2017 IEEE International Parallel and Distributed Processing Symposium Workshops

When Good Enough Is Better: Energy-Aware Scheduling for Multicore Servers

Xinning Hui*, Zhihui Du[a]*, Jason Liu†, Hongyang Sun‡, Yuxiong He§, David A. Bader¶
*Tsinghua National Laboratory for Information Science and Technology
Department of Computer Science and Technology, Tsinghua University, Beijing, China
[a]Corresponding Author's Email duzh@tsinghua.edu.cn
†School of Computing and Information Sciences, Florida International University, Florida, USA
‡Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN, USA
§Microsoft Research, Redmonds, WA, USA
¶College of Computing, Georgia Institute of Technology, Atlanta, GA, USA

***Abstract*—Power is a primary concern for mobile, cloud, and high-performance computing applications. Approximate computing refers to running applications to obtain results with tolerable errors under resource constraints, and it can be applied to balance energy consumption with service quality. In this paper, we propose a *"Good Enough (GE)"* scheduling algorithm that uses approximate computing to provide satisfactory QoS (Quality of Service) for interactive applications with significant energy savings. Given a user-specified quality level, the *GE* algorithm works in the *AES (Aggressive Energy Saving)* mode for the majority of the time, neglecting the low-quality portions of the workload. When the perceived quality falls below the required level, the algorithm switches to the *BQ (Best Quality)* mode with a compensation policy. To avoid core speed thrashing between the two modes, *GE* employs a hybrid power distribution scheme that uses the *Equal-Sharing (ES)* policy to distribute power among the cores when the workload is light (to save energy) and the *Water-Filling (WF)* policy when the workload is high (to improve quality). We conduct simulations to compare the performance of *GE* with existing scheduling algorithms. Results show that the proposed algorithm can provide large energy savings with satisfactory user experience.**

***Keywords*-approximate computing, scheduling algorithm, power efficiency, multicore servers**

I. Introduction

Large-scale cloud data centers and high-performance computing systems consume an exorbitant amount of energy. The electricity consumption of U.S. data centers in 2013 is estimated to have reached 91 billion kilowatt-hours. Future computing systems must adhere to strict power limits. For example, the U.S. Department of Energy has set a target power consumption of 20 MW to achieve Exascale computing in the next decade [3, 4]. Being able to handle large-scale computing and data processing with stringent energy budget is a critical challenge.

It has been observed that one can save a great deal of energy when applications can be computed in an approximate rather than exact fashion [17]. Fortunately, many such applications exist. Many interactive services, such as video rendering, web search, financial data analysis, process monitoring, and GPS tracking, can tolerate some degree of inaccuracy, where small errors in the result do not affect the overall quality of perceived user experience. In this context, *approximate computing* provides a great opportunity to solve many power-constrained computing problems.

Indeed, recent research has shown that different data of the same size may contribute differently to the achievable quality [12]. In addition, the marginal quality gain decreases with increased data processing volume. This phenomenon of *diminishing returns* was proposed in economics as early as 1776 [2], and it now plays an important role in many fields including computing. Without loss of generality, one can formulate the trend of diminishing returns as a concave function that maps the processed data of a service to its quality contribution. In our study, we take advantage of approximate computing to achieve *good enough* quality with less data (and workload), and in addition, exploit the law of diminishing returns to maintain the quality level with less energy consumption by selecting the most quality-efficient part of the input data or workload to execute.

We explore the characteristics of interactive applications, which can tolerate approximate results with concave quality functions, and design a "*Good Enough (GE)*" scheduling algorithm to reduce the power consumption on multicore servers. In contrast to the existing "*Best Effort (BE)*" scheduling paradigm (e.g., [9, 14]), our *GE* algorithm may discard part of the workload that contributes little to the quality even when the power budget is sufficient to complete the jobs. We call this service providing regime the *AES (Aggressive Energy Saving)* mode. Due to the unpredictable nature of jobs that arrive randomly for online scheduling, the *AES* mode may not always guarantee the desired quality. With online monitoring of the user experience, once the service quality is perceived to have fallen below the required level, *GE* will adopt a compensation policy and switch to the *BQ (Best Quality)* mode, which attempts to achieve the highest possible quality by completing all jobs so as to meet the overall quality demand of user experience.

The switching between the *AES* mode and the *BQ* mode, however, may adversely consume additional energy when the workload is light, due to the fact that the compensation

978-1-5386-3408-0/17 $31.00 © 2017 IEEE
DOI 10.1109/IPDPSW.2017.38

policy may introduce core speed thrashing. Moreover, applying significantly different speeds at the cores using DVFS (Dynamic Voltage and Frequency Scaling) can inadvertently boost energy consumption. To further reduce energy without sacrificing quality, the *GE* algorithm adopts a hybrid power distribution policy. Specifically, when the workload is light, an *Equal-Sharing (ES)* policy is used, which distributes the power equally among the cores. In this case, it can significantly reduce the energy consumption in *BQ* mode without losing quality. On the other hand, when the workload is high, a *Water-Filling (WF)* policy is used, which favors the cores with smaller power demands to improve the quality.

To evaluate the performance of the *GE* algorithm, we conducted simulations and compared the results with those from two other quality control policies, namely, power control policy and speed control policy. The power control policy reduces the total power budget for a given workload to support the specified quality. The speed control policy limits the maximum available speed to support the specified quality. The *BQ* mode is selected to schedule the jobs under both policies. Experiments were also designed to compare *GE* with an existing best effort energy-efficient algorithm and other widely used scheduling policies. The simulation results show that the *GE* algorithm can provide sufficient quality (90%) with up to 23.9% energy savings compared with these existing solutions.

The major contributions of this paper are as follows:

- We design a *GE (Good Enough)* scheduling algorithm that takes advantage of approximate computing to significantly reduce the energy consumption. The algorithm strategically selects the most quality-efficient parts of the jobs to execute while achieving sufficiently high quality with less energy.
- We design a quality compensation policy to guarantee a user-specified quality level. The policy dynamically switches the execution between two execution modes (*Aggressive Energy Saving (AES)* mode and *Best Quality (BQ)* mode) with the help of online monitoring of the service quality.
- We design a hybrid power distribution scheme that uses the *Equal-Sharing (ES)* policy and the *Water-Filling (WF)* policy to distribute the power budget among the cores depending on the workload. This scheme avoids the thrashing overhead on energy consumption between the two execution modes especially when the workload is light.

The rest of this paper is organized as follows. In Section II, we formulate the scheduling problem. The *GE* algorithm is introduced in details in Section III, followed by its performance evaluation in Section IV. In Section V, we summarize the related work. Finally, in Section VI, we conclude this paper and briefly outline the future work.

II. Problem Formulation

To formulate the problem of energy-efficient scheduling, we first develop a model for what we call *"good enough"* services on multicore servers.

A. Good Enough Services

Consider a set of n service jobs: $\mathcal{J} = \{J_1, J_2, \cdots, J_n\}$. Every job $J_j \in \mathcal{J}$ has a start time s_j, a deadline d_j, and a processing demand p_j (representing the data that needs to be processed). Each job can only be executed between its start time and its deadline. A job can be partially processed without completing the entire processing demand p_j. In this paper, we focus on applications, such as web search, which have a similar response time requirement. For simplicity, we assume the same duration in which a job can be processed, i.e., $d_j - s_j$ is the same for all jobs. For convenience, suppose the jobs are ordered by their starting time, i.e., $s_1 \leq s_2 \leq \cdots \leq s_n$. The above assumption suggests that the jobs also have *agreeable* deadlines, i.e., $d_1 \leq d_2 \leq \cdots \leq d_n$. For jobs with agreeable deadlines, it is highly desirable to schedule them in a *non-preemptive* manner in order to reduce the context switching overhead; that is, jobs are processing by the *Earliest Deadline First (EDF)* order without preemption (at least on a single core). We focus our attention to only such schedules in this paper.

An important property of *"good enough"* services is that jobs can be partially processed, thus they may return results with some quality loss (again, considering a web search). For each job J_j, we use c_j to denote the processed volume during $[s_j, d_j]$, where $c_j \leq p_j$. The processed volume contributes to the perceived quality of the job [12]. We define a *quality function* $f : R^+ \rightarrow R^+$ that maps the processed volume of a job (a positive number) to a value that represents the perceived quality of the (partially) executed job, which is normalized between 0 and 1. The average quality achieved by executing a set of jobs is defined to be:

$$Q(\mathcal{J}) = \frac{\sum_{J_j \in \mathcal{J}} f(c_j)}{\sum_{J_j \in \mathcal{J}} f(p_j)}.$$

The specific form of quality function for different applications may be different. We use a concave function to capture the trend of diminishing returns, which is a common characteristic among such applications. Without loss of generality, in this paper, we take the following concave quality function:

$$f(x) = \frac{1 - e^{-cx}}{1 - e^{-cx_{\max}}}, \tag{1}$$

where c is a constant multiplier that determines the concavity of the function, and $x_{\max}$ denotes the upper bound on the processing demand of the jobs.

B. Multicore Servers

We consider multicore servers with core-level dynamic voltage and frequency scaling (DVFS). A server is composed of m cores, $\{M_1, M_2, \cdots, M_m\}$, each of which can be set to run at a different frequency. We assume that the total power consumption of the server is the sum of the power consumption of all cores[1]. The power consumption of each core consists of a dynamic part and a static part. The dynamic power is a (convex) function of the core's speed [28]. We adopt a well-established model: $P_{dynamic} = a \times s^{\beta}$, where $a > 0$ is a scaling factor and $\beta > 1$ is an exponent parameter [6, 7, 15, 28]. While the static power is an important component of the power consumption, we assume that the cores cannot be individually shut down during execution, so the static power is the same for all the cores. As such, we can ignore static power in the power consumption formulation for the scheduling algorithms.

Consider that a total budget H is given for the server's dynamic power, which can be distributed arbitrarily among the cores. The power consumption of core M_i at time t is denoted by $P_i(t)$. The total power $P(t)$ of all cores at time t should satisfy $P(t) = \sum_{i=1}^{m} P_i(t) \leq H$. The total energy consumption E of a schedule is the total power integrated from the start time of the first processed job to the deadline of the last processed job, i.e., $E = \int_{s_1}^{d_n} P(t)dt$. In the scheduling model, we assume that a job can be assigned to any core of the server, but once the job has been assigned to a core, it cannot be migrated to other cores.

C. Good Enough Quality and Scheduling Objective

In our scheduling problem, the user can specify a *good enough* quality, or Q_{GE}, such as 90% or 95%. This quality will be used by the scheduling algorithm as a constraint such that the average quality achieved by executing a job set should be no smaller than Q_{GE}. The objective is to minimize the total energy consumption subject to this constraint. Formally, the scheduling problem can be formulated as an optimization problem:

$$\text{minimize } E$$
$$\text{s.t. } Q(\mathcal{J}) \geq Q_{GE}$$

If we set $Q_{GE} = 100\%$, it means that we do not allow any loss of quality. The traditional *best effort* scheduling can be treated as a special case of the optimization problem proposed here. Our *Good Enough* scheduling applies when $Q_{GE} < 100\%$. Although a user can specify any acceptable quality demand, in general, more energy can be saved with less Q_{GE}.

III. Good Enough Scheduling

In this section, we present the Energy-Aware "Good Enough (GE)" Scheduling algorithm for multicore servers.

[1]We omit the power consumptions of memory, cache and other subsystems in this study.

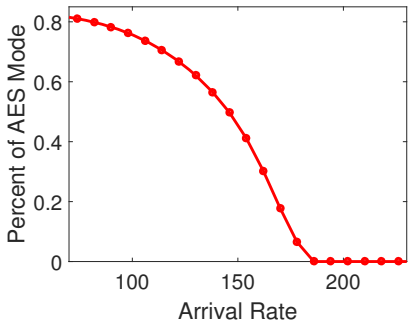


Figure 1. The execution time percentage of the AES mode.

We first provide an overview of the *GE* algorithm, following by an explanation of different execute modes and power distribution policies. The entire algorithm is presented at the end of this section.

A. Overview of the Algorithm

GE is a job scheduling algorithm aiming to save as much energy as possible, provided it can meet the user-specified quality requirement. The algorithm takes advantage of approximate computing and also the law of diminishing returns, in that the algorithm only executes the most quality-efficient part of a given workload in order to reduce energy consumption.

Given a user-specified quality level, *GE* first stays at the *Aggressive Energy Saving (AES)* mode, in which the algorithm discards the low quality part of each job as much as possible to save energy. At the same time, the overall quality will be monitored continuously upon each scheduled job. If the overall quality becomes less than the user-specified quality level, a compensation policy will be applied, in which case the scheduling algorithm switches to the *Best Quality (BQ)* mode to improve the quality. Once the overall quality can satisfy the given quality demand, *GE* will switch back to *AES* mode again to save energy.

The compensation policy is necessary to maintain the quality level, but it may lead to core speed thrashing between the *AES* and *BQ* execution modes when the total workload is light, which can be detrimental to energy saving. We employ an *Equal-Sharing (ES)* power distribution policy to address this issue, to keep the speed difference among the cores within a desirable range.

B. Aggressive Energy Saving Mode

The main aspect of the algorithm is to apply approximate computing to support a practical user-specified quality level to save energy. This is in contrast to other energy-aware methods where full quality is required. The *AES* mode of our algorithm adopts a job cutting policy to discard the low quality portions of the jobs based on a given *good enough* quality Q_{GE} before they are executed on each core.

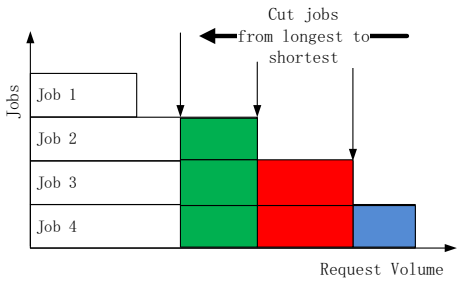


Figure 2. Job cutting of four jobs from longest to the shortest.

This *AES* execution mode can significantly save energy for our scheduling algorithm as long as the algorithm can work mostly in the *AES* mode during job execution, which is indeed the case. Fig. 1 shows the percentage of time that the *GE* scheduling algorithm stays in the *AES* mode for a typical job set. When the workload is light, the algorithm can mostly run in the *AES* mode to save energy. This is the major reason that the algorithm turns out to be able to significantly reduce energy consumption.

In the *AES* mode, the algorithm cuts jobs in response to the user-specified quality. According to the concave quality function, which observes the law of diminishing returns, we can see that processing the head of each job would contribute more to the quality than processing the tail of the job with the same amount of workload. Consequently, our algorithm executes the most quality-preserving part of the job at the head while cutting its tail.

The job cutting is performed starting from the longest job. We call it the *Longest-First (LF)* job cutting policy. Fig. 2 shows an example with four jobs of various lengths. The job cutting algorithm is described as follows:

1) Sort the jobs according to their proceeding demands.
2) At each iteration, cut the longest job (there could be multiple jobs with the same length) to make the remaining workload the same as the second longest job; update the overall quality Q after cutting.
3) If $Q > Q_{GE}$ (where Q_{GE} is the user-specified quality level), we continue with another iteration by going back to the previous step and cut the now longest job(s) until Q becomes no greater than Q_{GE}.
4) If $Q = Q_{GE}$, the job cutting is complete.
5) If $Q < Q_{GE}$, let $\mathcal{U}$ and $\mathcal{C}$ denote the set of uncut jobs and the set of cut jobs after this iteration, respectively. Calculate the total quality $F_{\mathcal{U}} = \sum_{J_j \in \mathcal{U}} f(p_j)$ of the uncut jobs, and the total quality $F_{\mathcal{C}} = \sum_{J_j \in \mathcal{C}} f(p_j)$ of cut jobs. The desired quality of each cut job is then given by $f(c) = (Q_{GE}\,(F_{\mathcal{U}} + F_{\mathcal{C}}) - F_{\mathcal{U}})/|\,\mathcal{C}\,|$. Use binary search on the concave quality function to find the proceeding demand c of each cut job such that its quality is $f(c)$.

A remaining issue is how to handle jobs that are already scheduled running. Once a job is assigned to run on a core of the server, it needs to stay on the same core. We solve this problem by considering a running job as a new one (with its original proceeding demand) upon a new schedule. We calculate its desired demand according to the same *LF* policy above. If the calculated demand is smaller than its remaining demand, this job will be cut accordingly. Otherwise, the job will continue to run with the remaining demand.

C. Best Quality Mode

During the *AES* mode, it is possible that some jobs in the current schedule may not be able to finish when a new event (e.g., job arrival) triggers the next schedule to start. In this case, some jobs will be discarded and the desired quality level may not be achieved.

To address this issue, we develop a compensation policy in our algorithm. We calculate the current quality whenever a new schedule is triggered. If the current quality is less than the user specified quality, the compensation policy will stop the job cutting and all jobs will be executed to completion in order to improve the total quality. We call this execution mode the *Best Quality (BQ)* mode. Once the quality is raised above the user-specified quality level, the execution mode will be switched back to the *AES* mode to save energy. Our experiments (described in Section IV-D) show that the compensation policy in *BQ* mode provides good quality guarantee for jobs in the long run.

D. Hybrid Power Distribution Policy

Although the compensation policy can provide the user specified quality guarantee, switching between *AES* and *BQ* may cause core speed thrashing when the workload is light. Core speed thrashing refers to the phenomenon that when the cores run at significantly different speeds using DVFS, or when a core switches between very high and very low speeds, the power consumption increases. The larger the speed variation, the worse it is for energy saving. This phenomenon is due to the fact that the power-speed relation is a convex function, where the minimal power consumption is obtained when the cores are running at the average speed [28]. In our case, core speed thrashing can cause significantly more energy consumption even though it may not reduce quality.

In order to solve this problem, we employ an *Equal-Sharing (ES)* policy, which aims at sharing the power budget equally among different cores to avoid large speed difference among them. *ES* policy can successfully reduce the energy consumption for light workloads without sacrificing the quality guarantee. During heavy workloads, the cores are assigned with significant work and would require power to run above the average speed in order to complete the work and meet the user specified quality. In this scenario, we employ the *Water-Filling (WF)* policy [9], which distributes the power budget by satisfying the low demand first and all

the remaining power is used to support heavy-loaded cores. Our previous work [9] has demonstrated that the *WF* policy is effective to guarantee the user specified quality.

Overall, our approach is a hybrid power distribution policy, which employs *ES* to save energy during light workload and employs *WF* to improve quality when the workload is heavy. The threshold for differentiating light and heavy workload is called the *critical load*, and the performance of the algorithm can be sensitive to the threshold.

E. The GE Algorithm

We now put different pieces together and describe the *GE* algorithm. *GE* is an online scheduling algorithm, so when new jobs arrive, they are first kept in a waiting queue. The scheduling algorithm then assigns the jobs to specific cores when certain conditions are met. These conditions are called *triggering events*, and they include "quantum triggering" (when the scheduler is run periodically at each scheduling interval), "idle-core triggering" (when a core becomes idle), and "counter triggering" (when the number of jobs in the waiting queue has reached a certain threshold).

When the scheduler runs, the jobs currently waiting in the queue are assigned to different cores in a batch. In this context, one can apply *Round Robin (RR)*, which is an effective policy despite its simplicity. However, to achieve a more balanced job distribution, we choose to use the *Cumulative Round Robin (C-RR)* policy [9]. The only difference is that *C-RR* is cumulative in the sense that it assigns jobs to the core where the last job distribution cycle stops. As such, it is expected to achieve a more balanced job distribution over the long run, which would be more desirable for both quality and energy.

After the jobs are assigned to the cores, the scheduling algorithm chooses the power distribution policy according to the current workload. If the current workload is heavier than the critical load, *WF* is selected as the power distribution policy; otherwise, *ES* is chosen.

Next, at each core, the scheduling algorithm calculates the quality and decides to run either in the *AES* mode or in the *BQ* mode. In the *AES* mode, the jobs assigned to the core are cut according to the user-specified quality level.

For all jobs assigned to a core, if the assigned power budget is not enough to satisfy the requests of all the workload, the existing *Quality-OPT* algorithm [14] is applied to calculate the most efficient part of the jobs to achieve the highest possible quality with limited power (this can be considered as a second cut to achieve the best quality).

Finally, the jobs assigned to each core are executed in order of their deadlines by the existing *Energy-OPT* algorithm [28] to achieve the least power consumption.

IV. Experimental Evaluation

In this section, we describe empirical studies to evaluate the proposed algorithm by comparing it with other widely used scheduling algorithms.

A. Evaluation Methodology

Our algorithm is based on *"good enough"* services on multicore systems that support core-level DVFS. We rely on simulations to model such architectures and use web search as the application for *"good enough"* interactive services. We evaluate the performance of the scheduling algorithm in the following aspects:

- Whether the *GE* algorithm can save more energy compared with other scheduling policies that operate in the same scenarios;
- Whether the compensation policy can guarantee the quality demand from the users;
- Whether the hybrid power distribution policy can avoid core speed thrashing to achieve more energy savings;
- Whether the proposed quality control policy can compete with power budget and speed control policies;
- How the quality function, the amount of power budget, the number of cores, and discrete speed scaling affect our algorithm.

We follow through our evaluation methods and elaborate the corresponding five sets of experiments as follows.

1) Comparing Scheduling Algorithms: Under the same multicore server and application models, we compare the performance of the *GE* algorithm with six other scheduling algorithms, namely, *OQ (Over Qualified), BE (Best Effort), FCFS (First-Come First-served), FDFS (First-Deadline First-served), LJF (Longest Job First) and SJF (Shortest Job First)*.

In particular, *OQ* sets the quality target to be 2% more than the user specified quality demand without compensation policy. *BE* always execute the jobs in the *BQ* mode and always employs the *WF* power distribution policy.

The other four algorithms are triggered whenever a core becomes idle, and a job in the waiting queue (with the earliest release time in *FCFS*, the earliest deadline in *FDFS*, the largest service demand in *LJF*, and the smallest service demand in *SJF*) is assigned to the core. The default power distribution policy for all four algorithms is *ES*. The job is executed with the slowest possible speed to finish before deadline to save energy. However, if the power supplied to the core is not enough to complete the job, the job will be executed with the highest available speed till the deadline is reached.

2) Impact of Compensation Policy: We compare our algorithm with the one without a compensation policy to see how the compensation policy can affect quality and energy. For the compensation policy, our algorithm will immediately switch to *BQ* mode to improve the quality once the perceived quality has fallen below the promised level. Without a compensation policy, the algorithm would never switch to *BQ* mode regardless of the quality.

3) Hybrid Power Distribution policy: We compare the two power distribution policies—*Water-Filling (WF)* and

Equal-Sharing (ES)—to show that *Equal-Sharing* can efficiently avoid core speed thrashing under light workload and *Water-Filling* can improve quality under heavy workload.

4) Comparing Control Policies: We compare the proposed quality control policy with two other policies, namely, power control policy and speed control policy. The *GE* algorithm employs the *good enough* quality Q_{GE} to determine the job cutting in our quality control policy. The power control policy employs the least power budget which can complete the quality guarantee of the jobs to direct power distribution. The speed control policy applies the minimum speed which can complete the quality guarantee of the jobs to limit the power distributed to all the cores.

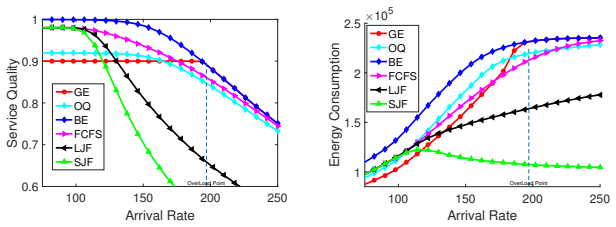
5) Sensitivity Studies: We study the sensitivity of our algorithm under the following scheduling scenarios:

- *Effect of quality function*: We show how quality functions with different concavity can affect the overall quality with the same energy consumption.
- *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 best performance.
- *Effect of discrete speed scaling*: We show how to support *GE* under discrete speed scaling model, and study its impact on quality and energy.

In the last study above, 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 no smaller than the chosen speed, subject to the total power budget. If in this case the power budget cannot support such a discrete speed, we will select the next lower discrete speed.

B. Simulation Setup

We model a web search server with $m = 16$ cores. We model the web search requests with partial evaluation support as follows. The arrival of the requests follows a Poisson process and the deadline of each request is defined to be $150ms$ after its arrival (delayed responses may affect user experience). The service demand of a request follows a 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, respectively. For simplicity, we use number of processing units instead of number of instructions to represent the service demands of the requests. We define the processing capability of a core executing at 1GHz in one second to be 1000 processing units. Our simulation results show this definition works with different parameter values, hence we only present the results with α = 3, $x_{\min}$ = 130 and $x_{\max}$ = 1000 (the mean service demand of a request can then be calculated to be 192 processing units). We set the concavity multiplier in Equation (1) to be $c = 0.003$. Taking different concave quality functions would not change the conclusion of our experimental results.



(a) (b)

Figure 3. Quality and energy comparison of different scheduling algorithms.

We set a total power budget of $H = 320W$, and a *good enough* quality of $Q_{GE} = 0.9$. We 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). The average speed for each core is 2GHz; that is, it can finish 2000 processing units in one second. In our simulations, we do not consider static power since it serves as a constant offset common to all scheduling algorithms.

Under this setting, we define the critical load for job arrival (to differentiate light load and heavy load) to be 154 requests per second, which means that on average the requests consume 77.8% of the server's total processing capacity with the given power budget and number of cores. The system is overloaded when job arrival rate is larger than 198 requests per second, in which case the arrival rate will exceed the total processing capacity of the server.

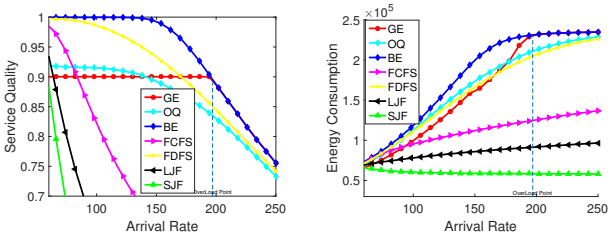
Our scheduling algorithm has three triggering events (described in Section III-E). We set the quantum trigger to be $500ms$ and the counter trigger to be 8 requests. The total simulation time for all experiments is set to be 10 minutes.

C. Comparing GE with Different Scheduling Algorithms

In this subsection, we show that using the same hardware architecture and application model, the *GE* algorithm not only guarantees the user specified quality demand but also achieves more energy savings than the other algorithms.

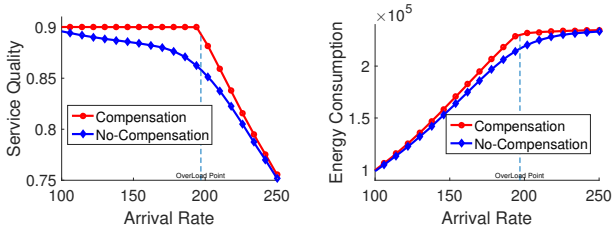
Fig. 3a shows that when the system is not overloaded, the *GE* algorithm can always achieve the stable quality at around $Q_{GE} = 0.9$. *BE* can achieve better quality but since the user-specified quality is expected to be good enough for guaranteed user experience, higher quality is considered unnecessary and cost more energy. *OQ* requires more power and cannot satisfy the quality demand when the workload is heavy. This shows that our compensation policy is more effective than the over qualified policy, which always tries to provide a little more quality beyond the user demand.

We can also conclude from Fig. 3a that the qualities of *LJF* and *SJF* are the worst among all the algorithms, because



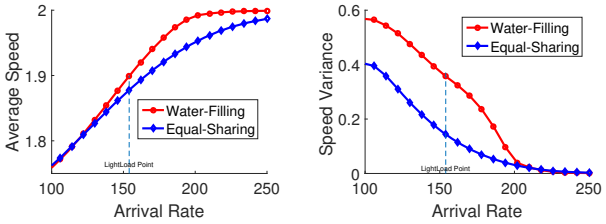
(a) (b)

Figure 4. Quality and energy comparison of different scheduling algorithms with random deadline intervals.



(a) (b)

Figure 5. Quality and energy comparison with and without quality compensation policy.



(a) (b)

Figure 6. The speed variation with different power distribution policies.

they perturb the arrival/deadline order of the jobs. For *LJF*, the longest job is first scheduled but it may have the furthest deadline, while some shorter jobs with earlier deadlines will have to be discarded, thus the quality becomes undesirable. The same is true with *SJF*. *FCFS* achieves relatively higher quality since it respects the deadline order of the jobs. As a result, it can finish more jobs.

Fig. 3b shows that *GE* algorithm costs the least amount of energy among all the algorithms that can satisfy the Q_{GE} requirement. This is because *GE* employs the *LF* job cutting policy to significantly reduce the unnecessary workload and the hybrid power distribution policy to avoid core speed thrashing for saving energy. In the best case, the *GE* algorithm is shown to achieve 23.9% energy reduction compared with the *BE* algorithm. The energy of *SJF* reduces with increasing workload because the algorithm may discard more long jobs with early deadlines. In this case, the short jobs are executed with slow speed for energy efficiency, but the long jobs may not have the chance to be executed before their deadlines. This also explains why *SJF* has the worst quality among all the algorithms.

Fig. 4 shows the results when the job model is modified to allow its service interval to change randomly between $150ms$ and $500ms$. In this experiment, we consider an additional algorithm *FDFS (First Deadline First Service)*, because the deadlines of the jobs may no longer be agreeable. *GE* algorithm is still shown to achieve stable quality around Q_{GE} with the least energy consumption. The results for *GE*, *OQ* and *BE* are similar to the ones shown in Fig. 3, because they consider all the jobs in each scheduling decision. The other algorithms, however, consider only one job at a time. *FCFS* performs extremely bad in this case because some jobs come first but their deadlines are late, which can lead to some urgent jobs to be discard. *FDFS* performs better than other algorithms because it observes the deadline orders of jobs, in which case more jobs can finish to improve the quality. *FDFS* also demonstrates that algorithms that schedule jobs in the order of deadline can achieve better stability and quality.

D. Impact of Compensation Policy

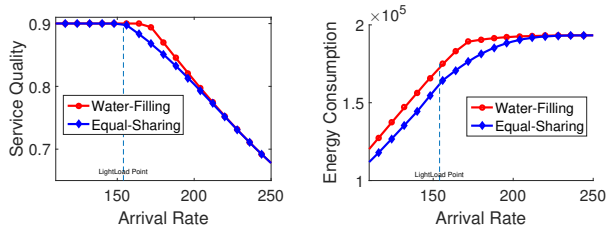
Fig. 5a demonstrates the importance of the quality compensation policy for providing the quality guarantee. The *LF* job cutting policy can aggressively save energy, but it may lead to quality loss. The compensation policy can switch from the *AES* mode to the *BQ* mode to boost quality when it has fallen below the given target Q_{GE}. Fig. 5b also shows that the compensation policy consumes a little more energy in order to achieve the promised quality.

E. Hybrid Power Distribution Policy

Fig. 6 shows that the average speed of *Water-Filling (WF)* is almost the same as that of *Equal-Sharing (ES)* when the load is light. However, the speed variance of *WF* is much larger than that of *ES*. Consequently, we conclude that *WF* together with the compensation policy can increase the energy cost due to the speed thrashing phenomenon under light load. *ES* can efficiently avoid core speed thrashing in this case to save energy consumption with the same quality compared with *WF* (shown in Fig. 7a).

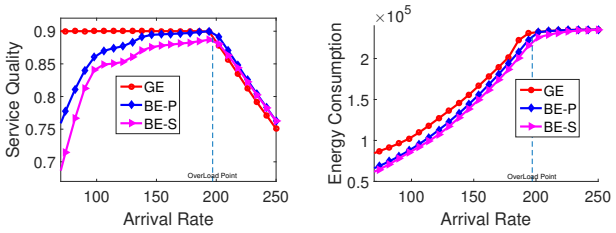
Fig. 6 also shows that both the average speed and the speed variance of *WF* are larger than those of *ES* when the load is heavy (but not very heavy). Because *WF* can take full advantage of the power budget while *ES* cannot exploit the unused power for lightly loaded cores, it explains why *WF* can achieve better quality than *ES* when the workload become heavy (shown in Fig. 7a).

Now, Fig. 7a clearly shows that *ES* can achieve the same quality as *WF* when the load is light. At the same time, Fig. 7b shows that *ES* consumes less energy because it can effectively avoid core speed thrashing. Thus, we employ the



(a) (b)

Figure 7. Quality and energy comparison with different power distribution policies.



(a) (b)

Figure 8. Quality and energy comparison with different control policies.

ES power distribution policy in this case. Fig. 7a also shows that *WF* achieves higher quality when the load is heavy. To guarantee quality, we therefore employ the *WF* power distribution policy in this case. This observation suggests that a hybrid power distribution policy not only avoids significant energy loss from speed thrashing under light load, but also achieves better quality under heavy load.

F. Comparing Control Policies

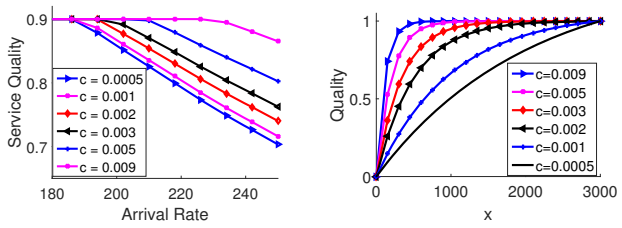
We implement two other policies to control the quality by providing limited power budget or speed in the *BE* algorithm. One is called *power control policy with BE (BE-P)*, which allocates the power according to the users' quality demands. The other is called *speed control policy with BE (BE-S)*, which sets the maximum core speed according to the users' quality demands.

Fig. 8 shows that the *GE* algorithm, which employs the quality control policy, outperforms the other two policies. *BE-P* performs better than *BE-S*, because the workloads of different cores are not balanced. In particular, *BE-P* employs *WF* to distribute the power budget, which helps to balance the power distribution based on the demands of the cores.

When the load is heavy, the performance of the three control policies is similar to one another. In this case, all the cores are overloaded and the algorithms can make full use of the given power budget or speed. Fig. 8b shows that *GE* consumes a little more energy than *BE-P* and *BE-S* to guarantee the user experience.

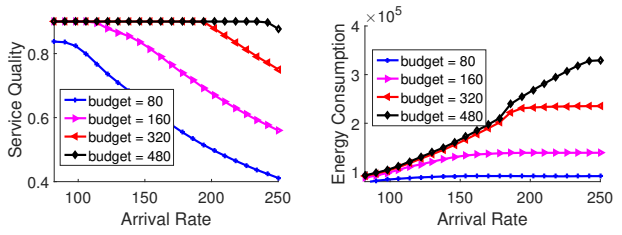
G. Sensitivity Studies

We now report the results of the five sets of sensitivity studies in the following.



(a) (b)

Figure 9. Effect of parameter c on the quality function and service quality of *GE*.



(a) (b)

Figure 10. Quality and energy of *GE* with different power budgets.

1) Effect of quality function: *GE* is designed for *"good enough"* services modeled with concave quality functions. The quality function used in Equation (1) has a parameter c that can affect the concavity of the function. Fig. 9 shows different quality functions with different c values. We observe that larger c increases the concavity of the quality function, which makes the partial evaluation of the *GE* algorithm more effective in gaining more quality for the same amount of work done, thus resulting in higher overall quality.

2) Effect of power budget: The effect of power budget is shown in Fig. 10. High power budget is not at all necessary when the load is light. When the load increases, *GE* can achieve more stable service quality with more power budget. Increasing load also increases the energy consumption till the total power reaches the given budget, after which point higher load has no impact on the energy consumption.

3) Effect of core count: In general, more cores in a server can help achieve higher quality with lower energy (we ignore the effect of static power here). This is because a larger core count can process more data given the same power budget due to the convexity of the power function. It also decreases the potential contention of the jobs at each core, in which case a job can be executed more slowly to save energy.

Fig. 11 shows the impact of different numbers of cores on both quality and energy. We observe that a small number of cores only achieves very limited quality, but consumes a lot of energy. The situation improves with increasing number of cores. The algorithm is able to achieve higher quality and lower energy till the system is saturated when additional cores bear no effect on the distribution of jobs.

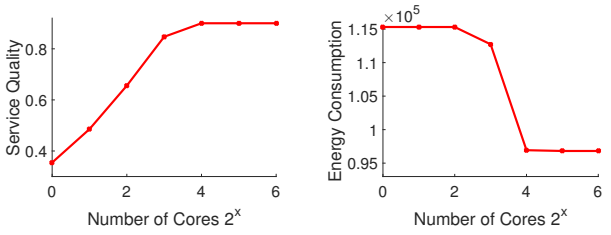


Figure 11. Quality and energy of *GE* with different numbers of cores.

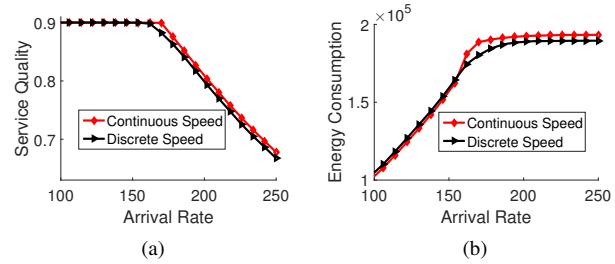


Figure 12. Quality and energy of *GE* with continuous and discrete speed scaling.

4) Effect of discrete speed scaling: In reality, the core speed cannot be changed continuously. Fig. 12 compares the quality and energy of *GE* with continuous and discrete speed scaling. Fig. 12a shows that discrete speed scaling can miss some quality because the cores cannot use the ideal speed to execute the scheduling solutions. Fig. 12b also shows that discrete speed scaling consumes marginally less energy for the same reason.

V. Related Work

In this section, we briefly review some existing work in three related areas.

Approximate Computing: Approximate computing, originally proposed in 1956, has been widely used in lossy compression and numerical computation [26]. Kugler [17] provided a summary of research in approximate computing. There exists work that focused on the kernel level to apply approximate computing to make computing more energy-efficient [8, 22, 24]. Others applied approximate computing at the system level to explore the accuracy of computation and performance-energy trade-offs [1, 5, 10, 16, 21, 25, 27]. Acun et al. [1] proposed a system to tackle power, reliability, and performance, where they use load balancing to achieve higher quality, use profiling data to reconfigure cache and turn off unused network to decrease energy consumption. Our previous work also attempted to achieve the best quality with less energy [9]. The algorithm, however, does not take advantage of approximate computing to reduce energy consumption, and it does not employ the hybrid power distribution policy employed in this paper.

Diminishing Returns: The phenomenon of diminishing returns was first discovered in economics [2], and now it has been widely adopted in computing. Han et al. [12, 13] applied approximate computing combined with diminishing returns and proposed AccuracyTrader and CLAP algorithms for low tail latency and high accuracy services. They proposed to aggregate the data from web search or the CF recommender system. They aggregated the data with high correlations to the result, which contributes more accuracy or quality. For error-tolerant services, diminishing returns can help achieve higher accuracy or quality when we schedule the most efficient part of the data under heavy load. In this paper, we adopted the idea of aggregated dataset and combined diminishing returns and approximate computing to achieve *good enough* computing.

DVFS for Energy Saving: DVFS has been widely explored for achieving higher energy efficiency [11, 19, 20, 29, 32]. Some research is based on system-level DVFS, where all cores must maintain the same speed [18, 23]. Other research is based on core-level DVFS, where each core can set the speed individually [9, 30, 31]. It has been shown that core-level DVFS can have a significant impact on the energy efficiency on multicore systems [31]. Yao et al. [28] first suggested that the power is a convex function of the processor speed, which presents a reasonable simplification for power-speed relationship. We explored this power-speed convex function and combined it with core-level DVFS to achieve greater energy efficiency.

VI. Conclusion

Energy efficiency is an important concern for future high-performance computing and data processing. In this paper, we proposed a novel online scheduling algorithm for *"good enough"* services on multicore servers to guarantee user-specified quality requirement rather than simply contriving to achieve the best quality. The proposed algorithm employs a job cutting policy and a compensation policy to guarantee user-specified quality, and relies on a hybrid power distribution scheme to eliminate the effect of core speed thrashing under light load conditions to save energy. Simulation results show that the algorithm can save up to 23.9% energy compared to the *best effort* scheduling algorithm with marginal and acceptable quality loss, thus responding strongly to the power challenge. We believe that the proposed energy saving method based on approximate computing can also be applied to other big-data applications and employed on different hardware platforms (such as many-core processors), and it will be our future work.

Acknowledgment

This research is supported in part by National Natural Science Foundation of China (Nos. 61440057, 61272087 and 61363019), MOE Research Center for Online Education Foundation (No. 2016ZD302), and the National

Key Research and Development Program of China (No. 2016YFB1000602). This work was also partially supported by NSF Grants ACI-1339745 (XScala) and CNS-1563883.

REFERENCES

[1] B. Acun, A. Langer, E. Meneses, H. Menon, O. Sarood, E. Totoni, and L. V. Kalé. Power, reliability, and performance: One system to rule them all. *Computer*, 49(10):30–37, 2016.

[2] S. Adam. *The wealth of nations*. Aegitas, 2016.

[3] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina, et al. The opportunities and challenges of exascale computing. *Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee*, pages 1–77, 2010.

[4] P. E. Bailey, D. K. Lowenthal, V. Ravi, B. Rountree, M. Schulz, and B. R. De Supinski. Adaptive configuration selection for power-constrained heterogeneous systems. In *43rd International Conference on Parallel Processing*, 2014.

[5] P. E. Bailey, A. Marathe, D. K. Lowenthal, B. Rountree, and M. Schulz. Finding the limits of power-constrained application performance. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 79, 2015.

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

[7] T. D. Burd and R. W. Brodersen. Energy efficient CMOS microprocessor design. In *Proceedings of the 28th Hawaii International Conference on System Sciences*, volume 1, pages 288–297, 1995.

[8] M. Carbin, S. Misailovic, and M. C. Rinard. Verifying quantitative reliability for programs that execute on unreliable hardware. 48(10):33–52, 2013.

[9] Z. Du, H. Sun, Y. He, Y. He, D. A. Bader, and H. Zhang. Energy-efficient scheduling for best-effort interactive services to achieve high response quality. In *Proceedings of the 27th IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 637–648, 2013.

[10] B. Gaudette, C.-J. Wu, and S. Vrudhula. Improving smartphone user experience by balancing performance and energy with probabilistic QoS guarantee. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 52–63, 2016.

[11] J.-J. Han, X. Wu, D. Zhu, H. Jin, L. T. Yang, and J.-L. Gaudiot. Synchronization-aware energy management for VFI-based multicore real-time systems. *IEEE Trans. Computers*, 61(12):1682–1696, 2012.

[12] R. Han, S. Huang, F. Tang, F. Chang, and J. Zhan. Accuracytrader: Accuracy-aware approximate processing for low tail latency and high result accuracy in cloud online services. In *Proceedings of the 45th International Conference on Parallel Processing (ICPP)*, pages 278–287, 2016.

[13] R. Han, S. Huang, Z. Wang, et al. Clap: Component-level approximate processing for low tail latency and high result accuracy in cloud online services. *IEEE Transactions on Parallel and Distributed Systems*, 2017.

[14] 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.

[15] M. Horowitz, T. Indermaur, and R. Gonzalez. Low-power digital design. In *1994 IEEE Symposium on Low Power Electronics: Digest of Technical Papers*, pages 8–11, 1994.

[16] Y. Inadomi, T. Patki, K. Inoue, M. Aoyagi, B. Rountree, M. Schulz, D. Lowenthal, Y. Wada, K. Fukazawa, M. Ueda, et al. Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 78, 2015.

[17] L. Kugler. Is good enough computing good enough? *Communications of the ACM*, 58(5):12–14, 2015.

[18] J. Li and J. F. Martínez. Power-performance considerations of parallel computing on chip multiprocessors. *ACM Trans. Archit. Code Optim.*, 2(4):397–422, 2005.

[19] K. Li. Scheduling precedence constrained tasks with reduced processor energy on multiprocessor computers. *IEEE Trans. Computers*, 61(12):1668–1681, 2012.

[20] P. Martí, C. Lin, S. A. Brandt, M. Velasco, and J. M. Fuertes. Draco: Efficient resource management for resource-constrained control tasks. *IEEE Trans. Computers*, 58(1):90–105, 2009.

[21] J. Mei, K. Li, A. Ouyang, and K. Li. A profit maximization scheme with guaranteed quality of service in cloud computing. *IEEE Trans. Computers*, 64(11):3064–3078, 2015.

[22] S. Misailovic, M. Carbin, S. Achour, Z. Qi, and M. C. Rinard. Chisel: Reliability-and accuracy-aware optimization of approximate computational kernels. *ACM SIGPLAN Notices*, 49(10):309–328, 2014.

[23] 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.

[24] G. Pekhimenko, D. Koutra, and K. Qian. Approximate computing: Application analysis and hardware design. Online available: www.cs.cmu.edu/ gpekhime/Projects/15740/paper.pdf.

[25] R. St Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger. General-purpose code acceleration with limited-precision analog computation. *ACM SIGARCH Computer Architecture News*, 42(3):505–516, 2014.

[26] J. Von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies*, 34:43–98, 1956.

[27] J. Wei and C.-Z. Xu. eQoS: Provisioning of client-perceived end-to-end QoS guarantees in web servers. *IEEE Trans. Computers*, 55(12):1543–1556, 2006.

[28] 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.

[29] H. Yu, Y. Ha, and B. Veeravalli. Quality-driven dynamic scheduling for real-time adaptive applications on multiprocessor systems. *IEEE Trans. Computers*, 62(10):2026–2040, 2013.

[30] 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.

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

[32] X. Zhong and C.-Z. Xu. Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee. *IEEE Trans. Computers*, 56(3):358–372, 2007.