the virtual machines. Table 1.1 summarizes configuration of physical machines. It also shows the virtual machines (VM) running on all physical machines and the roles played by those VMs. All VM run same version of Linux ($2.6.18 - 92.el5xen$). 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). Database machines run MySQL. *Nop03* and *Nop10* both have Dynamic Voltage and Frequency Scaling (DVFS) capability that allows administrator to tune the complete physical node or its individual cores for desired performance level. Xen Hypervisor (`http://www.xen.org/`) was used to create and manage physical resources (CPU and RAM) for cluster of Virtual Machines (VMs) on these physical servers.

We used *Daytrader* [3], as our representative application. Daytrader comes with a client that can drive a trade scenario 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, this application puts most of the load on the database server. To emulate business enterprise loads in highly dynamic environment, we modified the main trade scenario servelet to allow us to shift the processing load of a request from the database node to the computing node. In the modified daytrader application, the web server generates a random symbol for each request from the symbol set of available stock names. Web server performs database query based on that symbol and returns the result of the query to the client. Also, it computes the integral sum of first N integers, where N is supplied through the client workload.

Due to limitation of workload pattern allowed (only uniform) in *Daytrader* client, we used the Httperf [5] benchmarking application client tool in all of our experiments. It provides flexibility to generate various workload patterns (Poisson, deterministic, and uniform) with numerous command line options for benchmarking. We modified Httperf to print the performance measurements of our interest periodically while running the experiment. At the end of

each sample period, modified version of *Httperf* prints out the detailed performance statistics of the experiment in terms of *total numbers of requests sent, minimum response time, maximum response time, average response time, total number of errors with types, and response time* for each request.

Monitoring: Specially developed python scripts and Xenmon [22] were used as monitoring sensors on all virtual and physical machines. These sensors monitor CPU, disk and RAM utilization of the nodes (physical and virtual) throughout the system execution and report data after each sampling interval as well as at the end of the experiment. System time was synchronized using NTP. The jitter in monitoring sensors across all servers was controlled using a PID controller as described in [19]. Modifications to the web server code allowed us to monitor web server performance in terms of *max threads active* in web server, *response time measured* at first tier and at the database tier for each incident request, and *average queue size* in the web server after each sampling interval. The client returns the measured maximum, minimum, and average response time during the sampling period. We specified 100 *seconds as the timeout value* for a request response for all of the experiments as it provides enough time to web server for serving a most of the web requests even at the maximum utilization of bottleneck resource. Any outstanding request after the timeout was logged as an error. It also returns the number of errors with their type (client timeout, connection reset, etc.) in each sample. Measurement of power consumption of a physical node was done with help of a real time watt meter.

1.5 Model Identification

As a result of different experiments described in Section 1.6, an extensive list of system parameters have been identified. This list is shown in Table 1.2. It contains three different types of parameters: *Control Variables, State Variables,* and *Performance Variables*. Control variables are those which can be controlled at runtime to tune the system towards the desired performance objectives. State variables describe the current state of system under observation. Performance variables are used to quantify QoS level of the system. Additionally, state variables are divided into two different categories: *Observable* and *Unobservable*. Observable variables can be measured directly through sensors, system calls or application related API, while unobservable variables cannot be measured directly, instead they are estimated within certain accuracy using existing measurements through various techniques at runtime. During our experiments, we used specially written sensors or different tools to measure observable variables, while unobservable variables (e.g. service time and delay) are estimated through the exponential Kalman filter described in Section 1.6.3.

TABLE 1.2: System parameters

Control Variables	**State Variables**	**Performance Variables**
CPU Frequency	CPU Utilization (Observable)	Average Response Time
Cap on Virtual machine resources.	Memory Utilization (Observable)	Power consumption in Watts
Load distribution percentage (in a cluster)	Service Time (Unobservable)	percentage of Errors
Number of Service Threads	Queue waiting Time (Unobservable)	
Number of Virtual Machines in Cluster	Queue Size on each server (Unobservable)	
	Number of Live Threads (Observable)	
	Peak Threads available in a JAVA VM (Observable)	

1.6 System Modeling

An accurate system model derivation is necessary to run a computing system efficiently in power saving environment. The derived model will depict the exact system behavior in terms of various performance objectives with changes in operating environment and controllable parameters. For identifying an accurate system model of our representative distributed multi-tier system (see section 1.4), extensive experiments have been performed and results have been analyzed. During these experiments, we analyzed the multi-tier system performance with respect to system utilization, various work load profiles, bottleneck resource utilization, and its impact on system performance. Additionally, we calculated work factor of our client requests with help of linear regression techniques described in [46]. Details of the modeling efforts are described below.

1.6.1 Power consumption

As a first step towards system model identification, the mutual relationship among physical CPU core utilization, CPU frequency, and power consumption of the physical server was identified. This work is an extension of the power modeling effort described in [20] to model the system power consumption with greater accuracy that can be utilized effectively in real time physical server deployment. Details of the experiment can be found in our technical report [44]. Fig. 1.2 shows the power consumed on one of the physical server *Nop03* with respect to the aggregate CPU core usage and CPU frequency. An

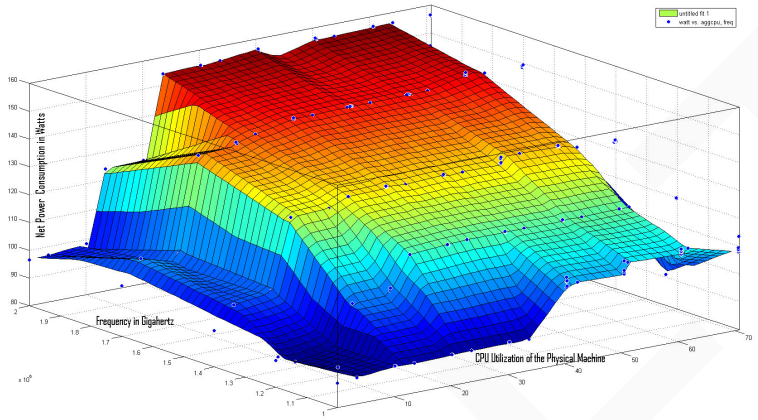


FIGURE 1.2: Power consumption on Nop03 vs CPU frequency and aggregate CPU core utilization. This plot uses linear interpolation to create the surface. Note, power consumption decreases as the frequency decreases.

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 across all of the physical cores (1 to 8) was kept same during each step. The consumed power was measured with help of a real time watt meter. 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 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. memory, 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. Combination of CPU frequency, and aggregate CPU core usage of the physical machine is used as a key of the lookup table to access the corresponding power consumption value. This aggregate power model was utilized further for the controlled experiments described in section 1.7 for predicting the estimated power consumption by the physical server at a particular setting of CPU core frequency and aggregate physical CPU utilization.

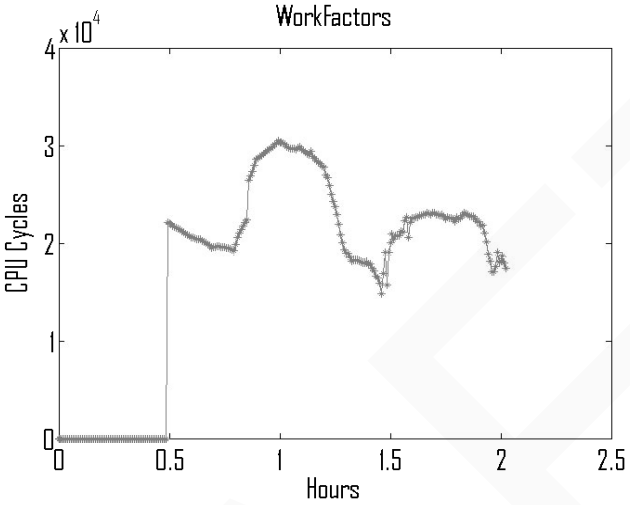


FIGURE 1.3: Work factor plot for request characteristic

1.6.2 Request characteristics

Httperf benchmark application code was modified to allow the generation of client requests to the web server, $Nop06$, at a pre-specified rate provided from a trace file. At Nop06, each request performed certain fixed floating point computation on the web server and then performed some random select query on the database machine. To better evaluate the nature of requests, we identified the number of CPU cycles needed to process the request using linear regression [46].

During any sampling interval T, if ρ is virtual CPU utilization, f is CPU frequency, W_{fc} is the work factor of the request (defined in terms of CPU clock cycles), λ is request rate, and ψ is system noise. Then, $\rho * f = \lambda * W_{fc} + \psi$. The average work factor was computed to be $2.5X10^4$ CPU cycles with coefficient of variation = 0.5. The variation in W_{fc} shows the variation in the nature of final request based upon the chosen symbol for query. Result of the experiment is shown in Fig. 1.3. Due to similar computational nature of all the requests incident on the web server in a given sample time, we can approximate the total computation time of all the requests, which in turn gives the average response time of the requests in a given sample time. We can use this average response time information to check the status of QoS objective (response time) in the web server.

1.6.3 Webserver characteristics

We aimed to identify the web server bottle necks and estimate the uncontrolled performance to compare later with the performance under the predictive controller. In this experiment, *Nop09* was the virtual machine (physical

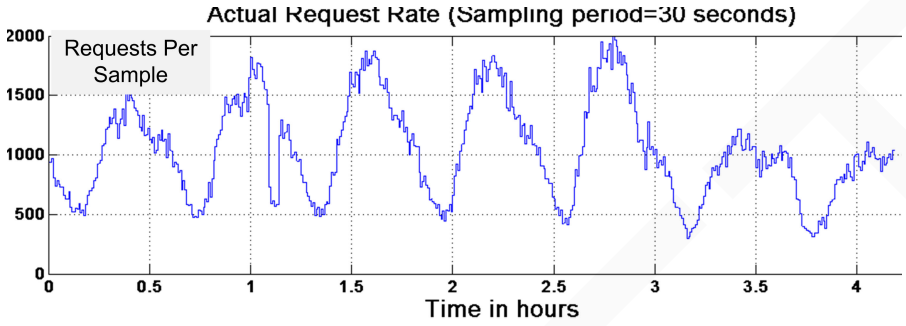


FIGURE 1.4: Http Workload based upon World Cup Soccer(WCS-98) applied to the web server

machine *Nop03*) running the first tier of *Daytrader* application. 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 physical 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 1.6.1. `MAX_JAVA_threads`, a parameter that sets the maximum concurrency limit was configured as 600 in the web server application. All CPU cores in physical server *Nop03* were operating at their maximum frequency 2.0 Ghz. The request trace (see Fig 1.4) in this experiment was based on the user request traces from the 1998 World Cup Soccer(WCS-98) web site [9]. Figure 1.14 shows the response time and power consumption as measured from this experiment. It also shows the CPU utilization at web server and aggregate CPU utilization of the physical machine. Notably, we saw that the CPU utilization at web server (Nop09CPU without controller) and aggregate CPU utilization of the physical machine (Nop03CPU without controller if we zoom in the line) follows a trend which is similar as the rate of requests made to the first tier. The power consumption curve was almost flat. We also noticed that the web server response time is correlated to the resident requests (system queue size) in the web server system (not shown here, but described in [44]).

Estimating Bottleneck Resource

To determine the bottleneck resource, we used a queuing approximation for a two tier system as shown in Fig. 1.5. λ is the incoming throughput of requests to an application. ρ is the utilization of the bottleneck resource. S is the average service time on the bottleneck resource. D is the average delay. T is the average response time of a request. The average waiting time for a request is $W = T - S - D$. We define a queue model with state vector $[S; D]$ and observation vector as $[T]$.

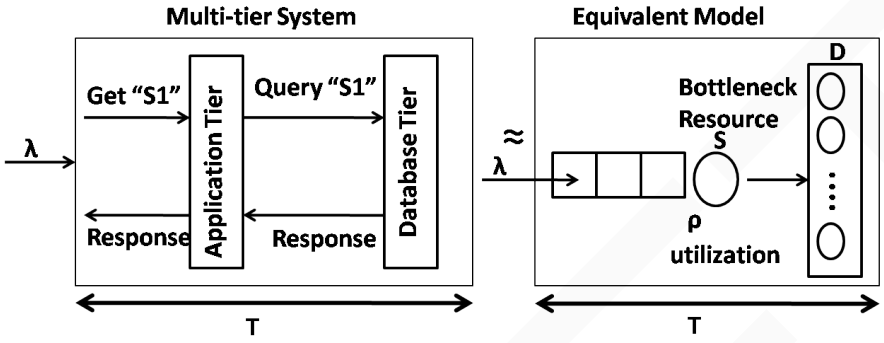


FIGURE 1.5: A queuing model for the two-tier system.

An exponential Kalman filter (KF) was 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/∞ 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 LPS queue model. However, as mentioned earlier, 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.

The KF equations, written in the term of exponentially transformed variables, $[x1 \in \Re; x2 \in \Re]$ s.t. $S = exp(x1)$ and $D = exp(x2)$ are as follows. Note that this transformation ensures $S, D \in \Re^{+}$: For a given timed index of observation, k, the equations $\begin{pmatrix} exp(\hat{x1}_k) \\ exp(\hat{x2}_k) \end{pmatrix} = \begin{pmatrix} exp(x1_{k-1}) \\ exp(x2_{k-1}) \end{pmatrix} + N(0, Q)$ and $T = exp(x1_k) * (1/(1 - \lambda_k * exp(x1_k))) + exp(x2_k) + V(0, R)$ define the state update dynamics and observation. N and V are Gaussian process and measurement noises with mean zero and covariances Q and R respectively. 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.

Figure 1.6 shows results of the off-line analysis of data generated through experiment 1.6.3 with help of Kalman filter described in current section. According to Fig. 1.6(sub-figure 1), the developed exponential Kalman filter tracks service time S and delay D at web server perfectly with low error variance as the experiment (section 1.6.3) progresses. Additionally, as per sub-figure 2, Kalman filter tracks bottleneck utilization as similar to CPU utilization of the system. However, we noticed that sometime, the bottleneck utilization might saturate at 1 without the CPU utilization reaching that value. In those cases, we discovered that the number of available system

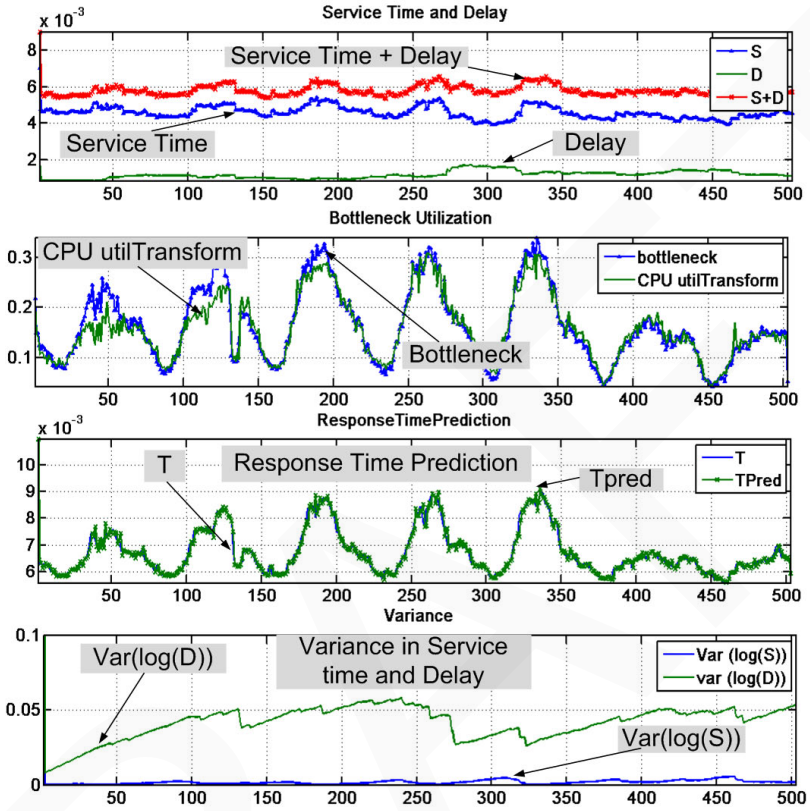


FIGURE 1.6: Offline Exponential Kalman filter output corresponding to the results from section 1.6.3. Service time and delay are in millisecond range. Response time is specified in seconds.

threads acted as the bottleneck. According to sub-figure 3, predicted response time from the Kalman filter T_{pred} and actual response time T also very close to each other, which indicates efficiency of the Kalman filter. Efficiency of the Kalman filter indicate that the developed filter is able to capture the response time dynamics of the web server system perfectly that will be used in on-line manner with the predictive control framework in next sub-section.

1.6.4 Impact of maximum usage of bottleneck resource

We aimed then to observe the affect of high bottleneck resource usage on system performance. Our test setup contains daytrader application which is a multi-threaded java based enterprise application hosted on *Web Sphere Application Server Community Edition (WASCE)* that listen on port number 8080 for incoming http requests. This daytrader application serves the incoming

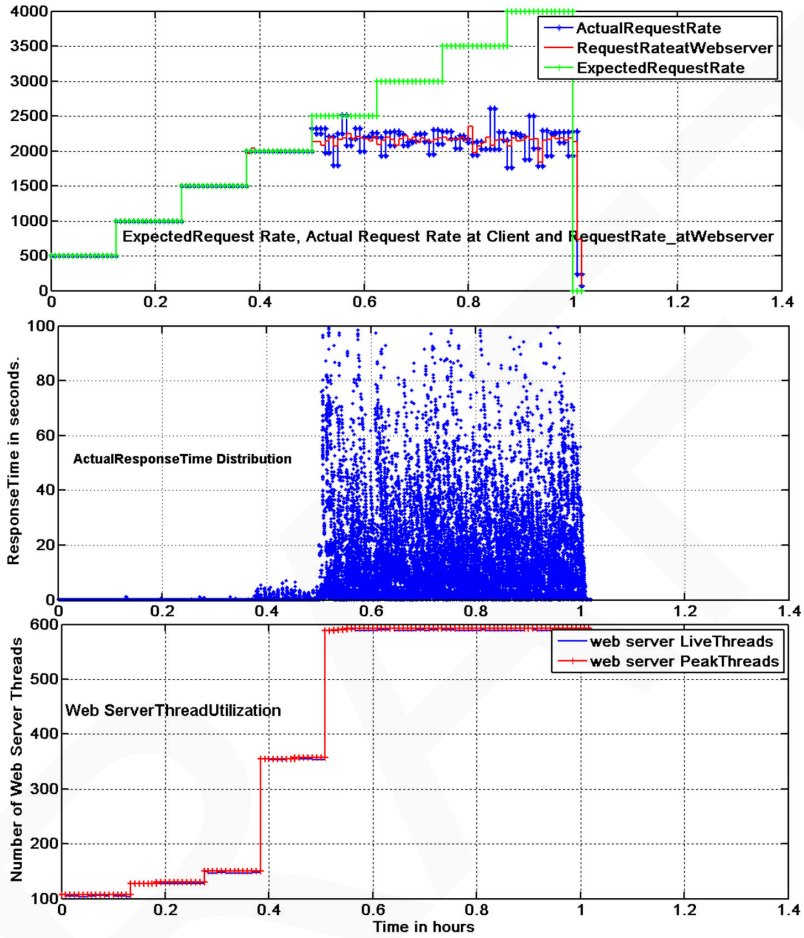


FIGURE 1.7: Impact of max utilization of bottleneck Resource on performance from section 1.6.4. MAX Thread =500.

requests through creating a new child JAVA thread or through existing JAVA thread if available in pool of free threads. A newly arrived http request for daytrader application is handed over to the newly created or free child thread for further processing. This child thread rejoins the pool of free threads once it finishes the processing. A limit on maximum number of created JAVA threads can be imposed in the web server that can also be considered as the maximum concurrency limit of the system. In case of unavailability of a free thread due to already achieved maximum thread limit and empty pool of free threads, newly arrived http request has to wait for thread availability that impacts the application performance severely in terms of response time. Therefore, we chose the settings that made the number of available threads as the system bottleneck.

We performed various experiments with different setting for Max JAVA Threads. This parameter sets the maximum number of threads that can be used for request processing. Based on our observation, there are typically 90 more system threads which are not accounted under this cap. Fig. 1.7 shows the results for the one experiment with max threads set to 500. This figure shows that at maximum utilization of the bottleneck resource, system performances decrease significantly and response time from the web server becomes unpredictable. Furthermore, this is the region, where the system transitions from a PS queue to a LPS queue system.

Once the system reaches the max utilization of the bottleneck resources, it restricts entry for more requests into the system resulting into max utilization of the incoming system queue which in turn results in rejection of the incoming client requests from the server. Therefore, to achieve predefined QoS specifications, system should never be allowed to reach the maximum utilization of bottleneck resource. Additionally, this boundary related to max usage of bottleneck resource can also be considered as "Safe Limit" of system operation.

1.6.5 Impact of limited usage of bottleneck resource

Next we observed the web server performance when the bottleneck resource utilization varies from minimum to maximum and back to minimum. This type of study provides knowledge regarding web server performance if bottleneck resource utilization is lowered from maximum limit through a controller that maintains the QoS objective of the multi-tier system.

The configuration settings for this experiment was same as experiment 1. MAX number of JAVA threads for experiment is 600. The client request-trace profile used for this experiment is shown in Fig. 1.8 subfigure 5. According to the results shown in the same figure (Fig. 1.8), system utilization (sub-figure 1) and performance in terms of response time (sub-figure 4) follows the trend of applied client request profile (sub-figure 6). We can also see the sudden jump in size of server queue (sub-figure 3), which indicates contention of computational resource among all of the pending requests inside the system. Sudden increase in RAM utilization is due to the increase in thread utilization of the system. Additionally, from the comparison of request rate and response time plot in Fig. 1.8, it is apparent that by lowering the system utilization and client load on the web server, web server can be brought back to state, where it can restore QoS objective of the system that can be consist of minimizing the system queue size and server response time.

Kalman Filter Analysis. Results of the experiment were analyzed with the help of Kalman filter described in section 1.6.3 and results of the analysis are shown in Fig 1.9. According to Fig. 1.9, the defined exponential Kalman filter tracks service time and delay at web server quite well with low variance as the experiment progresses. One can notice the regions where the bottleneck resource utilization approaches unity but the CPU utilization is less than one.

Upon further investigation of those time samples, the bottleneck resource was found to be the maximum number of Java threads available in web server. This limit can be changed by the 'MAX JAVA Threads' configuration setting. The goal of any successful controller design for performance optimization of the system will be to drive the system to work in the stable region (where the bottleneck resource utilization is less than unity). During the experiment, when predicted utilization of the bottleneck resource is more than one, Kalman filter does not update its states.

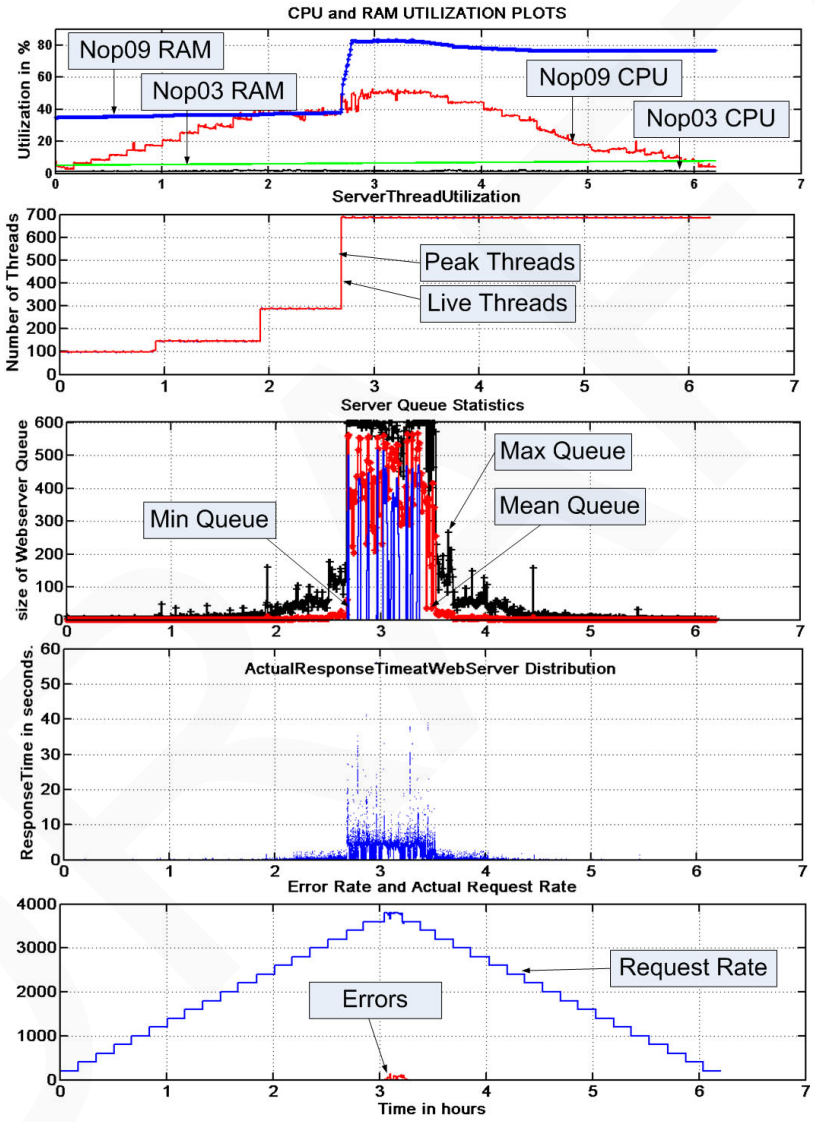


FIGURE 1.8: Web server behavior while limiting the use of bottleneck resource from section 1.6.5.

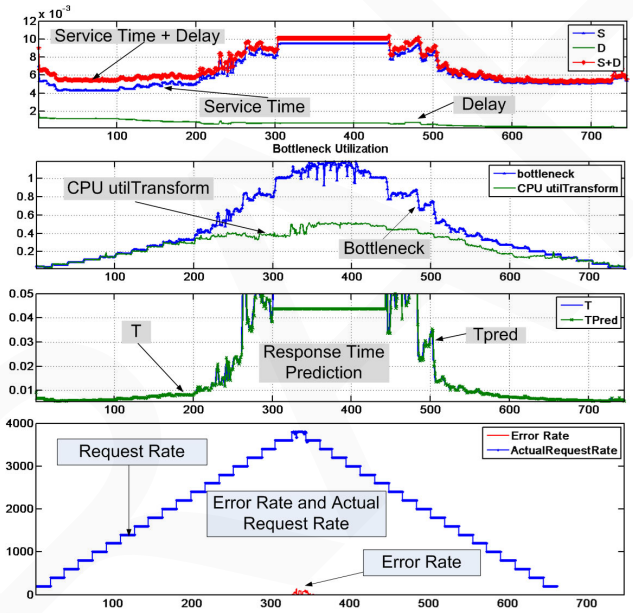


FIGURE 1.9: Offline KF Analysis of the results from Fig. 1.8 of section 1.6.5. Service time and delay are in milliseconds. Response time is specified in seconds.

1.7 Power Management using Predictive Control

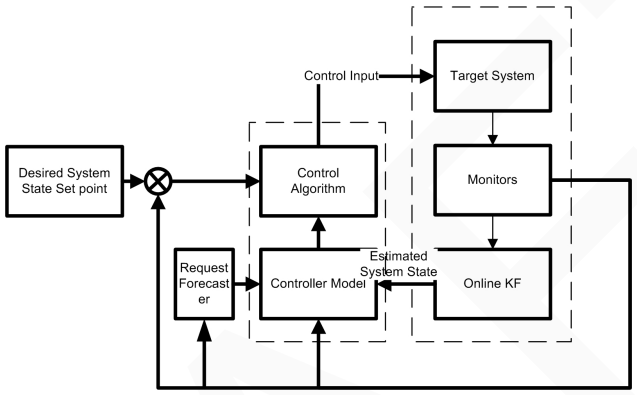


FIGURE 1.10: Elements of the applied predictive control framework

This section describes the implementation of an online predictive controller that uses the kalman filter and the queuing model identified in previous section (see Figure1.10). This controller is similar to the *L0 Controller* described in [30] and predicts the aggregate response time of the incident requests and the estimated power consumption during the next sample time (look-ahead horizon N) of the system based on different possible combinations of control inputs (CPU core frequency). It optimizes the system behavior in terms of QoS objectives by continuous observation of the underlying system and choosing the best control input for the system in next sample interval.

System Variables. Although there are a large number of system parameters listed in section 1.6, we have chosen a small set of most important parameters for our predictive controller to show the performance of our modeling approach. The chosen control input is the *CPU core frequency* due to its impact on the system performance in multiple dimensions for response time of the system and power consumption. *System queue size* and *response time* and *power consumption* were the chosen state variables as they are the typical performance variables in web service industry used to define a typical multi-dimensional service level agreement (SLA). Other experiments (reported in detailed report [44]) indicate that the higher value of application queue represents contention in computational resources of the application and total response time value indicates system's capability process the requests lying in system queue in a timely manner. Therefore, we try to minimize the application queue size and total response time as one of the component in cost function J (described later in this section).

Plant Model. The queuing model identified in the previous section was used to estimate the state of the managed system.

Controller Model. In order to combine the power consumption quality of service and the predicted response time, the controller uses a different internal system model. The controller model uses the estimated system state, predicted response time and predicted power consumption to make the system decisions. The system state for this experiment, $x(t)$, at time t can be defined as set of system queue $q(t)$ and response time $r(t)$, that is, $x(t) = [q(t); r(t)]$. The queuing system model is given by the equation: $\hat{q}(t+1) = max\{q(t) + (\hat{\lambda}(t+1) - \frac{\alpha(t+1)}{\hat{W}_f}) * T, 0\}$ and $\hat{r}(t+1) = (1 + \hat{q}(t+1)) * \frac{\hat{W}_f}{\alpha(t+1)}$, where at time t, $q(t)$ is the queue level of the system, $\lambda(t)$ is the arrival rate of requests, $r(t)$ is the response time of the system, $\alpha(t)$ is a scaling factor defined as $u(t)/u_{max}$ where $u(t) \in U$ is the frequency at time t (U is the finite set of all possible frequencies that the system can take), u_{max} is the maximum supported frequency in the system, $\hat{W}_f$ is the predicted average service time (work factor in units of time) required per request at the maximum frequency. Online Kalman filter estimates the service time $\hat{S}_t$ of the incident request at current frequency $u(t)$, which is scaled against the maximum supported frequency of the system to calculate the work factor ($\hat{W}_f = \hat{S}_t * \frac{u(t)}{u_{max}}$). $E(t)$ is the system power consumption measured in watts at time t.

Estimating Environment Inputs. The estimation of future environmental input and corresponding output of the system is crucial. In this experiment, an *autoregressive moving average* model was used as estimator of the environmental input as per following equation. $\lambda(t+1) = \beta * \lambda(t) + \gamma * \lambda(t-1) + (1 - (\beta + \gamma)) * \lambda(t-2)$, where β and γ determines the weight on the current and previous arrival rates for prediction.

Control Algorithm and Performance Specification: In this work we use a limited look ahead controller algorithm, which is a type of model predictive control. Starting from a time t_0, the controller solves an optimization problem defined over a predefined horizon ($t = 1...N$) and chooses the first input $u(t_0)$ that minimizes the total cost of operating the system J in future prediction horizon. Fomally, the chosen control input $u(t_0) = \arg\min_{u(t) \in U}(\sum_{t=t_0+1}^{t=t_0+N} J(x(t), u(t)))$. During this work, we limited the horizon to $N = 2$ as there is a significant computation cost associated with a longer horizon.

The cost function, J, at time t, is the weighted conjunction of drift of system state $x(t)$, ($x(t) = [q(t); r(t)]$) from the desired set point x_s, of the system state ($x_s = [q*, r*]$ where $q*$ = desired maximum queue size, $r*$ = desired maximum response time) and power consumption $E(t)$ (desired power consumption is 0). Formally, $J(t) = Q * \|x(t) - x_s)\| + R * \|E(t)\|$, where Q and R are user specified relative weights for the drift from the optimal system state x_s and power consumption $E(t)$, respectively. The power consumption $E(t)$ is predicted with the help of *lookup table* generated in section 1.6.1 based

upon the current frequency of the CPU core and aggregate system utilization of the physical server.

1.7.1 Power consumption and response time management

This section uses the concepts introduced in the earlier sections for developing a control structure to manage server power consumption while maintaining the predefined QoS requirement of minimum response time under a time varying dynamic workload for daytrader application hosted in virtualized environment(see section 1.4). Following sections will give details of these experiments.

Experiment Settings: Experimental settings and incoming request profiles were kept similar to Section 1.6.3 for direct comparison between the web server performance with and without the controller implementation. A local monitor running on the on the VM (Nop09) hosting web server collected, processed, and reported performance data after every SAMPLE_TIME (30 seconds) to the controller running on the physical host machine (*Nop03*). These performance data includes average service time at web server (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 SAMPLE_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 requests from the system in the current sample duration.

System State for Predictive Control: We used the exponential Kalman filter described earlier in section 1.6.3 to track the system state online. The two main parameters received from the filter are the current service time S and predicted response time T_{pred}. These values are then plugged into the model described in the previous section. The power model described in Section 1.6.1 was used to estimate the system (physical node of web server) power consumption. With help of these system and power model, the predictive controller provides the optimal configuration of the system in terms of CPU core frequency. Performance of the online controller directly depends upon the accuracy of Kalman filter estimation of the parameters of the web server application model and the power consumption model of the physical system.

For this experiment, we chose optimal system state set point to be $x_s = [q*, r*]$ where $q* = 0$ and $r* = 0$, which shows our inclination towards keeping system queue and response time both minimum. Q and R (user specified relative weights for cost function) were chosen as 10000 and 1 respectively to penalize the multi-tier system a lot more for increment in queue size and response time compared to the increment in power consumption. Additionally, look ahead horizon value N is 2 for the current experiment. Request forecasting parameters β and γ were equal to 0.8 and 0.15 respectively to put

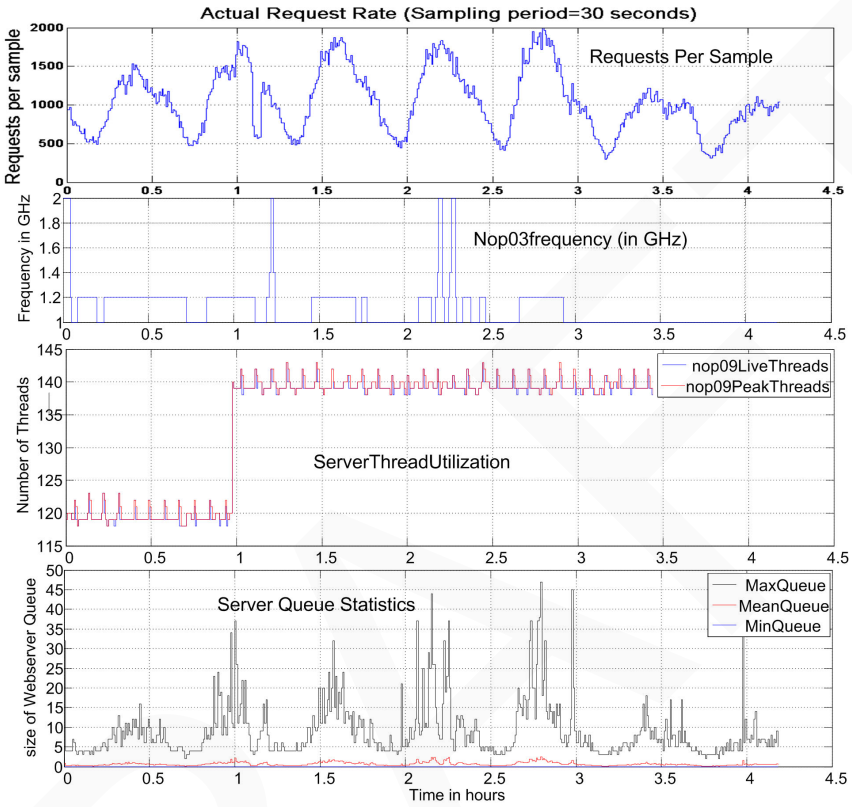
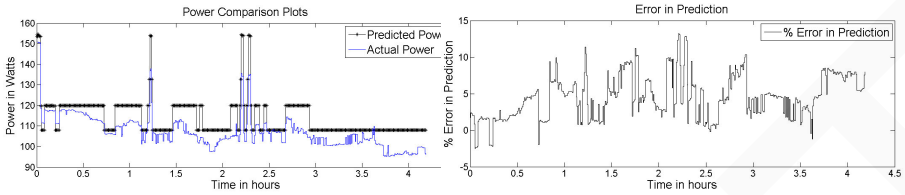


FIGURE 1.11: Web server behavior with controller as per section 1.7.1: sampling period=30 seconds.

maximum weight on the current arrival rate to calculate future arrival request rate.

Results from the experiment are shown in Fig. 1.11 and 1.14, while plots of the estimates from the online Kalman filter are shown in Fig 1.13. Additionally, direct comparison of the response time statistics and power consumption from section 1.6.3 and 1.7.1 are shown in figure 1.14. Validation of our power consumption model defined in section 1.6.1 is done by comparing the power consumption estimated by predictive controller against actual power consumption reported by real time watt meter (see figure 1.12).

Observations from Figure 1.11 and Figure 1.14: The aggregate CPU utilization and memory utilization (sub-figure 1) of web server and database tier are shown in Fig. 1.11. Nop03 CPU core frequencies during the experiment is shown in Fig. 1.11 (sub figure 2) and java thread utilization of the web server is shown in sub-figure 3. sub-figure 4 shows the queue size of the web server through the method described in section 1.7. The most interesting plot in



(a) Predicted and actual power consumption in web server from section 1.7.1

(b) Error in predicting power consumption compared to actual in section 1.7.1

FIGURE 1.12: Comparison of power consumption for actual Vs predicted through predictive controller in section 1.7.1.

figure 1.11 is sub-figure 2, which shows the change in frequency of the CPU core from the controller to achieve predefined Qos requirements based upon the control steps taken by observing the system state and estimating the future environmental inputs. After direct comparison of sub-figure 2 and sub-figure 1 from Figure 1.11, we can see that Nop03 CPU core frequency is changed as the incident request rate at web server changes. Additionally, controller chose 1.2 Ghz frequency for the CPU core until there was some sudden increase or decrease in the incident request rate. Furthermore, controller does not change the frequency of the core too often, even when the incident request rate is changing continuously, which shows the minimal disturbance in the system operation due to predictive controller. The power consumption plot for No03 is shown in fig 1.14(sub-figure 3), while Statistics (max and min) of observed response time at web server are shown in Figure 1.14 (sub-figure 1 and 2).

1.7.2 Performance analysis

According to the Fig. 1.13, online Kalman filter tracks average response time of the incident requests and bottleneck utilization with high accuracy. The estimated service time of the incident requests by the Kalman filter shows minimal variation. According to sub-figure 3, predicted response time from the Kalman filter Tpred and actual response time T are also very close to each other, which indicates efficiency of the Kalman filter. The controlled version runs at a lower frequency most of the times, which results into considerable amount of power saving (18%) over a period of four hours of experiment (fig 1.14 sub-figure 3) compared to the baseline experiment shown in section 1.6.3. The controller changes the frequency of the CPU core at very few occasions, but it is able to identify the sudden increase in the incident request rate which reflects adaptive nature of the controller in case of dynamic load conditions.

According to Fig. 1.14 (sub-figure 1 and 2), even after the presence of a local controller and slow running system (lower frequency), response times at web server is in the similar range in both of the cases. It shows that controller while managing to decrease power consumption does not affect QoS

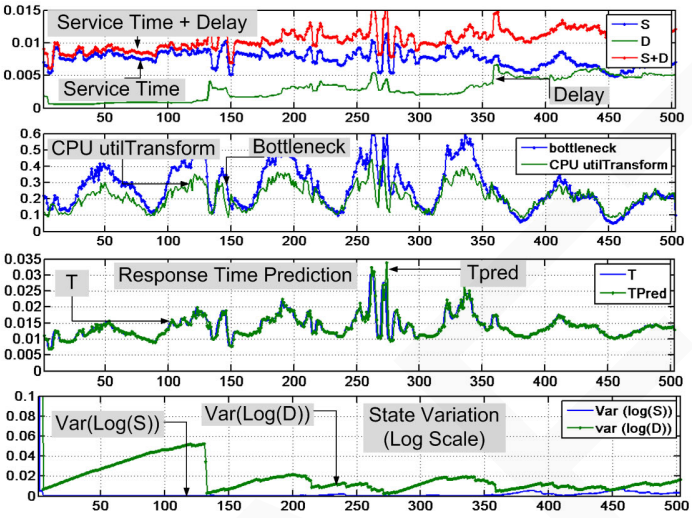


FIGURE 1.13: Online exponential Kalman filter output corresponding to the experiment from section 1.7.1 (figure 1.11 and 1.14). Service time and delay are in millisecond range. Response time is specified in seconds.

objectives of the system negatively. Furthermore, it is visible that there is negligible memory and CPU overhead due to the controller (Fig. 1.14 subfigure 4). The overhead in virtual CPU utilization over web server Nop09 can mostly be attributed to the lower physical core frequency. According to figure 1.12, the power model described in section 1.6.1, estimates the power consumption in the physical machine *Nop03* quite well with only 5% average error in prediction that indicates its effectiveness. Additionally, Java thread utilization is less in case of controller, which indicates that even after slowing down the system, incident requests are getting served in time without much contention of computational resources. Furthermore, we found out that the, mean server queue statistics is also in the same range for both of the cases (details about mean queue statistics in the no controller case are available in [44]).

Observations in previous paragraph indicate that the current system model captures the dynamics of the multi-tier web server (*daytrader*) well. Additionally, the system model uses typical control inputs, state variables, and performance measurements of the multi-tier web service domain for achieving QoS objectives that makes proposed framework suitable for any multi-tier web service system.

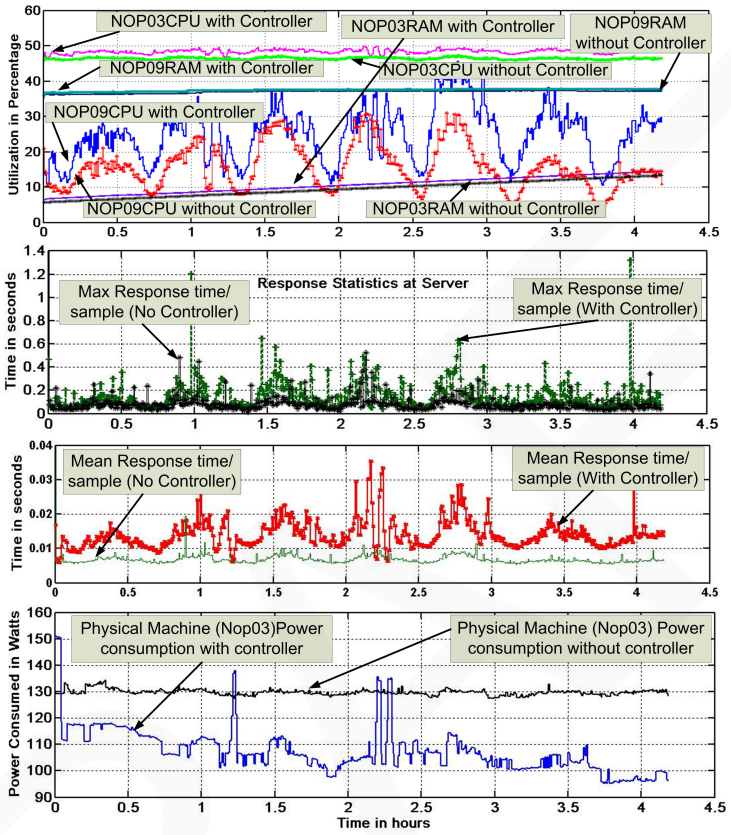


FIGURE 1.14: Comparison of of results with controller 1.7.1 and without controller 1.6.3: Sampling period=30 seconds. Std deviation for all response measurements=0.02 seconds (without controller), 0.019 seconds(with controller)

1.8 Conclusion and Discussions

We have presented a simple and novel approach to develop models with low variance for multi-tier enterprise systems. We showed that the developed model can be integrated with a predictive control framework for dynamically changing the system tuning parameters based on the estimated time varying workload. According to the results shown in section 1.7.1, the developed system model in terms of Kalman filter tracks the system performance on-line with high accuracy. Additionally, the proposed power consumption model of the system used by the controller predicts the overall physical server power

consumption well (95% accurate). Using this model we showed that we can optimize system performance and achieve 18% reduction of power consumption in four hours of experiment in single server without affecting the response time too much. Furthermore, the experimental results (CPU and RAM consumption with and without the controller) indicates that the proposed approach has low run-time overhead in terms of computational and memory resources. We further plan to extend this framework and validate its performance over a cluster of multi-tier computing systems in hierarchical fashion as described in [30].

Enormous research has been done by research industry and academia to make computing system infrastructures power aware by applying power management policies. These policies include hardware redesign, application level controller, efficient cooling system and shutting down the unused servers in a data center. Furthermore, these days virtualization technologies are also used for saving the infrastructure and operating costs of the computing infrastructure. Virtualization enables usage of idle CPU cycles by multiple servers while sharing the hardware resources at same time to reduce the overall power consumption of the system and savings in hardware cost and hosting space. Due to increased power consumption in data centers, there is tremendous need for profiling of data centers with respect to hot spots, air conditioning, and active server usage. Gradually, data center vendors are taking best practices to increase the efficiency of their infrastructure. These best practices include identifying the power consumption in components with respect to impact on performance, enabling the power awareness features, appropriately sized server farms, shutting down unused servers, and removing the unused old servers from the infrastructure [4]. Shutting down the unused servers and installation of an efficient cooling system is easy to accomplish while other methods, which need careful observation of system behavior with changes in environment, are complex and need a skilled architect to design the power efficient policies that can be applicable for various scenarios. In conclusion, it is not necessary to replace the existing hardware with a new power-aware hardware, instead a significant amount of power saving can be achieved easily by employing power aware application (system) controllers and policies on the existing systems.

Bibliography

[1] `http://www.gartner.com/it/page.jsp?id=1368614`.

[2] `http://www.gartner.com/it/page.jsp?id=1442113`.

[3] `http://cwiki.apache.org/GMOxDOC20/daytrader.html`.

[4] `http://www.thegreengrid.org/Global/Content/white-papers/Five-Ways-to-Save-Power`.

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

[6] S. Abdelwahed, N. Kandasamy, and S. Neema. Online control for self-management in computing systems. In *Proc. RTAS*, pages 365–375, 2004.

[7] S. Abdelwahed, S. Neema, J. Loyall, and R. Shapiro. A hybrid control design for QoS management. In *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, pages 366–369, 2003.

[8] T.F. Abdelzaher, K.G. Shin, and N. Bhatti. Performance guarantees for web server end-systems: a control-theoretical approach. *Parallel and Distributed Systems, IEEE Transactions on*, 13(1):80–96, Jan 2002.

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

[10] Manish Bhardwaj, Rex Min, and Anantha Chandrakasan. Power-aware systems, 2000.

[11] L. Bisdounis, S. Nikolaidis, O. Koufolavlou, and C.E. Goutis. Modeling the cmos short-circuit power dissipation. In *Circuits and Systems, 1996. ISCAS '96., 'Connecting the World'., 1996 IEEE International Symposium on*, volume 4, pages 469 –472 vol.4, May 1996.

[12] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Computer Architecture, 2000. Proceedings of the 27th International Symposium on*, pages 83 – 94, 2000.

[13] A. Cervin, J. Eker, B. Bernhardsson, and K. Arzen. Feedback-feedforward scheduling of control tasks. *J. Real-Time Syst.*, 23(1–2), 2002.

[14] Anton Cervin, Johan Eker, Bo Bernhardsson, and Karl-Erik. Feedback–feedforward scheduling of control tasks. *Real-Time Syst.*, 23(1/2):25–53, 2002.

[15] S. L. Chung, S. Lafortune, and F. Lin. Limited lookahead policies in supervisory control of discrete event systems. *IEEE Trans. Autom. Control*, 37(12):1921–1935, December 1992.

[16] V. Delaluz, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin. Memory energy management using software and hardware directed power mode control. Technical report, 2000.

[17] Yixin Diao, Joseph L. Hellerstein, Sujay Parekh, Hidayatullah Shaikh, Maheswaran Surendra, and Asser Tantawi. Modeling differentiated services of multi-tier web applications. *MASCOTS*, 0:314–326, 2006.

[18] Bruno Diniz, Dorgival Guedes, Wagner Meira, Jr., and Ricardo Bianchini. Limiting the power consumption of main memory. In *Proceedings of the 34th annual international symposium on Computer architecture*, ISCA '07, pages 290–301, New York, NY, USA, 2007. ACM.

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

[20] Abhishek Dubey, Rajat Mehrotra, Sherif Abdelwahed, and Asser Tantawi. Performance modeling of distributed multi-tier enterprise systems. *SIGMETRICS Performance Evaluation Review*, 37(2):9–11, 2009.

[21] Dimitris Economou, Suzanne Rivoire, and Christos Kozyrakis. Full-system power analysis and modeling for server environments. In *In Workshop on Modeling Benchmarking and Simulation (MOBS*, 2006.

[22] Diwaker Gupta, Rob Gardner, and Ludmila Cherkasova. Xenmon: Qos monitoring and performance profiling tool. Technical report, HP Labs, 2005.

[23] J. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. Wiley-IEEE Press, 2004.

[24] Yu Hu, Qing Li, and C.-C.J. Kuo. Run-time power consumption modeling for embedded multimedia systems. In *Embedded and Real-Time Computing Systems and Applications, 2005. Proceedings. 11th IEEE International Conference on*, pages 353 – 356, 2005.

[25] I. Hur and C. Lin. A comprehensive approach to dram power management. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 305 –316, 2008.

[26] Russ Joseph, Margaret Martonosidepartment, and Electrical Engineering. Run-time power estimation in high performance microprocessors. In *In International Symposium on Low Power Electronics and Design*, pages 135–140, 2001.

[27] 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.

[28] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *ICAC '09: Proceedings of the 6th international conference on Autonomic computing*, pages 117–126, New York, NY, USA, 2009. ACM.

[29] S. Kamil, J. Shalf, and E. Strohmaier. Power efficiency in high performance computing. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1 –8, 2008.

[30] N. Kandasamy, S. Abdelwahed, and M. Khandekar. A hierarchical optimization framework for autonomic performance management of distributed computing systems. In *Proc. 26th IEEE Int'l Conf. Distributed Computing Systems (ICDCS)*, 2006.

[31] Delaluz Sivasubramaniam Kandemir, V. Delaluz, A. Sivasubramaniam, N. Vijaykrishnan, and M. J. Irwin. Scheduler-based dram energy management. In *In Proceedings of the 39th Conference on Design Automation*, pages 697–702. ACM Press, 2002.

[32] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36:41–50, January 2003.

[33] Leonard Kleinrock. *Theory, Volume 1, Queueing Systems*. Wiley-Interscience, 1975.

[34] Z. Qian Z. Wang R. P. Dick Z. Mao L. Zhang, B. Tiwana and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proc. Int. Conf. Hardware/Software Codesign and System Synthesis*, 2010.

[35] Eun Lee, Indraneel Kulkarni, Dario Pompili, and Manish Parashar. Proactive thermal management in green datacenters. *The Journal of Supercomputing*, pages 1–31, 2010. 10.1007/s11227-010-0453-8.

[36] Weiping Liao, Lei He, and K.M. Lepak. Temperature and supply voltage aware performance and power modeling at microarchitecture level. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(7):1042 – 1053, 2005.

[37] Liu, X. Zhu, S. Singhal, and M. Arlitt. Adaptive entitlement control of resource containers on shared servers. In *Proc. 9th IFIP/IEEE Int'l Symp. Integrated Network Management (IM)*, 2005.

[38] Chenyang Lu, Guillermo A. Alvarez, and John Wilkes. Aqueduct: Online data migration with performance guarantees. In *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*, page 21, Berkeley, CA, USA, 2002. USENIX Association.

[39] Chenyang Lu et al. Feedback control real-time scheduling: Framework, modeling and algorithms. *Journal of Real-Time Systems*, 23:85–126, 2002.

[40] Z. Lu et al. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *Intl Conf. Compilers, Architectures, & Synthesis Embedded Syst. (CASES)*, pages 156–163, 2002.

[41] J. M. Maciejowski. *Predictive Control with Constraints*. Prentice Hall, London, 2002.

[42] Xiaorui Wang Matthias Eiblmaier, Rukun Mao. Power management for main memory with access latency control. Febid '09, San Francisco, CA, USA., 2009. ACM.

[43] Rajat Mehrotra, Abhishek Dubey, Sherif Abdelwahed, and Asser Tantawi. Integrated monitoring and control for performance management of distributed enterprise systems. *Modeling, Analysis, and Simulation of Computer Systems, International Symposium on*, 0:424–426, 2010.

[44] Rajat Mehrotra, Abhishek Dubey, Sherif Abdelwahed, and Asser Tantawi. Model identification for performance management of distributed enterprise systems. Technical Report ISIS-10-104, Institute for Software Integrated Systems, Vanderbilt University, April 2010.

[45] Markus Rupp Mostafa E. A. Ibrahim and Hossam A. H. Fahmy. A precise high-level power consumption model for embedded systems software. *EURASIP Journal on Embedded Systems*, 2011, 2010.

[46] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi. Cpu demand for web serving: Measurement analysis and dynamic estimation. *Performance Evaluation*, 65(6–7):531–553, June 2008.

[47] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. In *Proc. IFIP/IEEE Int. Symp. on Integrated Network Management*, 2001.

[48] David A. Patterson and John L. Hennessy. *Computer Organization and Design, The Hardware/Software Interface, 4th Edition*. Morgan Kaufmann, 2008.

[49] D. Roberts, T. Kgil, and T. Mudge. Using non-volatile memory to save energy in servers. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 743 –748, 2009.

[50] T. Simunic and S. Boyd. Managing power consumption in networks on chips. In *Proc. Design, Automation, & Test Europe (DATE)*, pages 110–116, 2002.

[51] Akshat Verma, Puneet Ahuja, and Anindya Neogi. Power-aware dynamic placement of hpc applications. In *Proceedings of the 22nd annual international conference on Supercomputing*, ICS '08, pages 175–184, New York, NY, USA, 2008. ACM.

[52] M. Woodside, T. Zheng, and M. Litoiu. The use of optimal filters to track parameters of performance models. In *QEST '05: Proceedings of*

the Second International Conference on the Quantitative Evaluation of Systems, page 74, Torino, Italy, September 2005.

[53] M. Woodside, Tao Zheng, and M. Litoiu. Service system resource management based on a tracked layered performance model. In *ICAC '06: Proceedings of the third International Conference on Autonomic Computing*, pages 175–184. IEEE Press, June 2006.

[54] John Zedlewski, Sumeet Sobti, Nitin Garg, Fengzhou Zheng, Arvind Krishnamurthy, and Randolph Wang. Modeling hard-disk power consumption. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 217–230, Berkeley, CA, USA, 2003. USENIX Association.

[55] Fan Zhang and Samuel T. Chanson. Power-aware processor scheduling under average delay constraints. In *Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pages 202–212, Washington, DC, USA, 2005. IEEE Computer Society.

[56] Jiheng Zhang, J. G. Dai, and Bert Zwart. Law of large number limits of limited processor-sharing queues. *Math. Oper. Res.*, 34:937–970, November 2009.

[57] Jiheng Zhang and Bert Zwart. Steady state approximations of limited processor sharing queues in heavy traffic. *Queueing Systems*, 60:227–246, 2008.

[58] Tao Zheng, Murray Woodside, and Marin Litoiu. Performance model estimation and tracking using optimal filters. *IEEE Transactions on Software Engineering*, 34(3):391–406, May–June 2008.

[59] Tao Zheng, Jinmei Yang, Murray Woodside, Marin Litoiu, and Gabriel Iszlai. Tracking time-varying parameters in software systems with extended kalman filters. In *CASCON '05: Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*, pages 334–345. IBM Press, October 2005.

[60] Victor V. Zyuban and Peter M. Kogge. Inherently lower-power high-performance superscalar architectures. *IEEE Trans. Comput.*, 50:268–285, March 2001.