Autonomic Multi-Agent Management of Power and Performance in Data Centers

Rajarshi Das
IBM Research
rajarshi@us.ibm.com

Jeffrey O. Kephart
IBM Research
kephart@us.ibm.com

Charles Lefurgy
IBM Research
lefurgy@us.ibm.com

Gerald Tesauro
IBM Research
gtesauro@us.ibm.com

David W. Levine
IBM Research
dwl@us.ibm.com

Hoi Chan
IBM Research
hychan@us.ibm.com

ABSTRACT

The rapidly rising cost and environmental impact of energy consumption in data centers has become a multi-billion dollar concern globally. In response, the IT Industry is actively engaged in a first-to-market race to develop energy-conserving hardware and software solutions that do not sacrifice performance objectives. In this work we demonstrate a prototype of an integrated data center power management solution that employs server management tools, appropriate sensors and monitors, and an agent-based approach to achieve specified power and performance objectives. By intelligently turning off servers under low-load conditions, we can achieve over 25% power savings over the unmanaged case without incurring SLA penalties for typical daily and weekly periodic demands seen in webserver farms.

Categories and Subject Descriptors

D.4.8 [**Software**]: Performance—*measurements, modeling and prediction, operational analysis*

General Terms

Data center, power measurement, multicriteria utility functions, policy-based management

Keywords

Green Data Center, Power Management, Energy Savings

1. INTRODUCTION

Energy consumption is a major and growing concern for customers, data centers, server vendors, government regulators and non-governmental organizations concerned with energy and environmental matters. To cite a recent and prominent example, the US Congress mandated a study of energy efficiency for servers and data centers [14]. Another clear sign of the growing interest in power management is the Green Grid, an industry consortium dedicated to improving data center power efficiency [20]. Recent trade press articles also make it clear that computer purchasers and data center operators are eager to reduce power consumption and the heat densities being experienced with current systems.

The widespread interest and concern regarding power management is attributable to several alarming trends [4, 10]:

- In 2005, data center servers accounted for 1.2% of electricity use in the United States, doubling since 2000.
- By 2008, 50% of existing data centers will have insufficient power and cooling.
- By 2008, power will be the second-highest operating cost in 70% of all data centers
- Data centers are responsible for the emission of tens of millions of metric tons of carbon dioxide emissions annually – more than 5% of the total global emissions.

In response to these concerns, researchers are exploring a myriad of different methods for reducing power consumption at a wide range of spatial and temporal scales. At the finest granularity, chip manufacturers are including controls that allow the CPU frequency and voltage to be reduced [13]. Methods for exploiting these controls at the server level are being implemented in firmware [12] and in operating systems (e.g. the Linux On-Demand Governor). Middleware is beginning to exploit such controls based on load balancing or virtualization at the level of entire server clusters [9]. Even at the data center-level approaches that go beyond static provisioning to active management techniques which are cognizant of how racks, cooling units, etc. are laid out in the data center have gained prominence recently.

There are several significant challenges to be met in developing a coherent power management strategy for a data center, of which we cite two. First, one must ensure that the various energy-savings techniques, when deployed separately and independently, will not work at cross purposes. Second, the problem of saving energy cannot be addressed independently of other important concerns such as performance targets and resource availability. For example, a good way to achieve high availability is to keep a large number of spare servers idle and at the ready in case another should fail, but these spare servers would be equally well coveted by a power manager (which might wish to turn them off) and a performance manager (which might want to use them to process more workload in parallel). How are these cross-disciplinary conflicts to be resolved?

Cite as: Autonomic Multi-Agent Management of Power and Performance in Data Centers, Rajarshi Das, Jeffrey O. Kephart, Charles Lefurgy, Gerald Tesauro, David W. Levine and Hoi Chan, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)- Industry and Applications Track*, Berger, Burg, Nishiyama (eds.), May, 12-16., 2008, Estoril, Portugal, pp. 107-114.

To these challenges we add an important constraint on the nature of the solution: one that is introduced by business reality. Previous research efforts have attempted to address simultaneous management of power and performance through centralized solutions that manage power and performance jointly. However, although perhaps feasible in prototypes, centralized approaches do not recognize the reality of today's IT environments, which typically contain multiple management products specialized to different disciplines including performance and power as well as other characteristics such as availability [15].

In the abstract, it seems clear that multi-agent systems should provide an excellent and fundamentally appropriate paradigm for addressing the data center management challenges we have cited, especially given the business constraints described above. Indeed, this position is consistent with our earlier claim [3] that autonomic computing is a *killer app* for multi-agent systems, given that data center management is one important, specific application domain for autonomic computing.

The purpose of this paper is to support our theoretical argument with a practical demonstration. Specifically, we describe an implemented system and experiments that demonstrate coherent automated management to specified power and application performance objectives in a medium-sized server cluster housed in a single IBM BladeCenter chassis. The agents are commercial or freeware products, or subcomponents thereof, to which has been added a thin layer of communication and some fairly simple algorithms.

The remainder of this paper is structured as follows. Section 2 sets the stage for our current work by providing a bit of historical perspective. In section 3, we describe our multi-agent architecture. Then, in section 4, we describe the results of two experiments that demonstrate successful management of the system to performance and power objectives that are specified using utility functions. Finally, in section 5, we summarize our main points and provide thoughts on future directions.

2. BACKGROUND

Industry is beginning to provide advanced power management features for servers, of which IBM Active Energy Manager (AEM) Version 3.1 extension[1]) of the IBM System Director (Version 5.20) [6] is an example. IBM Systems Director features open, standards-based design and broad platform and operating support to enable customers to manage heterogeneous environments from a central point. Director is typically run on a dedicated server and remotely monitors other servers in the data center. In our work, we measure power and control the server performance state using internal, prototype software that is somewhat in advance of the power management features available commercially today.

AEM provides power measurements for servers by using either power measurement circuitry built into IBM servers, or by communicating with rack-level power distribution units to monitor AC power of 3rd party servers. Based on such information, customers can allocate lower power usage on select IBM servers by two different techniques: *power capping* and *power savings*. Power capping lets users set a maximum power level per system while power savings mode lets users manage power usage by running the processors on the server at their lowest frequency and/or voltage setting. Today, AEM is manually operated by the systems administrator and does not have the capability to use performance information to make power management decisions on demand. In our work, we show that, by casting an advanced prototype of AEM as an agent and situating it in a multi-agent systems with other agents possessing the knowledge and capability to manage performance and other aspects of system behavior, we can achieve autonomic management of power and performance in accordance with specified objectives.

[1]AEM was formerly known as Power Executive.

Our approach differs from previous literature on performance and power management in several respects. First, we power servers off when they are not idle, which is inherently able to achieve greater energy savings than clock frequency manipulation techniques that have been studied previously [9]. Moreover, we differ from previous approaches that use intelligent placement of virtual machines or applications to consolidate servers [2] in that we employ load balancing to route work away from servers, allowing them to be powered off. To our knowledge, we are unique in offering a multi-agent, more de-centralized approach, which we believe is increasingly necessary as the system scales up in size, as well as an implemented system in which real (as opposed to simulated) servers are powered on and off, and in which actual power measurements are taken.

3. ARCHITECTURE

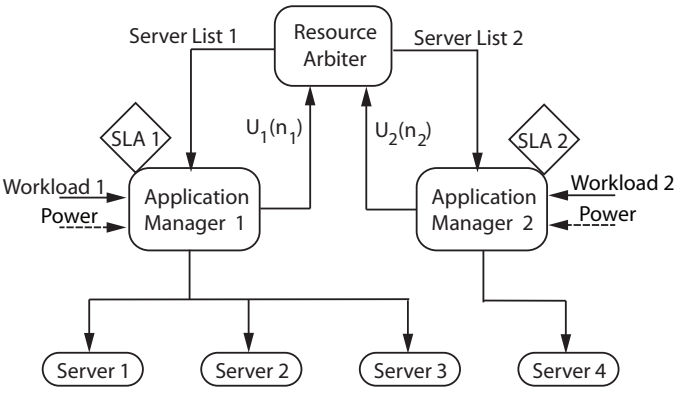

Figure 1: Unity architecture for optimal resource allocation in a data center.

This paper builds upon our prior "Unity" architecture for implementing self-management in a data center via interactions amongst a population of autonomous agents [16]. Unity supports multiple, logically separated *Application Environments*, each providing a distinct application service. Each Application environment is represented by an *Application manager* agent, which is responsible for local performance optimization within the application and communicating with a *Resource Arbiter* agent regarding resource needs (Figure 1). Agent-based resource allocation in the data center is achieved as follows. Each Application Manager i periodically computes and reports to the Arbiter a utility function $U_i(n_i)$ estimating expected business value to the application of receiving an allocation of n_i homogeneous servers. Business value within an application may be defined, e.g., by a performance-based Service Level Agreement (SLA), which stipulates payments or penalties as function of one or more performance metrics. Unity assumes that all

utility functions share a common scale of valuation, such as money. Given the most recent utility information, the Resource Arbiter periodically solves for the globally optimal allocation maximizing total expected business value summed over the applications. The Arbiter then conveys a list of assigned servers to each application, which are then used in dedicated fashion until the next allocation decision.

Our current research aims ultimately to extend our Unity architecture to encompass optimization over multiple competing criteria: application performance and power consumption. We propose to achieve this by including a power cost model in the utility functions $U_i(n_i)$, and extending the Arbiter's calculations to include an optimal number of servers N_{off} to be powered off in the data center, as well as the above computation of optimal allocated servers $\{n_i\}$ to each application. This paper describes our initial experiments in which performance and power tradeoffs are made solely within a single application. In this case there was no need to employ the full Unity architecture for optimizing across multiple applications.

Figure 2 provides a high-level overview of an Application Manager in our prototype data center. In brief, a Workload Generator produces a single workload with dynamically varying intensity, which is routed by a Workload Distributor to a set of blade servers contained in a single IBM BladeCenter chassis. The Workload Distributor's routing policy is set within the Performance Agent. Power consumption on each blade is managed dynamically in accordance with a control policy set by the Power Agent. A Coordination Agent manages interactions between the Performance Agent and the Power Agent in accordance with a power-performance policy generated by the Policy Generator. The Policy Generator takes as input a user-specified power-performance utility function and a system model derived from power and performance measurements. In the current implementation it is a simple optimization algorithm that is run just once prior to the beginning of the experiment. The workload generator, workload distributor, and agents are implemented using a mixture of open-source software and internally developed tools and firmware; these are described next.

3.1 The Performance Agent

The Performance Agent in our setup was primarily based on Apache 2.2.6, a robust, commercial-grade, featureful, open-source HTTP (Web) server [19]. One of the blades is set up as a reverse proxy server (using the Apache extension modules `mod_proxy` and `mod_proxy_balancer`) to perform load balancing among the HTTP servers in the data center. The `mod_proxy_balancer` module uses its Request Counting scheduler algorithm to distribute the HTTP requests. The scheduler can be reconfigured dynamically through a web interface at `http://proxy-server-name/balancer-manager` to change the share of total workload a particular server receives. If the share for an HTTP server is zero, it is considered to be off-line. Once the HTTP server is off-line, Apache can be stopped and the server can be powered down by the Power Agent. Naturally, this sequence of actions is reversed when starting the HTTP server and such coordinated sequence of control actions was managed by the Coordination Agent in our data center as explained in Section 3.3. Finally, we made use of Apache's base module `mod_status` to determine how well a server is performing by obtaining current server statistics via the webpage `http://server-name/server-status`.

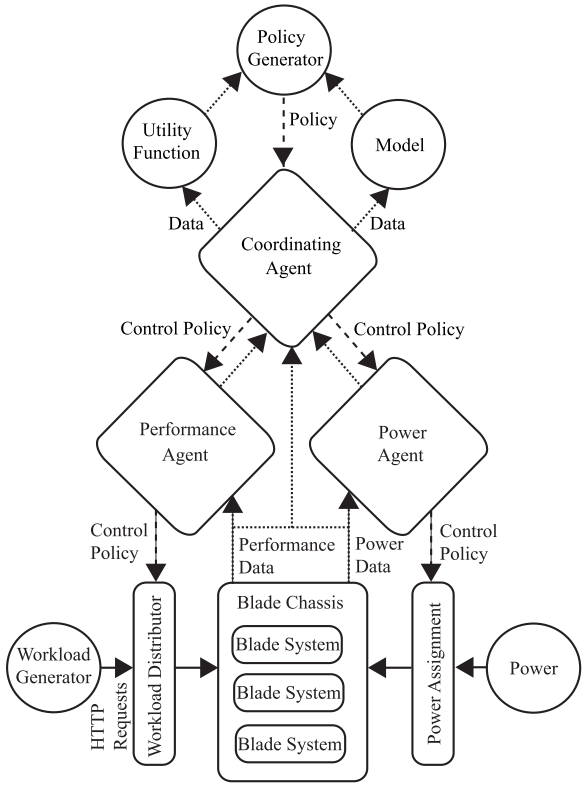

Figure 2: Overview of an Application Manager in our prototype data center with the Coordination Agent managing interactions between the Power Agent and Performance Agent.

3.2 The Power Agent

The Power Agent in our data center was based on an IBM internal program called Energy Management Tool (EMT). EMT is written entirely in `[Incr Tcl]` scripting language and provides a way to quickly prototype advanced power management features. Individual blade servers run internally developed prototype firmware that provide basic functions such as monitoring a server's power consumption and setting a server's power cap [12]. EMT uses these functions by communicating with this prototype firmware over Ethernet using the industry-standard IPMI protocol [8]. IPMI, which defines a set of common interfaces to computer hardware and firmware and operates independently of the operating system, allows administrators to remotely manage a system even when monitored system is not powered on. The service processor on our blade servers which runs the IPMI communication stack remains powered on, even when the blade is turned "off". The power measurements are taken by a power measurement circuit built-in to IBM blade servers which measures the entire DC power of the blade [12]. This is the same method by which the IBM Active Energy Manager product provides server power measurements. The power measurements are accurate to 0.1 W.

In previous work [9], we exploited the ability of EMT to throttle the CPU clock to achieve energy savings in the range of 10%. In contrast, in this work we achieve greater energy savings via a more drastic approach: powering individual servers on and off. We modified EMT to power blades up and down automatically by sending IPMI commands `Chas-`

`sis Control - power up` or `Chassis Control - initiate soft shutdown` to individual servers. When a blade server is turned "off", it consumes under 10W of power. The soft shutdown command invokes the normal shutdown procedure of the operating system.

In addition, we extended EMT to monitor the BladeCenter blower (fan) speeds and the temperature of each processor core on the blades. The temperature of each core in the Intel Woodcrest processor is measured by digital temperature sensors built into the cores. The temperature values are obtained by reading the IA32_THERM_STATUS register in the Intel Woodcrest Xeon processor. Periodically, EMT uses ssh protocol to login to each server and run a program we wrote to read the IA32_THERM_STATUS registers.

If the Power Agent receives multiple power-on or power-off control signals from the Coordination Agent within too short a span of time, it may choose to ignore them, or it may cancel an operation that is already in progress. It communicates to the other agents the power state of the individual servers as necessary.

3.3 Coordination Agent

We designed and implemented a Coordination Agent in Java to not only coordinate the interactions between the Performance Agent and the Power Agent, but also to provide a coherent control policy to manage the data center based on predefined utility functions and a system model. The Coordination agent was placed on a separate server and was configured to have no-password ssh (a secure clone of rsh with RSA encryption based authentication) access to all servers in the data center.

The information flowing between the Coordination Agent and the Performance Agent or the Power Agent can be broadly categorized into two types: (i) monitored performance and power data, and (ii) control signals for policy enforcement. The list of various information flows between the agents in our data center is presented in Table 1.

3.4 Policy Generator

The Policy Generator is not an agent in the current implementation, but it plays an important role in the multi-agent system because it derives the power-performance policy that governs the control signals sent by the Coordination Agent to the Power and Performance agents.

The Policy Generator takes as input a power-performance utility function and a system model. As in prior work [9], the utility function has the form:

$$U_{pp}(\mathbf{Perf}, \mathbf{Pwr}) = U_{\mathrm{Perf}}(\mathbf{Perf}) + \epsilon * U_{\mathrm{Pwr}}(\mathrm{Pwr}) \qquad (1)$$

where ϵ is a tunable coefficient expressing the relative value of power and performance objectives, **Perf** is a vector of performance metrics of interest (e.g. response time and/or throughput for one or more service classes, CPU utilization), and **Pwr** is a vector that represents all energy-related attributes of interest (e.g., the total power consumption and the set of intake and/or processor temperatures)[2]. In the experiments reported in this paper, we take **Perf** to be a simple scalar RT representing the response time of a single service class, while **Pwr** is a simple scalar Pwr representing the total power used by all of the servers in the cluster.

The system model expresses the performance and power vectors as functions of the control parameters and observable metrics, both of which can be represented as vectors:

$$\mathbf{Perf}(\mathbf{ControlParams}, \mathbf{Observables}) \qquad (2)$$

$$\mathbf{Pwr}(\mathbf{ControlParams}, \mathbf{Observables}) \qquad (3)$$

As will be illustrated in Section 4.3, the system model can be learned from data[3]. We use an off-line approach in which the model is learned during an initial training episode during which the system is subjected to a controlled workload designed to explore the space of environmental conditions that are likely to hold under typical system operation. An online approach, in which the system model would be adapted during system operation, could also be employed.

The Policy Generator substitutes the system model into the utility function, resulting in a system-level utility function

$$U_{pp}(\mathbf{Perf}, \mathbf{Pwr}) = U'_{pp}(\mathbf{ControlParams}, \mathbf{Observables}) \qquad (4)$$

Then, the Policy Generator computes the policy. There are a number of possible approaches. The Policy Generator can simply send the system-level utility expressed in Eq. 4. Then, when the Coordinating Agent observes a specific system state, it can compute the set of control parameters that maximizes U'_{pp}, and pass the appropriate components to the Performance and Power agents. A second approach, which we employ, is for the Policy Generator to precompute the optimal set of control parameters sets for each possible set of observables. This information is stored in a lookup table, which constitutes the policy used by the Coordination Agent. If the observable space contains continuous variables, then the policy cannot be pre-computed for all possible observables, and the optimal control parameters are evaluated on a grid. These values are stored in a lookup table, and the Coordination Agent uses them as the basis for interpolation or extrapolation.

4. EXPERIMENTS

First, we describe our experimental setup, including specifics on the platforms and their configuration. Then, after detailing the workload and how it is generated, we discuss off-line experiments that we used to establish system models. Finally, we describe results of experiments in which we dynamically turned servers on and off in an effort to save energy while maintaining an acceptable level of performance.

4.1 Experimental setup

4.1.1 Data center Hardware and Topology

Our prototype two-tier data center consisted of four IBM BladeCenter HS21 blade servers featuring two 3.0 GHz dual-core Intel Woodcrest Xeon processors and 1 GB memory per blade, all residing in a single chassis. The BladeCenter supports dynamically powering individual blades on or off without interfering with operation of other blades in the chassis. Apache was installed and configured on one blade to

[2]Note that the constant ϵ is not strictly necessary, since the function U_{Pwr} can be scaled. However, in practice it is useful to define the two utilities independently and then use ϵ to adjust their relative importance.

[3]Of course, as delineated in our earlier work [9], the system model can also be based on queueing theory or control theory.

Performance			
Monitoring		**Control**	
Attribute	Platform (`Command`)	Attribute	Platform (`Command`)
%CPU Utilization	Unix (`top`)	Server Schedule	`mod_proxy_balancer` (`En/Disable`)
Process Queue Length	Unix (`top`)	Share of Workload	`mod_proxy_balancer` (`lbfactor=?`)
Response Time	Workload Generator, Apache	Urgency of Workload	`mod_proxy_balancer` (`lbstatus=?`)
Throughput	Workload Generator, Apache	Start/Stop Server	Apache (`apachectl start/stop`)
Workload Intensity	Workload Generator, Apache	Multithreading on Cores	OpenMP (`OMP_NUM_THREADS=4`)
Power			
Monitoring		**Control**	
Attribute	Platform	Attribute	Platform (`Command`)
Power	Server power measurement circuit	Power up/down Server	IPMI (`Chassis Control power up`)
Ambient Temperature	BladeCenter Management Module	Power Capping	set by EMT prototype firmware [12]
Fan Speed	BladeCenter Management Module	Power Saving	set by EMT prototype firmware
CPU Temperature	Intel Xeon temperature sensor		
CPU Frequency	EMT prototype firmware		
CPU Voltage	EMT prototype firmware		

Table 1: Monitoring Data and Control Signals between Agents

function as the reverse proxy server with the load balancer (by modifying its Apache webserver's `httpd.conf` file). On the remaining three blades, the power-intensive LINPACK application was installed, and Apache was configured as an application server to process transactional requests to LINPACK. To reduce the effects of network latency, we also used additional blades mounted on the same chassis to host the Workload Generator, the Power Agent (AEM) and the Coordination Agent. The server hosting the Coordination Agent had root no-password `ssh` access to all blades in the data center. In terms of available power a BladeCenter chassis has two power domains and is configured with four 2000 W power supplies total. Each power domain is redundantly connected to two of the power supplies so that in the event of a single supply failure, the domain continues operating with the remaining power supply.

4.2 Workload Generation

4.2.1 Workload

While we have experimented in the past with I/O-bound and memory-bound transactional workload, here we focus solely on CPU-bound HTTP workload. More precisely, our data center was designed to provide a linear equation solver service. Upon the arrival of each HTTP request, the Apache webserver invoked the Intel Optimized (SMP) LINPACK Benchmark 10.0 `linpack_xeon64` [7] application which ran on the same server as Apache itself. This version of LINPACK, which solves a dense system of linear equations $\mathbf{Ax} = \mathbf{b}$, used the Intel Math Kernel Library and has been highly tuned for maximum performance on 64-bit Intel processor-based systems. Each HTTP request sets both the number of linear equations in $\mathbf{Ax} = \mathbf{b}$ and the leading dimension of the two dimensional matrix $\mathbf{A}$ to 1000. Moreover, in order to simultaneously leverage all four cores on the blade servers, the CGI script ran `linpack_xeon64` in a 4-way multithreaded fashion by first setting `OMP_NUM_THREADS=4`.

4.2.2 Workload Trace

Our emulation of time-varying demand experienced by data centers in the real-world was based on traces of all HTTP requests to the NASA Kennedy Space Center webserver in Florida in July and August of 1995. These weblog traces, which are freely available at the Internet Traffic Achive [11], fully preserve the originating host and the HTTP requests with timestamps that have 1 second resolution. We processed the logs to measure the number of unique users (by originating URL) in every 10 minute interval and obtain a new time-series $\mathcal{T}_c$ to quantify the number of clients logging into the data center at each time interval[4]. The time series of clients $\mathcal{T}_c$ clearly showed regular periodicities at both daily and weekly time scales, with peak-to-mean load ratios of ≈ 2 in each case. The combination of two time-scales resulted in significantly different peak workload conditions on weekdays and weekends, so we distinguished the two timeframes in our experiments.

4.2.3 Workload Generator

Since the number of clients defined the magnitude of the workload intensity in all our experiments, we configured our Java-based multithreaded workload generator to generate HTTP requests from an adjustable number of clients in a closed queueing network, all of which have exponentially distributed think times with a mean of 1 second. To complete the emulation of real-world workload, the workload generator then adjusted the number of clients in each time interval according to the time-series $\mathcal{T}_c$. In our experiments, the workload generator adjusted the number of clients every minute. Therefore, we drove the original NASA weblog trace through our system 10 times faster than the original rate.

4.3 Multicriteria Utility Function

The Performance Agent's utility function was defined using the following sigmoid function:

$$U_{\text{Perf}}(\mathbf{Perf}) = 1000 * \left(1 - \frac{1}{1 + e^{0.01*(\text{RT}-\text{RT}_0)}}\right) \qquad (5)$$

where RT was the average response time in milliseconds during a given time interval and RT_0 was the target response time. Thus the utility $U_{\text{Perf}}(\text{RT})$ ranges between 0 and 1000, and is equal to 500 when $\text{RT} = \text{RT}_0$.

Initial experiments showed that, for the maximum intensity of workload in the time-series $\mathcal{T}_c$ we consider in this

[4]We rescaled $\mathcal{T}_c$ by a factor of 3.5 to match our data center setup.

paper (= 55 clients), all three servers are necessary to provide an average response time of 1000 milliseconds. This response time was set as the SLA target: $\mathrm{RT}_0 = 1000$ milliseconds.

The Power Agent's utility function was defined as a linear function of the sum total of power consumed by all of the servers:

$$U_{\mathrm{Pwr}}(\mathrm{Pwr}) = -1.2 * Pwr \tag{6}$$

where Power is measured in watts. Note that, unlike the performance utility, there is no lower bound on $U_{\mathrm{Pwr}}(\mathrm{Pwr})$. If none of the servers were powered on, the power utility would reach its upper bound of zero. However, the option of turning all of the servers off is never taken in our prototype. The Proxy Server and at least one LINPACK server are always available, and therefore the upper bound on $U(\mathrm{Pwr}) \approx -400$, corresponding to the power consumed under the lowest workload conditions.

The multicriteria utility function to be maximized by the Policy Generator was then represented as

$$U_{pp}(\mathrm{RT}, \mathrm{Pwr}) = 1000 * \left(1 - \frac{1}{1 + e^{0.01*(\mathrm{RT}-1000)}} - 0.0012 * Pwr \right) \tag{7}$$

4.4 Model Learning and Policy Generation

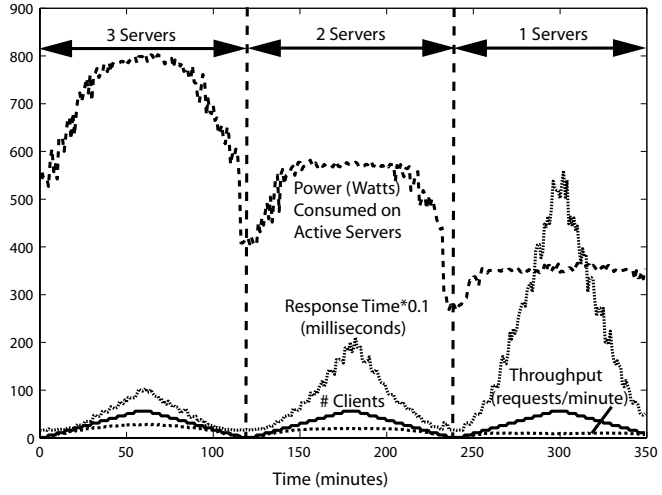

Figure 3: Experiment to determine model-based policy, displaying as a function of time and as the number of servers turned on is decreased in steps from 3 to 1, the a) number of clients, b) throughput, c) response time (scaled down by a factor of 10), and d) power consumption of all servers when idle servers are turned off.

As detailed earlier, in this work we focus on energy savings in the data center by powering individual servers on and off. In principle, we can represent the control parameters as a vector in which each component represents the power state of an individual server. However, given that the servers are all identical in our setup, we can collapse the vector into a scalar: the *number* of servers powered on. Thus the Coordination Agent's policy can be represented as a function mapping the observable state (or a portion of the observable state deemed relevant, such as the number of clients) into a number of servers to be powered on. The Coordination Agent relays the identities of the desired servers to the Power Agent and Performance Agent, which then take individual actions to ensure that the indicated servers can be powered on or off. Since the proxy server has to be powered on all the time to distribute workload in the data center, only the LINPACK servers can be turned or or off by the policy.

The Coordination Agent's policy was computed by the Policy Generator by the method described in Section 3.4. To obtain the model, we monitored each of the performance and power attributes listed in Table 1 on all four blades as we varied the number of servers powered on and the workload intensity. Figure 3 presents the time-series of the workload intensity, the number of powered servers, the average response time, the throughput and the consumed power in the entire data center observed in this experiment[5]. Although the Coordination Agent had access to information about a large number of attributes, as in our prior work [9], we first attempted to derive a policy based on a single data center state variable: the workload intensity. This experiment allowed us to characterize the relationship of the various attributes including response time and power as a function of the number of clients. To show the efficacy of our approach, for each level for servers powered on, we directly fitted the experimental data for response time and power vs. the number of clients to simple quadratic models without any pre/post-processing as shown in Figure 4(a-b). With these models in hand, we could estimate the joint utility function values of performance and power for all levels of workload intensity using Equation 7. Figure 4(c) plots these estimated utility values as a function of the number of clients for the three control settings (i.e., number of servers to turn on). Since the goal is to maximize utility, in this case it is easy for the Policy Generator to derive the following policy:

when $C < 17$, power on one server;
when $17 \leq C < 34$, power on 2 servers;
when $34 \leq C < 57$, power on 3 servers;
when $C > 57$, power on 1 server

where C is the number of clients in the system. Although we do not have experimental data when workload intensity is more than 56, the utility estimates in Figure 4(c) suggests that the system would maximize utility by actually turning on only one server in such adverse workload conditions. Since $U_{\mathrm{Perf}}(\mathrm{RT})$ is bounded from below by zero, further increasing the workload under such conditions results in negligible loss of utility when only one is server turned on. On the other hand, when three servers are turned on in such scenarios, $U_{\mathrm{Perf}}(\mathrm{RT})$ continues drop sharply as the response time is close to the target of 1000 milliseconds.

4.5 Experimental results

The results in this paper focus on four consecutive days (96 hours) from the NASA data starting on the first Thursday in July 1995 and ending on the following Sunday; due to temporal scaling in replaying the workload time-series $\mathcal{T}_c$, our experiments took 9.6 hours, or 576 minutes.

We performed two independent experiments, both of which entailed replaying the tape of workload intensity $\mathcal{T}_c$. In the first experiment, all four servers were powered on all of the time. In the second experiment, the servers were powered on or off as dictated by the pre-computed policy.

[5]Thus the power measurements include the proxy server which was always turned on.

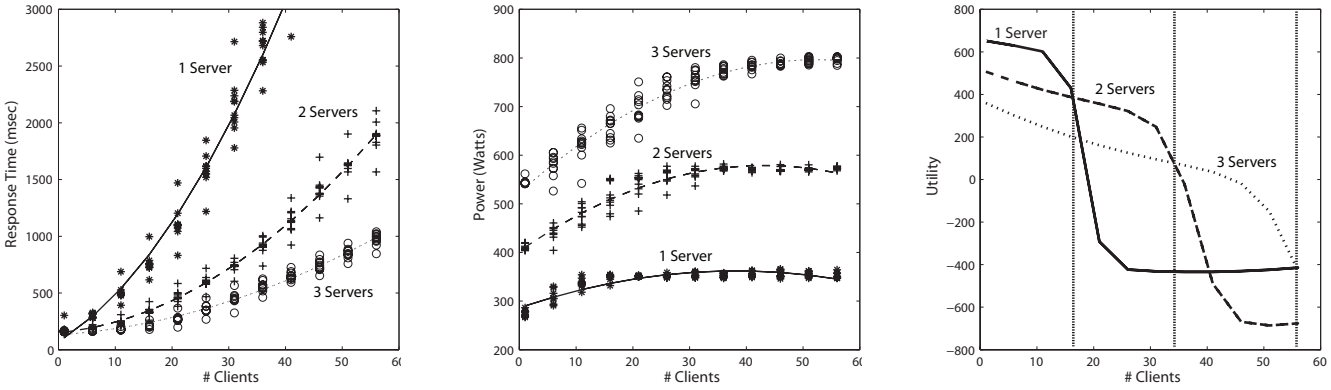

Figure 4: Observed (a) response time, (b) power consumption, and (c) estimated utility values for 1 server, 2 servers and 3 servers powered on. The utility values for policy derivation are obtained from quadratic fits of the data as shown in (a) and (b).

Period	Energy (kWh) without Policy	Energy (kWh) with Policy	Energy Saved
Weekdays	16.9	13.5	20.1%
Weekends	14.6	8.5	41.8%
Week	113.7	84.5	25.7%

Table 2: Average energy (kilowatt-hour) usage in our data center with 4 IBM HS21 blades with and without power management policy.

In our experiments, we find that blades running Linux can take anywhere from 5 to 10 minutes to boot up and resume processing workload. Thus, if we actually turned off servers in these experiments, the temporal scaling by a factor of 10 is equivalent to the servers taking up to 100 minutes per reboot, which is clearly unrealistic. Therefore, in the second experiment only, the Power Agent was modified so that it would not execute the request from the Coordination Agent to power servers off. However, it interacted with the other agents as if it had done so, and reported the power state as if it took 1 minute (10 minutes of simulated time) to power on a server in response to a request. It reported the power consumption during the "off" period as 10 W (rather than the actual power in the idle state, which is much higher). The simulated latency of 10 minutes provides a slightly pessimistic estimate of the amount of energy savings we can attain. All other agents perform as usual, e.g. the Performance Agent stops sending HTTP requests to servers that are "powered down" by the policy.

Figure 5 presents the time-series of workload intensity (number of clients), number of powered servers, response time, throughput and power from both experiments. The figure shows that, through the coordinated interactions between the Performance Agent, the Power Agent and the Coordination Agent, the data center is able to turn servers on or off based on the derived policy. As would be expected, the policy correctly powers on all three servers during peak workload conditions and turns off servers under low conditions of low workload. Although the policy results in increased response time, it successfully met the SLA target in 574 of 576 (99.65%) one-minute intervals in the experiment. On the other hand, there is little difference between the throughput of the data center in the two experiments. Figure 4 helps to explain this observation. When the workload intensity is low, additional servers do not increase throughput, as all of the demand is being satisfied. On the other hand, when workload intensity is high, all three LINPACK servers are powered on, so there can be no difference from the unmanaged case. In between these extremes (and indeed even at these extremes), one can see that the response time is very much more sensitive to the number of clients than the throughput, particularly as throughput begins to saturate. Since the power policy is designed to keep response time at an acceptable level, it will have the side-effect of keeping the throughput from getting very deep into saturation.

Figure 5 demonstrates that the model-based policy can significantly reduce power consumption. Table 2 explores this issue further by summarizing the energy usage on weekdays and weekends based on our experiments. The weekly figures, which are estimated by weighting the aggregate results from weekdays and weekends, show that even in a small data center environment it is possible to attain over 25% energy savings using a simple power management policy without incurring SLA penalties for typical daily and weekly periodic demands experienced in data centers.

5. CONCLUSIONS

Our primary aim in this work has been to develop and demonstrate the feasibility of applying multi-agent approaches to managing power and performance in server clusters through a prototype in which the agents are based on commercial or commonly available shareware products or research prototypes thereof. Our testbed employs a commercially available IBM BladeCenter, and realistic HTTP traffic governed by time-varying demand logged at a NASA web site. Our system makes management decisions in real time, based on live system measurements, taking into account multiple consequences of the decisions (e.g., on power consumption and application performance). Specifically, we demonstrated that our approach can achieve power savings of more than 25% while simultaneously meeting the application's performance target as specified in its SLA.

The off-line model-building approach used in this paper, which empirically measures the consequences of all possible management decisions (number of servers powered on) over all possible values of independent system state variables

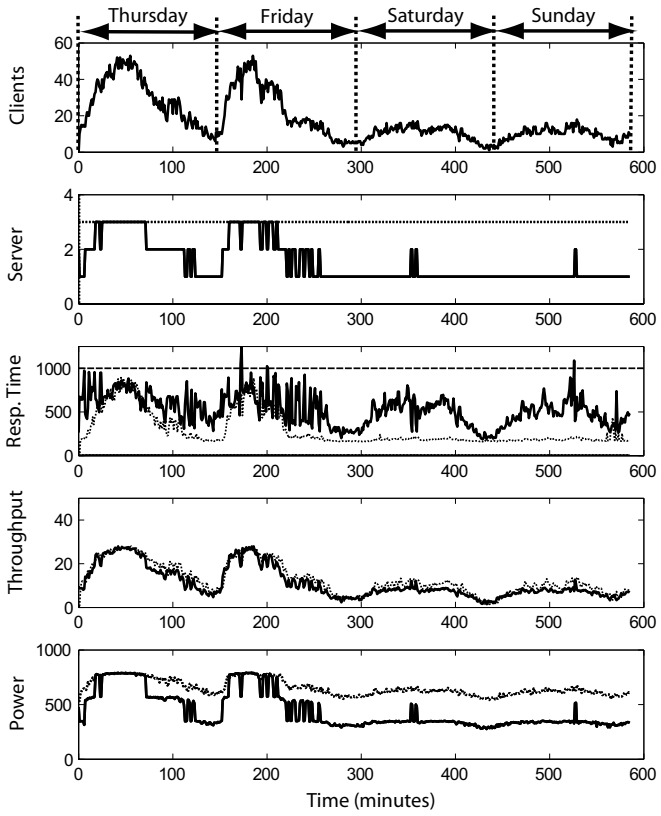

Figure 5: System behavior on 4 days (Thursday-Sunday) under the same trace of workload intensity (number of clients) with (solid line) and without (dotted line) power management policy in effect: number of powered servers, response time (msec), throughput (requests/minute), power consumption (watts).

(number of clients), is only feasible in low-dimensional state and action spaces. Hence this approach will not be viable as we scale our testbed in size and complexity. Instead, we plan to make use of more advanced machine learning methods that rely more heavily on function approximation and are capable of learning management policies by observing live trajectories through the system state and action space. One such approach is Hybrid Reinforcement Learning [18, 17], which has shown promising initial results in learning policies in high-dimensional state spaces for computational resource allocation, and for simultaneous management of performance and power consumption by dynamically modulating CPU clock speeds. Other related approaches which we expect to be of benefit include model-based reinforcement learning approaches (e.g., [1]) which simultaneously learn state-transition models along with expected value functions.

In future work, we will extend our prototype system to include multiple applications, and will make use of our Unity architecture and a global Resource Arbiter to make system-wide management decisions regarding the total number of servers to be powered on/off, as well as the number of servers to be allocated to each application. We expect the Unity approach to be highly effective as long as individual application states have negligible impact on each other. This assumption should be valid provided that applications do not share servers, and that any thermal couplings between applications are either negligible or unmanaged. However, in cases where the data center employs an explicit thermal management discipline, or allocates VMs to applications instead of whole servers, the Unity architecture may need to be modified so that interacting applications may coordinate to jointly optimize total business value. An intriguing Machine Learning approach which holds promise in such scenarios is Coordinated Reinforcement Learning [5].

6. REFERENCES

[1] P. Abbeel et al. An application of reinforcement learning to aerobatic helicopter flight. In *Proc. of NIPS-06*, 2006.

[2] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management*, pages 119–128, 2007.

[3] R. Das, I. Whalley, and J. O. Kephart. Utility-based collaboration among autonomous agents for resource allocation in data centers. In *AAMAS*, pages 1572–1579, 2006.

[4] Gartner Inc. Gartner Says 50 Percent of Data Centers Will Have Insufficient Power and Cooling Capacity by 2008. Press Release, November 29, 2006.

[5] C. Guestrin, M. Lagoudakis, and R. Parr. Coordinated reinforcement learning. In *Proc. of ICML-02*, pages 227–234, 2002.

[6] IBM Corp. IBM Director Agent Extensions: Active Energy Manager. http://www-03.ibm.com /systems/management/director/extensions/ actengmrg.html, 2007.

[7] Intel Corp. Intel Math Kernel Library 10.0 - LINPACK. http://www.intel.com/cd/software /products/asmo-na/eng/266857.htm, 2007.

[8] Intel Corp. et al. Intelligent platform management interface specification v2.0. http://www.intel.com/design/servers/ipmi/pdf/ IPMIv2_0_rev1_0_E3_markup.pdf, 2006.

[9] J. O. Kephart et al. Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs. In *Proc. of ICAC-07*, 2007.

[10] J. G. Koomey. Estimating total power consumption by servers in the U.S. and the world. http://enterprise.amd.com/Downloads/ svrpwrusecompletefinal.pdf, 2007.

[11] Lawrence Berkeley National Laboratory. The internet traffic achive. http://ita.ee.lbl.gov/, 2005.

[12] C. Lefurgy, X. Wang, and M. Ware. Server-level power control. In *Proc. of ICAC-07*, 2007.

[13] A. Naveh et al. Power and thermal management in the intel core duo processor. *Intel Technology J.*, 10(2), May 2006.

[14] U. E. P. A. E. S. Program. Report to congress on server and data center energy efficiency. http://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf, 2007.

[15] O. F. Rana and J. O. Kephart. Building effective multivendor autonomic computing systems. *IEEE Distributed Systems Online*, 7(9), 2006.

[16] G. Tesauro et al. A multi-agent systems approach to autonomic computing. In *AAMAS*, pages 464–471. IEEE Computer Society, 2004.

[17] G. Tesauro et al. Managing power consumption and performance of computing systems using reinforcement learning. In *Proc. of NIPS-07*, 2007.

[18] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *Proc. of ICAC-06*, pages 65–73, 2006.

[19] The Apache Software Foundation. Apache HTTP server version 2.2. http://httpd.apache.org/docs/2.2/, 2007.

[20] The Green Grid Consortium. The green grid. http://www.thegreengrid.org/, 2007.