

# *PowerTracer*: Tracing requests in multi-tier services to diagnose energy inefficiency

Lin Yuan<sup>1</sup>, Gang Lu<sup>1</sup>, Jianfeng Zhan<sup>1</sup>, Haining Wang<sup>2</sup>, and Lei Wang<sup>1</sup>

<sup>1</sup> Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

<sup>2</sup> Department of Computer Science, College of William and Mary Williamsburg, US

**Abstract.** As energy has become the major proportion of data center cost, it is essential to diagnose energy inefficiency inside data centers. The focus of this paper lies in the development of innovative techniques of diagnosing energy-inefficiency and saving power. Our contributions are three-fold: first, we propose a generalized methodology of applying request tracing approach for energy inefficiency diagnosis and power saving in multi-tier service systems. Second, on the basis of insights gained from request tracing, we develop an energy-inefficiency debugging tool that pinpoints the root causes of energy inefficiency and an accurate DVFS control mechanism that combines an empirical performance model and a simple feedback controller. Third, we design and implement a prototype of the proposed system, called *PowerTracer*, and conduct extensive experiments on a three-tier platform to validate its effectiveness. Our debugging tool diagnoses several state-of-the-practice and state-of-the-art DVFS control policies and uncovers existing energy inefficiencies. Meanwhile, our experimental results demonstrate that *PowerTracer* outperforms its peers in terms of power saving.

**Keywords:** multi-tier web server system, request tracing, debugging energy inefficiency, saving power

## 1 Introduction

Data centers have experienced significant growth in their scales and the complexity of the applications running inside centers. As the services provided by data centers scale up, data center owners have been facing with the increasing challenge of minimizing the capital and operations cost of a data center. To date energy cost has been a major part of the total cost of ownership (TCO) of a data center. With the continuing decrease of hardware price, the portion of energy cost will grow even larger in the near future. Therefore, uncovering the root causes of energy inefficiency inside data centers and finally improving energy efficiency, which we call *diagnosing energy inefficiency*, has become a critical task for data center administrators. The focus of this paper lies in the development of innovative techniques for diagnosing energy inefficiency and saving power in a multi-tier service platform, since most of services in data centers adopt a multi-tier architecture [9].

On one hand, a variety of request tracing approaches have been proposed to diagnose performance problems of multi-tier services of black boxes [16] [41] [34] or white boxes [12] [14] [15] [33] [40]; however, none of them take power management into account. On the other hand, significant efforts have been paid to improve energy efficiencies from three different perspectives, (1) dynamic voltage and frequency scaling (DVFS) [7] [1] [2] [10], (2) dynamic cluster reconfiguration by consolidating services through request distribution [7] [21], and (3) server consolidation through moving services to virtual machines [26] [25] [30]; however, there are subtle differences between diagnosing energy inefficiency and improving energy efficiency: traditional power saving approaches do not focus on how to uncover root causes of energy inefficiency, which may be caused by DVFS control policies or misconfigurations. Only after we have pinpointed the root causes, can we significantly improve energy efficiency.

In this paper, we propose a generalized methodology of applying request tracing approach for energy inefficiency diagnosis and power saving in multi-tier service systems. Through library interposition or kernel instrumentation, a request tracing tool can obtain *major causal path patterns* (in short, patterns), which represent *repeatedly executed* causal paths that account for significant fractions, in serving different requests and capture the server-side latency, especially the service time of each tier in different patterns. Thus, we can fully understand the role of each tier played in serving different requests and pinpoint the root causes of energy inefficiency with the support of power metering. Meanwhile, the request tracing approach also provides a sound basis for energy saving. First, in comparison with a brute-force approach, it decreases the time cost of performance profiling that is necessary to develop an accurate performance model; second, it reduces the controller complexity. Both the performance model and the controller are critical components in many power saving systems.

We design and implement a prototype of the proposed system, called *PowerTracer*, leveraging online request tracing for energy inefficiency diagnosis and accurate DVFS control. In general, PowerTracer consists of an energy-inefficiency debugging tool and a power saving system for multi-tier services. Two different request tracing tools PreciseTracer [41], which proposes a precise request approach, and WAP5 [34], which accepts imprecision of a probabilistic correlation method, are used in PowerTracer, respectively, one in kernel and the other in library. We conduct extensive experiments on a three-tier platform. For diagnosing energy inefficiency, we use several case studies to demonstrate the efficacy of PowerTracer, including state-of-the-art DVFS control policy—SimpleDVS [7], state-of-the-practice DVFS control policy—OnDemand [17], and state-of-the-art DVFS control policy that leverages the *average* service time of each tier without classifying major causal path patterns. With regard to power saving, we use two typical workloads: RUBiS (Rice University Bidding System) [4] and RUBBoS [52] to evaluate the effectiveness of PowerTracer in terms of three metrics: total system power savings compared to the baseline, request deadline miss ratio, and average server-side latency. Our experimental results demonstrate that PowerTracer not only uncovers existing energy inefficiencies but also outperforms its

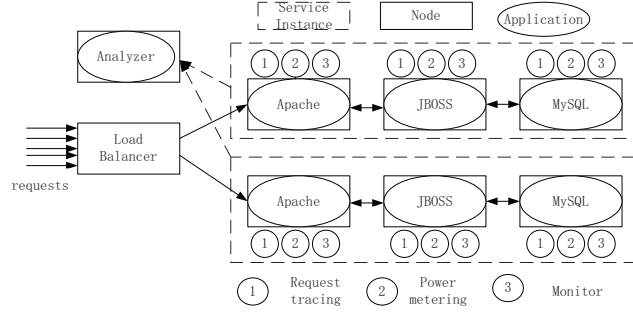


Fig. 1: The architecture of the energy inefficiency debugging tool.

peers [7] [17] in power saving. Moreover, PowerTracer equipped with PreciseTracer outperforms that equipped with WAP5, indicating that higher accuracy of request tracing leads to more power saving.

The remainder of this paper is organized as follows. Section 2 details the basic working mechanisms, and the system architecture of PowerTracer. Section 3 describes the system implementation. Section 4 presents the experimental results. Finally, Section 6 draws a conclusion.

## 2 PowerTracer System Architecture

PowerTracer consists of two parts: an energy inefficiency debugging tool and a power saving system.

Our targeted service applications use a multi-tier architecture, and services are replicated or distributed on a cluster of servers [9]. We call an instance of multi-tier service a *service instance*, which normally includes an Apache server, a JBOSS Server, and a MySQL database. The load balancer like LVS [49] is responsible for distributing requests to many service instances for load balancing.

### 2.1 Energy Inefficiency Debugging Tool

As shown in Fig.1, the energy inefficiency debugging tool of PowerTracer consists of four modules: online request tracing, monitor, power metering, and analyzer.

**Online Request Tracing.** We observe that a request causes a series of *interaction activities* in the OS kernel or the shared library, e.g. sending or receiving messages. Those activities happen under specific contexts (processes or kernel threads) of different components. We record an activity of sending a message as  $S_{i,j}^i$ , which indicates a process  $i$  sends a message to a process  $j$ . We record an activity of receiving a message as  $R_{i,j}^j$ , which indicates a process  $j$  receives a message from a process  $i$ . Our concerned activity types include: *BEGIN*, *END*,

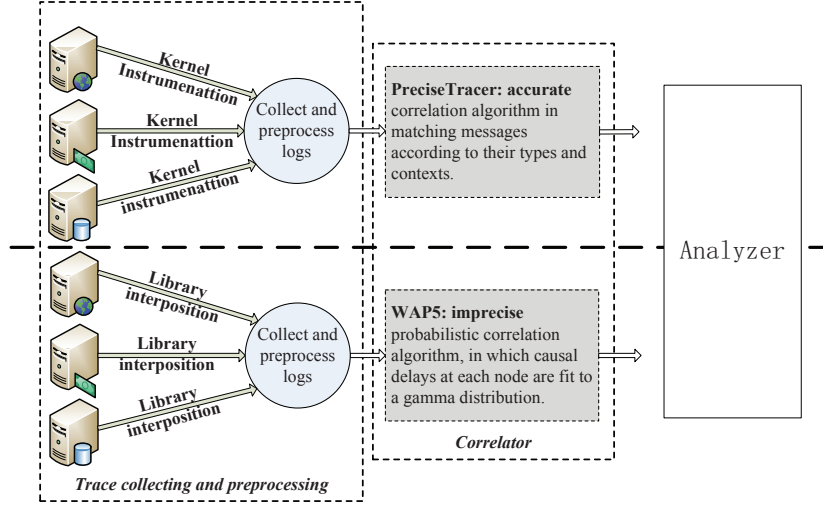


Fig. 2: Two tracer architectures.

*SEND*, and *RECEIVE*. The *SEND* and *RECEIVE* activities are those of sending and receiving messages. A *BEGIN* activity marks the start of servicing a new request, while an *END* activity marks the end of servicing a request.

When an individual request is serviced, a series of activities that have causal or happened-before relationships constitute a causal path. For each individual request, there exists a causal path. For a request, the server-side latency can be defined as the time difference between the time-stamp of *BEGIN* activity and that of *END* activity in its corresponding causal path. The service time of each tier can be defined as the accumulated processing time between *SEND* activities and *RECEIVE* activities. For each tier, its role in serving a request can be measured in terms of service time percentage, which is the ratio of service time of the tier to the server-side latency.

After reconstructing those activity logs into causal paths, we classify the causal paths into different *patterns*. Then, we use main patterns to represent repeatedly executed causal paths, which account for significant fractions. For each pattern, we compute the average server-side latency and the average service time of each tier. In addition, by observing the number of *BEGIN* activities, which mark the beginning of serving new requests, we can derive the current load level of the services. So, for a service instance, the online performance data provided by the request tracing system include: main patterns, the server-side latencies, the service time of each tier, and current loads.

As shown in Fig.2, in PowerTracer, we have implemented two instrumentation mechanisms: OS kernel instrumentation based on SystemTap [41], and library interposition. While correlating interaction activities into causal paths, we integrate PowerTracer with two tracing algorithms: the PreciseTracer algorithm [41], which is an accurate tracing algorithm, and the WAP5 algorithm

[34], which accepts imprecision of probabilistic correlations. The source code of two tracers can be found at <http://www.ncic.ac.cn/~zjf/#opensource>.

**Analyzer.** A new component called *Classifier* is responsible for classifying a large variety of causal paths into different patterns, and extracting online performance data for main patterns according to their fractions.

Our classification policy of causal paths addresses the following two problems: first, it is difficult to utilize each individual causal path from massive request traces as guidelines for debugging energy inefficiency or DVFS modulation; second, causal paths are different and the overall statistics of performance information (e.g., the average server-side latency used in [7]) of all paths hide the diversity of patterns. PowerTracer provides several ways to classify causal paths. One is classifying causal paths into different patterns according to their shapes. The other solution uses a k-means clustering method [8] to classify causal paths into patterns based on the size of the first message sent by a client, since the first message includes a URL and the length of the URL can uniquely identify the work to be performed by the tiers.

We use a four-tuple  $\langle \text{pattern ID}, \text{the average server-side latency of pattern ID}, \text{the average service time per tier}, \text{the current load} \rangle$  to represent the online performance data of a pattern in a multi-tier service. In our study, for RUBiS workload, we observe that there are more than hundred patterns, but the top ten patterns take up a large fraction of paths, more than 88%. Therefore, with the support of power metering, we use the online performance data provided by the request tracing module as guidelines for debugging energy inefficiency or building both the performance model and the controller module. Section 4.2 presents three case studies of debugging energy inefficiency.

**Monitor and Power Metering.** The other two modules, the monitor and power metering modules, are simple. The monitor module reports the resource status of each node. With the support of fine-grain resource accounting, like Linux /proc, the monitor module can report the resource consumption of each tier of services. The power metering module is responsible for collecting power consumption of each node. We just use QINGZHI 8775A power analyzer to measure the power consumption of each node.

## 2.2 Accurate DVFS Control

As shown in Fig.3, the power saving system of PowerTracer consists of three major modules: online request tracing, performance profiling, and controller.

The online request tracing module is described above. The performance profiling module aims to establish an empirical performance model off-line under different load levels. The model should capture the relationship between service performance and CPU frequency setting of each node. With the online performance data produced by the request tracing module, we can locate the dominated tier by checking the service time percentage of each tier in different

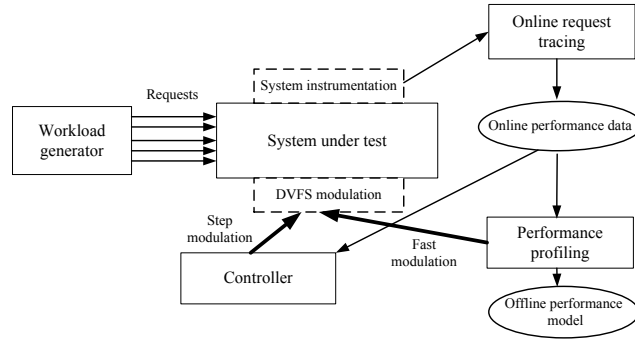


Fig. 3: The architecture of the power saving system

patterns. Then, we mainly observe the effect of DVFS modulation of the node, on which the dominated tier is deployed. In this way, we can ease the establishment of the performance model.

Based on the performance model, the controller module quickly calculates the optimal setting of DVFS modulation for different load levels. For each service instance, the controller module only relies on the single-node DVFS modulation as opposed to varying multiple CPU frequencies simultaneously at control periods that need the adaption.

**Performance profiling.** On the basis of the approach used in [32], our approach aims to create detailed performance profiles of multi-tier services with different DVFS modulations under varying workloads. A tier profile captures each tier’s performance characteristics as a function of the DVFS modulation and varying loads. In order to profile a tier, we deploy a test environment and vary the DVFS setting of a node, on which each tier is deployed. We then apply a variety of loads and collect each tier’s performance characteristics, independent of other tiers. After acquiring the measurement data, we derive the appropriate functions mappings from DVFS modulation and load levels to each tier’s performance metrics. The purpose of performance profiling is to create an empirical performance model, which is used for the fast modulation procedure. The power model can be represented by Equation (1) as follows.

$$\begin{cases} D_{L,1} = \sum_{j=1}^M f_{L,1,j}(F_j) + \gamma_{L,1} \\ \vdots \\ D_{L,N} = \sum_{j=1}^M f_{L,N,j}(F_j) + \gamma_{L,N} \end{cases} \quad (1)$$

$N$  is the number of *major* patterns singled out by our request tracing system.  $M$  is the tiers of a multi-tier service instance, namely three here.  $D_{L,i}$  ( $i = 1, 2, \dots, N$ ) is the average server-side latency of pattern  $i$  when the current load is  $L$ , and  $\gamma_{L,i}$  ( $i=1, \dots, N$ ) is the network latency of pattern  $i$  when the current

load is  $L$ . Function  $f_{L,i,j}(F_j)$  represents the average service time of each tier  $j$  in pattern  $i$ , when the clock frequencies run with  $F_j(j=1,\dots,M)$  and the current load is  $L$ .

For each load level, the function set  $f$  is derived as follows: first, through online request tracing, we measure the service time of each tier while traversing all discrete CPU frequency values offered by each server; second, we derive functions  $f_{L,i,j}$  between the average service time of each tier in major patterns and the CPU frequencies of nodes by the normal quadratic polynomial fitting tool. The models fit well, and the fitting coefficient  $R^2 > 97\%$ .

Through request tracing and performance profiling, we can ease the establishment of the empirical performance model. For each workload, e.g. given the transition table and the number of current clients, we create a function set  $f$  offline. For a three-tier service deployed on the testbed that offers four CPU frequency levels, if we consider 10 different load levels (from zero to the upper bound of the load), we need to do  $10 \times 4 \times 3$  times of experiments to gain the function set  $f$ . Since our request tracing tool can obtain the service time of each tier in major causal path patterns and decide the dominated tiers in terms of the service time percentage of each tier, our system can effectively decrease the time cost of performance profiling experiments by mainly scaling the CPU frequencies of the tiers that dominate the server-side latency. For example, for RUBiS, the average service time percentage of the web server, the application server, and the database server are 0.11%, 17.63%, and 82.26%, respectively. So, we consider the database server as the dominated tier. In this way, we can reduce the times of experiments to  $10 \times 4 \times 1$ .

**Controller design.** Similar to the control-theoretic terminology used in [27], we refer to the server cluster and deployed multi-tier service being controlled as *the target system*, which has a set of metrics of interest, referred to as *measured output*, and a set of control knobs, referred to as *control input*. The controller periodically (at each control period) adjusts the value of the control input such that the measured output (at each sampling period) can match its desired value—referred to as *reference input* specified by the system designer. We refer to the difference between the measured output and the reference input as *control error*. The controller aims to maintain control error at zero, in spite of the disturbance in the system.

In our feedback controller, the target system is the multi-tier server cluster. The control inputs are the new clock frequency vector. The measured outputs are the server-side latencies of the top main  $N$  patterns, which are presented as a vector as well. We define  $TH_i$  ( $i=1,\dots,N$ ) as the server-side latency threshold of pattern  $i$ . The reference inputs are the desired server-side latency *threshold zones* for the main patterns.  $LP$  and  $UP$  represent the lower and upper threshold factors. The controller module makes the measured outputs fall within the latency threshold zone by adjusting the values of the control inputs periodically. The controller module consists of two main procedures: fast modulation and step modulation. During the fast modulation procedure, by tracing the current load,

PowerTracer can decide the current load level and compute the optimal clock frequency of each server, with reference to the given server-side latency threshold zones of the main patterns according to Equation (1).

After the fast DVFS modulation, it is the step of modulation loop. In our system, step modulation periods are composed of alternate *sampling* periods and *control* periods. We model our system as Equation 2.

$$\overrightarrow{D(t+1)} = F(\overrightarrow{D(t)}, \overrightarrow{f(t)}) \quad (2)$$

$\overrightarrow{D(t)}$  is the vector that represents the average server-side latencies of each pattern in the  $t_{th}$  sampling period.  $\overrightarrow{f(t)}$  is the vector that represents the CPU frequency levels of the nodes in the  $t_{th}$  period.  $\overrightarrow{F(t)}$  is the vector that represents the transition functions of each pattern from states of the  $t_{th}$  period to that of the  $(t+1)_{th}$  period.

We design the step modulation procedure as follows. In the  $t_{th}$  sampling period, we use the request tracing module to obtain four-tuple online performance data of the top  $N$  patterns.

By comparing the measured server-side latencies of the top  $N$  patterns with the latency threshold zones, the controller module modulates the DVFS setting based on the following procedure.

For pattern  $i$  of each service instance, if  $D_i(t)$  exceeds its upper threshold  $UP*TH_i$ , the controller module chooses tier  $j$  that has the maximum service time to step up its CPU frequency and record the new frequency value into the frequency vector. The frequency values of the other tiers remain intact. For any patterns  $i$  ( $i = 1, \dots, N$ ) whose  $D_i(t)$  fall below their lower threshold  $LP*TH_i$ , the controller module chooses tier  $j$  that has the minimum service time to step down its CPU frequency and record the new frequency into the frequency vector. The frequency values of the other tiers remain intact.

The approach that we take the measured server-side latencies of the top  $N$  main patterns as the controlled variables, enables the majority of requests meet service-level agreement. A small percentage of requests is ignored so as to achieve the most power reduction. The step modulation procedure can be formally defined in Equation 3.

$$\begin{cases} f_j(t+1) = f_j(t) + 1 & \exists i | (D_i(t) > UP * TH_i) \wedge (ST_{i,j}(t) \text{ is the maximum}) \\ f_j(t+1) = f_j(t) - 1 & \forall i | (D_i(t) < LP * TH_i) \wedge (ST_{i,j}(t) \text{ is the minimum}) \\ f_j(t+1) = f_j(t) & \text{otherwise} \end{cases} \quad (3)$$

In Equation 3,  $ST_{i,j}(t)$  represents the service time of tier  $j$  in pattern  $i$  in the  $t_{th}$  period. For  $f_j, +1$  indicates stepping up CPU frequency with one level, while  $-1$  indicates stepping down CPU frequency with one level.

### 3 System Implementation

As Fig.4 shows, PowerTracer in total includes seven major components: Tracer, Power Modeler, Controller, Scaler, Monitor, Power Meter, and Analyzer. The

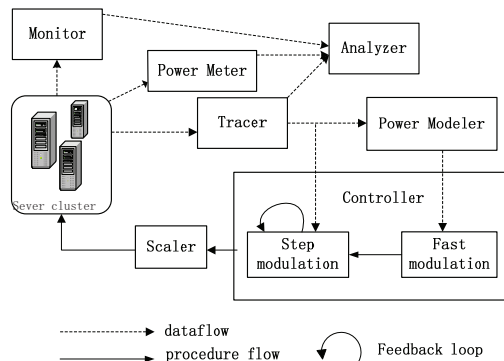


Fig. 4: The architecture diagram of PowerTracer.

energy-inefficiency debugging tool of PowerTracer includes Tracer, Monitor, Power Meter, and Analyzer, while the power saving system of PowerTracer includes Tracer, Power Modeler, Controller, and Scaler.

We implement the two request tracing systems described in Section 2.1 and integrate them into PowerTracer as a module called Tracer. As shown in Fig.4, Tracer reads all service logs and outputs four-tuple performance data of the top  $N$  patterns.

Through changing the DVFS modulation, PowerModeler analyzes the four-tuple performance data of the top  $N$  patterns, and then outputs the power model in the file *Pre\_model*.

For different load levels, Controller runs the fast modulation based on the *Pre\_model* file. Then, Controller runs step modulation loops, which are composed of alternate sampling and control periods. In the sampling period, Tracer is called to output four-tuple performance data for the top  $N$  patterns. In the control period, Controller makes decisions on changing clock frequencies as the approach presented in Section 2.2. Scaler, which runs on each tier, is called by Controller to set clock frequencies. The clock frequency configuration is invoked by setting frequency scaling governor of the Linux kernel and recording new frequency into *scaling\_setspeed* file.

Power Meter is responsible for measuring power consumption of each node. Monitor periodically reads resource accounting information maintained by the operating system, like Linux */proc*, and reports the resource status of each node. Tracer, Power Meter, and Monitor periodically send their collected data to Analyzer, which is responsible for debugging energy inefficiency.

## 4 Evaluation

We use two three-tier web applications RUBiS [4] and RUBBoS [52] to evaluate the efficiency of our approach for both debugging energy inefficiency and saving power. RUBiS is a three-tier auction site prototype modeled after eBay.com,

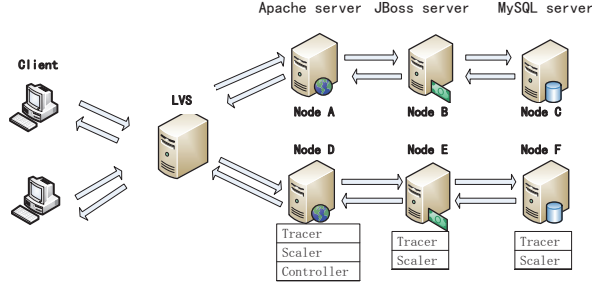


Fig. 5: The deployment diagram of three-tier platform.

developed by Rice University. RUBBoS is a bulletin board benchmark modeled after an online news forum like Slashdot, also developed by Rice University.

#### 4.1 Experimental Setup

The testbed is an heterogeneous nine-node platform composed of Linux-OS blade servers. In the following experiments, we deploy one or two service instances, each of which includes an Apache server, a JBOSS Server, and a MySQL database, on the testbed as shown in Fig. 5. Nodes A and D are deployed as the web server tier; Nodes B and E are deployed as the application server tier, and Nodes C and F are deployed as the database server tier. All the server nodes are DVFS-capable. Nodes A and B both have two capable processors, which support the frequencies at 1.0, 1.8, 2.0 and 2.2GHz. Nodes C and D both have eight capable processors, which support the frequencies at 0.8, 1.1, 1.6 and 2.3GHz. Nodes E and F both have 16 capable processors, which support the frequencies at 1.6, 1.7, 1.8, 2.0, 2.1, 2.2 and 2.4GHz.

As shown in Fig. 5, Tracer, Scaler, Monitor, and Power Meter are deployed on all the three server tiers, but Controller and Analyzer are only deployed on the web server tier rather than the other two server tiers. Note that the service delay at the web server tier imposes the least impact on request performance. In our experiments, the default tracer is set to PreciseTracer.

#### 4.2 Case Studies of Debugging Energy Inefficiency

In this subsection, we report three case studies of debugging energy inefficiency with PowerTracer. In the experiments, we use the mixed workload of RUBiS and set the number of clients to 500 to generate workloads. We deploy the three-tier servers on Nodes A, B and C as shown in Fig. 5. Each workload includes three stages, of which we set up ramp time, runtime session, and down ramp time as 5 seconds, 300 seconds, and 5 seconds, respectively. For control systems, we set the control period to 6 seconds. For PowerTracer, we set the sampling period to 6 seconds. During each sampling period, PowerTracer collects the performance data for 1 second to lower the overhead.

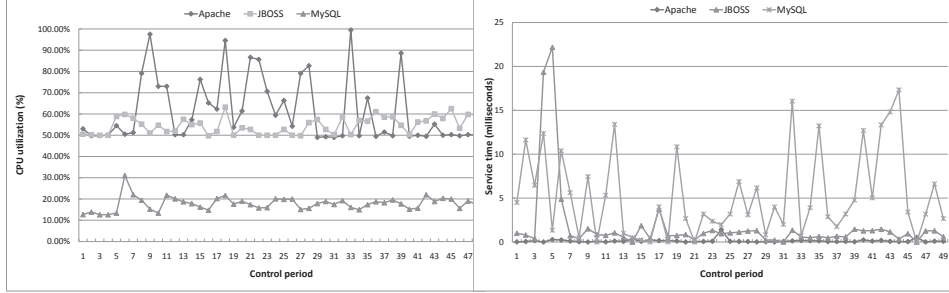


Fig. 6: The CPU utilizations of each tier equipped with SimpleDVS.

Fig. 7: The service time of each tier equipped with SimpleDVS.

**SimpleDVS.** The SimpleDVS DVFS control algorithm presented by Horvath *et al.* [7] takes CPU utilization as the indicator in determining which server's clock frequency should be scaled, and implements a feedback controller based on the DVS mechanism. SimpleDVS chooses the measured server-side latency as the input and decides how to scale clock frequency of the servers. Its scaling policy is that if the measured latency is below a lower threshold, then step down clock frequency of the server which has the minimum CPU utilization; if the measured latency is above an upper threshold, then step up clock frequency of the server which has the maximum CPU utilization.

For SimpleDVS, we set the latency threshold, the upper latency threshold factor,  $UP$ , and the lower latency threshold factor,  $LP$ , to 16.3483 milliseconds, 1.2 and 0.8, respectively.

With PowerTracer, we find out that at most of time the CPU utilization rate of the Apache tier is the highest, shown in Fig. 6, however the service time of the JBOSS tier, shown in Fig.7, dominates over that of the Apache tier. Meanwhile, we also observe that the CPU utilization rate of the MySQL tier is the lowest, while its service time is the longest at most of time. This observation indicates that the service time of each tier is not consistent with its CPU utilization, and hence it is not optimal for SimpleDVS to step down or up clock frequency of the server which has the maximum or minimum CPU utilization to affect the server-side latency. In Section 4.3, our experiments will show that taking the service time of each tier as the indicator for DVFS modulation can save more power than that of SimpleDVS, which confirms that taking CPU utilization as the indicator for DVFS modulation is the root cause of energy inefficiency of SimpleDVS. More details can be found at our technical report [50]. Fig.8 shows the server-side latencies of the service equipped with SimpleDVS.

**Ondemand governor.** The Ondemand governor, the most effective power management policy offered by Linux kernel, can vary CPU frequency based on CPU utilization.

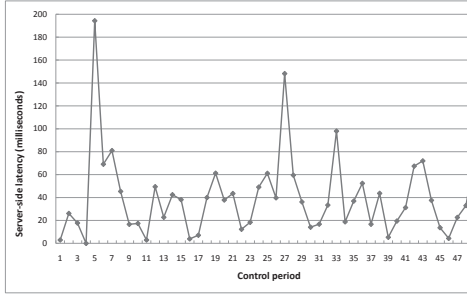


Fig. 8: The server-side latencies of the service equipped with SimpleDVS.

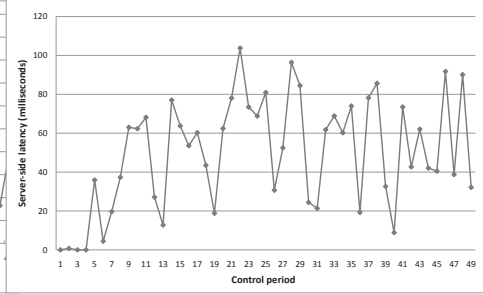


Fig. 9: The server-side latencies of the service equipped with Ondemand.

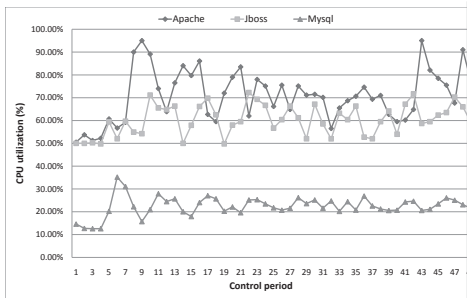


Fig. 10: The CPU utilizations of each tier equipped with Ondemand.

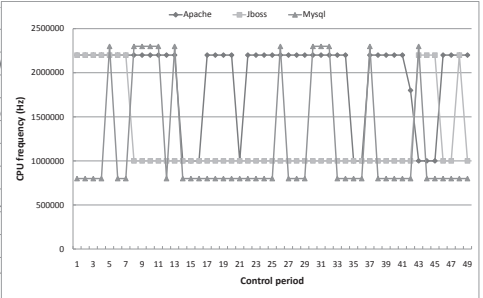


Fig. 11: The frequencies of three nodes adjusted by Ondemand governor.

Fig. 9, Fig.10, and Fig.11 show that the server-side latencies, the CPU utilizations of each tier, and the CPU frequencies of each node are dynamically changed in different control periods, respectively, due to the use of Ondemand governor.

Compared with that of SimpleDVS, Fig. 9 shows that the server-side latency of the service equipped with Ondemand is higher. At most of time, the server-side latency of the service equipped Ondemand is higher than 50 milliseconds, while the server-side latency of the service equipped with SimpleDVS is lower than 50 milliseconds. This observation indicates that without coordination of DVFS control actions on each node, Ondemand performs poorly in controlling the server-side latencies. Meanwhile, we also observe that at most of time, the CPU utilization of each tier with Ondemand is higher than that of SimpleDVS. Our additional experiment results in [50] also show that the service with Ondemand consumes more power than that of SimpleDVS. These observations demonstrate that the lack of the coordination of DVFS control actions on each node is the root cause of energy efficiency of Ondemand.

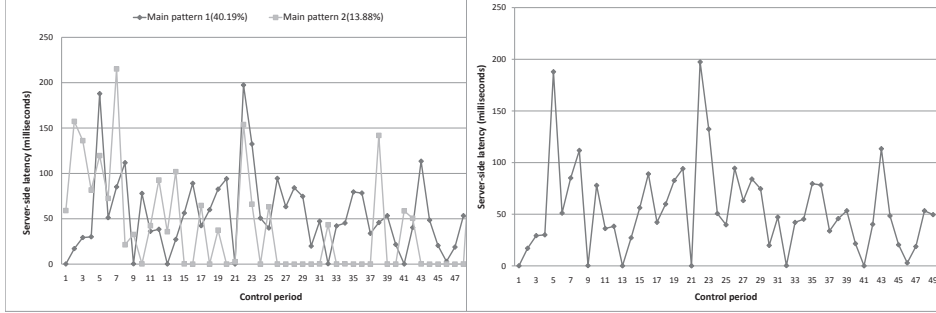


Fig. 12: The server-side latencies of the top two patterns equipped with PowerTracer\_NP.

Fig. 13: The average server-side latency of the service equipped with PowerTracer\_NP.

**PowerTracer without classifying main causal path patterns.** In particular, we implement a modified version of the power saving system introduced in Section 2.2, which we call *PowerTracer\_NP*. *PowerTracer\_NP* chooses the average server-side latency as the measured output, instead of  $N$  individual server-side latencies of the top  $N$  patterns.

For *PowerTracer\_NP*, we set the latency threshold, the upper latency threshold factor,  $UP$ , and the lower latency threshold factor,  $LP$ , to 16.3483 milliseconds, 1.2 and 0.8, respectively.

Fig. 12, Fig.13, and Fig.14 show the server-side latencies of the top two patterns—pattern 1 and pattern 2, the average server-side latency of all paths, and the CPU utilizations of each tier, respectively. The causal paths of pattern 1 and pattern 2 take up 40.19%, 13.88% of all causal paths, respectively.

From Fig. 12, we can see that even under the control of *PowerTracer\_NP*, pattern 1 and pattern 2 perform very differently, and hence we cannot implement an accurate DVFS control policy without classifying main causal path patterns. In Section 4.3, the experiments will show that choosing individual server-side latencies of the top  $N$  patterns as the measured output can save more power than that of *PowerTracer\_NP*, which confirms that choosing the average server-side latency as the measured output is the root cause of energy inefficiency of *PowerTracer\_NP*.

### 4.3 The Effectiveness of Accurate DVFS Control

We conduct four groups of experiments to evaluate the effectiveness of accurate DVFS control of *PowerTracer*, namely static workload experiment, dynamic workload experiment, multi-service-instance experiment, and WAP5 v.s. *PreciseTracer* experiment.

We set the clock frequency of all servers to the maximum as *the baseline*. We use three metrics to evaluate our system: the total system power savings compared to the baseline, the request deadline miss ratio, and the average server-

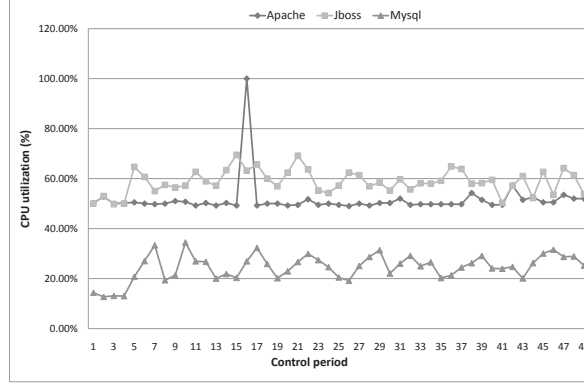


Fig. 14: The CPU utilizations of each tier equipped with PowerTracer\_NP.

side latency. Note that the power consumption under the baseline is not fixed for different load levels, and higher load will lead to higher power consumption even with the same clock frequency. We assume the server-side latency under the baseline is  $\vec{SL}$ . For PowerTracer,  $\vec{SL}$  is a vector representing the server-side latencies of the chosen main patterns. For the request deadline miss ratio, we predefine a server-side latency deadline, which is under the SLA constraints. We compare the results of PowerTracer with those of the three other algorithms: the SimpleDVS algorithm presented by Horvath *et al.* [7], the Ondemand governor offered by Linux kernel, and the modified version of the power saving system—PowerTracer\_NP.

We set both the sampling period and the control period as 6 seconds. We set the upper latency threshold factor,  $UP$ , and the lower latency threshold factor,  $LP$ , as 1.2 and 0.8, respectively. We trace the servers' performance statistics while setting the servers with different frequency values for different workloads, and use the normal quadratic polynomial fitting to derive the performance model.

**Static workload.** We have performed experiments on RUBBoS and the two workloads of RUBiS—read\_only workload and read\_write mixed workload. In the experiments, we investigate the effects of the server-side latency threshold and the number of main patterns upon the server cluster power consumption and other performance metrics.

*RUBBoS workload.* In the experiments, we only deploy a service instance on a three-node platform composed of Nodes A, B and C as shown in Fig.5. For the RUBBoS workload, we set the number of clients to 100, 200, 300, 400 and 500, respectively. Each workload includes three stages, of which we set up ramp time, runtime session and down ramp time as 5 seconds, 300 seconds and 5 seconds, respectively. The deadline is set to 0.5 second for RUBBoS mixed workload. For PowerTracer, we set the server-side latency threshold as  $2 \times \vec{SL}$ , where  $\vec{SL}$  is

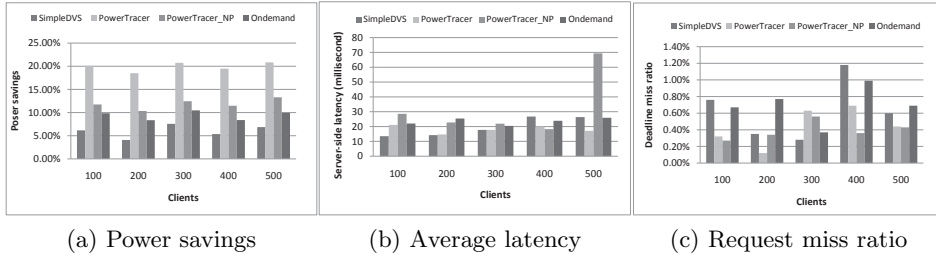


Fig. 15: A comparison of four power management algorithms in static workload. The x-axis represents the number of clients set by RUBBoS client emulator.

(6.31, 74.59, 112.50, 38.22, 27.83, 150.20, 72.19) milliseconds, and set the number of main patterns as 3 (the total number of patterns is 7).

Fig. 15 (a), (b), and (c) present the total system power savings compared to the baseline, the average server-side latencies, and the request deadline miss ratios of the four algorithms, respectively, when the number of clients varies from 100 to 500. We can see that PowerTracer clearly gains the highest power saving, and can reduce the power consumption by 20.83% when the number of clients is 500. The SimpleDVS has the lowest power reduction. The SimpleDVS and Ondemand governor both have high request deadline miss ratio.

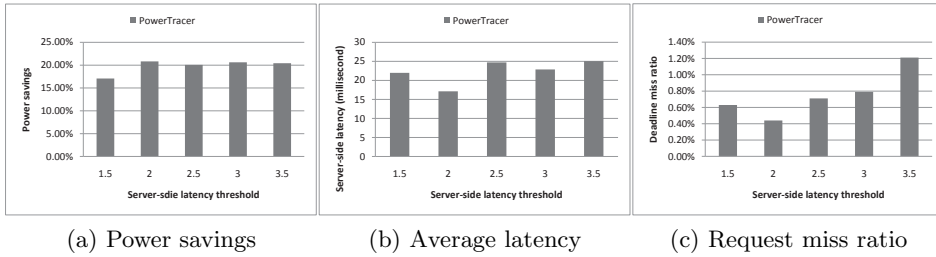


Fig. 16: Three evaluation metrics of PowerTracer under five latency thresholds. The x-axis represents five latency threshold levels. Fig.16a illustrates the total system power savings compared to the baseline. Fig.16b illustrates the average server-side latency of all requests. Fig.16c illustrates the deadline miss ratio of all requests, we set deadline to 0.5 second as above.

Fig. 16 (a), (b), and (c) demonstrate how the performance varies when the reference input in PowerTracer—the server-side latency threshold varies. We set the number of clients to 500 and select three main patterns according to their fractions from the seven patterns. For the seven patterns,  $\vec{SL}$  is (6.31, 74.59, 112.50, 38.22, 27.83, 150.20, 72.19) milliseconds. We set five latency thresholds

as 1.5, 2, 2.5, 3, and  $3.5 \times \overrightarrow{SL}$ , respectively. We can see that when the latency threshold is  $2 \times \overrightarrow{SL}$ , PowerTracer gains the highest power saving and the lowest request deadline miss ratio and average server-side latency among all the five groups of experiments. Therefore, choosing an appropriate latency threshold is critical to PowerTracer.

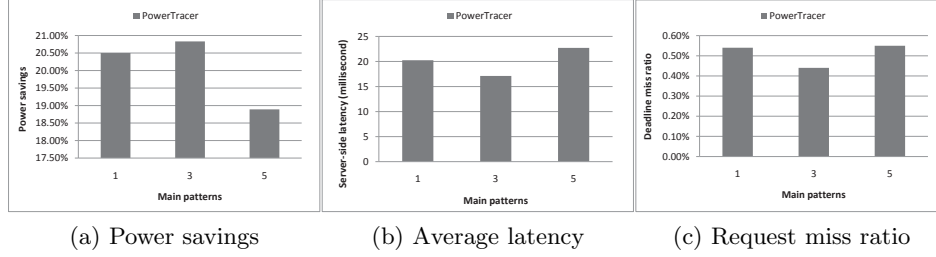


Fig. 17: Three evaluation metrics of PowerTracer with different number of main patterns. The x-axis represents the chosen number of the main patterns. Fig.17a illustrates the total system power savings compared to the baseline. Fig.17b illustrates the average server-side latency of all requests. Fig.17c illustrates the deadline miss ratio of all requests, we set deadline to 0.5 second as above.

Fig. 17 (a), (b), and (c) show the performance of PowerTracer when a different number of main patterns is chosen out of the seven major patterns. For these experiments, we set the number of clients to 500 and the latency threshold to  $2 \times \overrightarrow{SL}$ . When we set the number of main patterns to three, PowerTracer gains the highest power saving and the lowest server-side latency and request deadline miss ratio. Comparing the performance of one main pattern and that of five main patterns, we can see that choosing different number of main patterns is critical to the performance of PowerTracer. For workloads like RUBBoS benchmark, we suppose that when choosing the half of number of patterns as the number of main patterns, PowerTracer can achieve the optimal performance.

*RUBiS read\_only workload.* In the experiments, we only deploy a service instance on a three-node platform composed of Nodes A, B and C as shown in Fig.5. For the RUBiS read\_only workload, we set the number of clients to 100, 200, 300, 400 and 500, respectively. Each workload includes three stages, of which we set up ramp time, runtime session and down ramp time as 5 seconds, 300 seconds and 5 seconds, respectively. The deadline is set to 0.5 second for RUBiS read\_only workload. For PowerTracer, we set the server-side latency threshold as  $3 \times \overrightarrow{SL}$ , where  $\overrightarrow{SL}$  is (2.583734, 20.585, 25.88851, 63.86756, 74.38303, 63.06882, 66.01713) milliseconds, and set the number of main patterns as 3 (the total number of patterns is 7). Fig. 18 (a), (b), and (c) present the total system power savings, the average server-side latencies, and the request deadline miss ratios of the four algorithms, respectively, when the number of clients varies from 100

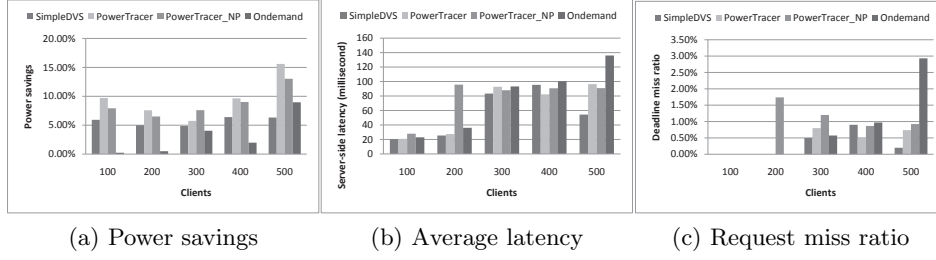


Fig. 18: A comparison of four power management algorithms. The x-axis represents the number of clients set by RUBiS client emulator. Fig. 18a illustrates the total system power savings compared to the baseline. Fig. 18b illustrates the average server-side latency of all requests. Fig. 18c illustrates the deadline miss ratio of all requests and the blank represents zero, we set deadline to 0.5 second for RUBiS read\_only workload.

to 500. In this set of experiments, we set the server-side latency threshold as  $3 \times \overline{SL}$ . For PowerTracer, *the number of main patterns* is 3, which indicates that the top 5 patterns are used as the guide for DVFS control. PowerTracer or PowerTracer\_NP gains the highest power reduction. The Ondemand governor gains the lowest power reduction except when the number of clients is 500. PowerTracer outperforms PowerTracer\_NP except when the number of clients is 300. When the number of clients is 500, PowerTracer gains the maximum power saving of 15.60% compared to the baseline, which is about 147% better than SimpleDVS and about 74% better than the Ondemand governor in terms of power saving.

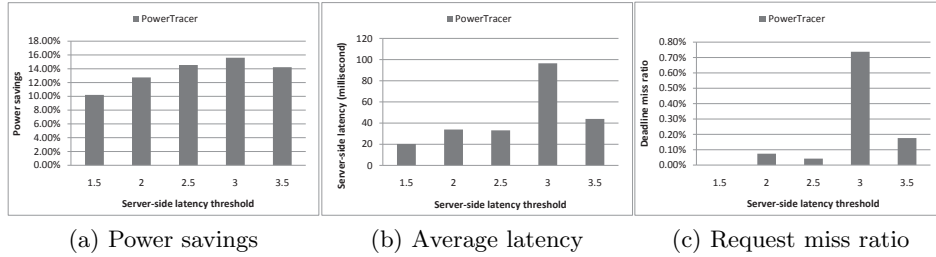


Fig. 19: Three evaluation metrics of PowerTracer under five latency thresholds. The x-axis represents five latency threshold levels. Fig.19a illustrates the total system power savings compared to the baseline. Fig.19b illustrates the average server-side latency of all requests. Fig.19c illustrates the deadline miss ratio of all requests, we set deadline to 0.5 second as above.

From Fig. 19(a), (b), and (c), we demonstrate how the performance varies when the reference input in PowerTracer—the server-side latency threshold, varies. We set the number of clients to 500 and select five main patterns according to their fractions from the top seven patterns. For the seven patterns,  $\overrightarrow{SL}$  is (2.583734, 20.585, 25.88851, 63.86756, 74.38303, 63.06882, 66.01713) milliseconds. We set five latency thresholds as  $1.5, 2, 2.5, 3,$  and  $3.5 \times \overrightarrow{SL}$ , respectively. We can see that when the latency threshold is  $2.5 \times \overrightarrow{SL}$ , PowerTracer is about 43% better than that when the the latency threshold is  $1.5 \times \overrightarrow{SL}$  in terms of the power saving. At the same time, under these two configurations, their server-side latencies and their request deadline miss ratios are close to each other. Therefore, choosing an appropriate latency threshold is critical to PowerTracer.

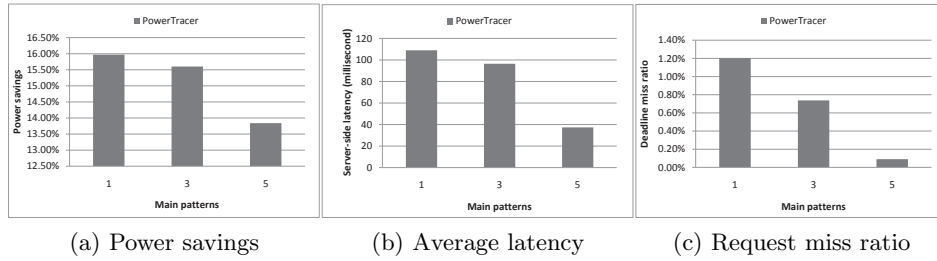


Fig. 20: Three evaluation metrics of PowerTracer with different number of main patterns. The x-axis represents the chosen number of the main patterns. Fig.20a illustrates the total system power savings compared to the baseline. Fig.20b illustrates the average server-side latency of all requests. Fig.20c illustrates the deadline miss ratio of all requests, we set deadline to 0.5 second as above.

Fig. 20 (a), (b), and (c) show the performance of PowerTracer when a different number of main patterns is chosen out of the seven major patterns. For these experiments, we set the number of clients to 500 and the latency threshold to  $3 \times \overrightarrow{SL}$ . When we set the number of main patterns to one, PowerTracer gains the highest power saving, but also the highest server-side latency and the highest request deadline miss ratio. Comparing the performance of one main pattern and that of five main patterns, we can see that choosing more patterns does not necessarily improve power savings and other metrics, and hence, we need to choose *the number of main patterns* based on different workloads so as to achieve the optimal power saving and performance.

*RUBiS read\_write mixed workload.* In the experiments, we only deploy a service instance on a three-node platform composed of Nodes A, B and C as shown in Fig.5. For the RUBiS read\_write mixed workload, we set the number of clients to 500, 600, 700, 800 and 900, respectively. Each workload includes three stages, of which we set up ramp time, runtime session and down ramp time as 5 seconds, 300 seconds and 5 seconds, respectively. The deadline is set to 0.5 second

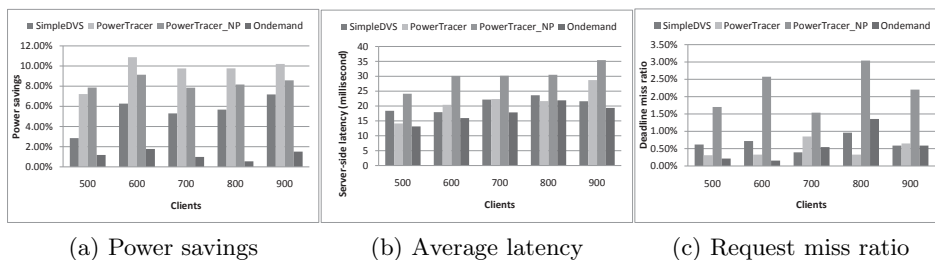


Fig. 21: A comparison of four power management algorithms. The x-axis represents the number of clients set by RUBiS client emulator. Fig. 21a illustrates the total system power savings compared to the baseline. Fig. 21b illustrates the average server-side latency of all requests. Fig. 21c illustrates the deadline miss ratio of all requests and the blank represents zero, we set deadline to 0.2 second for RUBiS read\_write mixed workload.

for RUBiS read\_only workload. For PowerTracer, we set the server-side latency threshold as  $5 \times \overline{SL}$ , where  $\overline{SL}$  is (0.15625, 9.207284, 13.67607, 11.8227, 28.14555, 22.5599, 22.7111, 22.50751) milliseconds, and set the number of main patterns as 3 (the total number of patterns is 8). Fig. 21 (a), (b), and (c) present the total system power savings, the request deadline miss ratios, and the average server-side latencies gained by the four algorithms, respectively, when the number of clients varies from 500 to 900. In these experiments, we set the number of the main patterns to 3 and the latency threshold to  $5 \times \overline{SL}$ . We can see that PowerTracer or PowerTracer\_NP gains the highest power saving. PowerTracer outperforms PowerTracer\_NP except when the number of clients is 500. The Ondemand governor has the lowest power reduction. When the number of clients is 600, PowerTracer achieves the maximum power saving of 10.88% compared to the baseline, which is about 74% better than SimpleDVS and about 515% better than the Ondemand governor in terms of power saving.

Fig. 22 (a), (b), and (c) illustrate how the performance varies when the reference inputs in PowerTracer—the latency threshold varies. For this set of experiments, we set the number of clients to 500 and select three main patterns out of the top eight patterns. For the top eight patterns,  $\overline{SL}$  is (0.15625, 9.207284, 13.67607, 11.8227, 28.14555, 22.5599, 22.7111, 22.50751) milliseconds. The results indicate that when the latency threshold is  $5 \times \overline{SL}$ , PowerTracer is only about 8% better than that when the latency threshold is  $3 \times \overline{SL}$  in terms of the power saving. Meanwhile, under these two configurations, their server-side latencies and their deadline miss ratios are close to each other.

Fig. 23 (a), (b), and (c) present the performance of PowerTracer when a different number of main patterns is chosen out of the top eight patterns. For these experiments, we set the number of clients to 500 and the latency threshold to  $5 \times \overline{SL}$ . Our results show that for the Read\_write mixed workload, PowerTracer achieves the optimal results, i.e., the higher power saving, the lower miss ratio,

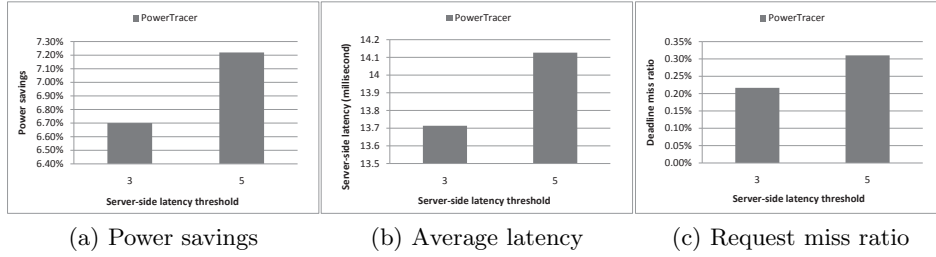


Fig. 22: Three evaluation metrics of PowerTracer under two latency thresholds. The x-axis represents two latency threshold levels. Fig.22a illustrates the total system power savings compared to the baseline. Fig.22b illustrates the average server-side latency of all requests. Fig.22c illustrates the deadline miss ratio of all requests, we set deadline to 0.2 second as above.

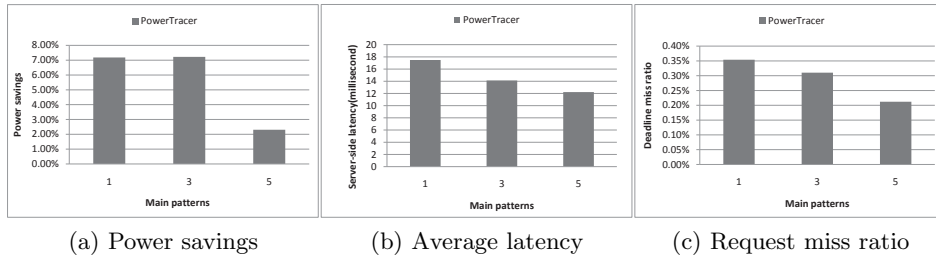


Fig. 23: Three evaluation metrics of PowerTracer with different number of main patterns. The x-axis represents the chosen number of the main patterns. Fig.23a illustrates the total system power savings compared to the baseline. Fig.23b illustrates the average server-side latency of all requests. Fig.23c illustrates the deadline miss ratio of all requests, we set deadline to 0.2 second as above.

and the lower server-side latency when we set the number of main patterns to three.

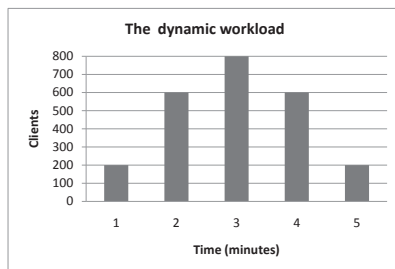


Fig. 24: The dynamic workload. The x-axis represents the running time of this experiment, and the y-axis represents the varying number of clients.

**Dynamic workload.** Based on RUBiS read\_write mixed workload, we emulate a dynamic workload by modulating the number of clients as Fig. 24 shows. The number of clients changes per each minute. For this experiment, we only deploy a service instance on a three-node platform composed of Nodes A, B, and C as shown in Fig. 5.

Fig. 25 (a), (b), and (c) present the total system power savings compared to the baseline, the average server-side latencies, and the request deadline miss ratios of the four algorithms, respectively. In these experiments, we set the number of the main patterns to 3 and the latency threshold to  $5 \times \overline{SL}$ . The deadline is set to 0.2 second. During the period when the number of clients is 200, the  $\overline{SL}$  is (2.66, 34.66, 50.07, 57.40, 39.47, 40.24, 43.82, 60.77). During the period when the number of clients is 600, the  $\overline{SL}$  is (0.09, 47.62, 45.97, 49.59, 79.91, 57.25, 50.92, 50.99). During the period when the number of clients is 800, the  $\overline{SL}$  is (0.10, 48.01, 44.21, 62.43, 82.50, 65.35, 51.61, 41.04). We can see that PowerTracer and PowerTracer\_NP (another version of PowerTracer without classifying major causal path patterns) achieve the much higher power saving than SimpleDVS and Ondemand. Though PowerTracer\_NP gains 1.1% higher power saving than PowerTracer, we can see from Fig. 25c that PowerTracer\_NP also has higher request deadline miss ratio than PowerTracer.

**Multi-service-instance experiment.** To demonstrate that our system is applicable on multi service instances, we deploy two service instances on the nine-node platform. We use RUBiS read\_only workload and set the number of clients to 500. Each workload also includes three stages, and the parameters are the same as those in Section 4.3. The deadline is set to 0.5 second. For PowerTracer, we set the server-side latency threshold as  $3 \times \overline{SL}$  milliseconds. The  $\overline{SL}$  for the first service instance is (2.58, 20.59, 25.89, 63.87, 74.38, 63.07, 66.02), and the

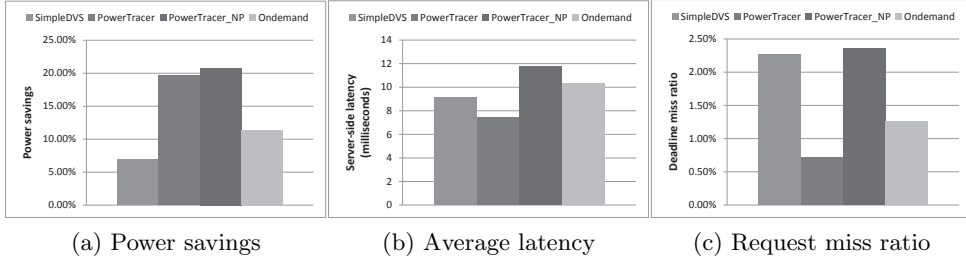


Fig. 25: A comparison of four power management algorithms in dynamic workload.

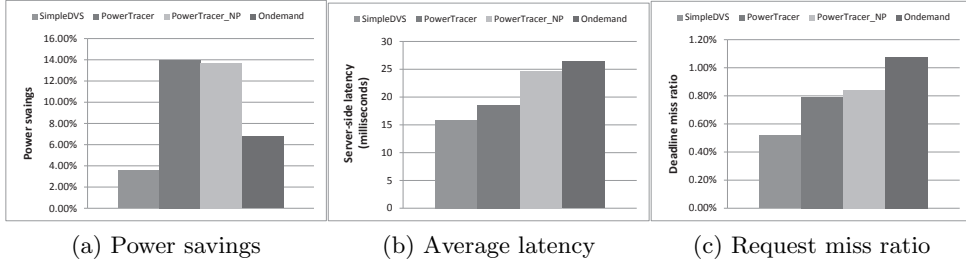


Fig. 26: A comparison of four power management algorithms with multiple service instances.

$\overrightarrow{SL}$  for the second service instance is (3.88, 30.88, 38.83, 95.80, 111.57, 94.60, 99.03). The number of main patterns is set to 5, while the total number of main patterns is 7. Please note that two service instances are deployed on nodes of different hardware configurations, and hence two service instances have different server-side latency thresholds.

Fig. 26 (a), (b), and (c) present the total system power savings compared to the baseline, the average server-side latencies, and the request deadline miss ratios of the four algorithms, respectively. The results show that PowerTracer reduces power consumption by 13.98%, the highest among all the four algorithms. Moreover, PowerTracer achieves the lowest request deadline miss ratio, which is below 0.80%.

Note that PowerTracer is based on our previous work PreciseTracer [43], in which we have demonstrated how to improve the scalability of the tracer through two mechanisms: tracing on demand and sampling. In addition, our experimental results in [43] show that PreciseTracer achieves fast responsiveness, and imposes negligible impacts on the throughput and the average response time of services, like RUBiS. Therefore, PowerTracer is promising in scalability.

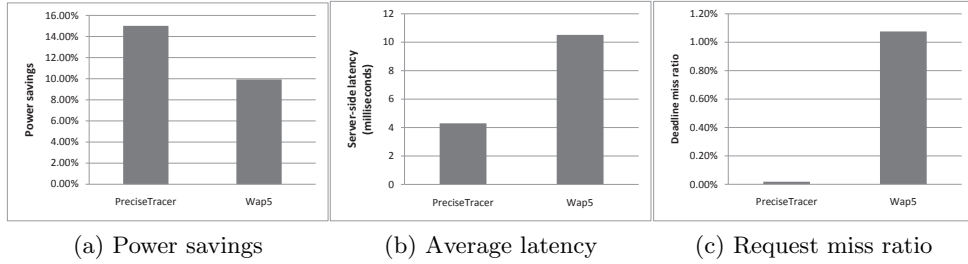


Fig. 27: A comparison of two different tracer: PreciseTracer vs WAP5.

**WAP5 v.s. PreciseTracer.** We further compare two tracers: WAP5 and PreciseTracer. As shown in [43], the accuracy of PreciseTracer is higher than that of WAP5. We use the performance data of each tracer to control power consumption of three-tier systems, and compare the total system power savings, the average server-side latencies, and the request deadline miss ratios. In the experiment, we use a three-tier platform composed of Nodes D, E, and F as shown in Fig. 5. We use the same workload as shown in Section 4.3. For PowerTracer, the server-side latency threshold and the number of main patterns are the same as those in Section 4.3.

Fig. 27 (a), (b), and (c) show the system power savings compared to the baseline, the average server-side latencies, and the request deadline miss ratio using the tracers of PowerTracer and WAP5, respectively. From Fig. 27a, we can see that PowerTracer equipped with PreciseTracer can save power by 15.01%, which is 1.51 times of that equipped with WAP5. Moreover, PowerTracer equipped with PreciseTracer has much lower request deadline miss ratio and server-side latency than that equipped with WAP5. Since WAP5 accepts imprecision of probabilistic correlations and uses the imprecise performance information, PowerTracer equipped with WAP5 gains low power efficiency.

#### 4.4 Discussion

For two different workloads of RUBiS and RUBBoS, the experiment results show that PowerTracer or PowerTracer\_NP outperforms its peers [7] [17], indicating that request tracing can improve the accuracy of DVFS control. At most of time, PowerTracer outperforms PowerTracer\_NP. This implies that monitoring the performance data of main patterns, instead of an average one, can also improve the accuracy of DVFS control. However, the optimal number of main patterns depends on different workloads, and we also observe that setting a higher latency threshold under a certain limit in PowerTracer improves power savings, the details of which can be found in our technical report [50].

**Potential for power saving.** So far, in most of commercial servers, CPU is the only energy proportional component, and our work is also confined by this limitation. Barroso *et al.* [11] showed that four components, including CPU,

DRAM, disk, and network switches, are the main sources of power consumptions in data center. Therefore, we believe that our accurate DVFS control will play a more important role in saving power consumption when the concept of DVFS is extended to the other system components.

**Potential use.** In this paper, we only provide a reference implementation for leveraging online request tracing for accurate DVFS control, however, our approach is promising for other two power saving approaches: dynamic cluster reconfiguration [7] [21], and server consolidation [26] [25] [30], since they can also benefit from accurate performance behavior monitoring.

## 5 Related Work

We summarize the related work from five perspectives.

**DVFS in server clusters.** Probably closest to our work is [7] by Horvath et al. They proposed a coordinated distributed DVS policy based on feedback controller for three-tier web server systems. However, this work fails to propose accurate DVFS control algorithms for reasons explained in Section 4.2. Horvath et al. [1] proposed a multi-mode energy management for multi-tier server clusters, which exploited DVS together with multiple sleep states. Horvath et al. [2] invented a service prioritization scheme for multi-tier web server clusters, which assigned different prioritized based on their performance requirements. Chen et al. [10] developed a simple metric called *frequency gradient* that allows prediction of the impact of changes in processor frequency on the end-to-end transaction response times of multi-tier applications.

**Dynamic cluster reconfigurations.** Rajamani et al. [21] improved energy efficiency by powering down some servers when the desired quality of service can be met with fewer servers. Elnozahy et al. [19] used *request batching* to conserve energy during periods of low workload intensity. Facing challenges in the context of connection servers, Chen et al. [3] designed a server provisioning algorithm to dynamically turn on a minimum number of servers, and a load dispatching algorithm to distribute load among the running machines.

**Virtual machine based server consolidation.** Dhiman et al [23] indicated that co-scheduling VMs with heterogeneous characteristics on the same physical node is beneficial from both energy efficiency and performance point of view. Wang et al. [18] proposed Virtual Batching, a novel request batching solution for virtualized servers with primarily light workloads. Wang et al. [25] proposed Co-Con, a cluster-level control architecture that coordinates individual power and performance control loops for virtualized server clusters. Padala et al. [29] developed an adaptive resource control system that dynamically adjusts the resource shares to individual tiers in order to meet application-level QoS goals. In their later work [30], Padala et al. present AutoControl, a resource control system that automatically adapts to dynamic workload changes to achieve application SLOs.

**Energy efficiency of specific systems.** Leverich et al [44] present their early work on modifying Hadoop to allow scale-down of operational clusters ((i.e.

operating at reduced capacity). Tsirogiannis et al [45] experiment with several classes of database systems and storage managers, and they find that within a single node intended for use in scale-out (shared-nothing) architectures, the most energy-efficient configuration is typically the highest performing one.

**Request tracing** There are two types of request tracing: black-box [34] [14] [13] [16] or white-box [15] [40] [33] approaches, on a basis of which we can develop our systems.

## 6 Conclusion

In this paper, we proposed a generalized methodology of applying request tracing approach for energy inefficiency diagnosis and power saving in multi-tier service systems. We developed an energy-inefficiency debugging tool that pinpoints the root causes of energy inefficiency and an accurate DVFS control mechanism that combines an empirical performance model and a simple feedback controller. A request tracing tool can characterize major causal path patterns in serving different requests and capture server-side latency, especially service time of each tier in different patterns, providing guidelines for debugging energy inefficiency. With regard to power saving, the advantage of the request tracing approach is two-fold: first, it decreases the time cost of performance profiling experiments; second, it decreases the controller complexity so that we can introduce a simple feedback controller, which only relies on the single-node DVFS modulation at a time for a service instance. Based on the request tracing approach, we presented a hybrid DVFS control algorithm that combines an empirical performance model for fast modulation at different load levels and a simpler controller for adaption. We developed a prototype of the proposed system, called PowerTracer, and conducted real experiments on a three-tier platform to evaluate its performance. Our experimental results demonstrated that PowerTracer not only uncovered existing energy inefficiencies but also outperforms its peers [7] [17] in power saving. Moreover, PowerTracer equipped with PreciseTracer outperformed that equipped with WAP5, indicating that higher accuracy of request tracing leads to more power saving.

## References

1. T. Horvath, *et al.* Multi-mode energy management for multi-tier server clusters. In Proc. PACT'08.
2. T. Horvath, *et al.* Enhancing energy efficiency in multi-tier web server clusters via prioritization. In IPDPS'07, pages 1–6.
3. G. Chen, *et al.* Energy-Aware Server Provisioning and Load dispatching for Connection-Intensive Internet Services. In Proc. NSDI' 08.
4. <http://rubis.objectweb.org>.
5. J. Chase, *et al.* Managing energy and server resources in hosting centers. In Proc. SOSP'01.
6. HEATH, T., *et al.* Energy conservation in heterogeneous server clusters. In Proc. of PPOPP' 05.

7. T. Horvath, *et al.* Dynamic voltage scaling in multi-tier web servers with end-to-end delay control. *IEEE Transactions on Computers*, vol. 56, no. 4, pp. 444-458, 2007.
8. J. A. Hartigan, *et al.* A k-means cluster algorithm, *Applied Statistics*, 28, 1979, pp. 100-108.
9. Urgaonkar, B., *et al.* An analytical model for multi-tier internet services and its applications. *SIGMETRICS Perform. Eval. Rev.* 33, 1 (Jun. 2005),
10. Chen, S., *et al.* Blackbox prediction of the impact of DVFS on end-to-end performance of multitier systems. *SIGMETRICS Perform. Eval. Rev.* 37, 4 (Mar. 2010), 59-63.
11. Barroso, L. A. *et al.* The Case for Energy-Proportional Computing. *Computer* 40, 12 (Dec. 2007), 33-37.
12. P. Barham, *et al.* Using Magpie for Request Extraction and Workload Modeling, In Proc. OSDI'04.
13. E. Koskinen, *et al.* BorderPatrol: Isolating Events for Precise Black-box Tracing, In Proc. EuroSys'08.
14. S. Agarwala, *et al.* E2EProf: Automated End-to-End Performance Management for Enterprise Systems, In Proc. DSN'07.
15. M.-Y. Chen, *et al.* Pinpoint: Problems Detection in Large, Dynamic Internet Service, In Proc. of DSN'02.
16. B.-C. Tak, *et al.* vPath: Precise Discovery of Request Processing Paths from Black-Box Observations of Thread and Network Activities, In Proc. USENIX'09.
17. <http://xlife.zuavra.net/index.php/70/#cpufreq-governors>
18. Y. Wang, R. Deaver and X. Wang, Virtual Batching: Request Batching for Energy Conservation in Virtualized Servers ", In Proceedings of IWQoS 2010.
19. M. Elnozahy, *et al.* Energy conservation policies for web servers. In Proc. of USIT'04.
20. E. Elnozahy, *et al.* Energy-efficient server clusters. In Proc. Workshop on Power-Aware Computing Systems, Feb. 2002.
21. K. Rajamani, *et al.* On evaluating request-distribution schemes for saving energy in server clusters. In Proc. IEEE International Symposium on Performance Analysis of Systems and Software, pages 111-122, 2003.
22. C. Rusu, *et al.* Energy-efficient real-time heterogeneous server clusters. In Proc. 12th IEEE Real-Time and Embedded Technology and Applications Symposium, pages 418-428, 2006.
23. Dhiman, G., Marchetti, G., and Rosing, T. 2009. vGreen: a system for energy efficient computing in virtualized environments. In Proc. of ISLPED '09. ACM, New York, NY, 243-248.
24. Y. Wang, *et al.* Power-Efficient Response Time Guarantees for Virtualized Enterprise Servers. In Proc. of RTSS 08. 303-312.
25. X. Wang, *et al.* Coordinating Power Control and Performance Management for Virtualized Server Clusters, *IEEE Transactions on Parallel and Distributed Systems*, 2010.
26. D. Meisner, *et al.* PowerNap: eliminating server idle power. In Proc. of ASPLOS '09. ACM, New York, NY, 205-216.
27. X. Zhu, *et al.* What does control theory bring to systems research?. *SIGOPS Oper. Syst. Rev.* 43, 1 (Jan. 2009), 62-69.
28. N. Tolia, *et al.* Delivering Energy Proportionality with Non Energy-Proportional Systems Optimizing the Ensemble. In *USENIX HotPower*, 2008.
29. P. Padala, *et al.* Adaptive control of virtualized resources in utility computing environments. *SIGOPS Oper. Syst. Rev.* 41, 3 (Jun. 2007), 289-302.

30. P. Padala, *et al.* Automated control of multiple virtualized resources. In Proc. EuroSys'09.
31. A. Verma, *et al.* Power-aware dynamic placement of HPC applications. In Proc. ICS '08.
32. Y. Chen, *et al.* SLA Decomposition: Translating Service Level Objectives to System Level Thresholds. In Proc. of ICAC'07.
33. B. H. Sigelman, *et al.* Dapper, a Large-Scale Distributed Systems Tracing Infrastructure, Google Technical Report dapper-2010-1, April 2010.
34. P. Reynolds, *et al.* WAP5: Black-box Performance Debugging for Wide-area Systems, in Proc. WWW'06.
35. M. K. Aguilera, *et al.* Performance Debugging for Distributed Systems of Black Boxes, in Proc. SOSP'03.
36. B. Tierney, W. Johnston, B. Crowley, G. Hoo, C. Brooks and D. Gunter, *The Net-Logger Methodology for High Performance Distributed Systems Performance Analysis*, in Proc. HPDC'98.
37. E. Thereska, B. Salmon, J. Strunk, M. Wachs, M. Abd-El-Malek, J. Lopez and G. R. Ganger, *Stardust: Tracking Activity in a Distributed Storage System*, in Proc. SIGMETRICS'06.
38. P. Reynolds, *et al.* Pip: Detecting the Unexpected in Distributed Systems, in Proc. NSDI'06.
39. A. Chanda, *et al.* Whodunit: Transactional Profiling for Multi-tier Applications, SIGOPS Oper. Syst. Rev. 41, 3, 2007, pp. 17-30.
40. R. Fonseca, *et al.* X-Trace: A Pervasive Network Tracing Framework, In Proc. NSDI'07.
41. Z. Zhang, *et al.* Precise Request Tracing and Performance Debugging of Multi-tier Services of Black Boxes, In Proc. DSN'09.
42. B. M. Cantrill, M. W. Shapiro, and A. H. Leventhal. *Dynamic Instrumentation of Production Systems*, In Proc. USENIX'04 of the USENIX ATC' 06, 2006.
43. B. Sang, *et al.* Precise, Scalable, and Online Request Tracing for Multi-tier Services of Black Boxes, Technical Report, available from our homepage: <http://www.ncic.ac.cn/~zjf>
44. J. Leverich, *et al.* On the energy (in)efficiency of Hadoop clusters. SIGOPS Oper. Syst. Rev. 44, 1 (March 2010), 61-65.
45. D. Tsirogiannis, *et al.* Analyzing the energy efficiency of a database server. In Proc. of SIGMOD '10.
46. R. Fonseca, *et al.* Quanto: tracking energy in networked embedded systems. In Proc.OSDI'08.
47. A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya. 2010. Virtual machine power metering and provisioning. In Proc. SoCC '10.
48. J. Zhan, N. Sun. 2005. Fire Phoenix Cluster Operating System Kernel and Its Evaluation. In Proc. Cluster'05.
49. Linux Virtual Server, <http://www.linux-vs.org/>
50. N. Mi, *et al.* Injecting Realistic Burstiness to a Traditional Client-Server Benchmark. In Proc. ICAC' 09.
51. <http://jmob.ow2.org/rubbos.html>