Model Evaluation of PAT: A Comprehensive Study

Zichen Xu [1], *Yicheng Tu* [2], *Xiaorui Wang* [1]
[1] *Electrical and Computer Engineering, The Ohio State University, U.S.A.*
{xuz,xwang}@ece.osu.edu,
[2] *Computer Science and Engineering, University of South Florida, U.S.A.*
ytu@cse.usf.edu,

USF CSE Technical Report
Number 12-039
April 2012

Department of Computer Science and Engineering
University of South Florida
Tampa, FL 33620
URL: http://www.cse.usf.edu/

Model Evaluation of PAT: A Comprehensive Study

Zichen Xu [1], Yicheng Tu [2], Xiaorui Wang [1]
[1] Electrical and Computer Engineering, The Ohio State University, U.S.A.
{xuz,xwang}@ece.osu.edu,
[2] Computer Science and Engineering, University of South Florida, U.S.A.
ytu@cse.usf.edu,

University of South Florida, CSE, CSE/12-039
April 2012

Abstract

Performance of data processing is without doubt the most important criterion in DBMS design. However, the increasing recurring energy cost gradually rivals the benefits of chasing after performance. Therefore, there are strong economic incentives to minimize power consumption within the performance budget, so that the energy cost could be best amortized. Such a goal is difficult to achieve in practice because the power cost consumption tends to vary significantly with actual database workloads. Effective power management solutions are needed to lower the power cost of database servers while maintaining desired performance.

Many modern hardware systems provide multiple modes with different power/performance tradeoffs. Among them, changing CPU frequency (and voltage) has been found to be a promising direction towards high energy efficiency in database systems. However, no efforts have been reported on dynamically changing CPU frequency for minimizing energy consumption under a performance bound. In this paper, we present Power-Aware Throughput Control (PAT), a software framework within DBMS that reduces the power consumption of database systems by implementing a feedback PI controller based on workload and system models on power and throughput. The design of PAT based on rigorous control-theoretic analysis brings performance guarantees that are not possible in heuristics-based database tuning techniques. A workload classification scheme based on fuzzy logic is also developed to capture key information of the workload and further improve the accuracy of system modeling and thus overall PAT performance. Also, comparing with open-loop CPU frequency control, *ad hoc* control and conventional feedback control, our approach achieves more energy savings regardless of the uncertainties of workload dynamics. We believe this work contributes to a solid foundation of the design and implementation of energy-aware DBMSs.

1 Introduction

The fast growth of energy-related research in data management systems is driven by the fact that *data centers are energy starving*. For example, data centers consumed 61 billion kWh of energy in US in year 2006 [33], and about 31GW of power around the world in 2011 [5]. The increasing maintenance cost of data centers is, to a great extent, caused by this rapid growth of energy consumption positively correlated with the growth of transactions and data volume in database services. In software system design, the maximum acceptable performance loss of data processing due to other processing requirements is called *performance budget*. Energy efficiency has now become the new key challenge in general purpose database system design. In a typical database service environment, resource utilization is relatively low in most of the time, as demonstrated by our experiments with a cluster of servers running one month of workload generated from SDSS – a well-known scientific database system [26]. In most (90%) of the servers' operating time, the CPU utilization is between 10% and 50 % (Figure 1). Similar results are reported in [1]: the average utilization of 5,000 servers is within the range of 20%-30% according to data collected in a 6-month period. The underutilization of resources translates into abundant opportunities for energy savings in database systems.

Manipulating power modes of many modern hardware provide great opportunities for energy reduction in software systems. In general, hardware running on low-power mode(s) consumes less power with a performance penalty. For example, Dynamic Voltage and Frequency Scaling (DVFS) is a popular mechanism in many modern CPUs to provide different power/performance modes – a cubic reduction in power density relative to performance loss is reported [25].

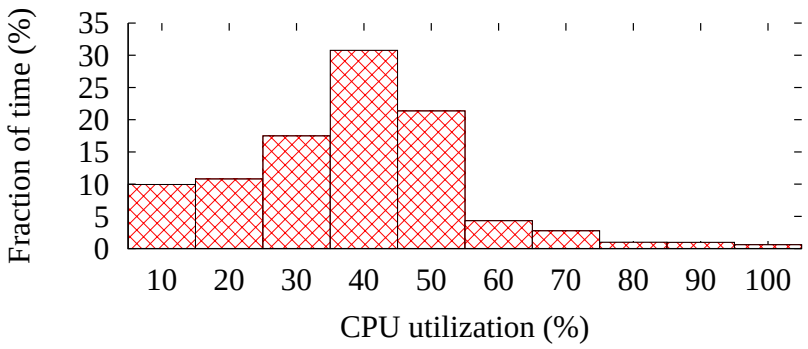

Figure 1: Peak CPU utilization of 10 servers running one month worth of SDSS traces. Each histogram bucket stands for the percentage of time when system runs under a certain utilization range

Various studies [1, 18] suggest that servers running on low-power modes show higher energy efficiency since the gain on power reduction overshadows the loss of performance. Recent reports have also identified the benefits of controlling hardware power modes in database systems. Poess et al. [23] examine the power breakdown of major components (CPU, Memory, Disk) of database servers in their active and idle modes, respectively. The results show that great potential of energy saving exists for DBMS operations. Lang et al. [15] suggests lowering CPU frequency to achieve better energy efficiency in DBMSs – they report a 46% power reduction with little performance degradation observed. In practice, the transition between different DVFS levels of a CPU can be done via an OS handle with a very small overhead [35].

1.1 Problem Statement

In this paper, we study the problem of saving energy in DBMSs via online adjustment of DVFS levels of CPUs, as a part of our efforts to design and implement *power-aware DBMSs* [40]. The design of such systems aims at minimizing power consumption under reasonable degradation of performance. Specifically, we tackle the following optimization problem:

Given a performance budget in terms of system throughput, how can we minimize CPU power consumption?

Here we assume that the database service provider (via the DBA) sets an acceptable throughput level, and failure to meet this budget will cause significant loss of service utility. In this paper, we use throughput as the only performance constraint since it is an important metric widely adopted in industry. Other performance metrics such as query response time are as important but beyond the scope of this paper. We focus on power control of CPU as it is found to be the main power consumer in database servers [22, 29].[1] Yet we believe the solutions proposed in this paper provide valuable insights for solving similar problems considering other performance constraints and hardware systems.

The main challenge is to provide performance guarantees in a DBMS under unpredictable system dynamics and resource consumption patterns of different database workloads. Well-known in database systems [32], such unpredictability is also verified by the extensive experiments we run on various database workloads generated from TPC benchmarks [28] and large-scale scientific database traces [26]. As a result, if the DVFS level is pre-determined based on static estimations of the workload's resource consumption, it can be easily set too low (causing a violation of performance budget) or too high (leading to less energy saving). Furthermore, we also found that the runtime fluctuations of DBMS resource use (e.g., I/O demand) can even change the effects of DVFS on throughput (see Section 2 for details). Therefore, our DVFS scheduler shall not only compute a valid processor frequency (given a desired throughput) but also adapt itself to the ever-changing system and workload dynamics.

[1] Hard disks is another type of hardware that waste a lot of energy in typical database servers. However, energy conservation in hard disks involves completely different mechanisms and challenges.

1.2 Our Solution and Contributions

We formulate the above optimization problem into a feedback control loop that periodically adjusts the DVFS level such that system throughput traces a target value even under fluctuations of workload and system characteristics. In sharp contrast to traditional methods in database tuning that depends on extensive experiments and manual tuning [3, 37], we adapt control-theoretic methods to achieve rigorous controller (i.e., DVFS scheduler) design based on extensive system modeling efforts. Specifically, such formal methods from control theory, when facing workload and system variations, provide theoretical guarantees on control performance such as stability, accuracy, and fast response, thus giving more power savings.

The application of control theory in our problem, however, is not trivial because it involves in-depth study of the target system and domain-specific controller design strategies. Thus, this paper focuses on the unique challenges in the design and implementation of the control loop as the solution to our problem in database systems. First, we need to understand the nature of the DBMS's throughput in response to changes of DVFS. In particular, we develop a quantitative model that describes the relationship between DVFS level and throughput via empirical and analytical *system identification* methods. Second, our control framework should be robust, meaning that its performance should not be affected by patterns of system/workload variations. This is achieved by our proportional-integral (PI) controller [9] design with provable accuracy and stability under disturbances. Third, as certain workload characteristics is found to play an important role in the whole control loop, we need to develop a query classification scheme at query compile time. Finally, there are also challenges in implementing various components in the control architecture. The technical contributions of this paper are summarized as follows.

- we design a unified control architecture to maintain system throughput of DBMS while minimizing total power consumption. The control-theoretic design guarantees system stability and maximal power saving;
- we implement the control architecture within PostgreSQL and evaluate it with extensive workloads generated from various TPC and scientific database benchmarks. Our experiments show that, without violating performance budget, our solution leads to much less power consumption than open-loop and OS-level DVFS control solutions;
- we study the complex relationships among workload statistics, throughput, DVFS level, and system power via extensive experiments. The observation reveals some essential facts that serve as the foundation of control architecture design;
- we design and implement a query classifier based on fuzzy set theory. Such classifier provides important information about the workload and such information plays a key role in achieving effective throughput control. The fuzzy-logic-based design also provides new insights to the classic problem of query clustering.

1.3 Paper Organization

The rest of the paper is organized as follows: we first discuss observations related to system modeling in Section 2; Section 3 introduces the control architecture as well as its components; Sections 4 and 5 present the design and analysis of the workload classifier and DVFS controller, respectively; Section 6 is dedicated to empirical evaluation of the proposed control strategy; Section 7 highlights our endeavour by comparing it with related work; finally, Section 8 concludes the paper.

2 System Characterization

In this section, we report the results of empirical studies we perform for better understanding of the aforementioned optimization problem. Through such experiments, we identify the main factors that play a role in the problem and (qualitative) relationships among such factors, thus pave the way to a concrete quantitative problem statement and system model required for control-theoretic reasoning. In our experiments, we feed the database with workloads generated from the TPC-H benchmark and SDSS database trace. The total size of our database (data is also extracted from TPC-H and SDSS) is 2TB. One thing to point out is that, although both TPC-H and SDSS are OLAP benchmarks, we create many queries that run very fast to simulate OLTP behaviour.[2] Here we skip more details of our experimental

[2] Specifically, we modify the selection predicates of the original TPC-H and SDSS queries such that they only touch a small portion of the database tables.

platform (those can be found in Section 6.1) and start our discussions on the results. Before doing that, let us first introduce the notations used throughout this paper by listing them in Table 1.

Table 1: Notations and symbols.

Symbol	Definition	Z-domain
d_{CPU}	demanding CPU resource	-
$d_{I/O}$	demanding I/O resource	-
u_{CPU}	average run-time CPU utilization	-
$u_{I/O}$	average run-time I/O resource	-
R_j	The j_{th} fuzzy rule	-
p_j	implication of fuzzy rule R_j	-
t_j	membership confidence	-
α	I/O utilization threshold	-
M, N	membership function parameters	-
i	control period index	-
T	control period	-
R_s	performance set point	-
λ	ratio of I/O intensive query	-
f	relative CPU frequency	$F(z)$
r	relative throughput	$R(z)$
A, B	system(DBMS) model coefficients	-
k_I, k_P	controller coefficients	k_I, k_P
$C(z)$	controller transfer function	$C(z)$
$G(z)$	system transfer function	$G(z)$

Note that, in this paper, we focus on the scenario of a *fully loaded* database system, in which the throughput will never be lower than the specified target value due to the lack of user queries.[3]

Figure 2 demonstrates the effects of DVFS levels on throughput and CPU power consumption. What we can immediately see is that the power consumption is positively related to the DVFS level, as we expected. Note that this relationship is very stable – the power readings in Figure 2b are recorded from experiments running under a few different workload intensity and workload types. On the other hand, the system throughput shows similar trend: when the DVFS level is higher, we got higher throughput. However, the response of throughput to DVFS differs in workloads with different ratio of I/O intensive queries. Each line in the Figure 2a stands for a particular workload and the lines have different slopes. Now let us go back to the optimization problem stated earlier. The problem seems trivial in a static environment: for a given throughput bound R_s, we can get the desired DVFS level from Figure 2a, and that DVFS level will be the solution. The reason for that is: the power-to-DVFS relationship shown in Figure 2b is a stable monotone. As a result, if we see a throughput higher than R_s, we unnecessarily waste power.[4] Therefore, to minimize power consumption, our task is to **keep the system throughput at exactly** R_s.

The real challenge in accomplishing the above task, however, is to maintain the desired throughput at all times. Like many complex systems, a database is hardly stable and predictable due to the inevitable errors in system behavior modeling, fluctuations in workload characteristics and system states [32, 37]. A salient problem is: the workload composition (i.e., the value of λ) may change over time and such changes are very unpredictable. In this paper, we will use a fuzzy-rule-based method to model such characteristics of the workload, and as we will see, such models cannot be 100% accurate. Non-workload factors can also show great impact on system behavior. For example, the experimental results shown in Figure 2a are obtained in an ideal environment in which no non-database jobs are scheduled. If the OS introduces tasks that compete with the DBMS for resources, throughput will undoubtedly decrease. A fundamental benefit of control-theoretic approach is that it gives us theoretical confidence on the effectiveness of the solution produced, even under uncertainties mentioned above (such are called *disturbance* in control terminology). A special note here is: any OS-level control mechanism without understanding the runtime workload statistics in DBMS is insufficient to solve our problem. In the remainder of this section, we further scrutinize our experimental results to

[3] We believe the lightly loaded scenario is easy to handle: we can simply turn down the DVFS to the lowest level.

[4] One the other hand, if we let the throughput to be lower than R_s, it is a violation of the performance budget.

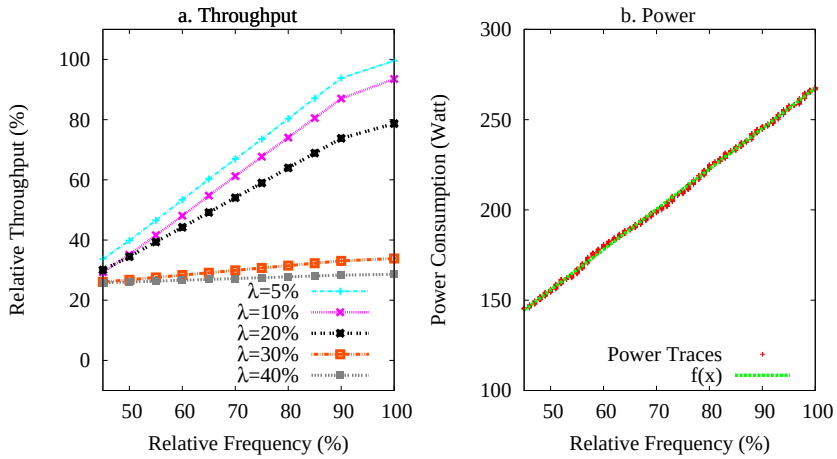

Figure 2: The effects of CPU frequency (i.e., DVFS level) on system throughput (a) and CPU power consumption (b). The five different workloads contain different ratios of I/O-intensive queries. The full (100%) throughput is the one recorded under 5% I/O intensive workload at 100% CPU frequency

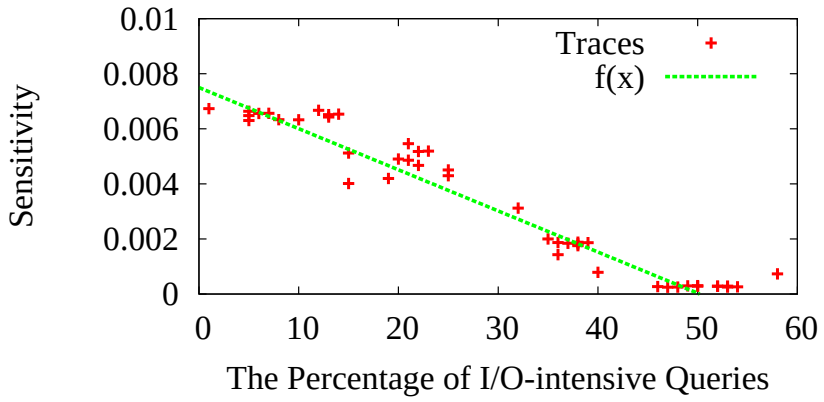

Figure 3: The relationship between workload's DVFS-to-throughput sensitivity and the percentage of I/O-intensive queries in the workloads

gain more insights in modeling the relationship between throughput and DVFS level - such model is the foundation of a successful control framework.

Linear DVFS-to-throughput relationship. An important observation from Figure 2a is: there exists a linear relationship between throughput and CPU frequency under all tested workloads (with different λ values). For all workloads, the goodness-of-fit accuracy metric for a linear model $R^2 = 1 - \frac{\text{variance(linear throughput prediction)}}{\text{variance(real throughput)}}$ is greater than 95%. Therefore, we can use the following linear model as a starting point of our system identification process:

$$r = Af + B \tag{1}$$

Furthermore, the CPU power consumption responds quickly to the changes of throughput (within seconds), and the such responses show no correlation to previous history. This concludes that the throughput and power consumption for any given workload are determined exclusively by the its processing pattern and resource demand, which is also independent from the statistics in any previous periods. These two factors, DVFS levels and workload characteristics, are the dominating factors in determining the performance and power tradeoff.

Role of workload type. Since we found that the portion of I/O-intensive queries in the workload is another key

factor in our problem, there is need to look deeper into the data shown in Figure 2a. In Figure 3, we replot the same experimental data to show how the workload statistics affects the database's response to DVFS levels. Such effects are quantified using a metric we call *sensitivity*, which is specifically defined as the ratio of throughput to CPU frequency. In other words, it is basically the slope of the lines shown in Figure 2a. Clearly, the relationship between such sensitivity and percentage of I/O-intensive queries can be approximated by a linear function (again, with a regression coefficient over 95%). Since the value λ is essential in our throughput control for power-saving purposes, it is necessary to identify each incoming query as either I/O-intensive or otherwise at runtime. The challenges and our solutions in query classification will be discussed in Section 4.

3 Power-Aware Throughput (PAT) Control Architecture

Figure 4 illustrates the control architecture as a unified framework to solve the power optimization problem. The main components of this framework form a feedback control loop, these include: the PI controller (Controller), the system throughput monitor (Plant) and the CPU frequency modulator (Actuator). The runtime goal of this loop is to maintain the actual throughput at a set point R_s. To do that, the control loop is invoked periodically to capture the system throughput and computes the new CPU frequency for the next period of time. The CPU frequency modulator will provide a reasonable DVFS state to set the CPU frequency accordingly. The CPU frequency change will, in turn, affect the throughput of the system in the next period. Specifically, the following steps are invoked at the end of each control period:

1. The throughput monitor measures system throughput $r(i-1)$ in the last period. The *control error* is computed as $\Delta r(i) = R_s - r(i-1)$;
2. The controller receives control error Δr, and the quantity λ from the workload classifier and uses them as inputs to compute the *control signal* $\Delta f(i)$, which is the change of CPU frequency for the next period;
3. The CPU frequency modulator receives the control output Δf, to calculate the new CPU frequency and apply this new frequency to the CPU.

The workload classifier works in a different frequency: whenever a query comes to the database, it fetches resource estimation information of that query from the query optimizer, computes the category the query belongs to, and updates λ.

Note that PAT is not a disruptive framework to the DBMS engine – it only needs to extract workload statistics from the existing DBMS (for the use of workload classifier) and record system throughput, and all other components can be implemented as modules isolated from the current DBMS. Furthermore, both the workload classifier and throughput monitor only read data from the DBMS thus do not interfere with the database operations.

There are many requirements in evaluating the performance of control. A database is a complex system and incoming workload pattern is unpredictable. Whether PAT can notice those changes and settle the system to *steady state* in which the control output is close to the setpoint within a predefined time (namely *settling time*), is an important goal in our control loop design. In control theory, the difference between the output (throughput in our case) and the setpoint is called *steady state error*. In PAT, we aim at a steady state error of zero since that translates into maximal power savings, and the controller in PAT can be designed to achieve that goal with guarantees. Our controller is also designed to meet several other important criteria of accuracy and performance, we will present the technical details related to controller design in Section 5.

3.1 PAT Components

In the remainder of this section, we briefly introduce the implementation of several other components of PAT.

3.1.1 CPU Frequency Modulator

We use Intel's SpeedStep technology to enforce the desired CPU frequency. In particular, under a Linux system, the change of CPU frequency is done by writing the new frequency into a system file. This process carries a very small overhead. An interesting issue is that the Intel Xeon CPU E5645 used in our testbed (as well as many others) supports only four discrete frequency levels. However, the new CPU frequency periodically set by PAT could be

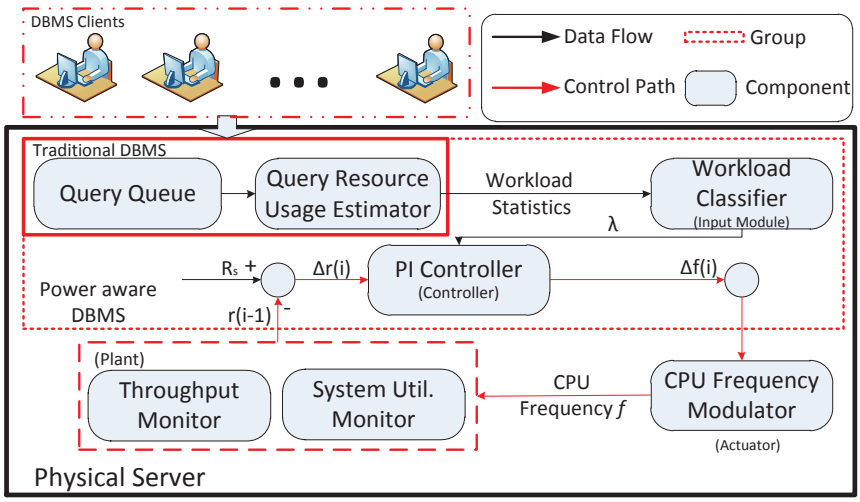

Figure 4: The power-aware throughput control architecture. The names in parentheses are given using control terminology

any (continuous) value within a range. Therefore, the modulator's task is to approximate the desired value using a combination of the supported discrete frequency levels. For example, to ensure a frequency of 2.23 GHz during a control period, the modulator would output pseudo continuous frequency signal using the sequence 2.67, 2, 2, 2.67, 2 and 2 on a timescale smaller than the control period. To do this, we implement a first-order delta-sigma modulator, which is commonly used in analog-to-digital signal conversion [16]. In each period, the controller sends the desired CPU frequency level to the modulator through a named pipe.

3.1.2 Throughput Monitor

The throughput monitor is implemented as a daemon program in the system that collects the number of tuples output from all queries within each period of time. The monitor traces every signal from the executor in PostgreSQL requesting a new tuple and accumulate it's counter C_i in period i if the finished process it is a child process from PostgreSQL that executes a query. The monitor maintains a list of n items which represents the throughput data from the past n periods. The collected data is sent to the controller running in the system.

3.1.3 System Utilization Monitor

The system utilization monitor is a root-authorized shell program that monitors system status (e.g., CPU utilization, system utilization, memory use) in all control periods. The recorded data is formatted into a local table file so that workload classifier obtain current system status.

3.1.4 Query Resource Consumption Estimator

The query resource usage estimator communicates directly with the query optimizer, which provides information about a query's execution plan and the associated costs in terms of number of tuples and demanding pages. In our system, the kernel of the PostgreSQL is hacked to provide us those information via a named pipe whenever a query comes into the system. With those information, the workload classifier could define which category the query belongs to and update the workload parameter λ.

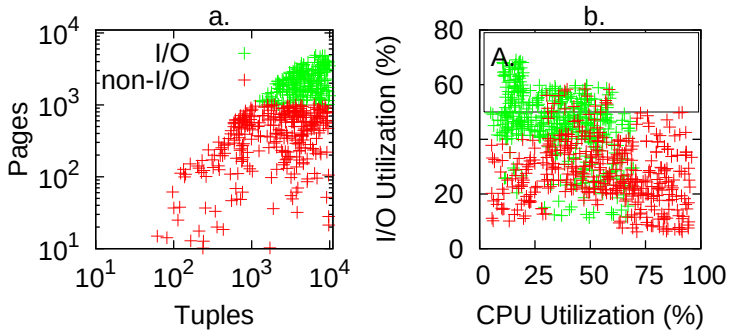

Figure 5: The estimated resource demand map of 500 different queries at compile time (a) and relative resource utilization map measured at runtime (b). Each node in the figure stands for a query. The circles A in (b) stand for the accurate region of I/O-intensive query group

4 Workload Classifier

As we learned from the observations in Section 2, the key factor in the workload statistics used in modeling the relationship of throughput and power is the percentage of I/O intensive queries in the workload, namely λ. Thus, to successfully build the system model to control in a composite workload, the model shall update the value of λ on-the-fly. To solve this problem, we introduce workload classification process into the system.

Workload classification is a process that identifies the characteristics of certain workloads from their resource consumption patterns. In our case, we need to know the resource use pattern of each incoming query so that we can identify whether this query is bounded by CPU or I/O. If a query has a runtime I/O utilization higher than a preset threshold α, it is considered an I/O-intensive query. In our experiments, the cutoff α is set to 50%, as suggested in [12]. However, we also show in Section 6.2.1 that, within a reasonable range, the actual value of α does not affect the effectiveness of our control. Now, the main challenge here is to provide accurate prediction of the I/O utilization at query compile (optimization) phase. For that, given the estimated resource consumption of a query, the workload classifier will have to map it into different (I/O-intensive or not) categories.

4.1 Main Challenge

The resource demand of query processing could be broken down into two parts - tuples processed by CPU d_{CPU} and pages demanded from I/O system $d_{I/O}$. As we know, I/O is normally the performance bottleneck of query processing. Thus, we could classify queries into two groups based on their estimated resource consumption: (1) I/O-intensive queries that requires accessing at least K pages in the storage system, and (2) non-I/O-intensive queries to which all other queries in the workload belong. The quantity K is a predefined threshold based on characteristics of the database server. For example, we could set K to 1,000 pages, which equals the size of the L3 cache in our server. Note that the resource consumption estimation is readily available from the query optimizer. Such a scheme is called *rule-based classification* and has been used in various data mining problems [17].

Figure 5a shows the classification results of 500 queries in our workload using the above rule-based method. Each query is labeled as either I/O-intensive (green dot) or non-I/O intensive (red dot). However, such a method can easily fail since the resource estimation given by the query optimizer can be very different from the actual consumption at runtime. Figure 5b shows the actual resource utilization in processing the same set of queries. We can see that only a fraction of the green queries are truly I/O-intensive ones (i.e., those in cluster A of Figure 5b) About 50% of queries reside in the Intersection region of green query set and red query set estimated in Figure 5a. The above empirical results clearly demonstrate the challenge of obtaining accurate classification, which is required for the prediction of the λ value, and eventually ensure the correctness of our throughput control. Even if we cannot provide a 100% accuracy, we need a classification scheme that reveals the general trend of the database workload.

One way to gain high accuracy of classification is to generate more rules in the classifier. For a static workload, this could be a good choice. To deal with real-world workload that is dynamic, infinite and unpredictable, the number of rules needed to guarantee the classification quality (assuming static rules will work) could be big and to some

extent become unmanageable. Since the conventional rule-based method fails to meet our requirement, we proposed a workload classification approach based on fuzzy set theory. Fuzzy-based method is particularly suitable for efficiently modeling systems with complex behaviors. It is designed to handle unpredictable environment with limited number of rules to reach sufficient accuracy [34, 27]. Our Fuzzy Workload Classifier (FWC) collects workload statistics and creates fuzzy rules to put queries into the a resource consumption category based on the resource demanding patterns.

4.2 Fuzzy Classifier Design

In FWC, Sugeno-type fuzzy rules [27] are generated from the clustered data for modeling database workloads. The input for the FWC is the resource demand vector (provided by query optimizer) of the incoming query and output is the aggregated estimation of runtime resource utilization. For the i–th query, its resource demand vector is denoted as $[d^i_{CPU}, d^i_{I/O}]^T$ and the estimated CPU and I/O utilization as $[u^i_{CPU}, u^i_{I/O}]^T$. The number of fuzzy rules is the same as the number of clusters in the estimated resource demand map [11]. In our case, there are two clusters: I/O-intensive and non-I/O-intensive. The member functions of the fuzzy rules are linear functions generated via rigorous mathematical tools, i.e., parameter fitting using estimated output and the actual utilization vector following a least-square approach. The rule base is constructed as follows:

R_j: IF $[d^i_{CPU}, d^i_{I/O}]^T \in$ cluster X_j, THEN $[u^i_{CPU}, u^i_{I/O}]^T = M_j[d^i_{CPU}, d^i_{I/O}]^T + N_j$

where X_j is the cluster determined by clustering technique, M_j and N_j are parameters from the fuzzy set associated membership functions obtained from the learning process. The symbol $\in$ stands for the distance between the node and the center of cluster X_j. It will be used to compute the implication of the output $[u^i_{CPU}, u^i_{I/O}]^T$ for the calculation of aggregation result. The procedures of workload classification are as follows:

1. *Evaluation of antecedents*: the input resource demand vector $[d_{CPU}, d_{I/O}]^T$ is computed to the appropriate fuzzy rule output $[u^i_{CPU}, u^i_{I/O}]^T$ via the corresponding membership functions M_j and N_j according to the rules above;

2. *Implication calculation of consequents*: implication p_j is performed on each fuzzy rule R_j by calculating how far the point is from the center of each cluster (the distance in the coordinate system) and the confidence t_j that the query belongs to fuzzy rule R_j based on the implication weight over all $\frac{\sum(p_j)-p_j}{\sum(p_j)}$;

3. *Aggregation result of consequents*: the outputs of all the fuzzy rules are aggregated and inversely translated into an average utilization vector $[\sum_j t_j u^i_{CPU}, \sum_j t_j u^i_{I/O}]^T$ from all rules with confidence t_j.

4.3 Use of the fuzzy rules

For better understanding the application of the fuzzy classification results, we present a case study as follows. Suppose a user sends in a query i to the database server when there are 100 other queries running. Let us further assume the λ is 30% and the cluster cutoff α is set to 50%. The query optimizer compiles query i and provides estimated resource demand $[5309, 100]^T$ for the chosen execution plan. Currently we have two rules R_1 and R_2 associated with the green and red clusters (as shown in Figure 5a), respectively. Their membership function parameters M_j and N_j are $\begin{pmatrix} 0.0007, & 0.054 \\ 0.0002, & -0.039 \end{pmatrix}, N_1 = \begin{pmatrix} 0.96 \\ 3.54 \end{pmatrix}$ and $M_2 = \begin{pmatrix} 0.0009, & -0.065 \\ 0.0006, & -0.042 \end{pmatrix}, N_2 = \begin{pmatrix} 2.08 \\ 1.14 \end{pmatrix}$. FWC receives the data from DBMS and fits it into different clusters. When query i fits in R_1, the output of rule R_1 is $[0.727, 0.699]^T$ with implication of 159.46 (the distance between node i and center of green cluster). Similarly, when the query fits in R_2, the consequence of rule R_2 is $[0.357, 0.125]^T$ with implication of 3659.23. Then the confidence of query i's belonging to R_1 and R_2 is $1 - \frac{159.46}{159.46+3659.23} = 95.9\%$ and $1 - \frac{3659.23}{159.46+3659.23} = 4.1\%$, respectively. Based on that, the aggregated output is $[u^i_{CPU}, u^i_{I/O}]^T = [0.712, 0.675]^T$. The estimated I/O utilization 0.675 is larger than threshold α and the query i is identified as an I/O intensive query. Then λ is updated to $\frac{100\times 30\%+1}{100+1} = 30.7\%$.

5 Controller Design

In this section we present the controller design process and system model based on system identification results from Section 2.

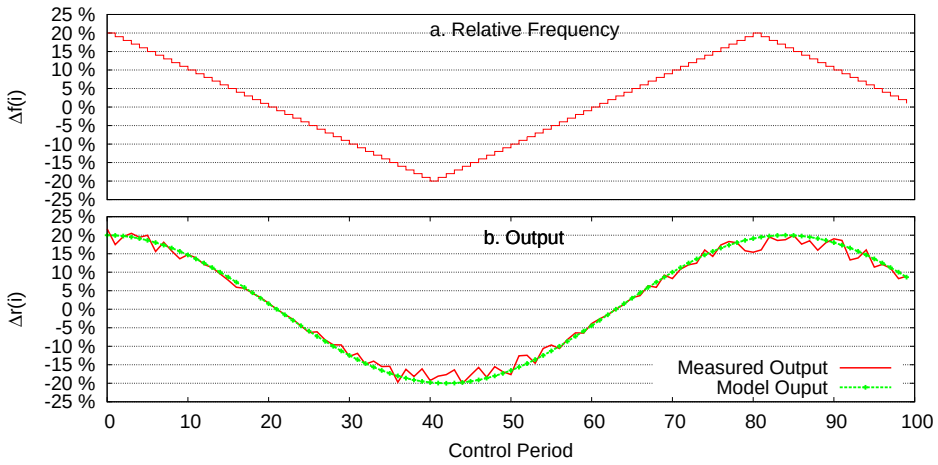

Figure 6: Control model validation

5.1 System Modeling

Building an accurate mathematical model of the system is of great importance to the whole controller design. We are interested in how to connect the active throughput r and the run-time CPU frequency f. In Section 2, we found an approximate linear relationship between active throughput and power consumption which are sampled in a period of time. Let us denote this period as T and the throughput within this period as $r(i)$. Then, given the current relative throughput data $r(i)$, our control goal is to guarantee that the $r(i)$ could be converged to the set point R_s after a finite number of control periods, or so called *settling time*. Note here, for better establishing the model we scale those two values into percentage. Thus, r and f are the relative throughput and frequency, respectively. For example, $f = 100\%$ means that CPU is running at its highest frequency. To avoid possible thresholding issues, we set the minimum CPU frequency to 45%.

Due to the complexity of DBMS systems, we adopt system identification techniques to model the system. The results turn out that the relationship between throughput and frequency is approximately linear, as we observed in Figure 2. Thus, with any workload, the system's power and throughput relationship can be defined as:

$$r(i) = \lambda A f(i) + B \tag{2}$$

where A is the effect of DVFS level on throughput; i stands for the i^{th} period, and λ is the ratio of I/O-intensive queries in the current workload. Equation (2) could be viewed in z-transform as:

$$R(z) = \lambda A F(z) \tag{3}$$

where $R(z), F(z)$ are the z-transform of signal $r(i), f(i)$, respectively. Thus, the system transfer function of the DBMS regarding to frequency change in Figure 2 is:

$$G(z) = \frac{R(z)}{F(z)} = \lambda A \tag{4}$$

In Figure 6a, to verify the accuracy of the model, we generate a different sequence of control inputs as a sine function of $\Delta f(i)$ and feed them into the system model. Then we collect modeled and measured output of relative throughput $\Delta r(i)$, as shown in Figure 6b. The comparison result of the measurement and prediction demonstrates that our model is sufficiently close to the real system output with goodness-of-fit metric $R^2 = 0.93377$.

5.2 Controller Design

The goal of the controller design is to meet the following criteria:

Stability: the throughput shall settle into a bounded range in response to a bounded reference input.

Zero Steady State Error: when the system enter the steady state (any property of the system is unchanging in time), the throughput shall settle to the set point, which is the performance budget in our case.

Short Settling Time: the system shall settle to the set point by the specified deadline.

Based on standard control theory, we design a *Proportional-Integral* (PI) control that has been widely adopted in industry control systems. We select PI controller because proportional control alone cannot totally remove the steady state error while the integral controller combined with the proportional controller can [6, 9]. Thus, an important advantage of PI control is that it can provide robust control performance despite considerable modeling error and input/output disturbances. The PI controller has the following form in the discrete time domain:

$$\Delta f(i) = k_P \Delta r(i) + k_I \sum_{1}^{i} (\Delta r(j)) \tag{5}$$

where $\Delta r(i) = R_s - r(i)$ is the control error at i^{th} period, and the control output signal $\Delta f(i)$ is the frequency offset to manipulate the throughput to the set point R_s. k_I and k_P are control parameters to the integral and proportional part, respectively. Those parameters can be analytically chosen to guarantee the system stability and zero steady-state error [6, 9]. From Eq. (5), we have the controller transform function in the z-transform as:

$$C(z) = \frac{z(k_I + k_P) - k_P}{z - 1} \tag{6}$$

The throughput controller is implemented to adjust CPU frequency based on the detected data from the throughput monitor. Based on the system identification, the parameters in Eq. (2) has the parameters $A = 0.514899$ and $B = 11.479$ while λ can be computed at runtime by FWC. Overall, the standard closed loop transfer function $\mathbf{F}(z)$ is

$$\mathbf{F}(z) = \frac{C(z)G(z)}{1 + C(z)G(z)} = \frac{\lambda A k_p (z-1) + \lambda A k_I z}{(1 + \lambda A(k_I + k_P))z - (\lambda A k_P + 1)} \tag{7}$$

The controller uses the Root-Locus method [6, 9] to guarantee stability and zero steady-state error. The poles of our closed-loop system are $-0.26 \pm 0.8i$. As both poles are inside the unit circle, the closed-loop system in our experiments is stable [6]. The final controller is Eq. (5) with control parameters $k_I = -0.5$ and $k_P = 1.06$. The details of pole placement and computation of control parameters can be found in later case study.

In addition to stability, we also perform the following analysis on the other design goals of our controller.

Settling Time. The settling time is the time elapsed starting from when the control system is out of steady state because of some system disturbances until the time when the control system has reentered the steady state. It is a user defined deadline requirement to ensure the system is within control. In our experiment, we require settling time to be 1 second because that is the frequency of reading power data from power meter.

Commonly defined in modern control theory, the control system settles when the output $\Delta r(i)$ is within the 5% range around the desired set point R_s. Thus, the required number of sample periods k (the time used to sample the output different for control system settle back to the steady state, the same length as control period) can be calculated as:

$$k \geq -\frac{ln 0.05}{ln|p|} \tag{8}$$

where p is the closed system pole $\frac{(\lambda A k_P + 1)}{1 + \lambda A(k_I + k_P)}$ as in Eq. (7). We could use it to derive a range of value in p when it is in the unit circle. Then the system is guaranteed to settle in the given deadline.

Overshoot Analysis. In control theory, an overshoot is when a signal or function exceeds its set point [20]. In our problem, it is the percentage of throughput exceeding the set point R_s. When an overshoot happened, controller will tune the DVFS to oscillate back to steady state. The decay ratio of this oscillation is called damping ratio ζ.

ζ can be obtained from the following equation according to [6]:

$$T_s = -\frac{ln(0.05)}{\zeta f} \tag{9}$$

where T_s is the settling time and f is the frequency of our control system. By calculation, the ζ is $0.27999 \approx 0.28$,

Also, ζ can be obtained from the maximum overshoot happened in the system:

$$PO = 100\% \times e^{\frac{-\zeta\pi}{\sqrt{1-\zeta^2}}} \tag{10}$$

where PO stands for percentage of overshoot. Deploying the value of $\zeta = 0.28$, we get the maximum overshoot percentage as 43%, which means based on a system with 1 second settling time, the control system could tolerate over 40% disturbance.

Selection of control period T. Also, we could modify Eq. (8) to consider how many periods are required for the throughput to settle within 100 tuples in the throughput when our experienced maximum throughput is over 2000 tuple/s:

$$k \geq -\frac{ln\frac{100}{2000}}{ln|p|} \tag{11}$$

Eq. (11) uses 100 out of 2000 tuple/s to calculate the minimum percentage of the set point to converge. Using the minimum pole value that we could get, we calculate a conservative value of k to be at least 6.23 which means the throughput will settle in 7 periods. In this way, we could obtain the upper bound of control period by dividing 1 second by 7 periods, which leads to 142.85 ms. The DVFS latency is about 10ms. Thus, it is safe to set a conservative control period T at 125ms.

Case Study of Controller Design Based on Pole Placement. In this case study, we set the desired convergence rate to ten sampling periods. This means the system, in response to dynamics, would converge to $e^{-4/k} = 67\%$ of the desired value in 10 control periods. We set the system damping r to 0.67, and set the desired closed-loop poles to be on the real axis, at $\theta = 0.6$. Thus, the desired closed loop characteristic equation (CLCE) is:

$$(z - re^{-j\theta})(z - re^{j\theta}) = z^2 - z + 0.36 \tag{12}$$

The controller transform function C(z) has one pole and the representative form is shown in Equ. 7. Then, the overall function Therefore, the closed-loop transfer function (CLTF) becomes:

$$\frac{C(z)G(z)}{1 + C(z)G(z)} = \frac{\lambda A k_p(z-1) + \lambda A k_I z}{(z + \lambda A(k_I + k_P))z - (\lambda A k_P + 1)} \tag{13}$$

The actual closed-loop characteristic equation is

$$(1 + \lambda A(k_I + k_P))z - (\lambda A k_P + 1) = 0 \tag{14}$$

Also, since we need the output to be exactly the same as the set point, the following requirement is also needed to be fulfilled:

$$\frac{\lambda A k_p(z-1) + \lambda A k_I z}{(z + \lambda A(k_I + k_P))z - (\lambda A k_P + 1)} \mid_{z=1} = 1 \tag{15}$$

We could obtain the control parameters using this method. In summary, the above design results in a closed loop system that enable the system function Equ.2 to produce the control signal for system control.

6 Performance Evaluation

In this section, we present the results of extensive experiments for studying the effectiveness of PAT.

6.1 Experimental Setup

Our testbed contains one database server, which runs PostgreSQL (version 8.3.18), and a client that generates SQL requests to server. Both machines run Ubuntu 11.10 with Linux kernel 3.0.0. The server is a DELL R710 equipped with Intel Xeon CPU E5645. The processors support frequency from 1.21Ghz to 2.4Ghz. The server and client are connected directly by Ethernet. The PostgreSQL [24] kernel is hacked to extract the runtime internal information and is made to fully support our customization. The client-side workload generator randomly draws tasks from a pool of 2,000 queries drained from the TPC-H benchmark [28] and SDSS databases [26]. The power consumption of the server is measured by a WattsUpPro power meter (with $\pm 1.5\%$ error) under a fixed sampling frequency of 1 Hz.

We implement three baselines for performance comparison with PAT. The first one, named OPEN-DVFS, is an *open-loop* solution for power efficiency that sets DVFS level to a fixed value based on the throughput set point. The second one, named Adhoc, is a simple ad hoc feedback control solution that tunes the frequency up/down based on the measured throughput difference. It determines the control signal following a predefined roadmap: when the throughput is higher (lower) than the set point, the frequency is set to be decreased (increased) by a fixed step size. The last one, named CTRL, contains every component of PAT except the workload classifier. Overall, the purpose of the experiments is to demonstrate that PAT outperforms other control baselines in terms of accuracy, flexibility and energy efficiency.

6.2 Experimental Results

6.2.1 Workload Classification

As shown in Section 2, the workload classifier is one key component in PAT design. To verify the performance of FWC, the database is fed with workloads containing a mixture of TPC-H queries. Each query's actual membership in a cluster is given by running them one by one and recording their resource consumption at runtime. We then create a training workload and multiple test workloads with the same size. We test two different categorization approaches: the I/O and non-I/O intensive classification as we used in PAT design, and the CPU and non-CPU intensive classification as a supplementary experiment.

The first experiment evaluates the classification performance of FWC in clustering CPU and non-CPU intensive queries. We run queries in the training set, collect the results, and keep training until the average mean error is below 10%. Then, we build the fuzzy set rules to identify CPU intensive query and those that are not. To verify that, anther two sets (1,000 queries each) of CPU and non-CPU intensive queries are created as the test workload. For CPU intensive workloads, FWC puts 91.4% of the queries into the right category and puts the other 8.6% into the wrong category. The errors probably come from the wrong weight of fuzzy probability assigned to each resource attribute. When the system is fed by a set of non-CPU intensive queries, FWC puts 99.7% of the queries into the right category while marking only three queries wrong.

The second experiment follows the same routine of the previous experiment on workload classification (Section 4). The classifier gives a 7.9% error in classifying I/O intensive queries. When facing a non-I/O intensive workload, the error is reduced by half – classification accuracy reaches 95.9%.

6.2.2 Comparison to OPEN-DVFS

The purpose of comparison between OPEN-DVFS and PAT is to show that a control scheme that sets DVFS levels statically based on the set point (R_s) is not a viable solution. We conduct this comparison experiment using a set point as $R_s = 65\%$ so that we set the CPU frequency as (1.76GHz) using Eq. (1). This setup is marked as OPEN-DVFS-2 in Figure 7. Also, we have another setup with the maximum CPU frequency marked as OPEN-DVFS-1. The static setup of DVFS cannot handle any disturbance from the workload and system. As observed in Figure 7b, the power curves of OPEN-DVFS-1/2 are similar with the workload throughput patterns in Figure 7a from period 1 to 20 and 40 to 100. At period 20, a Fibonacci program has been injected into the system that occupies lots of CPU resource, until period 40. When the CPU resource is demanded by the competing processes, the throughput of DBMS drops down fast in the two OPEN-DVFS setups, as illustrated in Figure 7a. The problem of the two OPEN-DVFS schemes is that they either waste a lot of energy (as in OPEN-DVFS-1) or suffers from significant loss of performance (as in OPEN-DVFS-2). It is very difficult to find an optimal DVFS value. In fact, we are not even sure there is an appropriate DVFS value as the OPEN-DVFS scheme does not respond to runtime dynamics at all. Thus, the static setup cannot provide guarantee of the throughput performance of DBMS nor ensure power saving when facing system disturbance. DVFS is only a technique that helps scaling the frequency in order to save some energy with possible performance hazard in software level. In our case, the victim is often the throughput performance of DBMS.

On the other hand, PAT behaves as it is designed with the given set point. It maintains the throughput around the set point to meet the performance budget while tuning down frequency of processors to preserve power when the throughput is too high. In period 1 to 20 and 40 to 100, the throughput fits the set point line in Figure 7a perfectly. The control step size varies when the workload pattern changes. As demonstrated in Figure 7b, the tuned step is reflected in the up-and-down power curve, which is against the workload pattern instead of following the pattern as OPEN-DVFS does. Also, when facing system disturbance from period 20 to period 40, the noise does hurt the throughput performance at the beginning when DBMS is under controlled by PAT. However, the system quickly settles back to

Figure 7: Real-time throughput (a) and power (b) readings of the system controlled by PAT and OPEN-DVFS with two different DVFS settings

the steady state within about 6 periods (Figure 7a). Also, to maintain the throughput performance, the processor is tuned up to handle the injected jobs while the performance of DBMS is persevered.

Comparing those two methods, PAT beats OPEN-DVFS when handling dynamics of DBMS and facing system disturbance. Within the performance budget, OPEN-DVFS-1 consumes 10% more power than PAT. OPEN-DVFS-2 saves 7% comparing with PAT with a sacrifice of more than 20% more performance loss. In all, OPEN-DVFS cannot handle a common case which is easily solved by PAT, shown in Figure 7. It is thus not a recommended solution for power-aware performance control in DBMS.

6.2.3 Comparison to Adhoc

In this experiment, an ad-hoc controller is used to replace the PI controller in PAT with all other parts untouched. We use this baseline to demonstrate that a feedback controller not designed with control-theoretic analysis lacks the desired properties such as zero steady state error. Figure 8 shows the average throughput of PAT and AdHoc tested under many set points ranging from 45% to 100%. Clearly, PAT can precisely meet the set point (reaching a 99% accuracy) while Adhoc fails to achieve a low steady state error – on average the reached throughput is higher than the set point by a margin of 8%.

Another comparison criterion of the two control methods is the energy saving. The experiment adopts both control methods on the same workload to meet the same set point. The difference of this experiment comparing with others in this paper is that the power data is read every 125ms (the same time length as control period) using another power meter. The collected data is shown in Figure 9. The system power is the almost the same before period 25 in both Figure 9a and Figure 9b. At period 25, those two control methods are deployed into the system. For Adhoc in Figure 9a, it keeps tuning the frequency back and forth to meet the set point. However, since the step size of Adhoc control is fixed and it is not started at a lucky point that fits the set point well, large amount of energy are wasted during the unnecessary tune-up. On the other hand, PAT could adjust its tuning step to settle into the steady state in the given time. Thus, it won't waste too much time in tuning up to meet the requirement. During the long experiment in Figure 9, we have experienced about 40% more energy savings from PAT comparing with the Adhoc control. Thus, the Adhoc is not the optimal control technique for us to reach energy efficiency.

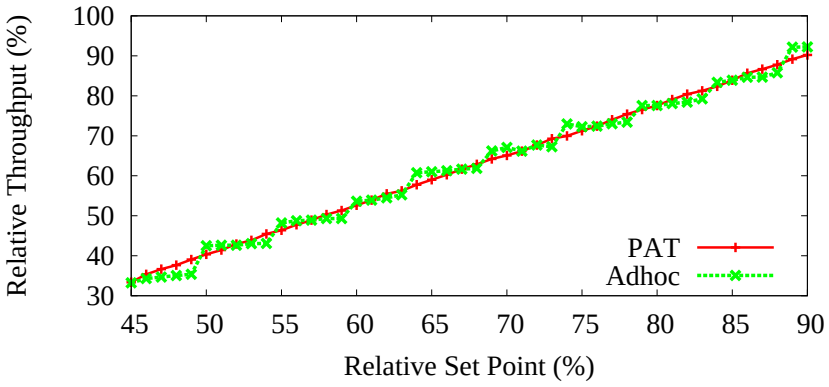

Figure 8: Steady state error recorded from running PAT and Adhoc. Each point represents a 5-minute run

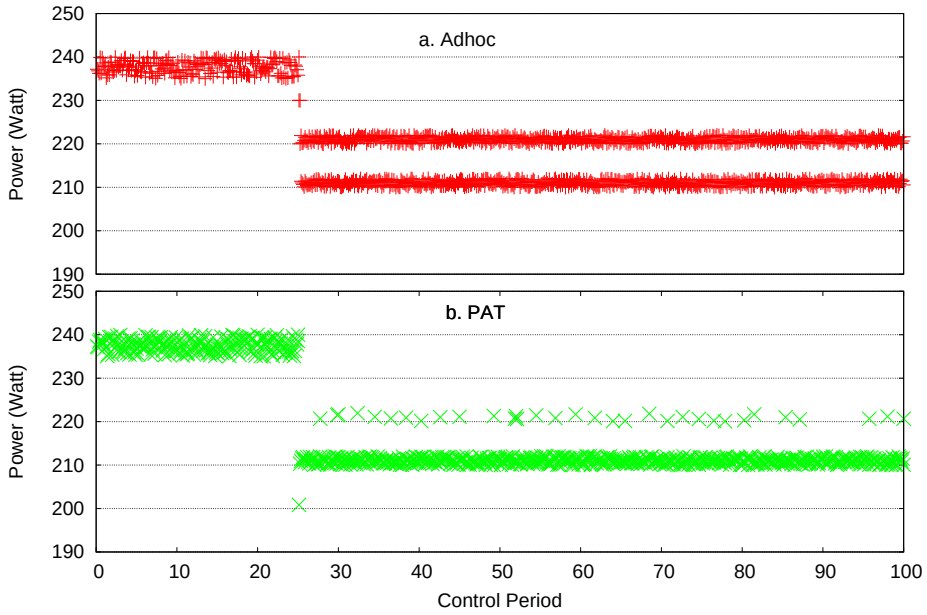

Figure 9: The detected power data of Adhoc control and PAT. Both control start at period 25.

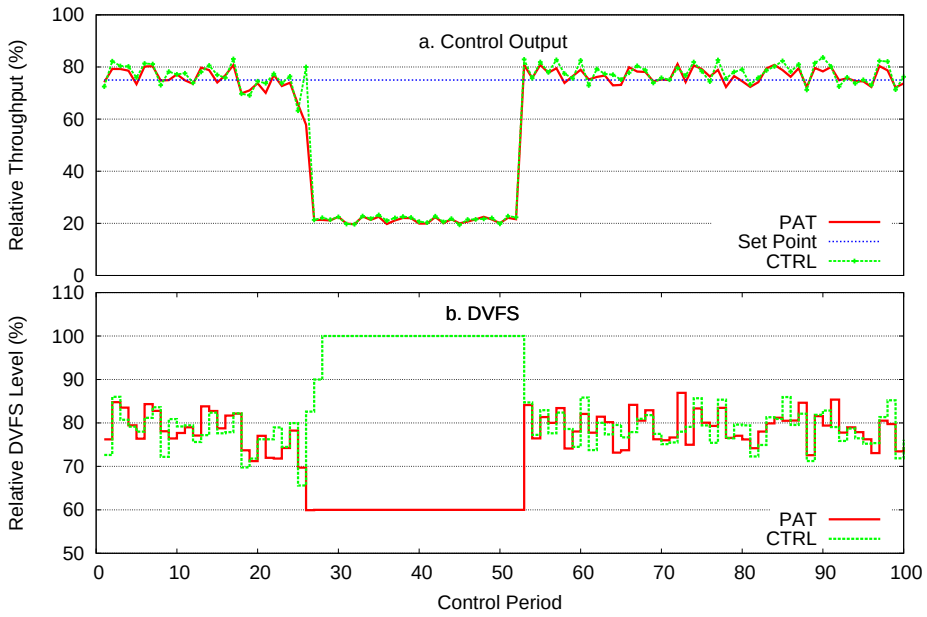

Figure 10: Throughput and power performance comparison of PAT and CTRL

6.2.4 Comparison to CTRL

In this experiment, we compare PAT with CTRL to show the effects of the workload classifier. Figure 10 summarizes the performance/power comparison between these two controllers. PAT reacted faster because it has the advantage of knowing the features of the incoming workloads to the system and adjusts itself in a shorter period of time than that of CTRL. Another difference of PAT and CTRL is their design purpose. PAT is designed to achieve maximum energy efficiency while CTRL is mainly focused on controling the throughput to be higher than or equal to the set point. One scenario that represents this design difference can be seen between control period 26 to 53 in Figure 10. At period 26, plenty of I/O intensive queries are sent into the system on purpose. As a result, the running workload becomes insensitive to the actuator (CPU frequency modulator). In order to meet the set point, CTRL tunes the CPU frequency to its maximal value while PAT is aware of what happened and gradually turned down the DVFS level to save power during this period. The energy consumption difference between PAT and CTRL is over 3.5kJ within these 27 control periods. Even without considering the above dramatic scenario, the average power consumption of PAT is only 88.6% of that in CTRL (180.8 watts vs. 203.3 Watts).

6.2.5 Energy Efficiency Considerations

All previous discussions and experiments focused on power consumption. In this experiment, we study the overall energy efficiency of PAT and other three control methods in the DBMS environment. Here energy efficiency is defined as $\frac{throughput}{power}$. We have four sets of workloads to simulate different runtime environment. SDSS200 is a general case with mostly non-I/O intensive queries and the queries have the same resource consumption pattern. Obviously, the OPEN-DVFS (named 'DVFS' in Figure 11) has the worst energy efficiency because of the large performance penalty that this method causes. The rest three methods including PAT perform relatively the same with less than 10% difference. For the second workload SDSS400, there are a lot of overshoots (utilization peaks) happening during the processing. Even though the Adhoc method is optimized with small step size, it cannot effectively set the performance within the budget in a relative short period of time. Thus, the energy efficiency of Adhoc drops down in this workload. In the third case of TPCH100, which contains many I/O intensive queries, the performance of PAT is slightly lower than that of SDSS200 because the workload become insensitive to the hardware scale-down technique. However, PAT still beats the other three in energy efficiency. In the last case (TPCH200), with the same set point as TPCH100, because the number of queries in the set are doubled, the DVFS is ineffective to affect this workload, the CTRL's energy efficiency quickly drops down because it is trying to pull up the DVFS level to meet the set point, similar at period 26 in Figure 10. Surprisingly, since the throughput of all other methods drop down, the energy efficiency of

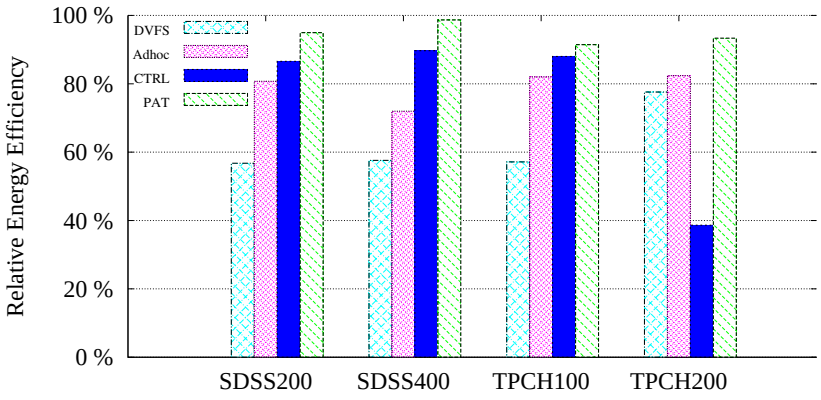

Figure 11: Energy efficiency of PAT control over other methods. Each workload is labeled with its content query source (TPCH or SDSS) and number of queries inside

OPEN-DVFS rises almost to the same level as Adhoc control. In all, we demonstrate in this experiment that PAT provides the best energy efficiency, thus reaches its design goal.

We also conduct experiments with different set points to check out the largest possible energy savings in those situations using PAT. We have five set points in this experiment with a 10% interval between two consecutive ones. Figure 12 demonstrates an acutely performance gap between each level. Note that, when the R_s is set to 100%, it is the original DBMS system without any control.

Based on those collected data, we did a small pivot analysis on the energy data. Examining energy saving data within this 100 periods shown in Figure 12, theoretically, the system consumes over 6000 Kj/H in a non controlled environment. When the controller is enabled with a 80% set point, the system's average throughput loses about 7.53% while energy saving rises to 23%. Gradually decreasing the set point to 60%, the system maintains 65.7% of the original throughput with 44% energy savings. According to our motivating example in Figure 1, in most cases, our servers run in a relative low CPU utilization. This technique introduced in PAT could help us gain great power and energy savings in the real environment. In general, our controller tends to save energy from two perspectives: (1) based on user-defined threshold, the controller reduces system power consumption by using DVFS technique. (2) the controller changes the system's capacity pattern so that it could eliminate the idle power waste in either CPU wait, I/O wait or both in the computing. Those experiments verify that our control model could save a great amount of energy based on user defined performance constraints. For the idle part of energy consumption from computing, the marginal energy cost when the limited workload finishes early and stays idle, which is often the case in query processing of web service and data storage.

6.2.6 The Effects of α

In our previous case study in Section 4, we understand the role of the FWC in the control architecture. An important parameter in the processing of FWC is the set of threshold α. In this experiment we would like to study the impact of this threshold. Whether we can set it to a static number and how much it will affect the system are the questions that we are looking for answers. In the experiment, we define two threshold learned from study [12]. They are 50% and 65%, respectively. The set point of controller is 90% and the system is fed with the same composite random workload. The result is shown in Figure 13.

The two curves nearly match each other perfectly during the 100 control periods. It is not a surprise that the controller could handle the error from workload classifier. It is designed to tolerate disturbance from system dynamics (maximum 40% overshoot in theory). If we take 50% threshold as the correct optimal number, then the 65% threshold setting could bring an average 15% error in value of λ. The feedback control will take it as a positive overshoot situation and update the relative control parameter to ensure the stability of the system. Thus, in this experiment, we show that

(1) Even though our workload classifier is accurate, the control process could handle at least 15% error from this stage;

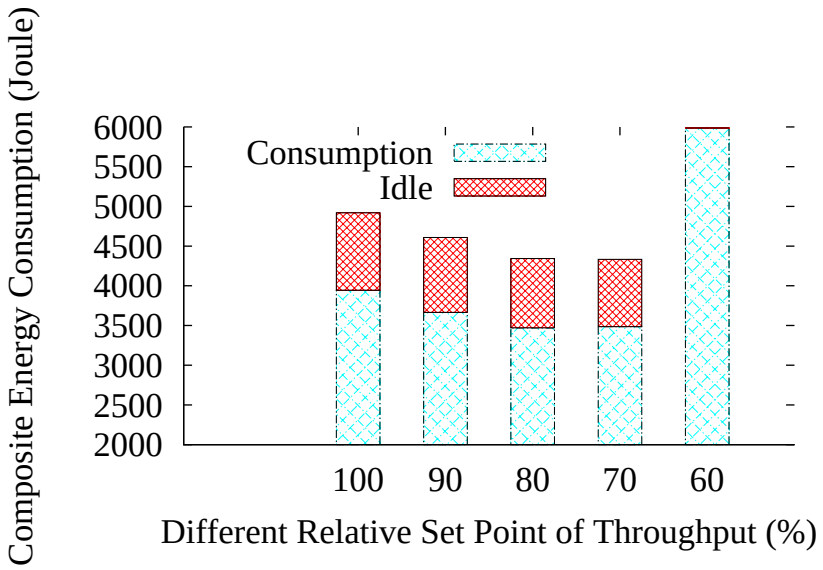

Figure 12: Energy consumption with multiple set points of throughput

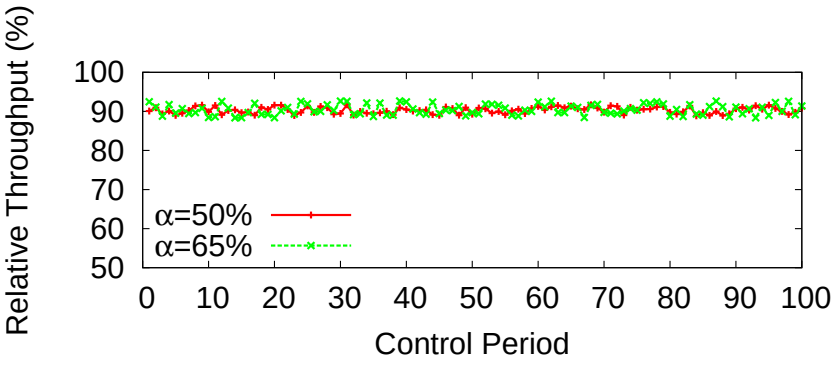

Figure 13: The performance of throughput control under two different identification thresholds using the same workload. The goodness-of-fit metric of the two curve here is 89%

(2) The value of threshold α is important, but we could set it to an experience arbitrary number.

7 Related Work

Reduction of energy consumption has become an active topic in the DBMS community. Harizopoulos et al. [8] and Graefe [7] introduce a new paradigm of DBMS design concerning energy efficiency. In our previous work [40], we found that there exists energy-efficient query plans in DBMS. Another work by Harizopoulos [30] suggests that most energy efficient plans come from DBMS running in the active low power mode. Poess and Nambiar [23] examines multiple storage components in the system for energy saving potentials. Lang et al. [14] claim that it is worthwhile to scale hardware performance to control in DBMS's query processing in the distributed environment. In contrast to their work, we argue that applying hardware scaling technique to DBMS design for energy-saving purposes is not a trivial task and propose a systematic solution that relies on rigorous control theory. As compared to heuristics-based strategies, our solution provides analytical assurance of control accuracy and system stability.

Application of mathematical control theory has been conducted in several topics in the DBMS area. Tu et al. [31] introduce this technique to handle load shedding in data stream systems by using a classic P feedback controller that successfully avoids noticeable streaming tuple delays with lower data loss. Kang et al. [13] creates chronos by applying a similar feedback model in controlling number of transactions to a baseline. Our paper, unlike those two, is the first paper targeting at power savings from preserving performance budget in database systems. It is inspired by the fact that most database servers are running in relatively low utilization and keep them running in this active low mode could help us gain great savings in energy, realize energy proportionality and avoid peak power spikes during the processing.

Recently, feedback control theory has been successfully applied to energy efficient control for data center servers in the system and hardware implementations [4, 35, 21]. Existing solutions of power and performance control for enterprise servers attempt to tackle the problem in two separate ways. Performance-oriented control solutions focus on altering power to meet the system-level performance budget while reducing power consumption in a best effort manner [19]. However, those solutions do not have any explicit internal information from software, such as DBMS. As a result, there would be possible undesired performance degradation. In the other way, power-oriented control solutions treat power as the first-class control target and maximize the performance within the power budget[4, 35, 38]. In DBMS, its throughput (number of tuples processed by second) could not be maximized by control in system/hardware level since the resource are evenly distributed (Round Robin scheduling in Linux). Thus, we need to build the control architecture that includes the software level.

Also, control theory has been adopted to manage many other application-level performance control problems. For example, Xu et al. [39] use predictive control for dynamically optimizing the resource allocation in a enterprise data center. Padala et al. [21] propose a two-layered architecture to control resource utilization and response time in a virtual computing environment. Their exploration results gain us a lot of insight from how operating system treat software (as a black box) in resource allocation and affecting their throughput.

Some prior work have been proposed to use DVFS/DVS/DFS to control power and performance. Wang et al. [36] publish their work in a VM cluster with a two-layer control for power savings using DVFS. Nevertheless, Horvath et al. [10] use DFS to control the end-to-end delay in multi-tier web servers. Zhu and Mueller [41] develop a scheduling algorithm for feedback control based on DVS. Bertini et al. [2] deploy dynamic DVS configuration to optimize power usage in the web servers.

Although the above work all use control theory (and even DVFS technique) to manage system performance while reducing power consumption, they cannot be directly applied to our work – the exact control problem and computing environment are different and, as shown in Section 5, controller design involves considerable amount of work in system modeling and the choice of control techniques differs under different problem statement and target environment.

8 Conclusions and Future Work

The contradictory requirements of high performance and low energy consumption have attracted a lot of interest in database system design. The power-modes of hardware systems provide an appropriate tool for lowering power cost yet the the demand for guaranteed performance needs to be entertained. DVFS is a popular mechanism embedded in many modern CPUs to enable power/performance tradeoffs. In this paper, we tackle the problem of maximizing energy savings via DVFS under a user-specified performance bound in database systems. We argue that such a problem is

non-trivial due to the dynamics in database workloads and environment therefore solutions based on traditional static optimization formulation are not effective.

In this paper, we reformulate the problem into a feedback control problem via empirical study of the relationships among power consumption, DVFS level, and system throughput. For that, we design and implement the PAT framework to adjust CPU frequency online. We use techniques derived from well-established control theory paradigm for designing controller based on system identification results. Our design, unlike heuristics-based adaptive solutions widely used in database tuning, provides guarantees in the performance of DVFS control. Another feature in PAT is that it collects workload statistics in a real-time manner and learns key information that is needed for effective control. Based on fuzzy set theory, such mechanisms are also backed up by mathematical guarantees. We implement PAT within the PostreSQL engine and empirical results demonstrate that our control solution can effectively reduce DBMS servers' power consumption and achieve higher energy efficiency while satisfying required throughput performance.

Immediate future work would be to make PAT handle more scenarios and taking advantage of other power saving handles in the DBMS. For example, we have found that by shifting the query optimization towards power, the database can run in lower power due to the existence of energy-efficient query plans. The relevant query optimizer parameter (i.e., preference of performance over energy) can thus be combined with DVFS level to form a 2-dimensional control signal for harvesting more energy reduction opportunities. Another direction would be to make the control loop more robust by considering dramatic changes of workload dynamics. For that, we can develop an adaptive control scheme in which two feedback loops are attached to each other. Considering performance, system utilization requirements and extensibility of functions, we envision a hybrid approach of performance control on I/O intensive workloads, which will be the future for DBMS power capping technique. We plan to focus on innovative ideas to enable dynamic power control storage options for database applications and utilize a hybrid storage system for the very near future.

References

[1] L. A. Barroso and U. Hölzle. The case for energy-proportional computing. *IEEE Computer*, 40(12):33–37, 2007.

[2] L. Bertini, J. C. B. Leite, and D. Mossé. Power optimization for dynamic configuration in heterogeneous web server clusters. *Journal of Systems and Software*, 83(4):585–598, 2010.

[3] K. P. Brown, M. Mehta, M. J. Carey, and M. Livny. Towards automated performance tuning for complex workloads. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 72–84, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[4] M. Chen, X. Wang, R. Gunasekaran, H. Qi, and M. Shankar. Control-based real-time metadata matching for information dissemination. In *RTCSA*, pages 133–142, 2008.

[5] DatacenterDynamics. http://www.datacenterdynamics.com/research/energy-demand-2011-12.

[6] G. F. Franklin, J. D. Powell, and M. L. Workman. *Digital Control of Dynamic Systems*. Addison-Wesley, 1990.

[7] G. Graefe. Database servers tailored to improve energy efficiency. In *Proceedings of the 2008 EDBT Wworkshop on Software Engineering for Tailor-made Data Management*, SETMDM '08, pages 24–28, 2008.

[8] S. Harizopoulos, M. A. Shah, J. Meza, and P. Ranganathan. Energy efficiency: The new holy grail of data management systems research. In *CIDR*, 2009.

[9] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[10] T. Horvath, T. F. Abdelzaher, K. Skadron, and X. Liu. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *IEEE Trans. Computers*, 56(4):444–458, 2007.

[11] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, 1999.

[12] M. A. Kandaswamy and R. L. Knighten. I/o phase characterization of tpc-h query operations. In *Proceedings of the 4th International Computer Performance and Dependability Symposium*, IPDS '00, pages 81–, Washington, DC, USA, 2000. IEEE Computer Society.

[13] K.-D. Kang, J. Oh, and S. H. Son. Chronos: Feedback control of a real database system performance. In *RTSS*, pages 267–276, 2007.

[14] W. Lang, R. Kandhan, and J. M. Patel. Rethinking query processing for energy efficiency: Slowing down to win the race. *IEEE Data Eng. Bull.*, 34(1):12–23, 2011.

[15] W. Lang and J. M. Patel. Towards eco-friendly database management systems. In *CIDR*, 2009.

[16] C. Lefurgy, X. Wang, and M. Allen-Ware. Server-level power control. In *ICAC*, page 4, 2007.

[17] W. Li, J. Han, and J. Pei. Cmar: Accurate and efficient classification based on multiple class-association rules. *Data Mining, IEEE International Conference on*, 0:369, 2001.

[18] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. Power management of online data-intensive services. In *ISCA*, pages 319–330, New York, NY, USA, 2011. ACM.

[19] R. G. Melhem, D. Mossé, and E. N. Elnozahy. The interplay of power management and fault recovery in real-time systems. *IEEE Trans. Computers*, 53(2):217–231, 2004.

[20] K. Ogata. *Modern Control Engineering*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 4th edition, 2001.

[21] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *EuroSys*, pages 13–26, 2009.

[22] M. Poess and R. O. Nambiar. Energy cost, the key challenge of today's data centers: a power consumption analysis of tpc-c results. *PVLDB*, 1(2):1229–1240, 2008.

[23] M. Poess and R. O. Nambiar. Tuning servers, storage and database for energy efficient data warehouses. In *ICDE*, pages 1006–1017, 2010.

[24] PostgreSQL. http://www.postgresql.org.

[25] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *ACM Transactions on Architecture and Code Optimization*, 1(1), 2004.

[26] Sloan Digital Sky Survey. http://cas.sdss.org/dr7/en/.

[27] T. Takagi and M. Sugeno. Fuzzy identification of systems and its applications to modeling and control. *Ieee Transactions On Systems Man And Cybernetics*, 15(1):116–132, 1985.

[28] Transaction Processing Performance Council. http://www.tpc.org.

[29] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the energy efficiency of a database server. In *SIGMOD Conference*, pages 231–242, 2010.

[30] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the energy efficiency of a database server. In *SIGMOD Conference*, pages 231–242, 2010.

[31] Y.-C. Tu, S. Liu, S. Prabhakar, and B. Yao. Load shedding in stream databases: A control-based approach. In *VLDB*, pages 787–798, 2006.

[32] P. Unterbrunner, G. Giannikis, G. Alonso, D. Fauser, and D. Kossmann. Predictable performance for unpredictable workloads. *Proc. VLDB Endow.*, 2(1):706–717, Aug. 2009.

[33] U.S. Environmental Protection Agency. Report to congress on server and data center energy efficiency. *Government Report*, August 2007.

[34] L. Wang, J. Xu, M. Zhao, Y. Tu, and J. A. B. Fortes. Fuzzy modeling based resource management for virtualized database systems. In *MASCOTS*, pages 32–42, 2011.

[35] X. Wang, M. Chen, C. Lefurgy, and T. W. Keller. Ship: Scalable hierarchical power control for large-scale data centers. In *PACT*, pages 91–100, 2009.

[36] Y. Wang, X. Wang, M. Chen, and X. Zhu. Partic: Power-aware response time control for virtualized web servers. *IEEE Trans. Parallel Distrib. Syst.*, pages 323–336, 2011.

[37] G. Weikum, A. Mönkeberg, C. Hasse, and P. Zabback. Self-tuning database technology and information services: from wishful thinking to viable engineering. In *VLDB*, pages 20–31, 2002.

[38] Q. Wu, P. Juang, M. Martonosi, L.-S. Peh, and D. W. Clark. Formal control techniques for power-performance management. *IEEE Micro*, 25(5):52–62, 2005.

[39] W. Xu, X. Zhu, S. Singhal, and Z. Wang. Predictive control for dynamic resource allocation in enterprise data centers. In *NOMS*, pages 115–126, 2006.

[40] Z. Xu, Y.-C. Tu, and X. Wang. Exploring power-performance tradeoffs in database systems. In *ICDE*, pages 485–496, 2010.

[41] Y. Zhu and F. Mueller. Exploiting synchronous and asynchronous dvs for feedback edf scheduling on an embedded platform. *ACM Trans. Embedded Comput. Syst.*, 7(1), 2007.