Energy-Efficient Scheduling for Best-Effort Interactive Services to Achieve High Response Quality

Zhihui Du*, Hongyang Sun†, Yuxiong He‡, Yu He*, David A. Bader§, Huazhe Zhang¶
*Tsinghua National Laboratory for Information Science and Technology
Department of Computer Science and Technology, Tsinghua University, Beijing, China
†School of Computer Engineering, Nanyang Technological University, Singapore
‡Microsoft Research, Redmonds, WA, USA
§College of Computing, Georgia Institute of Technology, Atlanta, GA, USA
¶School of Information and Communication Engineering, Beijing University of Post and Telecommunication, Beijing, China

***Abstract*—High response quality is critical for many best-effort interactive services, and at the same time, reducing energy consumption can directly reduce the operational cost of service providers. In this paper, we study the quality-energy tradeoff for such services by using a composite performance metric that captures their relative importance in practice: Service providers usually grant top priority to quality guarantee and explore energy saving secondly. We consider scheduling on multicore systems with core-level *DVFS* support and a power budget. Our solution consists of two steps. First, we employ an equal sharing principle for both job and power distribution. Specifically, we present a "*Cumulative Round-Robin*" policy to distribute the jobs onto the cores, and a "*Water-Filling*" policy to distribute the power dynamically among the cores. Second, we exploit the concave quality function of many best-effort applications, and develop *Online-QE*, a myopic optimal online algorithm for scheduling jobs on a single-core system. Combining the two steps together, we present a heuristic online algorithm, called *DES* (Dynamic Equal Sharing), for scheduling best-effort interactive services on multicore systems. The simulation results based on a web search engine application show that *DES* takes advantage of the core-level *DVFS* architecture and exploits the concave quality function of best-effort applications to achieve high service quality with low energy consumption.**

***Keywords*-Energy efficiency; Scheduling algorithm; Quality of service; Multicore systems**

I. Introduction

Many large-scale interactive services—including web search, video-on-demand, financial, recommendations, map search, and online gaming—require high service quality within a short response time, which is critical for a service provider to stay competitive and win customers [15]. In addition, these services are often hosted on thousands of machines, and hence it is also crucial to reduce unnecessary energy consumption, which can reduce the operational cost and increase the profit of the service provider, not to mention its impact on the carbon footprint and global environment.

These interactive services possess two properties: they are often best-effort in nature, and must respond within a rigid deadline. (1) A best-effort request can be partially executed and will produce partial results. Given additional resources, such as processing time, the results will improve in quality. We use a quality function to map the processing time a request receives to its quality value, and the shape of such a quality function is usually non-decreasing and concave as shown in Figure 1. (2) The processing for a request is limited by time

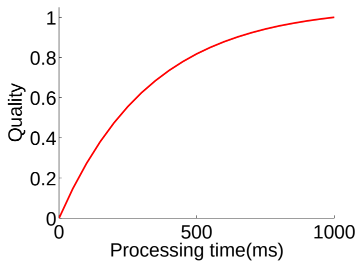


Figure 1. An example quality function that maps the processing time a request receives to its quality value.

or resource constraints, e.g., it has a deadline. For example, a web search engine often strives to answer a query within 150ms with at least 95% of the quality that could be obtained if there is no deadline constraint.

In this paper, we study the problem of scheduling best-effort interactive services on multicore systems for quality-energy tradeoff. We introduce a composite performance metric $\langle quality, energy\rangle$ to capture their relative importance in practice: Service providers usually grant top priority on service quality to guarantee user experiences; under a quality guarantee, they also want to make the system more cost effective by reducing the energy consumption. In other words, $\langle quality, energy\rangle$ first ranks the schedules based on the service quality, and then for those producing the same quality, it prefers the one with the lowest energy. Such a lexicographic order [12] respects the relative importance of quality and energy from the perspective of service providers.

We focus on scheduling best-effort interactive services on emerging processors that support *core-level DVFS* (Dynamic Voltage & Frequency Scaling). Compared to most existing processors that only support *system-level DVFS* [20][22], where all cores on a chip must share the same speed, core-level *DVFS* allows each individual core to have its own speed/power level with little additional overhead [26][27]. While the entire system is bounded by a power budget, the new architecture apparently provides more flexibility for both quality and energy optimization.

We present a heuristic online algorithm, called *DES* (Dynamic Equal Sharing), which possesses two important properties that differentiate it from other scheduling algorithms: (1) *DES* takes advantage of the core-level *DVFS* feature for better power distribution among the cores in order to handle the service demand variation of the requests. Specifically, cores with heavy load (executing long or more requests) can consume more power and run faster to achieve higher response quality while cores with light load (executing short or less requests) can spend less power and run slower to improve the energy efficiency. (2) *DES* exploits the best-effort feature of interactive services to optimize the execution of the requests. In particular, the quality of many best-effort applications is a concave function of a request's processing time [14]: response quality improves with increasing resources but with diminishing returns. *DES* runs the most profitable portions of the requests under heavy load in order to gain more quality with the same energy expenditure.

With the above intuition, *DES* divides the multicore scheduling problem into two sub-problems. The first sub-problem concerns how to distribute the requests onto the cores and how to distribute the power budget among the cores. The second sub-problem concerns how to execute the assigned requests on a single core with the given power budget. *DES* integrates our solutions to the two subproblems, and can be considered as *DES = C-RR + WF + Online-QE*. That is, it uses "*Cumulative Round-Robin*" policy and "*Water-Filling*" policy for job and power distributions, respectively, and then it applies a myopic optimal algorithm *Online-QE*, which is inspired by an offline optimal algorithm, for scheduling the requests on each individual core.

We use simulation to evaluate the performance of *DES* and to compare it with a few widely accepted scheduling algorithms. Using requests from web search engine as the driving workload, the results show that *DES* takes advantage of the core-level *DVFS* architecture and the concave quality function of best-effort applications to achieve high quality with low energy consumption. In particular, at light load, *DES* achieves about 2% more quality than other scheduling algorithms (which is significant for large-scale interactive services) with roughly the same energy as the compared scheduling algorithms. At heavy load, the quality improvement is even more as *DES* can better utilize the power budget and the energy resources. For the same target quality of 0.9, *DES* achieves up to 69% higher throughput than the other scheduling algorithms.

The contributions of this paper are the following: (1) We present a new metric $\langle quality, energy \rangle$ to evaluate the quality-energy tradeoff of online services. (2) We develop an optimal offline algorithm on single-core systems with respect to $\langle quality, energy \rangle$, and present a myopic online algorithm based on the offline optimal. (3) We propose an heuristic online scheduling algorithm to schedule requests on multicore systems with core-level *DVFS* support and a power budget. (4) We use simulation to evaluate the performance of our algorithm and validate the results on real systems.

The rest of this paper is organized as follows. Section II formulates the scheduling problem. Section III presents the single-core offline algorithm and its optimality proof, as well as a single-core online algorithm. Section IV presents our main algorithm for multicore systems, followed by its performance evaluation in Section V. Section VI discusses some related work, and Section VII concludes the paper.

II. Problem Formulation

A. Best-Effort Interactive Services

We consider the following model for best-effort interactive services [15]. There is a set $\mathcal{J} = \{J_1, J_2, \cdots, J_n\}$ of n interactive requests or jobs, and each job $J_j \in \mathcal{J}$ is characterized by a *release time* r_j, a *deadline* d_j, and a *service demand* w_j (the required number of CPU cycles). Each job can only be processed between its release time and deadline. We assume that the deadlines of the jobs are *agreeable*, that is, a job with later release time has a later deadline. This is true for many applications such as search engines and video-on-demand servers, where jobs usually have similar response time requirements.

Another important property of best-effort interactive services is the support of partial evaluation. Let p_j denote the *processed volume* (the number of processed cycles) job J_j receives during $[r_j, d_j]$, and it need not equal to its full service demand, that is, partial execution is allowed with $p_j \leq w_j$. In a schedule, if $p_j = w_j$, we say that job J_j is *satisfied*. Otherwise, the job is said to be *deprived*. A *quality function* $f : R \to R$ maps the processed volume of a job to a quality value gained by (partially) executing the job. We assume that the quality function f is *monotonically increasing* and *strictly concave*. For simplicity of analysis, we also assume that the same function f applies to all jobs in $\mathcal{J}$. Such concave quality functions are common due to the effect of diminishing returns; many best-effort interactive services such as search engines and video-on-demand servers exhibit such properties [15]. The total quality gained by executing a set $\mathcal{J}$ of jobs is then given by $Q = \sum_{J_j \in \mathcal{J}} f(p_j)$.

B. Multicore Server

We consider a multicore server supporting core-level *DVFS*. A server is composed of a set $\{M_1, M_2, \cdots, M_m\}$ of m cores, and each core supports independent *DVFS* and can have a different speed from other cores. The total power consumption of a core consists of both dynamic power and static power, i.e., $P = P_{dynamic} + P_{static}$. The dynamic power is usually a convex function of the core's speed [25], and we employ the model $P_{dynamic} = a \times s^{\beta}$, where $a > 0$ is the scaling factor and $\beta > 1$ is the power parameter. The static power $P_{static} = b$ is a non-negligible constant with $b > 0$. In this paper we do not allow an individual core to shut down when the server is running, so the static power will then become a common offset to all scheduling algorithms. Therefore, except when validating the results using a real system in Section V-G, we ignore static power in the rest of this paper and only consider the effect of dynamic power for comparing different scheduling algorithms.

For the dynamic power, the server has a total *budget* H which can be distributed arbitrarily among the cores. Let $P_i(t)$ denote the power consumption of core M_i at time t. The total power $P(t)$ of the system, which is given by the sum of power of all cores at time t, should be bounded by the power budget H, i.e., $P(t) = \sum_{i=1}^{m} P_i(t) \leq H$. The *energy consumption* E for scheduling a set $\mathcal{J}$ of jobs is the total power consumed from the release time of the first job to the deadline of the last job, i.e., $E = \int_{r_1}^{d_n} P(t)dt$. Note that the energy above refers to the one contributed by the dynamic power only while the static energy consumption in interval $[r_1, d_n]$ will be a constant for all scheduling algorithms in our model.

To reduce context-switching overhead, we consider *non-migratory* scheduling, that is, a job can be executed by any core in the server; but once started, it cannot be migrated to a different core. The total processed volume of a job is decided by the processing time the job receives as well as the speed of the core executing the job. For job J_j, let $s_j(t)$ represent the speed of the core that is processing the job at time t. If job J_j is not being processed at time t, we have $s_j(t) = 0$. The total processed volume of job J_j is given by $p_j = \int_{r_j}^{d_j} s_j(t)dt$.

C. Performance Metric and Scheduling Model

To address both quality and energy concerns, we introduce a composite performance metric – $\langle quality, energy\rangle$ – as our scheduling objective. As interactive service providers, the top priority is to offer high quality of service to improve user experiences. With a quality guarantee, they also want to make the system more energy efficient. The composite metric $\langle quality, energy\rangle$ captures precisely this objective by first ranking the schedules based on the total quality, and then for those under the same quality chooses the one that minimizes the total energy consumption. In other words, a *lexicographic order* is imposed on the $\langle quality, energy\rangle$ pair when comparing different schedules. An optimal solution with respect to this performance metric should produce the highest total quality Q among all valid schedules, and it should also consume the minimum amount of energy E among those schedules that maximize the total quality.

In this paper, we consider both offline and online scheduling for the single-core scenario. An offline model assumes that a scheduler knows complete information about all jobs including future arrivals. Although not practical in reality, it inspires the design of our online algorithms for both a single core and multicore systems. In the online model, a scheduler uses only the information of arrived jobs including their service demands, release times and deadlines without requiring any information of future jobs.

III. Algorithms for Single-core system

This section discusses scheduling for a single-core system. We first introduce an offline algorithm and prove its optimality with respect to $\langle quality, energy\rangle$. Inspired by this optimal offline algorithm, we then present a myopic optimal online algorithm, which turns out to be an important component when designing our multicore scheduling algorithm in Section IV.

A. Optimal Offline Algorithm

We present an offline algorithm, called *QE-OPT*, and prove that it produces an optimal schedule with respect to $\langle quality, energy\rangle$ under a given power budget.

Preliminaries: *QE-OPT* generates an optimal schedule in two steps: The first step determines the processing volume for each job in the job set to maximize the total quality. The second step determines the speed at which each job will be processed to minimize the overall energy.

The design of the two steps is inspired by two algorithms, namely, *Energy-OPT* [25] and *Quality-OPT* [15], respectively.[1] Each algorithm gives an optimal schedule for a simpler metric. *QE-OPT* is an amalgamation of these two algorithms, and we will show that it is optimal with respect to $\langle quality, energy\rangle$. We first give an overview of these two algorithms to help us construct and understand *QE-OPT*.

Energy-OPT uses *DVFS* to schedule jobs on a single core to minimize the energy consumption without power budget constraint. Since there is no power limit, all jobs can be satisfied and there is no need to consider partial evaluation. Two key concepts are defined for *Energy-OPT*:

- *Interval intensity*: The intensity of an interval $I = [z, z']$ is defined as
$$g(I) = \frac{\sum w_i}{z' - z},$$
where the sum is taken over all jobs in $\mathcal{J}_I = \{J_j | [r_j, d_j] \subseteq [z, z']\}$. It is also called the average speed of interval I.
- *Critical interval*: We call I^* a critical interval if I^* maximizes $g(I)$ among all intervals in $\mathcal{J}$. The set $\mathcal{J}_{I^*}$ of jobs falling in I^* is the critical group of $\mathcal{J}$. The average speed of a critical interval is called critical speed.

Since *Energy-OPT* attempts to satisfy all jobs in $\mathcal{J}$, its main intuition is to find the most intensive interval, or the critical interval I^*, to schedule first. Due to the convex nature of the power function, the most energy-efficient speed to run $\mathcal{J}_{I^*}$ would be $g(I^*)$ because the core cannot run any slower to finish all the jobs in I^* before their deadlines. After deciding the first critical interval and assigning the schedule for this critical group, *Energy-OPT* removes the interval and the critical group from $\mathcal{J}$, adjusts the release time and the deadline for other jobs that partially overlap with this interval, and then repeats the process of finding the next critical interval for the remaining jobs recursively.

Quality-OPT, on the other hand, schedules jobs on a single core with a fixed speed to maximize the total service quality. Since the core's speed cannot be changed and the jobs need to be processed before their deadlines, some jobs can only be partially evaluated when the workload is high. As with *Energy-OPT*, two important concepts are defined for *Quality-OPT*:

- *Deprived mean (d-mean) of an interval*: The *d-mean* $\tilde{p}(I)$ of an interval $I = [z, z']$ is defined as
$$\tilde{p}(I) = \frac{z' - z - \sum_{J_j \in S(I)} w_j}{|D(I)|},$$
where $S(I)$ and $D(I)$ denote the set of satisfied jobs and the set of deprived jobs in interval I, respectively. The type of each job is determined by comparing its service demand with a temporary value of *d-mean* in an iterative fashion (See [15] for details). Note that if all jobs in an interval I can be satisfied, i.e., $|D(I)| = 0$, its *d-mean* is simply defined as $\tilde{p}(I) = \infty$.

[1]*Energy-OPT* is also referred to as the *YDS* algorithm in the literature [4], and *Quality-OPT* is also called *Tians-OPT* in [15].

- *Busiest deprived interval*: An interval I^* is called the busiest deprived interval if I^* minimizes $\tilde{p}(I^*)$ among all intervals of job set $\mathcal{J}$.

Since the quality functions of all jobs are concave and identical, the intuition behind *Quality-OPT* is that it will achieve the highest quality for the jobs in I^* by satisfying all jobs in $S(I^*)$ and allocating an equal processing volume (or *d-mean*) to each deprived job. As with *Energy-OPT*, after finding the first busiest deprived interval and assigning the schedule for this group of jobs, *Quality-OPT* removes this interval and scheduled jobs, adjusts the release time and the deadline for other jobs that partially overlap with this interval, and recursively schedules the remaining jobs by searching for the next busiest deprived interval or until all jobs are satisfied with the fixed core speed.

***Algorithm QE-OPT**:* *QE-OPT* addresses a more complex problem by combining the benefits of *Quality-OPT* and *Energy-OPT*, which are responsible for maximizing quality and minimizing energy, respectively. In particular, *QE-OPT* first runs *Quality-OPT* with the maximum core speed, i.e., full power budget, to determine the processing volume for each job. This step guarantees to deliver the maximum quality. The minimum energy is then ensured by running *Energy-OPT* on top of the schedule produced by *Quality-OPT*. The following shows the two steps of *QE-OPT* in detail:

1) Apply *Quality-OPT* on job set $\mathcal{J}$ using the maximum core speed to determine the processing volume p_j for each job $J_j \in \mathcal{J}$. Then, for each job, adjust its service demand to be its processed volume, i.e., $w_j \leftarrow p_j$, without changing the release time and deadline. Call the new job set $\mathcal{J}'$.
2) Apply *Energy-OPT* on the new job set $\mathcal{J}'$ with the adjusted service demands to determine the speed s_j to execute each job $J_j \in \mathcal{J}'$ on the core.

Since both *Quality-OPT* and *Energy-OPT* produce non-preemptive schedules when the deadlines of the jobs are agreeable, the final schedule computed by *QE-OPT* also executes each job without preemption in the order of their arrivals. A straightforward implementation of *Energy-OPT* takes $O(n^3)$ time, and the complexity of *Quality-OPT* is $O(n^4)$, where n is the total number of jobs. Thus, a straightforward implementation of *QE-OPT* takes $O(n^4)$ time. In Section III-B, we show that when jobs arrive in an online manner, a simpler implementation of *QE-OPT* is possible with lower complexity.

***Optimality Analysis**:* We show that *QE-OPT* produces an optimal schedule with respect to the metric $\langle quality, energy \rangle$. Before proving its optimality, we first show that the schedule produced by *QE-OPT* is always feasible.

Theorem 1: *QE-OPT* produces a feasible schedule for any job set on a single-core system.

Proof: We prove the feasibility by showing that any critical speed used by *Energy-OPT* for job set $\mathcal{J}'$ is not more than the maximum core speed, denoted by s^*. Since the critical speed is non-increasing over critical intervals and the speed within a critical interval does not change [25], we need only consider the first critical interval I^* found by *Energy-OPT*. Suppose the speed of this interval is larger than the maximum core speed, i.e., $g(I^*) > s^*$. Consider the set of jobs in I^*. While both *Quality-OPT* with fixed speed s^* and *Energy-OPT* with variable speed are able to complete every job in I^*, *Energy-OPT* runs at speed $g(I^*)$ thus consumes more energy than *Quality-OPT*. This contradicts the fact that *Energy-OPT* produces a minimum energy schedule for any critical job group. ■

Theorem 2: *QE-OPT* produces an optimal schedule with respect to $\langle quality, energy \rangle$ on a single-core system.

Proof: Since *Quality-OPT* computes the maximum quality on a core with fixed speed, and the first step of *QE-OPT* uses the fastest core speed, *QE-OPT* gives the maximum possible quality for the job set. Moreover, since the quality function is strictly concave, according to convex optimization [6], the optimal quality is uniquely determined by the assigned job processing volumes calculated by *Quality-OPT*. From Theorem 1, we know that the same optimal quality is preserved by *Energy-OPT* without violating the power budget. Since *Energy-OPT* gives a minimum energy schedule for any critical job group, the schedule produced by *QE-OPT* also minimizes the energy among those with maximum quality. ■

B. Myopic Optimal Online Algorithm

While *QE-OPT* gives an optimal solution to our scheduling problem, it requires complete information of the jobs including future arrivals. This section presents a practical online algorithm, called *Online-QE*, based on the basic principles of *QE-OPT* but with lower complexity.

Online-QE simply computes the optimal schedule based on the current knowledge of the jobs in the system, including jobs that have arrived so far and whose deadlines have not yet expired. The algorithm is invoked in an online manner whenever some triggering events occur (see Section IV-E). Upon each invocation, *Online-QE* recomputes a new schedule using *QE-OPT*. Apparently, the overall schedule is not globally optimal for the whole set of jobs; *Online-QE* guarantees a *myopic optimal* solution for the set of available jobs at each invocation.

There is, however, one issue we need to address to even guarantee myopic optimal schedule on the set of ready jobs. That is, a new invocation can be requested when the currently running job is not yet completed according to the previous schedule. We address this issue by readjusting the release time of the job to ensure correct calculations of both *Quality-OPT* and *Energy-OPT*. Let t denote the current time when *Online-QE* is invoked. Let $\mathcal{J}_t$ denote the set of ready jobs at time t, and let $J_1 \in \mathcal{J}_t$ denote the job that is currently running, and suppose that the processed volume of J_1 so far is $\bar{p}_1$. Before invoking the algorithm, we first adjust the release time of J_1 to be $r_1 = t - \bar{p}_1/s^*$, where s^* denotes maximum core speed under the given power budget, and adjust the release time of any other job $J_j \in \mathcal{J}_t$ to be $r_j = t$. Keeping all the other attributes of the jobs unchanged, call the new job set with adjusted release times $\mathcal{J}'_t$. *Online-QE* performs two steps:

1) Apply *Quality-OPT* on $\mathcal{J}'_t$ using the maximum core speed to determine the processing volume p_j for each job $J_j \in \mathcal{J}'_t$. Then, for the currently running job J_1, readjust its release time to be the current time t, and

adjust its service demand to be $w_1 = p_1 - \bar{p}_1$. If $w_1 \leq 0$, remove J_1 from $\mathcal{J}'_t$. For any other job $J_j \in \mathcal{J}'_t$, adjust its service demand to be $w_j = p_j$. Call the new job set $\mathcal{J}''_t$.

2) Run *Energy-OPT* on the new job set $\mathcal{J}''_t$ to determine the speed s_j to execute each job $J_j \in \mathcal{J}''_t$ on the core from current time t onwards.

Following the analysis of *QE-OPT*, it is not hard to see that *Online-QE* gives a feasible and myopic optimal schedule for the set of ready jobs at each invocation. In particular, the adjustments of the jobs' release time before both steps of the algorithm ensure the correct calculations of the optimal processing volume and the speed for each job, assuming that there is no future arrivals and the computed schedule is only applied to the jobs from the current time onwards.

Moreover, the computational complexity of *Online-QE* can be reduced to $O(n^2)$ at each invocation, where n denotes the total number of ready jobs when the algorithm is invoked. This is because all jobs except the currently running one can now be considered as having the same release time. Therefore, both *Energy-OPT* and *Quality-OPT* need only consider the $O(n)$ intervals instead of all $O(n^2)$ possible intervals.

Finally, the schedule produced by *Online-QE* for the entire job set is also non-preemptive and it works even when the power budget of the core can change at each invocation. On a multicore system, such scenario can happen with dynamic power distribution across cores. This allows our single-core algorithm to be employed as an independent procedure in scheduling multicore systems as described in Section IV-D.

IV. Algorithm for Multicore System

This section discusses scheduling on multicore systems with a dynamic power budget that can be distributed arbitrarily among the cores. The offline version of this problem is NP-hard, since a simpler problem where the objective is to minimize the energy consumption without a power budget has been shown to be NP-hard [1]. In this section, we focus on the online version of the problem by presenting a heuristic algorithm, called *DES* (Dynamic Equal Sharing). We evaluate the performance of *DES* in Section V.

A. Overview of DES

Two key aspects of our multicore scheduling algorithm are to distribute the ready jobs onto the cores and to distribute the power budget among the cores. These are two challenging tasks because we need to consider their impacts on both quality and energy. To achieve good job and power distributions, *DES* applies the principle of *equal sharing*, which turns out to be an effective strategy to address both quality and energy concerns. By doing so, the algorithm effectively divides the multicore scheduling problem into many independent single-core problems, and the sub-problem for each individual core can then be solved by using the *Online-QE* algorithm presented in Section III-B.

B. Job Distribution Policy

To maximize quality, it is desirable to distribute the jobs evenly among the cores so as to balance the workload on each core. This will minimize job contention so that more jobs can be potentially satisfied and more work can be completed, which directly contributes towards higher total quality. To minimize energy with a convex power function, we want to run each job as slowly as possible while meeting its deadline. Apparently, having less job contention on each core enables us to achieve this goal. Thus, even distribution of the jobs also helps reduce the overall energy consumption.

With this intuition, we employ a simple and efficient "*C-RR*" (Cumulative Round Robin) policy to equally share the ready jobs among the cores at each invocation of the algorithm. The policy is cumulative in the sense that it starts to assign the ready jobs from the core where the last job distribution cycle stops. Compared with non-cumulative *RR*, which distributes jobs starting from the same core at each invocation, "*C-RR*" can lead to much balanced job distribution in the long run. Since our algorithm is non-migrating, once a job is distributed to a core, it will stay on the core till completion or when it is discarded due to partial evaluation.

C. Power Distribution Policy

After job distribution, each core can produce its own schedule by using the *Online-QE* algorithm. However, the total requested power from all cores may exceed the total power budget of the system. In such case, we need an effective policy to distribute the power.

Since the power function is convex, the sum of speeds of all the cores is maximized when the power is equally shared among them. This will maximize the total amount of work that can be done in a time unit, thus potentially improves the chance to complete more work and achieve higher quality. Therefore, the key insight for power distribution is, once again, equal sharing. Due to workload variations, however, a core may need less power than the designated equal share. In this case, it will be more energy-efficient to provide only what the core demands and assign the remaining power budget to other cores with higher power demands. Based on this intuition, we propose a dynamic power distribution policy, called "*WF*" (Water-Filling).

As the name suggests, "*WF*" policy is analogous to filling water into containers of different heights. Here, the total amount of water represents the total power budget, and the height of a container represents the requested power of the corresponding core. Imagine that the bottom of the containers are connected so that water can flow freely among them. "*WF*" policy is equivalent to pouring water from the top of the containers until all are fully filled or there is no more water. Figure 2 illustrates this process on a 4-core system. In this example, core 4 requires less than the equal share so that it gets what it demands. Cores 1, 2 and 3, having higher demands, equally share the remaining power.

More formally, let h_i denote the requested power from core M_i and let a_i denote the assigned power for core M_i. Initially, set $a_i = 0$ for all $1 \leq i \leq m$. Given a total power budget H, the "*WF*" policy distributes the power to the cores iteratively as follows:

1) Suppose there are m' unsatisfied cores whose requested power remains nonzero. If $m' = 0$, the policy terminates. Otherwise, let $h_{\min}$ denote the minimum requested power among these cores.

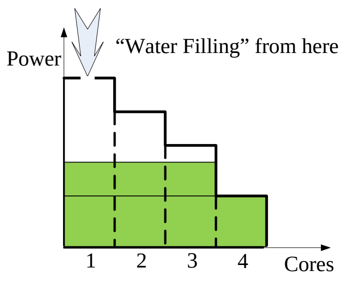


Figure 2. "*WF*" policy for distributing power budget among cores.

2) If $h_{\min} \cdot m' \geq H$, then evenly distribute the remaining power budget to these cores, i.e., $a_i = a_i + H/m'$, and terminate. Otherwise, add $h_{\min}$ power to each of these cores, i.e., $a_i = a_i + h_{\min}$. Then, update the remaining power budget $H = H - h_{\min} \cdot m'$ and the request of each core $h_i = h_i - h_{\min}$. Go back to Step 1).

D. DES Algorithm

We now describe our *DES* (Dynamic Equal Sharing) algorithm. The basic idea is to use the job and the power distribution policies to divide one global multicore scheduling problem into several local single-core problems. Then, we employ the single-core online algorithm presented in Section III-B on each individual core. In a sense, *DES* can be considered as *DES* = *C-RR* + *WF* + *Online-QE*, which presents a package of solutions, and each component addresses a different aspect of a complex scheduling problem.

DES is invoked to recompute the schedule whenever some triggering events occur (See Section IV-E). The input is a set of ready jobs that have arrived since the last invocation, and the output is a schedule that defines when a job runs on which core and at what speed. Suppose *DES* is invoked at time t, and the following shows the scheduling steps:

1) *Ready-job-distribution*: Employ the "*C-RR*" policy to distribute the ready jobs onto the cores. Adjust the release time of all jobs to the current time t.
2) *Budget-free-independent-core-scheduling*: Assuming unlimited power, use *Energy-OPT* to calculate a schedule on each core. Let $P_i'(t)$ denote the power required at time t in this schedule for core M_i. Since all jobs are now released at current time t, according to *Energy-OPT*, the calculated power on each core decreases monotonically over time, i.e., $P_i'(t') \leq P_i'(t)$ for all $t' \geq t$. If the total power at time t can meet the budget, i.e., $P'(t) = \sum_{i=1}^{m} P_i'(t) \leq H$, we can complete all jobs under the power budget and the algorithm terminates; otherwise, proceed to Step 3).
3) *Dynamic-power-distribution*: Employ the "*WF*" policy to distribute the total power budget H among the cores based on the required power $P_i'(t)$ of each core M_i at time t. Let $P_i(t)$ denote the distributed power to M_i.
4) *Budget-bounded-independent-core-scheduling*: For each core M_i, employ the single-core *Online-QE* algorithm with power budget $P_i(t)$ to calculate a schedule.

Steps (1) and (3) distribute the jobs and the power budget to the cores respectively, and Steps (2) and (4) schedule the jobs on each individual core. In particular, Step (2) first calculates an optimistic schedule for each core without assuming a power budget. If such a schedule does not violate the power constraint, all jobs can be satisfied and we are done; otherwise, Step (3) resolves the power competition among the cores and Step (4) produces a feasible schedule that meets the power budget with some partially evaluated jobs.

E. Triggering Events for Grouped Scheduling

Instead of using *Immediate Scheduling* (IS), which recomputes a new schedule whenever a job arrives, we employ *Grouped Scheduling* (GS) for *DES*. With GS, arriving jobs are first stored in a waiting queue, and they are only assigned to cores when certain scheduling events are triggered. GS reduces scheduling overhead; it also helps to improve the quality of scheduling decision by considering multiple requests together. The following shows the three types of triggering events we use in this paper:

- *Quantum trigger*: The scheduler is triggered periodically after each scheduling quantum.
- *Idle-core trigger*: An idle core triggers the scheduler to start assigning more jobs.
- *Counter trigger*: The scheduler is triggered when certain number of jobs have been accumulated in the queue.

V. Performance Evaluation

We evaluate the performance of our *DES* algorithm on different architectures and compare it with different algorithms. First, the simulation results show that *DES* can take advantage of both the modern hardware architecture and the feature of the best-effort applications to achieve high quality and low energy. Second, even for the same architecture and application, *DES* outperforms other widely accepted scheduling algorithms. The sensitivity study then shows how the quality function, power budget and number of cores can affect the performance of our algorithm. Finally, validation on a real system shows the accuracy of the simulations, thus suggesting the reliability of our results.

A. Evaluation Methodology

Our *DES* algorithm targets at best-effort interactive services on multicore architectures that support fine-grained core-level *DVFS*. In our evaluation, we simulate such an architecture and use the web search engine as an example to model interactive services. We evaluate the performance of *DES* from the following five aspects: (1) Does it take advantage of the core-level *DVFS* to achieve high quality with low energy? (2) Given the same architecture, does *DES* take advantage of the partial evaluation feature of best-effort applications (e.g. web search) to achieve higher quality? (3) Based on the same architecture and application, does *DES* improve quality and save energy compared with other scheduling policies? (4) How is the performance of *DES* affected by different quality functions, amount of power budget, number of cores and discrete speed scaling in the system? (5) How reliable are the simulation results when validated on real systems with realistic power models? We elaborate our evaluation methodologies of the five sets of experiments as follows.

Taking Advantage of Modern Multicore Architectures: We implement *DES* on three processor architectures with different levels of *DVFS* support. The aim is to evaluate both quality and energy of *DES* on these architectures, and to show its advantage on the core-level *DVFS* systems.

- *No-DVFS*: This architecture cannot support any *DVFS* to achieve energy saving. To implement *DES* on *No-DVFS* architecture, we ignore Steps (2) and (3) of the algorithm as well as the second step of the single-core *Online-QE* algorithm. In other words, the algorithm simply assigns the ready jobs to cores using the "*C-RR*" policy, and employs *Quality-OPT* to optimize quality on each core.
- *S-DVFS* (System-level *DVFS*): This architecture provides limited *DVFS* support, where all cores can change their speeds all must share the same speed at any time. To implement *DES* on *S-DVFS*, we set the power requirements of all cores calculated in Step (2) of *DES* to the maximum power requested by any core, so Step (3) will assign the same power to each core. If the total power requirement is less than the power budget, unlike *No-DVFS*, *S-DVFS* is able to save energy by running all cores at a slower speed. The second step of the single-core *Online-QE* algorithm is also ignored.
- *C-DVFS* (Core-level *DVFS*): This is the most flexible architecture that allows each individual core of the system to run at different speed at any time. It can provide fine-grained *DVFS* support and our *DES* algorithm is designed for this architecture.

Taking Advantage of Best-Effort Interactive Services: Web search is a good example of best-effort interactive service, where we can obtain some quality even if a job is only partially executed by its deadline. For a job that cannot be partially evaluated, however, we will not get any quality if it is not executed to completion. To show how *DES* can take advantage of this feature, we will vary the proportion of the jobs with best-effort support to compare the different qualities obtained with the same power budget.

Comparing with Different Scheduling Algorithms: Under the same hardware architecture and application models, we compare the performance of our *DES* algorithm with three widely used scheduling policies, namely *FCFS* (First-Come, First-served)[2], *LJF* (Longest Job First) and *SJF* (Shortest Job First). All three algorithms are triggered whenever a core becomes idle, and a job in the ready queue (with earliest release time in *FCFS*, with largest service demand in *LJF*, and with smallest service demand in *SJF*) will be assigned to the core. The job will be executed with the slowest possible speed to finish it before deadline to save energy. If the power supplied to the core is not enough to complete the job, it will be executed with the highest available speed till its deadline. The default power distribution policy for all three algorithms is static equal sharing, i.e., all the cores will be given the same power budget, similarly to the situation under the *S-DVFS* architecture. We also compare *DES* with these algorithms when they are augmented with "*WF*" power distribution.

[2]*FCFS* is equivalent to *EDF* (Earliest Deadline First) policy, since we assumed that the jobs' deadlines are agreeable.

Sensitivity Study under Different Scenarios: We study the sensitivity of *DES* under the following scheduling scenarios:

- *Effect of quality function*: We show how quality functions with different concavity can affect the total achieved quality with the same consumed energy.
- *Effect of power budget*: We show the tradeoff between quality and energy and its implications when different power budgets are used, especially under heavy load.
- *Effect of number of cores*: We show how different numbers of cores can affect quality and energy, and the optimal number of cores to use for the best performance.
- *Effect of discrete speed scaling*: We show how to support *DES* under discrete speed scaling model, and study its impact on quality and energy.

In the last sensitivity study, we modify the power distribution policy in order to support discrete speed scaling. After performing the "*WF*" power distribution and starting from the core with the lowest assigned power, we rectify the speed to a discrete value closest to but not less than the continuous one, subject to the total power budget. If the power budget cannot support such a discrete speed, we will select the next lower discrete speed instead.

Validation on Real System: We validate the simulation results on a real system to evaluate the accuracy of the energy consumption. In this study, a scheduling trace of the *DES* algorithm under discrete speed scaling is reproduced on a 8-node multicore cluster. To match our simulation settings with the actual system, we also adopt a more practical power consumption model including both static and dynamic power. We use regression method to obtain a power function based on a set of measured $\langle speed, power\rangle$ pairs on the real system. Under this practical power function, we compare the result of our simulations with the one measured in the system.

B. Simulation Setup

We model a web search server with $m = 16$ cores and model the web search requests with partial evaluation support as follows. The arrival of the requests follows the Poisson process and the deadline of each request is defined to be 150ms after its arrival (later responses may affect user experience). The service demand of a request follows bounded Pareto distribution with three parameters α, $x_{\min}$ and $x_{\max}$, which represent the Pareto index, the lower bound and the upper bound on the service demand (for simplicity, we use how many processing units instead of how many instructions to represent the jobs' demands), respectively. We define the processing capability of a core executing at 1Ghz in one second to be 1000 processing unit. Our simulation results show consistency with different parameter values, hence we only present the results with $\alpha = 3$, $x_{\min} = 130$ processing units and $x_{\max} = 1000$ processing units (the mean service demand of a request can then be calculated to be 192 processing units).

We use the following family of quality functions:

$$q(x) = \frac{1 - e^{-cx}}{1 - e^{-1000c}}, \quad (1)$$

where c is a multiplier constant determining the concavity of the function. Figure 7(a) visualizes the quality functions with different values of c. We use the default value of $c = 0.003$ except in Section V-F where we perform sensitivity study on the impact of different quality functions. The total quality presented in our simulation results are normalized against the maximum possible quality that can be obtained.

We set a total dynamic power budget of $H = 320W$ and apply the dynamic power function $P_{dynamic} = a \times s^{\beta}$, where $a = 5$ and $\beta = 2$ are constants and s is the core speed (in terms of Ghz). So the average speed for each core is $\sqrt{20/5} =$ 2Ghz, and it can finish 2000 processing units in one second. In our simulation, we do not consider static power since it serves as a common offset to all scheduling algorithms.

Under this setting, we quantitatively define the workload to be light when the job arrival rate is less than 120 requests per second, which means that on average the requests consume 72% of the server's total processing capacity with the given power budget and number of cores. We also define heavy load to be when the job arrival rate is larger than 180 requests per second, which already exceeds the total processing capacity of the server in the ideal case.

For the triggering events, we use all three triggers described in Section IV-E and set the quantum trigger to be 500ms and use a counter trigger of 8 requests. The simulation time for all the experiments is 1800 seconds.

C. Performance of DES on Different Architectures

We evaluate *DES* on three architectures: *No-DVFS*, *S-DVFS*, and *C-DVFS*. The results show that *C-DVFS* produces the highest quality and consumes the lowest energy, which demonstrates the benefits of our "*WF*" power distribution policy and the effectiveness of *DES* on exploiting modern architecture with fine-grained *DVFS* support.

Figure 3(a) shows that *C-DVFS* always achieves the best quality among the three architectures (the qualities of *S-DVFS* and *No-DVFS* are very close to each other). Under light load, *C-DVFS* is much better than *S-DVFS* and *No-DVFS*. In this case, *C-DVFS* can achieve nearly full quality (by processing each request to near completion) but *S-DVFS* and *No-DVFS* will lose about 2% of the quality (some requests will be discarded before completing the search), which is considered significant in large-scale search engines. The reason is that although *DES* tries to distribute the jobs evenly on each core, it cannot completely eliminate the load imbalance because of the variance in the jobs' service demand. Both *No-DVFS* and *S-DVFS* lack architectural support to dynamically allocate enough power for those cores experiencing temporal high loads. With *C-DVFS*, *DES* can now exploit the hardware flexibility by allowing heavily loaded cores to have more power than lightly loaded ones in order to alleviate the load imbalance and hence return more search results.

With increasing loads, the quality will start to decrease for all architectures, but *C-DVFS*'s quality degrades much slower than those of *S-DVFS* and *No-DVFS*. Under heavy load, the qualities of the three architectures become similar because all cores will be heavily loaded, and the power budget is not sufficient to support the cores to achieve more quality. In this case, the best strategy is to give each core the same power, diminishing the difference between static and dynamic power distribution.

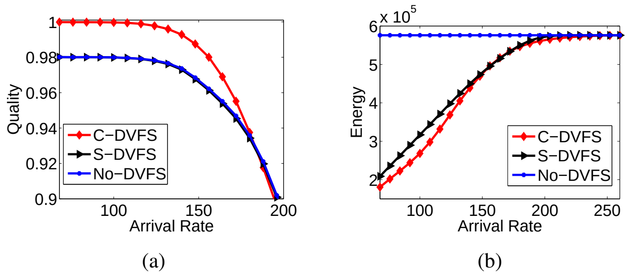


Figure 3. Quality and energy of *DES* on different architectures.

Figure 3(b) shows that *DES* also takes advantage of the *C-DVFS* architecture to save energy. Because *No-DVFS* does not support any *DVFS*, it consumes the maximum energy given by the budget. *S-DVFS* architecture can save energy when the peak power of the cores is less than the budget. *C-DVFS* achieves further saving by reducing the speed of an individual core according to its own load. Under light load, both *C-DVFS* and *S-DVFS* can save energy but *C-DVFS* saves more. In particular, *S-DVFS* saves at least 35.6% of the dynamic energy compared with *No-DVFS*, and *C-DVFS* further saves about 6.8% on top of *S-DVFS*.[3] Again under heavy loads, the entire dynamic power budget is used to execute the jobs regardless of the architectures, and they consume the same amount of energy.

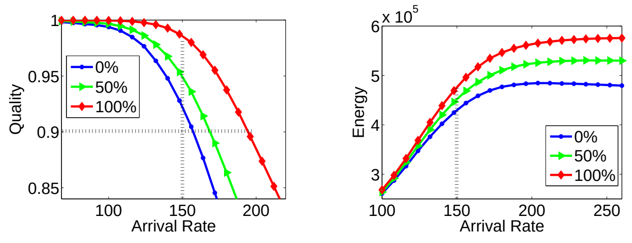
D. Performance of DES with Different Job Execution Models

We show that *DES* indeed benefits from jobs that have partial evaluation support. In this set of simulations, we vary the proportion of the jobs that can be partially evaluated. Three different scenarios are presented under the same load conditions. One is that all the jobs cannot be partially executed; the second is that 50% of the jobs can be partially executed and the last scenario is that all jobs can be partially executed. For a job that cannot be partially evaluated, the algorithm first checks if it can be completed in full under the current schedule. If not, the algorithm discards this job and computes a new schedule based on the remaining jobs.

Figure 4 shows that the more jobs that can support partial evaluation, the more quality we will get under the same load, and the more energy will be consumed because more work needs be done in order to achieve higher quality. It clearly shows that our *DES* algorithm can take advantage of the partial evaluation feature of the jobs and can explore the power budget efficiently to improve quality.

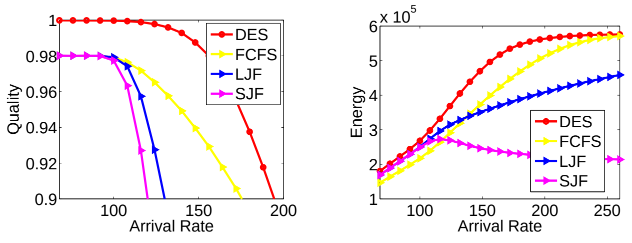
As shown in Figure 4(a), under light load, all three cases can achieve almost full quality with low energy consumption because the power budget is sufficient to finish all the jobs. With increasing workload, their qualities all decrease, but the case with 100% partial evaluatable jobs decreases much slower. The horizonal dotted line in Figure 4(a) shows that to achieve the same normalized quality of 0.9, the 100% case can support a workload with an arrival rate as high as 194 requests per second, while the 50% case can support an arrival rate of 168 (a reduction of about 13.4% in workload), and the

[3]The total energy saving also depends on the proportion of static power. For example, if static power takes up 60% of the total power, then *C-DVFS* saves about $(35.6\% + 6.8\%) \times 40\% = 16.96\%$ of the total energy.



(a) (b)

Figure 4. Quality and energy of *DES* for jobs with different proportions of partial evaluation support.



(a) (b)

Figure 5. Quality and energy comparison for different scheduling algorithms.

0% case can only support an arrival rate of 158 (a reduction of about 18.5%). On the other hand, the vertical dotted line shows that under the same workload with an arrival rate of 150, the 100% case achieves a normalized quality as high as 0.99, the 50% case achieves a quality of 0.95, and the 0% case only achieves a quality of 0.93.

E. Comparing DES with Different Scheduling Algorithms

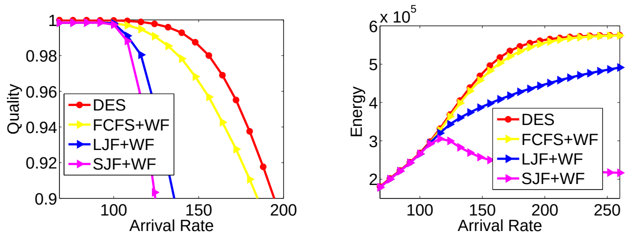
We have shown that our *DES* algorithm can take advantage of the core-level *DVFS* architecture and the features of best-effort applications to achieve high quality. In this subsection we will show that under the same hardware architecture and application model, *DES* can take advantage of its power distribution policy to achieve better performance than three widely accepted scheduling algorithms: *FCFS*, *LJF*, and *SJF*.

In the first experiment, we employ the default static power sharing policy for the other three algorithms. Figure 5(a) shows that the quality of *DES* is always better than the other algorithms. Even under light load, the qualities of *FCFS*, *LJF* and *SJF* are about 2% less than *DES*, which is significant for large-scale interactive services. This is because *DES* has a global view in job distribution and speed scaling, and at the same time it is supported by the dynamic "*WF*" power distribution, which provides the flexibility to optimize for more jobs in one schedule. The other algorithms, however, only optimize for one job in one schedule and use static power distribution. In particular, the "*WF*" power distribution is effective because of the variance in jobs' service demands even under light load. In such case, "*WF*" borrows some excessive power budget from the lightly-loaded cores to support the heavily-loaded ones, and therefore achieves better resource utilization, which eventually translates to higher quality.

Among the three algorithms, the qualities of *LJF* and *SJF* are the worst, because they disturb the arrival/deadline order of the jobs. In either case, jobs with earlier deadline may be discarded in favor of jobs with later deadline, while they could have been co-scheduled to achieve better quality. *FCFS* achieves relatively higher quality since it respects the deadline order of the jobs so it can finish more jobs.

For the same quality, say 0.9, *DES* can support an arrival rate as high as 196, while *FCFS*, *LJF* and *SJF* support an arrival rate of 164, 132 and 116 respectively. It means that the throughput of *DES* is about 20%, 48% and 69% higher than those of *FCFS*, *LJF* and *SJF*.

Figure 5(b) shows that, under light load, *DES* achieves significantly higher quality with little extra energy consumption. When the load increases, the energy consumption of all algorithms (except *SJF*) will start to increase, and *DES* incurs more energy in order to complete more work to sustain its quality. The energy of *SJF* reduces with increasing load because it will discard more long jobs with early deadlines. In this case, the short jobs will be executed with very slow speed for energy efficiency, but the long jobs will not have any chance to be executed before they expire. This also explains why *SJF* has the worst quality among all algorithms.



(a) (b)

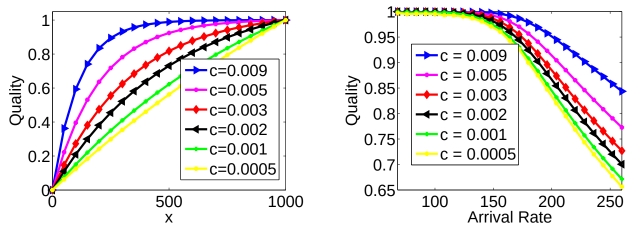
Figure 6. Quality and energy comparison for different scheduling algorithms with "*WF*" power distribution.

In another experiment, we enhance the three comparing algorithms with "*WF*" power distribution. Figure 6 shows that they can all benefit from this dynamic policy. Specifically, under light load, all the "*WF*"-enhanced algorithms can achieve nearly full quality, which is a significant improvement compared to the results in Figure 5(a). When the load becomes high, *DES* still maintains its quality advantage over the other algorithms, due to its global view in job distribution and speed scaling. In particular, *DES* schedules all jobs in the ready queue at each invocation while *FCFS*, *LJF* and *SJF* select only one job to schedule, so they cannot efficiently make use of the power budget to achieve high quality in total.

F. Sensitivity Study

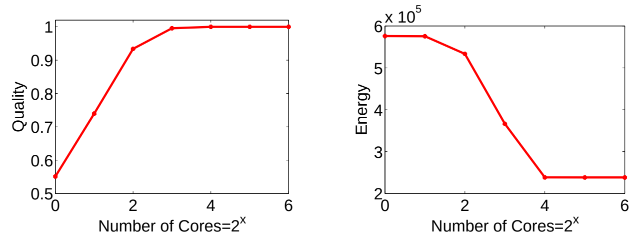
We perform some sensitivity study on our *DES* algorithm in this subsection. The results show: (1) With the same power budget, we can achieve more quality if the applications exhibit a more concave quality function; (2) For the same target quality, more power budget can support higher load, with increased energy consumption; (3) With the same power budget, an optimal number of cores can be used to achieve the best quality with the least energy under a specific load; (4) Discrete speed scaling has little impact on the performance of *DES*.

Effect of different quality functions: DES is designed for best-effort interactive services with concave quality functions.



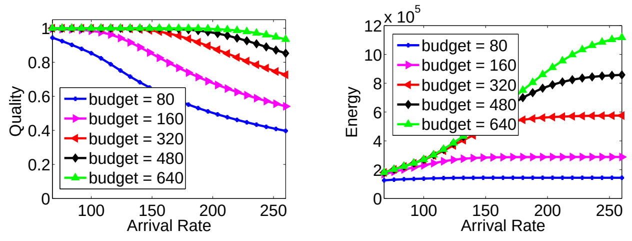
(a) (b)

Figure 7. Different quality functions and impacts on the quality of *DES*.



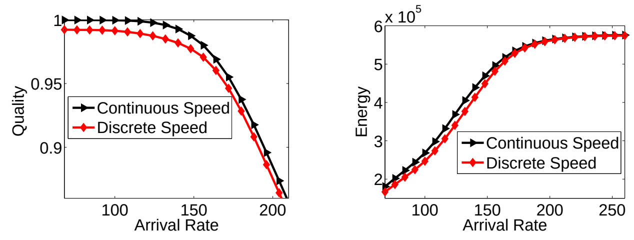
(a) (b)

Figure 9. Quality and energy of *DES* with different numbers of cores.



(a) (b)

Figure 8. Quality and energy of *DES* with different power budgets.



(a) (b)

Figure 10. Quality and energy with continuous and discrete speed scaling.

The resulting quality is highly affected by the concavity of the function. Figure 7(a) shows different quality functions by choosing different values for the parameter c given in Eq. (1). As can be seen, a larger c gives a more concave function, and therefore gets more quality from (partially) executing the same fraction of the job. Because of this, under the same schedule, a larger c results in higher quality than a smaller c, as shown in Figure 7(b). The energy consumption will not be affected by the quality functions.

Effect of different power budgets: Figure 8 depicts the impact of different power budgets on both quality and energy. The results show: (1) When load is heavy, with more power budget, *DES* is able to achieve higher quality under the same load, or support higher load in order to sustain the same quality; when load is light, however, high power budget is not necessary. (2) The energy consumption will increase as the load increases until the total power reaches the given budget. From then on, a higher load will no longer affect the energy while the quality will start to degrade.

Effect of different numbers of cores: Both quality and energy will benefit from having more cores in the system. Due to the convexity of the power function, a larger number of cores will increase the total processing capability under the same power budget. It also decreases the potential contention of the jobs on each core so that a job can be executed more slowly to save energy. Figure 9 shows the impact of different numbers of cores on both quality and energy when the arrival rate is 90 (Similar curves are observed for other loads). We can see that a small number of cores only obtains very limited quality and consumes a lot of energy. The effect of both quality and energy improves as more cores are added to the system. Such improvement reaches saturation when no more concurrent jobs can be executed with additional cores. In our experiment, 16 cores are sufficient to sustain high quality with low energy. In practice, adding more cores will change the hardware setting, which will lead to more static power consumption, so the effect of energy saving will be compromised to some extent by the increased static energy.

Effect of discrete speed scaling: Figure 10 compares the the quality and energy of *DES* under continuous and discrete speed scaling. The discrete implementation loses some quality because it is not able to use the ideal speed to complete as much work as originally planned in the continuous case. But at the same time, the energy consumption of the discrete version is also less than that of the continuous one. Under light load, the difference in quality is about 1% and the difference in energy is less than 7.6%, which is relatively high. This is because of the variance in service demands and the limit in a core's highest speed. For example, even when the total power budget is sufficient, some long requests cannot be completed due to the discrete speed limit of the cores, but they can be completed under continuous speed scaling since the cores can run in any speed. The differences in both quality and energy become smaller as the arrival rate increases. Under heavy load, the both of differences are reduced to less than 0.5% between the two implementations.

G. *Validation on Real Systems*

This last subsection validates the accuracy of our simulation results, in particular the energy consumption, by comparing it against the one measured in a real system. For this purpose, we implement *DES* using discrete speed scaling and apply the scheduling solutions from simulation on the real system for energy measurement.

The system we use is a 8 nodes multicore cluster. Each node has two Quad-Core AMD Opteron(tm) Processor 2380. Each core's speed can be set as 800Mhz, 1300Mhz, 1800Mhz, or 2500Mhz independently. The corresponding power consumptions are 11.06W, 13.275W, 16.85W and 22.69W, respectively. The cluster is equipped with the PowerPack software [13] as well as the necessary hardware to measure its practical energy consumption. Adopting the power model $P = a \times s^{\beta} + b$, we

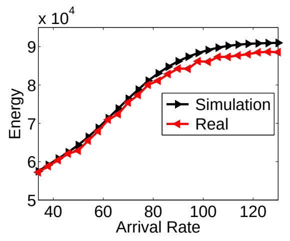


Figure 11. Energy comparison between simulation and real system implementation.

can get $a = 2.6075$, $\beta = 1.791$ and $b = 9.2562$, which are then used to drive our simulation. The power budget is set as 152W and the simulation time for each arrival rate is 10min. Figure 11 shows the energy measurements from both simulations and the real system. The results are very close to each other, despite the fact that there could be additional scheduling overheads in the actual system implementation. This suggests that our simulations provide accurate and reliable results on the performance of the *DES* algorithm.

VI. Related Work

In this section, we review some related works on scheduling interactive services for energy minimization and quality maximization.

Energy Minimization with DVFS: DVFS has been a widely adopted technique to achieve higher energy efficiency. Yao et al. [25] initiated the study of energy minimization by scheduling requests on a single *DVFS*-enabled processor. Assuming that power is a convex function of the processor speed, they provided an optimal offline algorithm as well as two online algorithms, which was shown to have good competitive ratios [4][25]. Refinement for the offline algorithm was later provided in [21] with lower computational complexity as well as with discrete processor speed. Albers et al. [1] recently considered the same scheduling problem but for multicore systems, where the speed of each core can be independently scaled. They provided both an optimal offline algorithm and a competitive online algorithm for this setting. All these results assumed no power budget, i.e., the speed of the processors can be scaled arbitrarily high, so all the jobs can be fully processed to achieve the maximum possible quality.

Quality Maximization under Overloads: Maximizing the service quality for interactive jobs on a fixed-speed processor has been the subject of many studies over the years. While the EDF (Earliest Deadline First) algorithm is known to produce an optimal schedule under light load, it can perform very badly in overloaded systems [24]. To handle overloads, many early works [19][5][18][17][23] assumed a strict scheduling model in which an uncompleted job before its deadline does not contribute any value towards the total quality. Inspired by the emerging workload on interactive services such as web search and video-on-demand, He et al. [15] recently considered a scheduling model that supports partial evaluation for the jobs. Assuming that the quality function is non-decreasing and strictly concave, they gave an optimal offline algorithm, and evaluated the proposed online algorithms also using web search as an example. Prior to this work, similar models have been studied in [8][9][10][11], which assumed a linear quality function for jobs with different priorities. All these results do not consider energy since the speed of the processor is assumed to be fixed.

Scheduling under a Power Budget: Many prior works on energy-efficient scheduling does not assume a power budget for *DVFS*-enabled systems. In practice, the total power is often constrained by the chip design and its cooling system. Prior work in [3][7] focused on the strict scheduling model on a single core with a power budget, and proposed online algorithms that achieve competitive performance for both quality and energy. Isci et al. [16] considered global power management for multicore systems based on the per-core load and a given power budget. Baek and Chilimbi [2] proposed a framework that trade off quality and energy by supporting different approximations for interactive services. Although their results are also based on partial executions, they did not consider scheduling and *DVFS*. To the best of our knowledge, no prior work studies scheduling interactive jobs that support partial evaluation with the objective of both quality maximization and energy minimization.

VII. Conclusion

In this paper, we have proposed a new performance metric $\langle quality, energy \rangle$ to measure the performance of scheduling algorithms with both quality and energy considerations. We have presented the *DES* algorithm for scheduling best-effort interactive services on multicore systems with a power budget. We showed that the algorithm can exploit the application features and can take advantage of the core-level *DVFS* architecture to enable new advances in quality improvement and energy saving. Simulation results demonstrate the benefits of *DES* in achieving high service quality with low energy consumption. Validation on a real system also verifies the accuracy and reliability of our simulation results.

Acknowledgment

We would like to thank Prof. Rong Ge of Marquette University for providing the PowerPack environment for conducting validation on real systems. This research is supported in part by National Natural Science Foundation of China (No. 61272087, No. 61073008, No. 60773148 and No. 60503039), Beijing Natural Science Foundation (No. 4082016 and No. 4122039), NSF Grants OCI-0904461 and OCI-1051537.

References

[1] S. Albers, F. Müller, and S. Schmelzer. Speed scaling on parallel processors. In *ACM Symposium on Parallelism in Algorithms and Architectures(SPAA)*, pages 289–298, San Diego, USA, 2007.

[2] W. Baek and T. M. Chilimbi. Green: a framework for supporting energy-conscious programming using controlled approximation. In *ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI)*, pages 198–209, 2010.

[3] N. Bansal, H. L. Chan, T. W. Lam, and L. K. Lee. Scheduling for speed bounded processors. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 409–420, 2008.

[4] N. Bansal, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM*, 54(1):3:1–3:39, 2007.

[5] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems*, 4(2):125–144, 1992.

[6] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[7] H.-L. Chan, W.-T. Chan, T.-W. Lam, L.-K. Lee, K.-S. Mak, and P. W. H. Wong. Energy efficient online deadline scheduling. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 795–804, 2007.

[8] E.-C. Chang and C. Yap. Competitive on-line scheduling with level of service. *Journal of Scheduling*, 6(3):251–267, 2003.

[9] F. Y. L. Chin and S. P. Y. Fung. Online scheduling with partial job values: Does timesharing or randomization help. *Algorithmica*, 37:149–164, 2003.

[10] F. Y. L. Chin and S. P. Y. Fung. Improved competitive algorithms for online scheduling with partial job values. *Theoretical Computer Science*, 325(3):467–478, 2004.

[11] M. Chrobak, L. Epstein, J. Noga, J. Sgall, R. van Stee, T. Tichý, and N. Vakhania. Preemptive scheduling in overloaded systems. *Journal of Computer and System Sciences*, 67(1):183–197, 2003.

[12] M. Ehrgott. A characterization of lexicographic max-ordering solutions. In *Proceedings of the 6th Workshop of the DGOR Working-Group Multicriteria Optimization and Decision Theory*, pages 193–202, 1997.

[13] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K.W. Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *IEEE Transactions on Parallel and Distributed Systems*, 21(5):658–671, 2010.

[14] Y. He, S. Elnikety, J. Larus, and C. Yan. Zeta: scheduling interactive services with partial execution. In *Proceedings of the Third ACM Symposium on Cloud Computing*, 2012.

[15] Y. He, S. Elnikety, and H. Sun. Tians scheduling: Using partial processing in best-effort applications. In *International Conference on Distributed Computing Systems(ICDCS)*, pages 434–445, 2011.

[16] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *IEEE/ACM International Symposium on Microarchitecture*, pages 347–358, 2006.

[17] C.-Y. Koo, T.-W. Lam, T.-W. Ngan, K. Sadakane, and K.-K. To. On-line scheduling with tight deadlines. *Mathematical Foundations of Computer Science (MFCS)*, 295(1-3):251–261, 2003.

[18] G. Koren and D. Shasha. Dover: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM Journal on Computing*, 24(2):318–339, 1995.

[19] E. L. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Annals of Operations Research*, 26(1-4):125–133, 1991.

[20] J. Li and J. F. Martínez. Power-performance considerations of parallel computing on chip multiprocessors. *ACM Transactions on Architecture and Code Optimization*, 2(4):397–422, 2005.

[21] M. Li, A. Yao, and F. Yao. Discrete and continuous min-energy schedules for variable voltage processors. *National Academy of Sciences*, 103:3983–3987, 2006.

[22] A. Naveh, E. Rotem, A. Mendelson, S. Gochman, R. Chabukswar, K. Krishnan, and A. Kumar. Power and thermal management in the intel core duo processor. *Intel Technology Journal*, 10(2):109–122, 2006.

[23] Z. Shi, C. Beard, and K. Mitchell. Competition, cooperation, and optimization in multi-hop CSMA networks. In *ACM symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks (PE-WASUN)*, 2011.

[24] J. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo. *Deadline Scheduling for Real-Time systems - EDF and related algorithms*. Academic Publishers, 1998.

[25] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 374–382, 1995.

[26] X. Zhang, K. Shen, S. Dwarkadas, and R. Zhong. An evaluation of per-chip nonuniform frequency scaling on multicores. In *USENIX Annual Technical Conference (USENIX ATC)*, pages 19–19, 2010.

[27] X. Zhao and N. Jamali. Fine-grained per-core frequency scheduling for power efficient-multicore execution. In *International Green Computing Conference (IGCC)*, pages 1–8, 2011.