CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2015; **00**:1–19
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Energy-efficient CPU frequency control for the Linux system

Michał P. Karpowicz[1,2]

[1] *NASK Research Institute, ul. Wawozowa 18, 02-796 Warsaw, Poland*
[2] *Institute of Control and Computation Engineering, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland, e-mail:* michal.karpowicz@nask.pl

SUMMARY

Efficiency of energy usage in computing systems improves, however, still not at the rate matching the climbing demand for computing capacity. To address this urging problem, computing elements of the latest generation, i.e. CPUs/GPUs, memory units, network interface cards, have been designed to operate in multiple modes with differentiated energy-consumption levels. Mode switching and high-frequency performance monitoring functions have also been exposed by co-designed abstract programming interfaces. The challenge of energy-efficient computing is to develop hardware control mechanisms taking advantage of the new capabilities. This paper aims at giving an insight into the structure of optimal energy-aware CPU frequency scaling rules. It gives a characterization of solutions to the optimal control problem of energy-efficient real-time packet inspection performed by a Linux server. A class of CPU frequency switching rules, exploiting dynamic voltage and frequency scaling mechanisms, is constructed based on experimentally identified model of server operations. The control rules are demonstrated to outperform the default CPU frequency scaling governor for the Linux kernel, both in terms of achievable power savings and service quality. Copyright © 2015 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: energy-efficient computing, optimal control, Linux, dynamic voltage and frequency scaling, dynamic programming, system identification

1. INTRODUCTION

Data centers, supporting cloud services and high performance computing, as well as networking infrastructure, composed of routers and switching elements, are known to consume an increasing amount of electrical energy. This trend is commonly believed to be shaping the development of information and communications technology (ICT). Let it be noted that over the period from 2005 to 2010 the energy consumption of data centers worldwide rose by 56%, reaching the level of approximately 1.5% of total electricity use in 2010. At the same time energy-efficiency of computing systems (FLOPS/W) has been improving*. However, what poses a problem is that the rate of improvement does not match the growth rate of demand for computing capacity, especially in Big Data applications. Much concern has also been expressed about the related carbon dioxide (CO_2) production. Recent data show that 2% of global emissions is due to the ICT industry† [1, 2, 3]. Unless radically new energy-aware solutions are introduced, both in hardware and software domain, it will not be possible to meet DARPA's 50 GFLOPS/W goal‡ by year 2020 [4, 5]. Indeed, limiting power consumption in data centers has become an important engineering problem [6, 7, 8].

*www.green500.org
†http://www.gartner.com/newsroom/id/503867
‡www.etp4hpc.eu

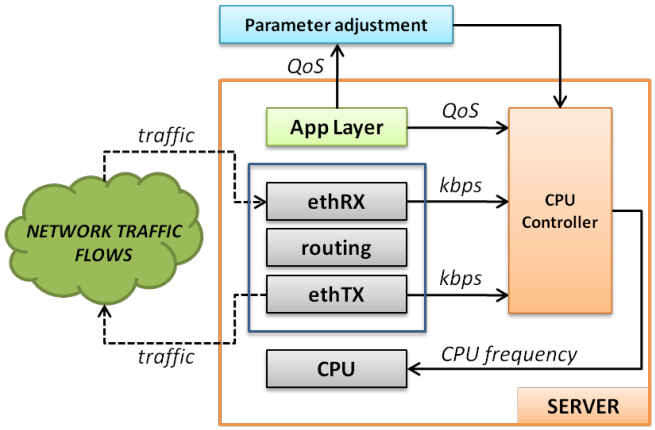

Figure 1. Control system architecture.

Average CPU utilization of a server in data center rarely approaches 100%, most of the time oscillating somewhere between 10% and 50% [9]. Over-provisioning of computing capacity, strongly correlated with the running CPU frequency, has been applied as a sound strategy to eliminate potential breakdowns caused by sudden workload fluctuations or internal disruptions, e.g. hardware or software faults. Unfortunately, although efficient from the point of view of service quality, the strategy may be a significant source of energy waste. The challenge of energy-efficient CPU frequency scaling, addressed in the present paper, is therefore to design control rules reducing power consumption subject to quality of service constraints in highly stochastic environment of a server operating system.

Dynamic voltage and frequency scaling (DVFS) mechanisms have been commonly used to reduce power usage in routers and application servers. Translation of the commands responsible for the CPU frequency control in the Linux system has been provided by the `cpufreq` kernel module [10]. The module allows to adjust performance of CPU by activating a desired control policy implemented as a CPU *governor*. Typically, the calculated control inputs are mapped to admissible performance P-states, as described by the ACPI standard[§], and passed to a processor driver (e.g. `acpi_cpufreq`). The sleeping state, or ACPI C-state, which CPU enters during idle periods, is independently and simultaneously determined by the `cpuidle` kernel module [11].

Recent versions of the Linux kernel[¶] deliver several build-in CPU governors. Two basic ones, `performance` and `powersave`, keep CPU at the highest and the lowest processing frequency, respectively. The `userspace` governor plays the role of proxy for custom system management applications, permitting user-space control of the CPU frequency. Equivalently, `cpupower` kernel utility can be used to communicate commands to the `cpufreq` module. Finally, the `ondemad` governor dynamically adjusts the CPU frequency in response to observed workload variations.

The `ondemand` governor is presently the default CPU frequency scaling mechanism for the Linux system. It performs feedback control operations according to user-defined settings, taking the CPU workload estimate as a system output. Whenever the workload exceeds user-defined upper-threshold level, the governor increases the operating frequency to the maximal feasible one. If the observed workload is below user-defined lower-threshold level, then the frequency is switched to the lowest feasible one at which the observed workload can be processed. If neither of the conditions holds, the frequency applied in the last sampling period is kept. Empirical studies have shown that the `ondemand` governor is capable of reducing power consumption without significant loss in computing performance. However, as far as the control-theoretic point of view is considered, the question of optimality of the design has not been extensively addressed.

[§] www.acpi.info
[¶] www.kernel.org

DOI: 10.1002/cpe

This paper aims at giving an insight into the structure of optimal energy-aware CPU frequency scaling rules. It presents a characterization of solutions to the control problem of energy-efficient real-time packet inspection performed by a Linux software router. A rationale behind the focus on this particular network monitoring application is twofold. First, packet-sniffing is legitimately and commonly used by network administrators for both security and management reasons. Information captured by the packet sniffers, such as `tcpdump`, allows to identify and eliminate data transmission problems, which in turn contributes to efficient network performance. Second, increasing capabilities of network traffic generators support design of system identification experiments revealing dynamics of computing operations performed under desired workload profile. Indeed, the results of system identification presented in this paper were partially obtained within the ECONET project[‖] fostering development of energy-efficient networking technologies [12, 13, 14]. The paper is intended to enrich the collection of design patterns exploiting DVFS mechanisms available in the Linux kernel. (It should be pointed out here that the design of idle logic, responsible for activating the sleeping state of a CPU core, was not directly addressed in this research.) Characterization of the structure of optimal energy-aware control policy, minimizing expected cost of network traffic processing by the Linux server, can be viewed as the main contribution of this study. Programming and engineering guidelines for the development of custom CPU controllers for the Linux kernel, compatible with the `cpufreq` module, are believed to be of general interest as well.

To give an engineering background for the presented investigations, in Section 2 the architecture of the `cpufreq` kernel module is first discussed. In particular, the design of feedback loop is described and a model of the `ondemand` control policy is derived. In Section 3 the addressed problem of optimal control is stated. Namely, a control policy is being searched for that adjusts service rate (frequency) of a processor performing real-time operations on the network packets incoming at variable random rate. The control objective is to minimize the expected total cost of operations performed over infinite control horizon. State and output equations are defined in Section 4. The main challenge here was to construct a system model that can be identified in the course of remote experiments subject to typically encountered measurement limitations, involving power-consumption sampling rate and communication delays. As a solution a linear approximation of the system is proposed including component describing conditional steady-state response of the server. In Section 5 the stated control problem is solved under additional regularity assumptions. It is demonstrated that there exists a stationary optimal control policy which is bounded from below by the best-response function minimizing short-term operational cost of the system. This function can be constructed based on identified power-consumption and performance profiles of the server. Finally, Section 6 presents the results of experiments comparing performance to the optimal control policy and the control policy of the `ondemand` governor.

Overview of results. The default CPU frequency governor for the Linux kernel, the `ondemand` governor, can be viewed as an efficient general purpose solution. However, it is possible to design a control policy which outperforms the `ondemand`-policy both in terms of achievable power savings and quality of system performance. Such a control policy is derived in the following sections as a solution to stochastic control problem. Its structure is characterized in terms of a short-term best-response function minimizing weighted sum of average costs of power-consumption and performance. The following interpretation can be given to the optimal frequency scaling rule. Whenever it is possible to process workload with the CPU frequency minimizing short-term operational cost, then this frequency should be selected. Otherwise, the system should set frequency optimizing long-term operational cost, bounded from below by the short-term-optimal frequency. This result holds for the class of systems whose performance and response profiles meet additional monotonicity assumptions. It is argued that these can be satisfied by the identified model of Linux server. The outcomes of experiments, validating the findings of theoretical studies, provide data based on which further hypotheses can be formulated.

[‖] www.econet-project.eu

DOI: 10.1002/cpe

Related work. Control engineering techniques used in the present paper can be found in [15, 16, 17, 18]. A survey of controller design patterns recently applied in various software design domains is presented in [19]. For an overview of performance analysis methodologies, metrics and energy monitoring mechanisms for data centers see [20, 21]. The architecture of modern multi-core processors has been discussed in [22] along with a description of voltage-frequency islands supporting the DVFS mechanisms.

Identification of power usage models in computing systems is currently quite problematic [23]. Inherent complexities, such as multiple cores, hidden device states, and large dynamic power components, all shaping the system's dynamics, make the system difficult to describe by means of linear models. Appropriate choice of modeling technique, both minimizing prediction errors and insensitive to measurement noises, is indeed challenging. Encouraging results of application server identification and controller design has been presented in [24]. Resource utilization is adapted to dynamic workload changes by a feedback controller taking CPU and disk usage as observed output, reference inputs there being provided by application level objectives. Computing process model identification technique, applied for the purpose of CPU frequency control design, has also been presented in [25]. Based on stochastic workload characterization, a feedback control design methodology is developed that leads to stochastic minimization of performance loss. The optimal controller is designed based on a solution to the problem of stochastic runtime performance error minimization for a selected class of applications. Supervised learning technique is used in [26] to identify the processors power dissipation and execution time per task (system output) based on characteristics of the tasks and the state of the service queue (input). A Bayesian classifier is constructed and used as a state transition model in a design of optimal voltage-frequency switching policy, this being derived with dynamic programming algorithm.

In [27] adaptive DVFS governors are proposed that take as their input the number of stalls introduced in the machine due to non-overlapping last-level cache misses, calculate performance/energy predictions for all possible voltage-frequency pairs and select the one that yields the minimum energy-delay performance. In [28] a design of performance optimizing controller is proposed for a single application server. The paper describes the use of MIMO techniques to track desired CPU and memory utilization while capturing the related interactions between CPU and memory. Decoding rate control based on DVFS, applied for multimedia-playback system design, is discussed in [29]. The proposed feedback-control system dynamically sets the minimum possible CPU frequency providing the desired throughput. In [30] a feedback controller has been proposed to solve the problem of low utilization of servers running I/O-intensive applications. To adjust CPU frequency the proposed control concept relies on energy-related system-wide feedback rather than on the CPU utilization levels. A DVFS technique that makes use of adaptive update intervals is proposed in [31]. The optimal control strategy constructed here is demonstrated to meet the workload processing deadlines given the workload arrival time. In [32] a DVFS-based control of thread execution is proposed as a technique capable of reducing memory bus and memory bank contention. The proposed policy changes the clock frequency and supply voltage of each core according to the observed number of cycles that a given thread impedes the memory accesses from other threads.

Minimization of the energy consumed by task scheduling and execution processes in grid environments equipped with DVFS mechanisms has been addressed in [33]. In particular, new scheduling objective functions have been proposed which incorporate cumulative energy consumed in the system. Dynamic resource provisioning for Big Data application scheduling is also discussed in [34]. A hybrid approach applied here may be viewed as an interesting way to include power usage minimization goal in system management processes. The problem of energy-efficient communication in data centers has been addressed in [35]. The approach taken is to minimize the total number of messages exchanged in the cluster. Finally, for a discussion of challenges of energy-efficiency benchmarking in HPC clusters see e.g. [36].

DOI: 10.1002/cpe

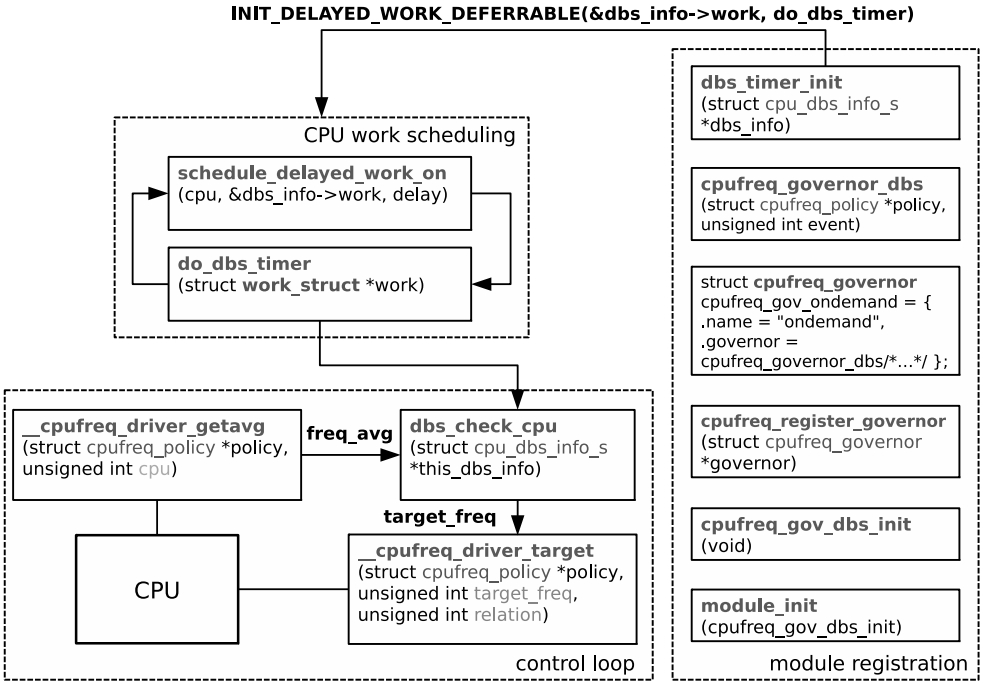

Figure 2. Architecture of the `ondemad` governor in the Linux kernel, version 3.4.

2. THE `ONDEMAND` GOVERNOR

Translation of commands for the CPU frequency control in the Linux kernel is performed by abstraction layer developed within the `cpufreq` module [10]. In the most typical setting the control inputs calculated by a given governor are mapped to admissible performance P-states, each corresponding to a feasible CPU frequency, and passed to the processor driver (e.g. `acpi_cpufreq`). In this Section the architecture of the `ondemand` governor (Fig. 2), the default Linux CPU controller, is discussed.

The `ondemand` governor is called periodically every sampling period defined in configuration file:

```
/sys/devices/system/cpu/cpufreq/ondemand/sampling_rate
```

Default sampling frequency is 1000 times the transition latency of processor. For the typical value of transition latency, which is 10^{-5} s, the sampling rate is 0.01 s. At the sampling instant the procedure presented in Listing 1 is executed. First, it estimates the workload faced by each CPU core during the last sampling period. The maximal observed workload is taken to represent the CPU workload. Second, it calculates and configures the CPU frequency for the next sampling period.

In order to estimate workload, the amount of time is measured that was used to execute instructions since the last procedure call (`busy_time`). This value is next subtracted from the total amount of time that elapsed since the last measurement was taken (`wall_time`). As a result idle time (`idle_time`) is obtained. In the Linux kernel time flow is registered by the variable named `jiffies`, storing the number of clock ticks since the system started. It is incremented at each timer interrupt, i.e. every $1/\mathtt{HZ}$ s, where `HZ` is the system variable defining (tick) rate of the system clock. The idle time of the CPU core is translated from `jiffies` to μs. Once the idle time is calculated, the workload is estimated as $\mathtt{load} = 100 \cdot (\mathtt{wall\_time} - \mathtt{idle\_time})/(\mathtt{wall\_time})$.

Further operations require the workload to be expressed in terms of measured processing frequency. To this purpose the value of variable `freq_avg` is calculated by referring to the CPU performance counters. In the case of Intel IA32 architecture the required information is provided by the ratio of values stored in registers `IA32_APERF` (0xe8) and `IA32_MPERF` (0xe7). The average

DOI: 10.1002/cpe

Listing 1 drivers/cpufreq/cpufreq_ondemand.c

```
static void dbs_check_cpu
  (struct cpu_dbs_info_s *this_dbs_info){
  /*...*/
  max_load_freq = 0
  for_each_cpu(j, policy->cpus){
    /*...*/
    cur_idle_time    = get_cpu_idle_time(j, &cur_wall_time);
    cur_iowait_time = get_cpu_iowait_time(j, &cur_wall_time);
    wall_time        = (unsigned int)(cur_wall_time-j_dbs_info->prev_cpu_wall);
    idle_time        = (unsigned int)(cur_idle_time-j_dbs_info->prev_cpu_idle);
    idle_time       -= iowait_time;
    /*...*/
    load             = 100*(wall_time - idle_time)/wall_time;
    freq_avg         = __cpufreq_driver_getavg(policy, j);
    /*...*/
    load_freq        = load * freq_avg;
    if (load_freq > max_load_freq)
      max_load_freq = load_freq;
  }
  OND(max_load_freq,up_threshold,down_differential);
}
```

processing frequency is given by the product of ratio of the counters and the maximal feasible CPU frequency [100·kHz], as presented in Listing 2. The CPU workload, `max_load_freq`, is defined by the maximum of values $\texttt{load\_freq} = \texttt{load} \cdot \texttt{freq\_avg}$, calculated per CPU core.

Once the workload is estimated, the procedure calls the control rule of the governor, denoted as `OND(...)` in the code listing above. The frequency to be applied in the next sampling period is calculated based on the value of `max_load_freq` variable and the values of switching thresholds:

```
/sys/devices/system/cpu/cpufreq/ondemand/up_threshold,
/sys/devices/system/cpu/cpufreq/ondemand/sampling_down_factor.
```

If the observed workload `max_load_freq` is higher than the upper-threshold value, `up_threshold`, then the operating frequency is increased to the maximal feasible one. On the other hand, if the observed workload is below the lower-threshold value, `up_threshold - sampling_down_factor`, then the operating frequency is set to the lowest level at which the observed workload can be supported.

Listing 2 drivers/cpufreq/mperf.c

```
unsigned int cpufreq_get_measured_perf(
  struct cpufreq_policy *policy,unsigned int cpu){
  /* ... */
  ratio  = calc_aperfmperf_ratio(&per_cpu(acfreq_old_perf, cpu), &perf);
  /* ... */
  retval = (policy->cpuinfo.max_freq * ratio) >> APERFMPERF_SHIFT;

  return retval;
}
```

Calculated target frequency, `target_freq`, is projected on the set of feasible control inputs, this being performed by the procedure in Listing 3. Depending of the value of `relation`, either the lowest upper bound or highest lower bound of the target frequency can be selected. Additional tuning paramters have also been introduced.

Let $\tilde{y}$ denote CPU workload (`load_freq`), $\mathbf{1}(x) = 1$ if $x > 0$ and $\mathbf{1}(x) = 0$ otherwise. Also, consider $[z]_V^+ \equiv \min\{v | z \leq v,\ v \in V\}$, where $V = \{\tilde{v}_0, ..., \tilde{v}_{n_P}\}$, $\tilde{v}_0 < \tilde{v}_1 < ... < \tilde{v}_{n_P}$. Given the switching thresholds $0 < \underline{\alpha} < \overline{\alpha} < 1$, evolution of frequency $\{v_k\}$ controlled by the `ondemand`

DOI: 10.1002/cpe

Listing 3 `drivers/cpufreq/freq_table.c`

```
int cpufreq_frequency_table_target(
  struct cpufreq_policy *policy,
  struct cpufreq_frequency_table *table,
  unsigned int target_freq, unsigned int relation,
  unsigned int *index) { /* ... */ }.
```

governor is described by the following equation:

$$\begin{aligned} v_k &= (\tilde{v}_{n_P} - v_{k-1})\mathbf{1}(\tilde{y}_k - \overline{\alpha}v_{k-1}) + ([\tilde{y}_k/\underline{\alpha}]_V^+ - v_{k-1})\mathbf{1}(\underline{\alpha}v_{k-1} - \tilde{y}_k) + v_{k-1}, \\ v_0 &= \tilde{v}_0. \end{aligned} \tag{1}$$

By the above definition, if the observed workload $\tilde{y}_k$ is higher than the upper-threshold value $\overline{\alpha}v_{k-1}$, then the operating frequency is increased to the maximal one, $\tilde{v}_{n_P}$. If the observed workload is below the lower-threshold value $\underline{\alpha}v_{k-1}$, then the frequency is set to the lowest level at which the observed workload can be supported, $[\tilde{y}_k/\underline{\alpha}]_V^+$. Consequently, the control policy of the governor is of the following structure:

$$\tilde{\mu}(\tilde{y}, v) = \begin{cases} [\tilde{y}/\underline{\alpha}]_V^+, & \text{if } \tilde{y} < \underline{\alpha}v, \\ \tilde{v}_{n_P}, & \text{if } \overline{\alpha}v < \tilde{y}. \end{cases} \tag{2}$$

In the remaining sections an optimal control structure is derived for a special class of problems. This structure is next compared to that of the `ondemand`-policy. It will be argued, based on the obtained theoretical and empirical results, that control policy $\tilde{\mu}$ can in fact be viewed as an efficient general purpose solution. To improve performance and reduce power consumption the characteristics of the addressed system should be taken into account in the control policy design, which is precisely the hypothesis to be explored.

3. CONTROL PROBLEM FORMULATION

The addressed problem is to find a control policy π adjusting frequency $v \in \mathbb{V}$ [GHz] of a CPU in the Linux server performing real-time inspection of network packets (of fixed length) incoming at random rate $w \in \mathbb{W} \subset \mathbb{R}_+$ [Mbps]. The control objective is to minimize the expected total cost $\sum_{k=0}^{N-1} \beta^k Q_\gamma(y_k, v_k),\ \beta \in (0,1)$, of operations generating CPU workload y_k [%] in each stage $k = 0, ..., N-1$ of the process. The system state, $x \in \mathbb{X}$ [Mbps], is assumed to represent a packet drop rate of the application layer, i.e. a packet inspection software (`tcpdump` in the case considered). State function, $f_{\boldsymbol{\theta}}$, describing dynamics of the system's response to disturbance w and control input v at a given state x, is assumed to be defined by a vector of parameters $\boldsymbol{\theta} \in \mathbb{R}^2$. Similarly, output function $h_{\boldsymbol{\vartheta}}$, describing observed output y as a function of state variable x, control input v and disturbance w, is assumed to be defined by a vector of parameters $\boldsymbol{\vartheta} \in \mathbb{R}^2$.

Given a set of model parameters $(\gamma, \boldsymbol{\theta}, \boldsymbol{\vartheta})$, the addressed control problem can be formulated for the system $\mathcal{M}(\gamma, \boldsymbol{\theta}, \boldsymbol{\vartheta}) = (Q_\gamma, f_{\boldsymbol{\theta}}, h_{\boldsymbol{\vartheta}})$ as follows:

$$\begin{aligned} \text{minimize} \quad & J_\pi(x_0) = \lim_{N\to\infty} \mathbf{E}\{\sum_{k=0}^{N-1} \beta^k Q_\gamma(y_k, v_k) \colon w_k \in \mathbb{W},\ w_k \sim \mathbf{Pr}_k\} \\ \text{s.t.} \quad & x_{k+1} = f_{\boldsymbol{\theta}}(x_k, v_k, w_k) \in \mathbb{X}, \\ & y_k = h_{\boldsymbol{\vartheta}}(x_k, v_k, w_k) \in \mathbb{Y}, \\ \text{over} \quad & \pi \in \{\{\mu_k\}_{k=0}^{N-1} | \mu_k \colon \mathbb{X} \times \mathbb{W} \to \mathbb{V}\}. \end{aligned} \tag{3}$$

Efficiency of control policy is defined by two objectives, namely, power consumption cost Q_p and packet processing cost Q_q. Both objectives are aggregated by stage cost function $Q_\gamma,\ 0 \le \gamma \le 1$,

DOI: 10.1002/cpe

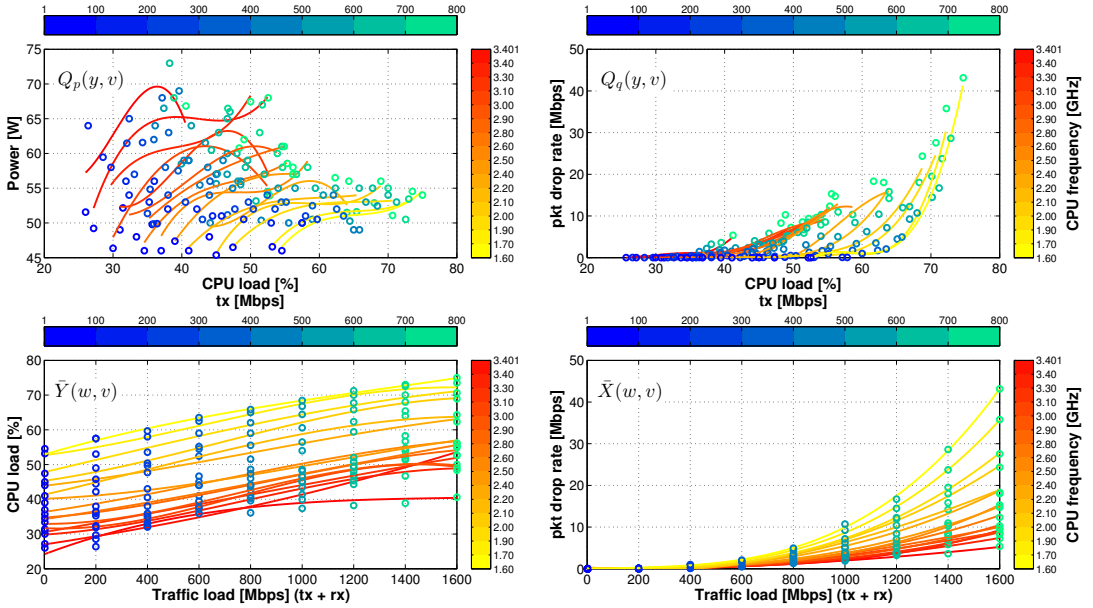

Figure 3. Performance profiles of a server.

where γ denotes weight assigned to the cost of power consumption. In the remainder of the paper the control objectives are aggregated by a weighted sum:

$$Q_\gamma(y,v) = \gamma(Q_p(y,v) + Q_p^0) + (1-\gamma)(Q_q(y,v) + Q_q^0), \tag{4}$$

where (Q_p^0, Q_q^0) define origins of objective set. Given a CPU frequency v and a packet processing workload y, function Q_p describes an average amount of power consumed by the system, whereas function Q_q denotes an average packet drop rate of the system. Two auxiliary functions are also introduced for the purpose of control policy characterization.

Definition 1
Cost function, $\hat{C}_\gamma \colon \mathbb{X} \times \mathbb{W} \to \mathbb{R}$, and best-response function, $\hat{\mu}_\gamma \colon \mathbb{X} \times \mathbb{W} \to \mathbb{V}$, are defined for $Q_p \colon \mathbb{Y} \times \mathbb{V} \to \mathbb{R}$ and $Q_q \colon \mathbb{Y} \times \mathbb{V} \to \mathbb{R}$ by a bounded and monotonic scalarizing function $Q_\gamma \colon \mathbb{Y} \times \mathbb{V} \to \mathbb{R}$ as follows:

$$\hat{C}_\gamma(x,w) = \min\{Q_\gamma(h_{\boldsymbol{\vartheta}}(x,v,w),v) \colon Q_q(h_{\boldsymbol{\vartheta}}(x,v,w),v) \le \overline{Q}_q, v \in \mathbb{V}\}, \tag{5}$$

$$\hat{\mu}_\gamma(x,w) = \arg\min\{Q_\gamma(h_{\boldsymbol{\vartheta}}(x,v,w),v) \colon Q_q(h_{\boldsymbol{\vartheta}}(x,v,w),v) \le \overline{Q}_q, v \in \mathbb{V}\}. \tag{6}$$

Function $\hat{C}_\gamma$ describes minimal value of the system operational (stage) cost that can be reached subject to quality of service bounds, $\overline{Q}_q$. Best-response function $\hat{\mu}_\gamma$ defines CPU frequency at which the value of cost $\hat{C}_\gamma$ is obtained.

4. SYSTEM MODEL IDENTIFICATION

The system model $\mathcal{M}(\gamma, \boldsymbol{\theta}, \boldsymbol{\vartheta})$ was identified based on the measurements of packet processing performance. In the course of experiments a `tcpdump` probe was forced to filter streams of UDP packets. The experiments were conducted remotely in the network of Linux software routers** consisting of seven DELL Precision T1650 nodes, each one powered by Intel Core i7-3770 processor with 16GB DDRe 1600MHz memory and Broadcom 5719 QP 1Gb NIC. Power

**PC-based Linux software router, Fedora 18, Linux kernel 3.6

DOI: 10.1002/cpe

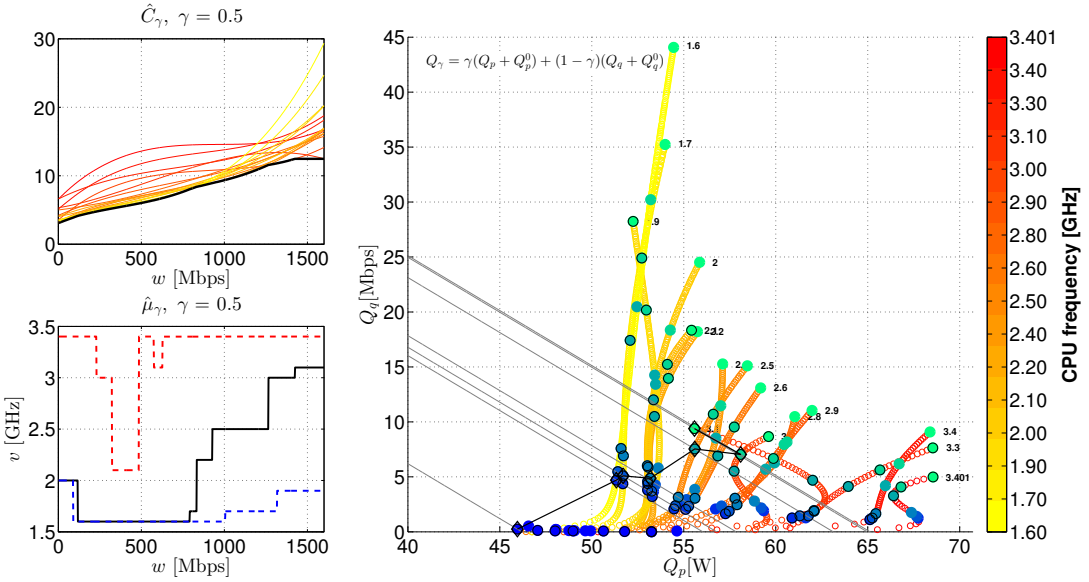

Figure 4. Approximations of $\hat{C}_\gamma$ and $\hat{\mu}_\gamma$ (left), and illustration of objective space (Q_p, Q_q) (right).

consumption monitoring was supported by a standard 1-phase electricity meter for active energy measurement equipped with RS485 interface. Measurements included processor workload, active power consumption and packet drop rate (which, to large extent, imposed the definition of system model).

The challenge of model identification was to extract maximally informative information from the measurements (i) collected remotely with relatively low frequency and (ii) to construct a sufficiently exact model of nonlinear and stochastic system dynamics. Measurement limitations came mainly from the delays in MODBUS session negotiations (power measurements), application layer sampling rate constraints and IP link delays; see e.g. [37] for a comparison of power measurement tools for computing systems. Stochastic and nonlinear dynamics result from internal task scheduling and interruption handling mechanisms of the Linux kernel networking stack, this being composed of a collection of buffers. For an overview of approaches to server model identification in the context of control system design see e.g. [38, 39, 40, 41]. Examples of models based on queuing theory, including buffer control policies, can be found e.g. in [42, 43]. Finally, let it be noticed that benchmarking capabilities are being constantly improved, especially in the Linux system environment, which sets up promising directions for further research.

In each experiment the step responses of the system were recorded. Precisely, the system operating at a fixed CPU frequency was forced to process traffic streams incoming at rates ranging from 1000 Mbps to 100 Mbps (Fig. 6). Based on the collected measurements two characteristics were next constructed, $\bar{X}$ and $\bar{Y}$, representing operating points for a fixed transmitted (tx) and received (rx) traffic workload, and CPU frequency, respectively. Figure 3 presents the cost profiles (Q_q, Q_p) and steady-state characteristics $(\bar{X}, \bar{Y})$. It follows that power consumption increases with the CPU frequency, whereas average packet drop rate decreases with the CPU frequency and increases with the CPU workload.

Figure 4 presents approximations of $\hat{C}_\gamma$ and $\hat{\mu}_\gamma$ (on the left, black lines). As can be seen, function $\hat{C}_\gamma$ is defined by a lower envelope of stage cost functions $Q_\gamma(\cdot, v)$ for a fixed frequency. Function $\hat{C}_\gamma$ is continuous, increasing and bounded. Best-response function $\hat{\mu}_\gamma$ is bounded from below by $\hat{\mu}_0$ (red line) and from above by $\hat{\mu}_1$ (blue line). It should also be noticed that $\hat{\mu}_\gamma$ is not necessarily monotonic. Objective space is presented in the right column of Figure 4. Level sets of scalarizing function Q_γ (gray lines) correspond to the frequency switching points defined by the best-response function $\hat{\mu}_\gamma$.

DOI: 10.1002/cpe

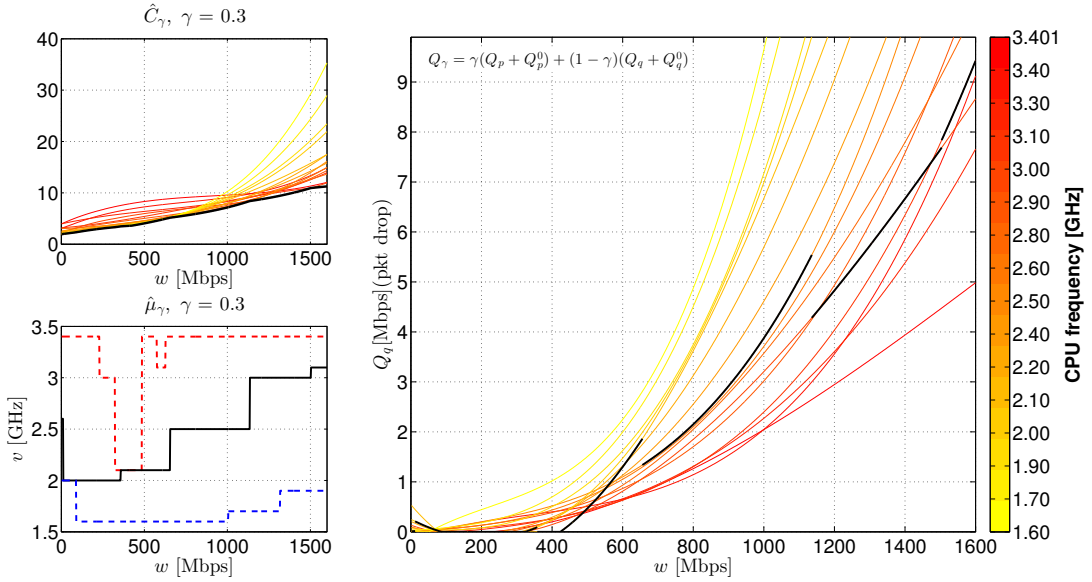

Figure 5. Illustration of saturation effect.

Figure 5 presents an interesting property of best-response function $\hat{\mu}_\gamma$, namely, a saturation condition $Q_q(y, \hat{\mu}_\gamma(x,w)) > 0$ (on the right, black line). Under low workload levels the system is capable of processing packets without losses, i.e. condition $Q_q(y, \hat{\mu}_\gamma(x,w)) = 0$ holds at CPU frequency $\hat{\mu}_\gamma(x,w)$. However, if the system load is heavy, the system is not able to process all packets, i.e. condition $Q_q(y, \hat{\mu}_\gamma(x,w)) > 0$ holds at frequency $\hat{\mu}_\gamma(x,w)$. As can be noticed, positive average drop rate is observed for CPU workload $y < 100\%$.

Given the technical limitations described above, the following simplified model was identified based on the collected measurements:

$$x_{k+1} = \max\{0, \theta_1 x_k + \theta_2 w_k + \bar{X}(w_k, v_k)\}, \tag{7}$$
$$y_k = \max\{0, \vartheta_1 x_k + \vartheta_2 w_k + \bar{Y}(w_k, v_k)\}, \tag{8}$$
$$\boldsymbol{\theta} = (\theta_1, \theta_2) \approx (0.1730, -0.0023),$$
$$\boldsymbol{\vartheta} = (\vartheta_1, \vartheta_2) \approx (0.0435, -0.0013).$$

State equation (7) and output equation (8) are both composed of two terms. The first one can be interpreted as a transient response model, approximating random behavior of the system in a neighborhood of the operating point, $x = \bar{X}(w,v)$ and $y = \bar{Y}(w,v)$. The second one, $\bar{X}(w,v)$ and $\bar{Y}(w,v)$, describes equilibrium point reached under constant load and control input. Figure 6 illustrates performance of the model, with `fit` denoting its normalized root mean square error.

5. CONTROL POLICY CHARACTERIZATION

A simplified characterization of a solution to the infinite horizon stochastic control problem (3) is derived in this section. Control policy π is constructed in two steps. First, a solution π^* to the finite-horizon formulation of the problem is obtained given the system model $\mathcal{M}(\gamma, \boldsymbol{\theta}, \boldsymbol{\vartheta})$. Second, it is shown that $\pi^* \to \pi$ as $N \to \infty$. The control policy is derived under the monotonicity assumption regarding the system response model. Namely, it is required that quality of service be increasing with the CPU frequency.

Assumption 1
Function $\bar{X}(w,v)$ is decreasing in frequency $v \in \mathbb{V}$.

DOI: 10.1002/cpe

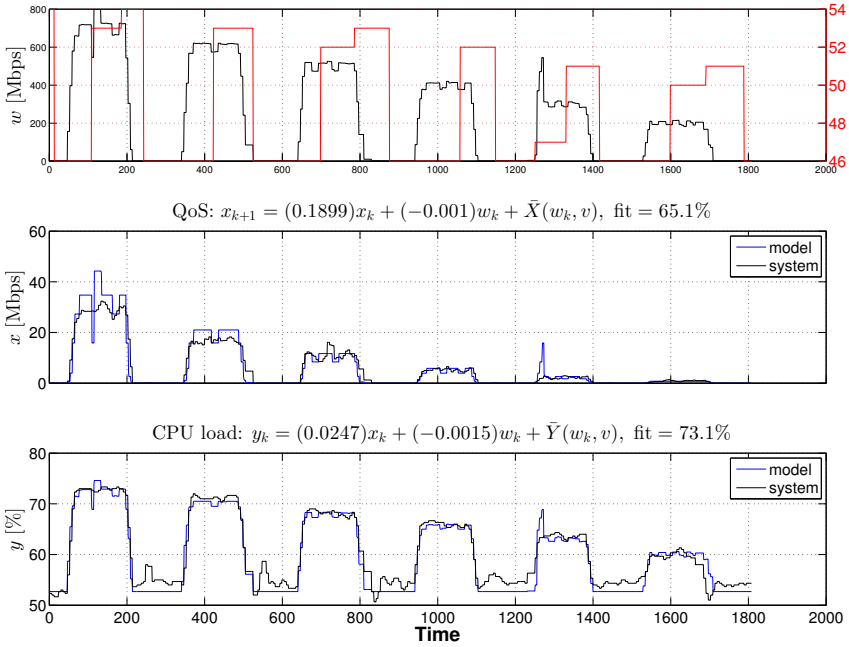

Figure 6. Illustration of step responses.

Consider the following problem:

$$\begin{aligned} \text{minimize} \quad & J_\pi(x_0) = \mathbf{E}\{\sum_{k=0}^{N-1} \beta^k Q_\gamma(y_k, v_k) \colon w_k \in \mathbb{W},\ w_k \sim \mathbf{Pr}_k\} \\ \text{s.t.} \quad & x_{k+1} = f_{\boldsymbol{\theta}}(x_k, v_k, w_k), \\ & y_k = h_{\boldsymbol{\vartheta}}(x_k, v_k, w_k), \\ \text{over} \quad & \pi \in \{\{\mu_k\}_{k=0}^{N-1} | \mu_k \colon \mathbb{X} \times \mathbb{W} \to \mathbb{V}\}, \end{aligned} \tag{9}$$

where $\mathcal{M}(\gamma, \boldsymbol{\theta}, \boldsymbol{\vartheta})$ is given by (4) and (7)-(8). Control policy π^*, being a solution to the problem, will now be derived by applying dynamic programming algorithm [18]. Specifically, the case of random disturbances being observed before the control signal is selected will be considered:

$$\begin{aligned} J_N(x_N) &= 0,\ k = N-1, N-2, ..., 0, \\ \mu_k(x_k, w_k) &= \arg\min_{v_k} \left[Q_\gamma(h_{\boldsymbol{\vartheta}}(x_k, v_k, w_k), v_k) + \beta J_{k+1}(f_{\boldsymbol{\theta}}(x_k, v_k, w_k)) \right], \\ J_k(x_k) &= \mathbf{E}\{Q_\gamma(h_{\boldsymbol{\vartheta}}(x_k, \mu_k(x_k, w_k), w_k), \mu_k(x_k, w_k)) \\ &\quad + \beta J_{k+1}(f_{\boldsymbol{\theta}}(x_k, \mu_k(x_k, w_k), w_k)) \colon w_k \sim \mathbf{Pr}_k\}. \end{aligned} \tag{10}$$

This approach is justified by the design of frequency switching mechanism supported by the `cpufreq` Linux kernel module. Indeed, the CPU frequency is calculated based on observations of idle time, this being treated as an estimate of the CPU workload, at the end of sampling period. algorithm (10) nicely describes the resulting sequence of events.

The following theorem presents a structure of control policy minimizing expected cost of workload processing in the analyzed computing system.

DOI: 10.1002/cpe

Theorem 1 (Finite horizon policy characterization)
Consider the following control rule:

$$\mu_k^*(x_k, w_k) = \begin{cases} \hat{\mu}_\gamma(x_k, w_k), & \text{if } f_{\boldsymbol{\theta}}(x_k, v_k, w_k) = 0, \\ \hat{\mu}_k^+(x_k, w_k), & \text{if } f_{\boldsymbol{\theta}}(x_k, v_k, w_k) > 0, \end{cases} \tag{11}$$
$$\hat{\mu}_k^+(x, w) = \arg\min\{Q_\gamma(h_{\boldsymbol{\vartheta}}(x, v, w), v) + \beta J(f_{\boldsymbol{\theta}}(x, v, w))\colon v \geq \hat{\mu}_\gamma(x, w)\},$$

where $k = 0, ..., N-1$. If Assumption 1 holds, then control policy $\pi^* = \{\mu_0^*, ..., \mu_{N-1}^*\}$ is an optimal solution to problem (9).

Proof
Consider dynamic programming algorithm (10). It will be demonstrated by induction that solution to problem (9) is given by equation (11). The following notation will be applied: $\Delta^v \varphi(y, \hat{v}) \triangleq \varphi(y, \hat{v} + \Delta v) - \varphi(y, \hat{v})$.

Suppose first that $J_N(x) = 0$. For $k = N-1$ we have:

$$\hat{\mu}_\gamma(x, w) = \arg\min_v Q_\gamma(h_{\boldsymbol{\vartheta}}(x, v, w), v) = \mu_{N-1}^*(x, w),$$
$$J_{N-1}(x) = \mathbf{E}\{\hat{C}_\gamma(x, w)\colon w \sim \mathbf{Pr}_{N-1}\}.$$

Furthermore, since cost function is monotonic and increasing with respect to state variable ($\partial h_{\boldsymbol{\vartheta}}/\partial x > 0$), function J_{N-1} is also monotonic and increasing:

$$\Delta^x J_{N-1}(x) = \mathbf{E}\{\hat{C}_\gamma(x + \Delta x, w) - \hat{C}_\gamma(x, w)\colon w \sim \mathbf{Pr}_{N-1}\} \geq 0.$$

For $k = N-2$ two cases must be considered. Let us take the control input $\hat{v} = \hat{\mu}_\gamma(x, w)$ and let $x_{N-1} = 0$. For any Δv we then have:

$$\Delta^v Q_\gamma(h_{\boldsymbol{\vartheta}}(x, \hat{v}, w), \hat{v}) + \beta \Delta^v J_{N-1}(f_{\boldsymbol{\theta}}(x, \hat{v}, w)) \geq 0.$$

This shows that optimality condition holds for $\hat{v}$. Suppose next that $x_{N-1} > 0$. In order to demonstrate that optimal control input $\hat{v}$ satisfies condition $\hat{v} \geq \hat{\mu}_\gamma(x, w)$, assume that $\hat{v} < \hat{\mu}_\gamma(x, w)$. By monotonicity of $f_{\boldsymbol{\theta}}$ and J_{N-1}, for $\Delta v = \hat{\mu}_\gamma(x, w) - \hat{v}$ we then obtain:

$$\Delta^v Q_\gamma(h_{\boldsymbol{\vartheta}}(x, \hat{v}, w), \hat{v}) + \beta \Delta^v J_{N-1}(f_{\boldsymbol{\theta}}(x, \hat{v}, w)) < 0,$$

which is a contradiction. Therefore, $\hat{v} \geq \hat{\mu}_\gamma(x, w)$ and:

$$J_{N-2}(x) = \mathbf{E}\{Q_\gamma(h_{\boldsymbol{\vartheta}}(x, \hat{\mu}_{N-2}^*(x, w), w), \hat{\mu}_{N-2}^*(x, w)) + \beta J_{N-1}(f_{\boldsymbol{\theta}}(x, \mu_{N-2}^*(x, w), w))\colon w \sim \mathbf{Pr}_{N-2}\}.$$

It remains to show that $\Delta^x J_{N-2}(x) \geq 0$. Several cases must be considered. Since functions $\hat{C}_\gamma$ and J_{N-1} are monotonic, for $f_{\boldsymbol{\theta}}(x + \Delta x, \hat{\mu}_\gamma(x + \Delta x, w), w) = 0$ and $f_{\boldsymbol{\theta}}(x, \hat{\mu}_\gamma(x, w), w) = 0$:

$$\Delta^x Q_\gamma(h_{\boldsymbol{\vartheta}}(x, \mu_{N-2}^*(x, w), w), \mu_{N-2}^*(x, w)) = \Delta^x \hat{C}_\gamma(x, w) \geq 0,$$
$$\Delta^x J_{N-1}(f_{\boldsymbol{\theta}}(x, \mu_{N-2}^*(x, w), w)) = 0.$$

If $f_{\boldsymbol{\theta}}(x + \Delta x, \hat{\mu}_\gamma(x + \Delta x, w), w) > 0$, then by definition of $\hat{C}_\gamma$:

$$\Delta^x Q_\gamma(h_{\boldsymbol{\vartheta}}(x, \mu_{N-2}^*(x, w), w), \mu_{N-2}^*(x, w))$$
$$= Q_\gamma(h_{\boldsymbol{\vartheta}}(x + \Delta x, \hat{\mu}_{N-2}^+(x + \Delta x, w), w), \hat{\mu}_{N-2}^+(x + \Delta x, w)) - \hat{C}_\gamma(x, w) \geq 0,$$
$$\Delta^x J_{N-1}(f_{\boldsymbol{\theta}}(x, \mu_{N-2}^*(x, w), w))$$
$$= J_{N-1}(f_{\boldsymbol{\theta}}(x + \Delta x, \hat{\mu}_{N-2}^+(x + \Delta x, w), w)) - J_{N-1}(0) \geq 0.$$

DOI: 10.1002/cpe

Assume next that $f_{\boldsymbol{\theta}}(x, \hat{\mu}_\gamma(x,w), w) > 0$. Given the structure of optimal control rule, it follows that:

$$\begin{aligned} &Q_\gamma(h_{\boldsymbol{\vartheta}}(x, \hat{\mu}^+_{N-2}(x,w), w), \hat{\mu}^+_{N-2}(x,w)) - \hat{C}_\gamma(x,w) \\ &\leq \beta \left[J_{N-1}(f_{\boldsymbol{\theta}}(x, \hat{\mu}_\gamma(x,w), w)) - J_{N-1}(f_{\boldsymbol{\theta}}(x, \hat{\mu}^+_{N-2}(x,w), w)) \right]. \end{aligned}$$

Therefore, frequency is switched from $\hat{\mu}_\gamma(x,w)$ to $\hat{\mu}^+_{N-2}(x,w)$ if an increase in stage cost can be compensated by a reduction in cost-to-go. This implies that condition $\Delta^x J_{N-2}(x) < 0$ cannot be satisfied for $f_{\boldsymbol{\theta}}(x, \hat{\mu}_\gamma(x,w), w) > 0$. To see that, assume to the contrary that $\Delta^x J_{N-2}(x) < 0$ and consider $\Delta x \to 0^+$. This however means that for some $\epsilon \to 0^+$ we have $J_{N-2}(x+\epsilon) < J_{N-2}(x)$, which contradicts optimality of $\hat{\mu}^+_{N-2}$ for x. Therefore, $\Delta^x J_{N-2}(x) \geq 0$.

Fix $k < N-2$ and let $\pi^*_{k+1} = \{\mu^*_{k+1}, ..., \mu^*_{N-1}\}$ be an applied control policy. By the arguments presented above, $\Delta^x J_{k+1}(x) \geq 0$ for μ^*_{k+1}. Let us now assume that μ^*_k satisfies condition (11). If $x_{k+1} = 0$ for $\hat{v} = \hat{\mu}_\gamma(x_k, w_k)$, then for any Δv:

$$\Delta^v [Q_\gamma(h_{\boldsymbol{\vartheta}}(x_k, \hat{v}, w_k), \hat{v}) + \beta J_{k+1}(f_{\boldsymbol{\theta}}(x_k, \hat{v}, w_k))] \geq 0.$$

Similarily, if $x_{k+1} > 0$, then optimal control input $\hat{v} \geq \hat{\mu}_\gamma(x_k, w_k)$. Indeed, if $\Delta v = \hat{\mu}_\gamma(x_k, w_k) - \hat{v}$ and $\hat{v} < \hat{\mu}_\gamma(x_k, w_k)$, then:

$$\Delta^v [Q_\gamma(h_{\boldsymbol{\vartheta}}(x_k, \hat{v}, w_k), \hat{v}) + \beta J_{k+1}(f_{\boldsymbol{\theta}}(x_k, \hat{v}, w_k))] < 0,$$

which is a contradiction. This proves optimality of control rule $\mu^*_k(x_k, w_k)$. The arguments presented above can now be repeated to prove that $\Delta^x J_k(x) \geq 0$. This completes the induction and shows that control policy $\pi^* = \pi^*_0 = \{\mu^*_0, ..., \mu^*_{N-1}\}$ is a solution to problem (9). □

Theorem 1 shows that control policy π^* is bounded from below by the best-response function $\hat{\mu}_\gamma$. Furthermore, the CPU frequency $v_k = \hat{\mu}_\gamma(x,w)$ is selected only if it does not increase operational cost of the system in the following control stages. If this condition does not hold, the CPU frequency is set to $v_k = \hat{\mu}^+_k(x,w) \geq \hat{\mu}_\gamma(x,w)$. Frequency switching to $\hat{\mu}^+_k(x,w)$ is applied at the workload level for which the corresponding increase in stage cost Q_γ is equal to the resulting reduction in cost-to-go J_{k+1}.

The following result shows that there exists a limit of cost function $J_{\pi^*}(x_0)$ generated by control policy π^* for $N \to \infty$. Furthermore, this limit is defined by a stationary control policy $\pi = \mu_\gamma$ providing solution to problem (3).

Corollary 2 (Infinite horizon control policy characterization)
For every $x_0 \in \mathbb{X}$ there exists limit:

$$J(x_0) = \lim_{N\to\infty} \mathbf{E}\{\sum_{k=0}^{N-1} Q_\gamma(h_{\boldsymbol{\vartheta}}(x_k, \mu^*_k(x_k,w_k), w_k), \mu^*_k(x_k,w_k)) \colon w_k \geq 0, w_k \sim \mathbf{Pr}_k\}, \tag{12}$$

where μ^*_k is given by (11). Function J is defined by a stationary control policy $\pi = \mu_\gamma$, where:

$$\begin{aligned} \mu_\gamma(x,w) &= \begin{cases} \hat{\mu}_\gamma(x,w), & \text{if } f_{\boldsymbol{\theta}}(x, \hat{\mu}_\gamma(x,w), w) = 0, \\ \hat{\mu}^+_\gamma(x,w), & \text{if } f_{\boldsymbol{\theta}}(x, \hat{\mu}_\gamma(x,w), w) > 0, \end{cases} \\ \hat{\mu}^+_\gamma(x,w) &= \arg\min\{Q_\gamma(h_{\boldsymbol{\vartheta}}(x,v,w), v) + \beta J(f_{\boldsymbol{\theta}}(x,v,w)) \colon v \geq \hat{\mu}_\gamma(x,w)\}. \end{aligned} \tag{13}$$

Control policy μ_γ is a solution to problem (3).

DOI: 10.1002/cpe

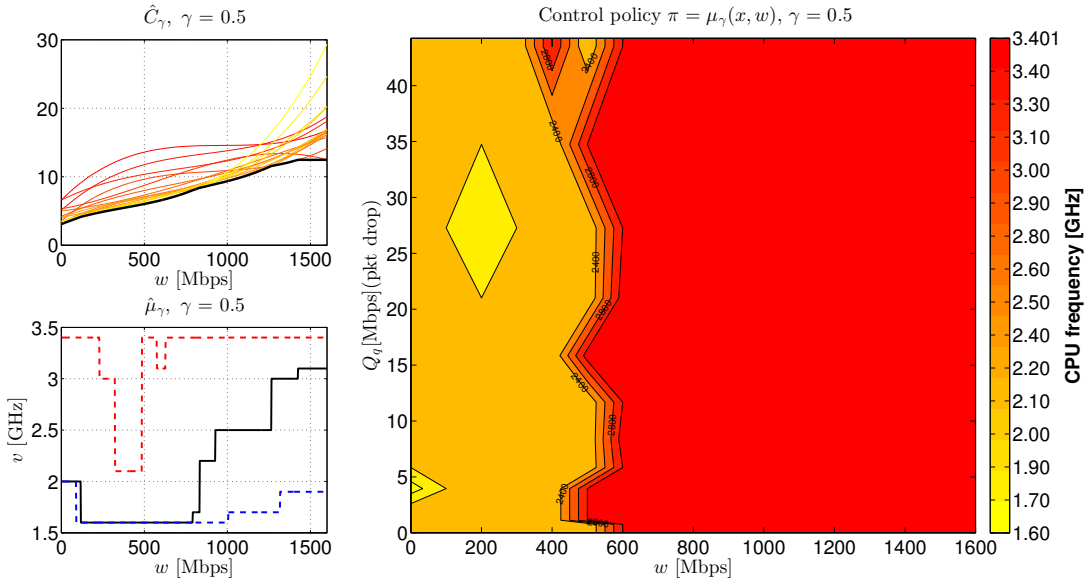

Figure 7. Example of control policy, $\mu_{0.5}$. Numerical design based on experimental data.

Proof
For control policy $\pi^* = \{\mu_0^*, ..., \mu_{N-1}^*\}$ we have:

$$J_{\pi^*}(x_0) = \mathbf{E}\{\sum_{k=0}^{K-1} \beta^k Q_\gamma(h_{\boldsymbol{\vartheta}}(x_k, \mu_k^*(x_k, w_k), w_k), \mu_k^*(x_k, w_k))\}$$
$$+ \lim_{N\to\infty} \mathbf{E}\{\sum_{k=K}^{N-1} \beta^k Q_\gamma(h_{\boldsymbol{\vartheta}}(x_k, \mu_k^*(x_k, w_k), w_k), \mu_k^*(x_k, w_k))\}$$
$$= V_K(x_0) + \lim_{N\to\infty} \mathbf{E}\{\sum_{k=K}^{N-1} \beta^k Q_\gamma(h_{\boldsymbol{\vartheta}}(x_k, \mu_k^*(x_k, w_k), w_k), \mu_k^*(x_k, w_k))\}.$$

Since $|Q_\gamma(h_{\boldsymbol{\vartheta}}(x, v, w), v)| \leq \overline{Q}_\gamma$, it follows that:

$$J_{\pi^*}(x_0) - (\beta^K \overline{Q}_\gamma)/(1-\beta) \leq V_K(x_0) \leq J_{\pi^*}(x_0) + (\beta^K \overline{Q}_\gamma)/(1-\beta).$$

Therefore, for any initial state $x_0 \in \mathbb{X}$ there exists function $J(x_0) = \lim_{K\to\infty} V_K(x_0)$ being a solution to Bellman equation. This completes the proof of optimality of $\pi = \mu_\gamma$. □

Control policy μ_γ can be given the following interpretation. Whenever it is possible to process the workload with the CPU frequency minimizing short-term operational cost, then this frequency should be selected. Otherwise, the system should set frequency optimizing long-term operational cost, bounded from below by the short-term-optimal frequency.

Control policy of $\tilde{\mu}$ of the `ondemand` governor, defined by (2), can be seen to have a structure which is similar to that of μ_γ. According to Corollary 2 this explains the observed efficiency of the `ondemand` governor. In fact, as a key conceptual difference between μ_γ and $\tilde{\mu}$, apart from the control dead zone and the observed output, one can view the frequency increasing part of the policy. The `ondemad` governor does not take into account the performance profiles of the system and applies the over-provisioning strategy to clear the instruction buffers. In contrast, control policy μ_γ exploits the knowledge of application performance and adjusts the frequency accordingly to meet the quality of service goals at minimal power-consumption cost.

DOI: 10.1002/cpe

Listing 4 User-space CPU controller

```
#!/bin/bash

declare -A mu                  # Control policy definition
# mu[xi,wj] = vk, xi = 1,...,nx, wj = 1,...,nw, vk = 1,...,nv
# ...
initControlLoop() { ... }      # Runtime parameters setup
getOutput() { ... }            # Read output feedback
ondemandPolicy() { ... }       # ondemand-policy
optimalPolicy()                # Optimal control policy
{ # ...
  target_freq=${freqs[${mu[$xi,$wi]}]}
  # ...
}
adjustControlPolicy() { ... } # Control policy adjustment
applyControlInput()            # Control input translation
{ # ...
  cpupower frequency-set -f $target_freq -r
  # ...
}
# ...
runControlLoop() {             # Main control loop
  while true
  do
    getOutput
    adjustControlPolicy
    applyControlInput
    sleep $sampling_rate
  done
  return
}
# ...
initControlLoop
runControlLoop
exit
```

6. RESULTS OF EXPERIMENTS

Performance of the control policy μ_γ was experimentally compared to that of the `ondemand`-policy $\tilde{\mu}$. For the sake of comparability both policies were implemented in the form of switching tables as a block of user-space process (app-level) operating at the sampling rate of 4 Hz. Listing 4 presents Bash code for the governor.

The experiments were conducted in the network of Linux software routers designed to demonstrate power-saving capabilities of energy-efficient network traffic engineering mechanisms (for YouTube demo see `http://youtu.be/srbmXioj538`). The routers were forced to process two network traffic profiles: noisy sine wave of adjustable frequency and noise level, and TCP/IP trace-based workload[††]. The first type of input signal was used to examine performance of the network under periodic noisy workload, commonly observed over long control horizons (days/weeks). The second one was used to examine performance of the network under workload usually observed over short timescales (seconds, minutes, hours).

Port-based source routing mechanism was applied to control traffic flows in the network. The mechanism implements functionality of the MPLS standard using the capabilities provided by the `iptables` filtering technique. In addition, each router was equipped with the `tcpdump`-based probe performing real-time packet inspection and QoS analysis. Both mechanisms, i.e. routing and packet inspection, generate CPU workload correlated with the amount of traffic forwarded by the router. As a result efficiency of the CPU frequency control contributed significantly to the overall

[††] `www.caida.org/data/`

DOI: 10.1002/cpe

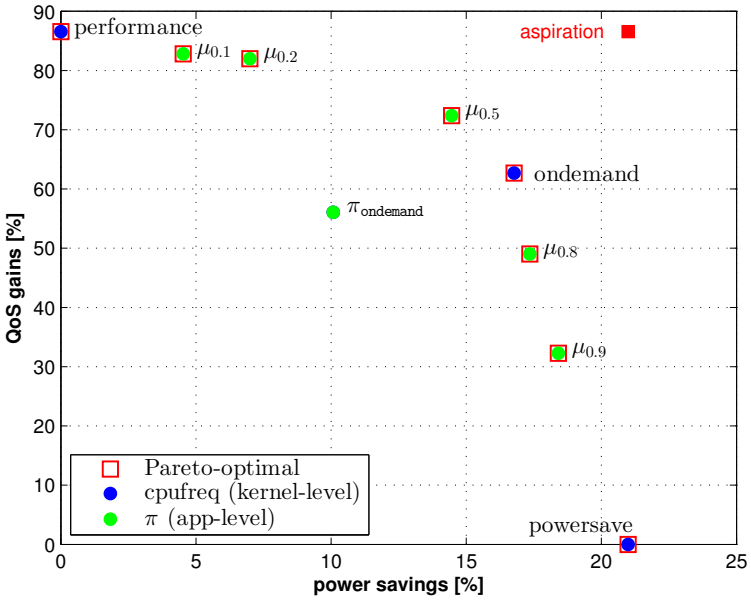

Figure 8. Comparison of control performance.

performance of the implemented traffic engineering mechanisms. Figure 8 presents the results of experiments. Examples of control input and workload trajectories are presented in Figure 9.

Control policy μ_γ, with $\gamma \in (0, 1)$, generates Pareto-optimal outcomes, outperforming those achievable by the `ondemand`-policy, $\pi_{\texttt{ondemand}}$, both in terms of power-savings and performance. Indeed, an appropriate choice of weight γ allows to obtain strictly dominating outcomes, which is the case for $\mu_{0.5}$. Given the experimental setting, the power-consumption of a server has been reduced by approximately 3% whereas the performance has been increased by approximately 16%, as compared to $\pi_{\texttt{ondemand}}$. Extreme values of γ yield outcomes similar to those generated by the `performance`- and `powersave`-policy, which is precisely what should be expected from the control policy with adjustable parameter playing the role of weight assigned to the cost of power-usage. This property opens the way for the design of adaptive controller, adjusting the parameters of control (namely, aggressiveness in power-saving) based on identified variations in process dynamics (e.g. in probability distribution of workload) or control objectives [44].

Finally, let it be noted that the user-space control process (List. 4) may itself generate a considerable CPU workload, dependent on the applied sampling rate. For this reason performance of the kernel-level implementation of the `ondemand` governor was also observed and taken into account. As can be seen, the outcomes obtained by the kernel-level governor are located near the Pareto-front (north-east boundary of the convex hull of the set of outcomes, Fig. 8) formed by the outcomes of the μ_γ policy. Based on this observation and the presented results of theoretical studies, it seems justified to expect Pareto-inferiority of the `ondemad`-policy with respect to the kernel-level implementation of μ_γ. Verification of this statement is left as a subject of further research.

7. CONCLUSIONS

New energy-aware computing elements (CPUs/GPUs, memory units, disks, network interface cards) have been designed to operate in multiple performance and idle modes of differentiated energy-consumption levels. Mode switching and performance monitoring functions have also been exposed by co-designed abstract programming interfaces (API). Quite obviously control-theoretic approach has been intensively applied to exploit these capabilities in power and software performance regulation. However, the solutions obtained so far have been limited by the lack of probes enabling

DOI: 10.1002/cpe

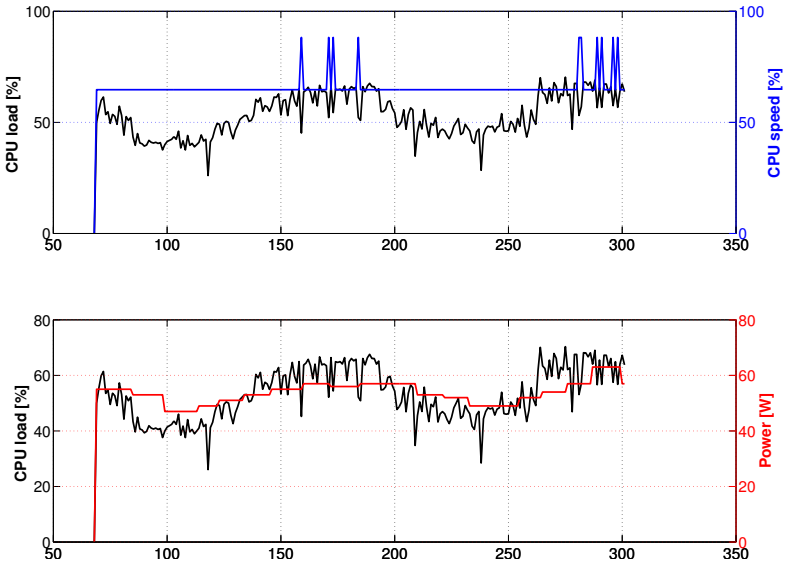

Figure 9. Example of workload (black) and control input (blue) trajectories, $\mu_{0.75}$.

measurements with desirable resolution. Much of the research effort has therefore been focused on the development of maximally informative experiments, revealing response profiles which can be useful from the viewpoint of controller design. The approach proposed in this paper proved successful in characterizing dominant (on average) features of the analyzed computing system. The results of conducted experiments strongly support this statement. Nonetheless, novel profiling capabilities of the Linux kernel can be expected to pave the way to more accurate models and even more efficient controllers. Indeed, software and hardware performance counters, such as those provided by the `perf_events` kernel module or `RAPL` CPU registers, can be sampled at rate exceeding 100 Hz. Identification of software models relating the low-level counters, such as the number of instructions per cycle, cache-misses, memory-loads, core power loads, to mention a few, for the purpose of workload forecasting and controller design seems to be a promising research topic. The structures of control rules described in this paper may be of considerable use, serving as a controller design patterns.

REFERENCES

1. Beloglazov A, Abawajy J, Buyya R. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems* 2012; **28**(5):755–768.
2. Koomey J. *Growth in data center electricity use 2005 to 2010*. Oakland, CA: Analytical Press, 2011.
3. Subramaniam B, Saunders W, Scogland T, Feng Wc. Trends in energy-efficient computing: A perspective from the Green500. *2013 International Green Computing Conference (IGCC)*, IEEE, 2013; 1–8.
4. Dongarra J, *et al.*. The international exascale software project roadmap. *International Journal of High Performance Computing Applications* 2011; **25**:3–60.
5. Kambatla K, Kollias G, Kumar V, Grama A. Trends in big data analytics. *Journal of Parallel and Distributed Computing* 2014; **74**(7):2561–2573, doi:10.1016/j.jpdc.2014.01.003.
URL http://www.sciencedirect.com/science/article/pii/S0743731514000057.
6. Zhang Q, Cheng L, Boutaba R. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications* 2010; **1**(1):7–18, doi:10.1007/s13174-010-0007-6. URL http://dx.doi.org/10.1007/s13174-010-0007-6.
7. Jing SY, Ali S, She K, Zhong Y. State-of-the-art research study for green cloud computing. *The Journal of Supercomputing* 2013; **65**(1):445–468.
8. Kim KH, Beloglazov A, Buyya R. Power-aware provisioning of virtual machines for real-time Cloud services. *Concurrency and Computation: Practice and Experience* 2011; **23**(13):1491–1505, doi:10.1002/cpe.1712. URL http://dx.doi.org/10.1002/cpe.1712.
9. Barroso LA, Hölzle U. The case for energy-proportional computing. *IEEE computer* 2007; **40**(12):33–37.
10. Pallipadi V, Starikovskiy A. The ondemand governor. *Proceedings of the Linux Symposium*, vol. 2, 2006; 215–230.

11. Pallipadi V, Li S, Belay A. cpuidle: Do nothing, efficiently. *Proceedings of the Linux Symposium*, vol. 2, 2007; 119–125.
12. Bolla R, Bruschi R, Carrega A, Davoli F, Suino D, Vassilakis C, Zafeiropoulos A. Cutting the energy bills of Internet Service Providers and telecoms through power management: An impact analysis. *Computer Networks* 2012; **56**(10):2320–2342.
13. Niewiadomska-Szynkiewicz E, Sikora A, Arabas P, Kamola M, Mincer M, Kołodziej J. Dynamic power management in energy-aware computer networks and data intensive computing systems. *Future Generation Computer Systems* 2014; **37**:284–296, doi:10.1016/j.future.2013.10.002.
URL http://www.sciencedirect.com/science/article/pii/S0167739X13002124.
14. Niewiadomska-Szynkiewicz E, Sikora A, Arabas P, Kołodziej J. Control system for reducing energy consumption in backbone computer network. *Concurrency and Computation: Practice and Experience* 2013; **25**(12):1738–1754, doi:10.1002/cpe.2964. URL http://dx.doi.org/10.1002/cpe.2964.
15. Astrom KA, Wittenmark B. *Computer-controlled systems: theory and design*. Dover Publications, Mineola, NY, 2011.
16. Soderstrom T, Stoica P. *System identification*. Prentice Hall International (UK), 1989.
17. Ljung L. *System identification*. Prentice Hall, Upper Saddle River, New Jersey, 1998.
18. Bertsekas DP. *Dynamic Programming and Optimal Control*. 3 edn., Athena Scientific, Belmont, MA, 2005.
19. Patikirikorala T, Colman A, Han J, Wang L. A systematic survey on the design of self-adaptive software systems using control engineering approaches. *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, IEEE, 2012; 33–42.
20. Benedict S. Energy-aware performance analysis methodologies for HPC architectures - an exploratory study. *Journal of Network and Computer Applications* 2012; **35**(6):1709–1719, doi:10.1016/j.jnca.2012.08.003.
URL http://www.sciencedirect.com/science/article/pii/S1084804512001798.
21. Wang L, Khan SU. Review of performance metrics for green data centers: a taxonomy study. *The journal of supercomputing* 2013; **63**(3):639–656.
22. Howard J, Dighe S, Vangal SR, Ruhl G, Borkar N, Jain S, Erraguntla V, Konow M, Riepen M, Gries M, *et al.*. A 48-core IA-32 processor in 45 nm CMOS using on-die message-passing and DVFS for performance and power scaling. *IEEE Journal of Solid-State Circuits* 2011; **46**(1):173–183.
23. McCullough JC, Agarwal Y, Chandrashekar J, Kuppuswamy S, Snoeren AC, Gupta RK. Evaluating the effectiveness of model-based power characterization. *USENIX Annual Technical Conference*, 2011.
24. Padala P, Hou KY, Shin KG, Zhu X, Uysal M, Wang Z, Singhal S, Merchant A. Automated control of multiple virtualized resources. *Proceedings of the 4th ACM European conference on Computer systems*, ACM, 2009; 13–26.
25. Wu B, Li P. Load-aware stochastic feedback control for DVFS with tight performance guarantee. *2012 IEEE/IFIP 20th International Conference on VLSI and System-on-Chip (VLSI-SoC)*, 2012; 231–236.
26. Jung H, Pedram M. Supervised learning based power management for multicore processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 2010; **29**(9):1395–1408.
27. Spiliopoulos V, Kaxiras S, Keramidas G. Green governors: A framework for continuously adaptive DVFS. *2011 International Green Computing Conference and Workshops (IGCC)*, IEEE, 2011; 1–8.
28. Gandhi N, Tilbury D, Diao Y, Hellerstein J, Parekh S. MIMO control of an Apache web server: modeling and controller design. *Proceedings of the American Control Conference*, vol. 6, 2002; 4922–4927.
29. Lu Z, Hein J, Humphrey M, Stan M, Lach J, Skadron K. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. *Proceedings of the 2002 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, ACM, 2002; 156–163.
30. Manousakis I, Marazakis M, Bilas A. FDIO: A feedback driven controller for minimizing energy in I/O-intensive applications. *Presented as part of the 5th USENIX Workshop on Hot Topics in Storage and File Systems*, USENIX, 2013.
31. Salehi ME, Samadi M, Najibi M, Afzali-Kusha A, Pedram M, Fakhraie SM. Dynamic voltage and frequency scheduling for embedded processors considering power/performance tradeoffs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 2011; **19**(10):1931–1935.
32. Kondo M, Sasaki H, Nakamura H. Improving fairness, throughput and energy-efficiency on a chip multiprocessor through DVFS. *ACM SIGARCH Computer Architecture News* Mar 2007; **35**(1):31–38.
33. Kołodziej J, Khan SU, Wang L, Kisiel-Dorohinicki M, Madani SA, Niewiadomska-Szynkiewicz E, Zomaya AY, Xu CZ. Security, energy, and performance-aware resource allocation mechanisms for computational grids . *Future Generation Computer Systems* 2014; **31**(0):77–92, doi:10.1016/j.future.2012.09.009.
URL http://www.sciencedirect.com/science/article/pii/S0167739X12001823, special Section: Advances in Computer Supported Collaboration: Systems and Technologies.
34. Vasile MA, Pop F, Tutueanu RI, Cristea V, Kołodziej J. Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing. *Future Generation Computer Systems* 2014; doi:10.1016/j.future.2014.11.019.
URL http://www.sciencedirect.com/science/article/pii/S0167739X14002532.
35. Bessis N, Sotiriadis S, Pop F, Cristea V. Using a novel message-exchanging optimization (MEO) model to reduce energy consumption in distributed systems. *Simulation Modelling Practice and Theory* 2013; **39**:104–120, doi:10.1016/j.simpat.2013.02.003.
URL http://www.sciencedirect.com/science/article/pii/S1569190X13000191.
36. Molka D, Hackenberg D, Schöne R, Minartz T, Nagel WE. Flexible workload generation for HPC cluster efficiency benchmarking. *Computer Science-Research and Development* 2012; **27**(4):235–243.
37. Diouri MEM, Dolz MF, Glück O, Lefèvre L, Alonso P, Catalán S, Mayo R, Quintana-Ortí ES. Assessing power monitoring approaches for energy and power analysis of computers. *Sustainable Computing: Informatics and Systems* 2014; **4**(2):68–82, doi:10.1016/j.suscom.2014.03.006.
URL http://www.sciencedirect.com/science/article/pii/S2210537914000171.

38. Wang Y, Wang X, Chen M, Zhu X. Partic: Power-aware response time control for virtualized web servers. *IEEE Transactions on Parallel and Distributed Systems* 2011; **22**(2):323–336.
39. Wang X, Wang Y. Coordinating power control and performance management for virtualized server clusters. *IEEE Transactions on Parallel and Distributed Systems* Feb 2011; **22**(2):245–259.
40. Lefurgy C, Wang X, Ware M. Server-Level Power Control. *The 4th IEEE International Conference on Autonomic Computing*, IEEE, 2007, doi:10.1109/ICAC.2007.35.
41. Meisner D, Gold BT, Wenisch TF. PowerNap: eliminating server idle power. *ACM SIGARCH Computer Architecture News* 2009; **37**(1):205–216.
42. Rusek K, Janowski L, Papir Z. Transient and stationary characteristics of a packet buffer modelled as an MAP/SM/1/b system. *International Journal of Applied Mathematics and Computer Science* 2014; **24**(2):429–442.
43. Gaidamaka Y, Pechinkin A, Razumchik R, Samouylov K, Sopin E. Analysis of an M|G|1|R queue with batch arrivals and two hysteretic overload control policies. *International Journal of Applied Mathematics and Computer Science* 2014; **24**(3):519–534.
44. Åström KJ, Wittenmark B. *Adaptive control*. Dover Publications, Mineola, NY, 2013.

DOI: 10.1002/cpe