Energy-Efficient Scheduling of Interactive Services on Heterogeneous Multicore Processors

Shaolei Ren†, Yuxiong He‡, Sameh Elnikety‡
†University of California, Los Angeles, CA ‡Microsoft Research, Redmond, WA

***Abstract*—A heterogeneous multicore processor has several cores that share the same instruction set architecture but run at different speeds and power consumption rates, offering both energy efficient cores and high-performance cores to applications. We show how to exploit such processors to make significant energy reduction to serve large interactive workloads such as web search by carefully scheduling requests. Scheduling is a challenging task. Intuitively, we want to run short requests on slow cores for energy efficiency and long requests on fast cores for timely responses. However, there are two key challenges: (1) request service demands are unknown; and (2) the most appropriate core to run a request may be busy.**

We propose an online algorithm, Fast-Preempt-Slow (FPS), which improves response quality subject to deadline and total power constraints. We conduct a simulation study using measured workload from a large commercial web search engine as well as using a variety of synthetic workloads to assess the benefits of FPS. Our results show significant benefits, achievable under a wide variety of conditions: The throughput of a heterogeneous processor is 60% higher than that of the corresponding homogeneous processor with the same power budget; equivalently, to support a large workload as in web search, FPS on the heterogeneous processors reduces the number of servers by approximately 40%.

I. Introduction

We show that running large-scale interactive services on heterogeneous multicore processors saves a significant amount of energy compared to running them on homogeneous multicore processors.

We focus here on interactive services which have the following characteristics: (1) Requests can be executed adaptively, permitting partial evaluation. (2) Since requests can be evaluated partially, there is a response quality profile that improves with the amount of processing. The quality profile is typically concave: Full evaluation gives quality 1.0, and halfway evaluation gives quality normally higher than 0.5. (3) Requests have deadlines since they originate in an interactive environment. This is an important class of workloads, and many large-scale interactive cloud services belong to this class. It constitutes a significant portion of data center workloads. Examples include web search, video-on-demand, map search and online gaming.

In practice, such workloads have two important properties. Although the service demand is bounded, the service demand distribution has a relatively large variance because many requests are short and a few are long. In addition, there are constraints imposed on processing the workload, expressed as quality requirement. For example, a web search provider may specify that requests must be processed within 120 ms deadline with an average quality 0.995.

Next, we turn to modern processors. Today, server-class processors have multiple identical cores, and we call such processors homogeneous CPUs. In the near future, CPUs will have heterogeneous cores, such that all cores run the same instruction set but they run at different speeds, and thus with different power rates: The faster the core, the more power it uses. The power consumption increases faster than speed; therefore, slow cores are energy-efficient while fast cores achieve high performance. We call such processors heterogeneous CPUs, for which several designs have been proposed [2], [3], [12]–[14], [23] and some manufacturers have announced production plans [1].

Due to the large variance of request service demand in interactive services, homogeneous CPUs have their limitations: By using energy-efficient slow CPUs, long requests cannot be completed before their deadline, thereby resulting in a degraded quality, whereas using high-performance fast CPUs consumes a high energy for short requests.

To meet quality requirement and save energy, we propose to run interactive services on heterogeneous CPUs. We show how to schedule requests, meet the timing and quality requirements and assess the energy savings by using heterogeneous CPUs. This is a hard problem to solve. The key idea is to run short requests on slow cores for energy efficiency and run long requests on fast cores to meet the deadline and quality requirements. We want to run each request on the slowest core that can meet the deadline and quality requirement, which minimizes request energy consumption while satisfying service quality. However, there are two key challenges: (1) We do not know request service demands. (2) There are multiple requests but only a limited number of cores. The most appropriate core to run a request may not always be available when the request needs it.

We address the challenges with two important techniques. (1) with unknown service demands, we introduce a key technique: *migrate a request from slower to faster cores during its execution*. Intuitively, short requests are likely to be completed on energy-efficient slow cores, leaving fast cores to process long requests. We also perform a formal analysis, which shows that migrating a single request from slower to faster cores during its execution produces the most energy-efficient schedule. (2) When multiple requests compete for resources, to decide which request gets a faster core and when, we introduce the urgency metric that estimates the desired minimum core speed to complete a request given its current status. We find that *the longer a request is processed, the higher the urgency.* The reason is that, when a request is processed more, its expected service demand increases and its available processing time before the deadline decreases. By assigning faster cores to requests with higher urgency, requests have greater chances to be completed prior to their deadlines, leading to a higher response quality.

Inspired by the two techniques, we develop an online algorithm called Fast-Preempt-Slow (FPS). When a fast core is available, FPS always promotes the most urgent request from a slower core to the faster core in a manner that improves the average response quality. FPS shows significant benefits. Under a fixed quality and deadline requirement, FPS improves the server throughput, and thereby reduces both the number of servers and the total energy required to process a large workload.

We conduct simulation studies using measured workload distribution and quality profile from a large commercial web search engine — Microsoft Bing — as well as using a variety of synthetic workloads and profiles. Our results demonstrate the benefits by using FPS on heterogeneous CPUs. A heterogeneous CPU using FPS improves the average response quality, reduces the quality variance, and improves the high-percentile response quality compared to other traditional scheduling algorithms. Under the same deadline and quality requirements, a heterogeneous CPU using FPS increases the throughput by 60% compared to that of a homogeneous CPU with the same power budget, thereby substantially reducing the number of servers and their corresponding energy consumption in a large-scale interactive service.

The main contributions of the paper are as follows: (1) We characterize an important class of interactive workloads for which heterogeneous CPUs achieve higher energy efficiency than homogeneous CPUs (Section II). (2) We propose an efficient algorithm, FPS, to schedule requests on a heterogeneous CPU to improve the response quality (Section III). (3) We conduct simulation studies to assess the performance of FPS. Our results show that using FPS, heterogeneous CPUs significantly outperform homogeneous ones in terms of improved quality, increased throughput and reduced energy (Section IV).

II. Scheduling Model

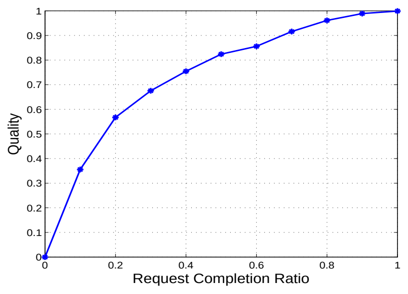
This section discusses request characteristics in interactive services in our environment and our measurement study on Bing. Then, we formalize the job and hardware model.

A. Job Characteristics

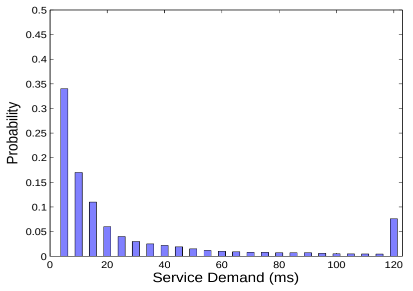
We consider interactive services with three characteristics:

(1) Adaptive execution. Adaptive execution is the flexibility of trading more resources for better quality. Without adaptive execution, a scheduler can either execute a request fully or reject it. In contrast, adaptive execution opens the possibility of partial evaluation in which some requests can be returned before completion, which is a favorable characteristic in many applications such as web search.

(2) Concave quality profile. Because requests can be executed adaptively, a quality profile measures the quality of the response. If a request is executed fully it receives quality 1. If the request is executed partially, it receives a lower quality. The relationship between the quality of the result and the used amount of computational resources is typically a concave function, which reflects diminishing returns.



(a) Quality profile.



(b) Service demand distribution.

Fig. 1: Measured web search workloads.

(3) Deadline. For online interactive services, users expect timely responses; long response times are not acceptable because they cause user dissatisfaction and loss of revenues [25]. Timing constraints are normally expressed as deadlines.

Many important interactive data center workloads possess these characteristics. For example, in finance servers with Monte-Carlo computations, increasing the number of samples reduces the uncertainty of the results. However, the incremental gain of an additional sample becomes smaller when the sample size becomes larger. In video streaming systems, basic layers are more important than refinement layers; the quality of received video streams improves monotonically with the number of received layers but exhibits diminishing returns [30]. Web search is another example, which we discuss next.

B. Measurement Study on Web Search Engine

We conduct a measurement study on Bing. We focus here on the web index serving system which receives user queries and returns the response. This system is a distributed interactive service, not to be confused with the batch-oriented index building system that crawls webpages and builds the web index. The web index contains billions of documents and thus, the index is partitioned and managed among thousands of search servers.

When a query request arrives, it is assigned to an aggregator, which sends the request to the search servers. Each search server returns its matched results to the aggregator, and the aggregator collects them and returns the top N webpages to user. The search servers are the dominant component and consume the majority (over 90%) of hardware resources, and hence our study focuses on search servers.

Search servers support adaptive execution with response deadlines. A server scans an inverted index looking for webpages that match the requested keywords and ranks the matching webpages. The more time the server spends in matching and ranking the documents, the better (i.e., more relevant) the search results get. If the search server does not finish searching for all matching webpages, it can still return the best matches found so far. The search server responds to the aggregator within 120 ms: the aggregator returns its collected results to users without waiting for the delayed responses from search servers.

Quality Profile. Web search has a concave response quality profile. We measure the quality profile in Fig.1(a) using 200K queries from a production trace. Here each request is a web search query including a set of keywords, and the response includes a set of links to the top documents matching the keywords. We obtain the results by running the queries several times with different completion ratios. The response quality compares the set of documents returned in the test with a golden set of base results that are obtained when each request is fully processed. The x-axis of Fig.1(a) is the *request completion ratio* which is calculated as the received processing time divided by its full processing time; the y-axis is the average response quality. The figure demonstrates that the quality profile is monotonically increasing and concave [1]. The concavity comes from the effect of diminishing returns: An inverted index lists important (e.g., popular) webpages first; thus, webpages matched earlier are more likely to rank higher and contribute more to the request quality.

Service Demand. We also measure service demand distribution of Bing requests as shown in Fig.1(b). The majority of the requests are short with service demand less than 40ms but there are still more than 10% long requests with service demand 100ms and above.[2] This diversity in request service demands has been observed in many workloads [18], [19], making heterogeneous CPUs a promising candidate to reduce energy consumption while meeting the deadline and quality requirements.

C. Job Model

A job, or a request, is specified by a tuple (t_a, d, w), where t_a, d, w are its arrival time, lifespan, and service demand, respectively. The job deadline is $t_a + d$ and any processing thereafter does not contribute to the response quality of the job. In many environments, servers do not differentiate requests, leading to the same delay requirement for all requests: we assume that the lifespan of all requests is the same. The service demand is the total work (i.e., CPU cycles) required to complete a request. The request processing time on a core is directly proportional to the inverse of the core speed.[3] We denote the maximum service demand by $\hat{w}$. The actual service demand of a job is unknown to the scheduler until the job is completed. However, the service demand distribution is available by using, for example, online or offline profiling [7], [8], [18]. Thus, $w \in [0, \hat{w}]$ is an unknown random variable, whose probability density function (PDF) and cumulative distribution function (CDF) are denoted by $f(w)$ and $F(w) = \int_0^w f(x)dx$, respectively. We allow a job to migrate from one core to another. In practice, job migration incurs a context switch overhead and our scheduler introduces a small number of job migrations.

A *quality function* $q(r) : \mathbb{R}^+ \rightarrow \mathbb{R}$ maps the request completion ratio (total processed work/service demand) to a quality value gained by executing the request. Each request may have a unique quality profile, which is often unavailable for an online scheduler, and sometimes can be determined only after completing the full request processing. Therefore, we use $q(r)$ to represent expected quality profile of a request, which can be measured through online/offline workload characterization.

D. Hardware Model

A heterogeneous CPU consists of $N \geq 1$ heterogeneous cores, indexed by $1, 2, \cdots, N$, respectively, which are non-uniform in terms of their processing speeds as well as power consumption. Without loss of generality, we assume that the i-th core speed s_i and power p_i satisfy $0 < s_1 \leq s_2 \leq \cdots \leq s_N$ and $0 < p_1 \leq p_2 \leq \cdots \leq p_N$ [7]. Moreover, we assume that a core with a faster processing speed consumes more energy to process a unit of work, i.e., $p_i/s_i \leq p_j/s_j$ for $1 \leq i \leq j \leq N$, since otherwise the faster core is more energy-efficient than the slower core and there is no need to include the slower core in the processor design.

E. Scheduling Objective

Our scheduling objective is to improve the total (or average) response quality of all requests. The benefits of quality improvement can be used to increase throughput and save energy. Higher quality at any given load indicates that, under a fixed quality and deadline requirement, a server can support higher load and its throughput improves, and thereby reducing both the number of servers and the total energy required to serve a large workload.

To fairly compare heterogeneous to homogeneous CPUs, we compare the servers with the same CPU power budget.

III. Scheduling Algorithm

This section proposes an online algorithm FPS to schedule interactive requests on heterogeneous CPUs. We first describe two challenges and the important techniques to address them. Then, we present our scheduling algorithm FPS. Finally, we discuss an enhanced early-termination feature of FPS, which further exploits the concavity of the request quality profile to improve the response quality.

A. Key Insights

Our scheduling algorithm is designed to improve server throughput while achieving a high response quality under a power budget. Intuitively, if the request service demand is known, we want to schedule long requests on fast cores, which ensures a timely high response quality, while scheduling short requests on slow cores, which improves energy efficiency of

[1]Except for a small interval around 0.6 possibly due to imprecision of the measurements.

[2]Service demand is measured on the fast core, the specification of which is provided in Section 5.

[3]We vary the relation between processing time, core speed and power rate in the performance evaluation study.

requests. An ideal scheduler will run every request on the slowest core that can meet the request deadline and quality requirement. However, there are two challenges. (1) Request service demand is often unknown: how can we schedule short requests to slow cores and long requests to fast cores? (2) There are multiple requests but only a limited number of cores. The most appropriate core to run a request may not always be available when the request needs it.

Challenge (1): The challenge is to run short requests on slow cores and long requests on fast cores without knowing the service demand. Naturally, without the service demand information, we cannot find the most appropriate core for a request before its execution. We propose to *migrate a request from slower to faster cores during its execution.* By doing so, short requests are likely to be completed on energy-efficient slow cores, saving fast cores to process long requests.

In addition to being intuitive, our proposed approach is also supported by a formal analysis in Theorem 1, which shows that migrating a single request from slower to faster cores during its execution produces the most energy-efficient schedule. In essence, minimizing the energy consumption of an individual request can be translated to maximizing the throughput of a server system under a given power budget, which is aligned with our scheduling objective.

Theorem 1: Suppose that a request has deadline d, service demand CDF F and quality profile function q that is monotonically increasing and concave. To meet any average quality requirement, the core speed for processing the request[4] is non-decreasing under the optimal schedule that minimizes the average CPU energy consumption of the request.

Proof. As the first step of the proof, we show how to meet request quality requirement. The request quality is related to the quality profile function q and the amount of the processed work before the deadline. Let us define the target work $\bar{x}$, which specifies the maximum amount of work that is completed prior to the deadline regardless of the actual service demand of the request. If the request has a total service demand less than $\bar{x}$, the request runs until completion, whereas the request is terminated at work $\bar{x}$ otherwise. Since the quality profile function q is monotonically increasing in $\bar{x} \in [0, \hat{w}]$, the average (expected) response quality increases from 0 to 1. Therefore, given an average quality requirement $0 \le r \le 1$, we can find a fixed target work $\bar{x} \in [0, \hat{w}]$ that satisfies the quality target.

After finding the target work $\bar{x}$, we formulate the problem of minimizing the average CPU energy consumption of a request as follows:

$$\min_{\mathcal{X}} \int_0^{\bar{x}} \left[1 - F(x)\right] \cdot \frac{p_{\mathcal{X}}(x)}{s_{\mathcal{X}}(x)} dx, \tag{1}$$

$$s.t., \quad \int_0^{\bar{x}} \frac{1}{s_{\mathcal{X}}(x)} dx \le d, \tag{2}$$

where $\mathcal{X}$ is a schedule that specifies which cores are used and in which order to process the single request, $s_{\mathcal{X}}(x) \in \{s_1, s_2 \cdots s_N\}$, and $p_{\mathcal{X}}(x) = p_i$, if $s_{\mathcal{X}}(x) = s_i$. Constraint (2) guarantees that the schedule $\mathcal{X}$ satisfies the deadline. We now prove the theorem by contradiction. Suppose that $s_{\mathcal{X}'}(x)$ minimizes (1) while, under the schedule $\mathcal{X}'$, the job is first processed by a faster core and then by a slower one. Thus, there exist x_1 and x_2 such that $0 \le x_1 < x_1 + dx \le x_2 < x_2 + dx \le \bar{x}$ and $s_{\mathcal{X}'}(x_1') > s_{\mathcal{X}'}(x_2')$, where $x_1' \in [x_1, x_1 + dx]$, $x_2' \in [x_2, x_2 + dx]$ and dx is a sufficiently small positive number.

Since faster cores consume more energy to process one unit of work than slower ones, we show that the following inequality holds:

$$\begin{aligned}
&[1 - F(x_1')] \cdot \left[\frac{p_{\mathcal{X}'}(x_1')}{s_{\mathcal{X}'}(x_1')} - \frac{p_{\mathcal{X}'}(x_2')}{p_{\mathcal{X}'}(x_2')}\right] \\
&+ [1 - F(x_2')] \cdot \left[\frac{p_{\mathcal{X}'}(x_2')}{s_{\mathcal{X}'}(x_2')} - \frac{p_{\mathcal{X}'}(x_1')}{s_{\mathcal{X}'}(x_1')}\right] \\
&= \left[\frac{p_{\mathcal{X}'}(x_1')}{s_{\mathcal{X}'}(x_1')} - \frac{p_{\mathcal{X}'}(x_2')}{s_{\mathcal{X}'}(x_2')}\right] \cdot [F(x_2') - F(x_1')] > 0.
\end{aligned} \tag{3}$$

Thus, we have $[1 - F(x_1')] \cdot \frac{p_{\mathcal{X}'}(x_1')}{s_{\mathcal{X}'}(x_1')} + [1 - F(x_2')] \cdot \frac{p_{\mathcal{X}'}(x_2')}{s_{\mathcal{X}'}(x_2')} > [1 - F(x_1')] \cdot \frac{p_{\mathcal{X}'}(x_2')}{s_{\mathcal{X}'}(x_2')} + [1 - F(x_2')] \cdot \frac{p_{\mathcal{X}'}(x_1')}{s_{\mathcal{X}'}(x_1')}$. By evaluating the integral in (1), we see that the expected energy consumption can be further reduced if we exchange the order of cores processing the $x_1'-$th cycle and the $x_2'-$th cycle, for $x_1' \in [x_1, x_1 + dx]$ and $x_2' \in [x_2, x_2 + dx]$, while keeping the rest of the schedule $\mathcal{X}'$ unchanged. This contradicts the assumption that $\mathcal{X}'$ minimizes (1) and hence, proves Theorem 1. ■

Inspired by Theorem 1, FPS migrates a request only from slower to faster cores when necessary.

Challenge (2): When two requests are competing for a faster core, FPS uses the faster core to run the more urgent request, which is also the request that has arrived earlier. We define the "urgency" of a request as follows.

Definition 1: The *urgency* of a request is defined as the expected minimum core speed to complete the request prior to its deadline. Mathematically, the urgency is expressed as follows:

$$\begin{aligned}
U &= \frac{\mathbb{E}\left\{w - w_0 \,|\, w \ge w_0\right\}}{r} \\
&= \frac{\int_{w_0}^{\hat{w}} w f(w \,|\, w \ge w_0) dw - w_0}{r},
\end{aligned} \tag{4}$$

where w_0 is the completed work, r is the remaining lifespan of the request, and $f(w \,|\, w \ge w_0)$ is the PDF of the service demand conditioned on that the request has completed w_0 work.

Urgency indicates the desired core speed to complete a request upon its deadline. Therefore, by assigning faster cores to requests with higher urgencies, jobs have greater chances to be completed prior to their respective deadlines. To show the trend on job urgency and motivate the techniques of FPS, Fig. 2 shows a lower bound on the urgency value where we assume that all the completed work of the service request is done by the fastest core and the request is processed immediately upon arrival. Fig. 2 depicts urgency versus the completed work for the measured service demand distribution in our search server, where the $x-$axis is the completed work (in ms) and the $y-$axis is the desired core speed normalized

[4]This theorem considers the case where the desired cores for a request are always available. The case when multiple requests are competing for cores is addressed in Challenge 2.

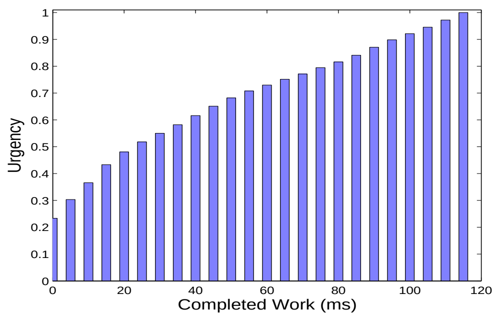


Fig. 2: Urgency versus completed work.

with respect to the fastest core speed.[5] *A key observation is that as the request is processed more, its urgency increases.* The reason is that, when a request is processed more, its expected service demand increases whereas its available time before deadline decreases. The general trend of the urgency curve is similar for other widely used service demand distributions such as exponential distribution and Pareto distribution. This observation motivates FPS to use faster cores to run the request that arrives earlier because that request has higher urgency.

B. FPS Algorithm

When a faster core is idle, FPS always promotes the most urgent job from a slower core and executes it on a faster core, where the most urgent job is also the job with the earliest arrival time and the earliest deadline. The pseudo-code of FPS is shown in Algorithm 1. Let us further demonstrate FPS by considering the following three example cases.

Case 1 (No jobs are available): A new job will be processed by core N, which is the fastest available core.

Case 2 (Only the fastest core is busy): A new job will be processed by core $N-1$, which is the second fastest core; it will be scheduled to core N after the earlier job leaves.

Case 3 (All the cores are busy): Whenever an existing job leaves because completion or deadline expiration, the job with the highest urgency being processed by a *slower* core will be scheduled to the core previously processing the job that leaves, and similar operations for the other jobs. The job waiting in the head of the queue will be scheduled to the slowest core. No job migrates between cores that have the same speed.

Algorithm 1 shows three properties of FPS:

- If a request migrates, it always migrates from a slower to a faster core. From Theorem 1, this property reduces the average energy consumption of each request, thereby improving the system throughput under a power budget.
- A faster core always runs a request with higher (or equal) urgency than that of any slower core. This property increases the chance of completing all requests before deadline and improves the response quality.
- When there are $1 \leq k < N$ requests where N is the number of cores, the fastest k cores are used to process these k requests, which improves the response quality.[6]

[5]During actual processing, request urgency is impacted by its waiting time in the queue and its execution history, making the urgency in Fig. 2 a lower bound.

Algorithm 1 FPS

```
Require: Active job queue $\mathcal{Q}$, core processing speeds $0 < s_1 \leq s_2 \leq \cdots \leq s_N$
 1: Assign urgencies to all jobs.
 2: $i \leftarrow N$
 3: while $i \geq 1$ do
 4:   if core $i$ is idle then
 5:     job $\mathcal{J}$ = job being processed on a slower core than core $i$
        (or waiting in the queue) and with the highest urgency
 6:     if job $\mathcal{J}$ is null then
 7:       break
 8:     else
 9:       schedule job $\mathcal{J}$ to core $i$
10:     end if
11:   end if
12:   $i - -$
13: end while
14: return
```

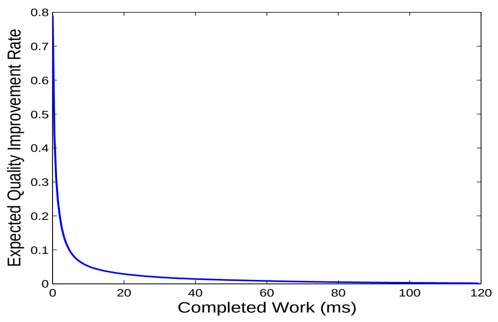


Fig. 3: Expected quality improvement rate versus completed work.

C. Early Termination Feature of FPS

We introduce an enhancement for FPS to further exploit the concavity of the quality profile and improve the response quality. The variation of FPS with the new feature is referred to as FPS-ET. As the concave quality profile exhibits diminishing returns, a job which has been processed for a long time may not contribute to the overall quality as much as a new job. Hence, by early terminating a job that can only bring marginal quality improvement, the core can be utilized to process "fresher" jobs that are more likely to bring higher quality gain and improve the overall response quality. To formally illustrate this point, we define the expected quality improvement rate as follows.

Definition 2: If a job has been processed for $x \in [0, \hat{w}]$ amount of work, its *expected quality improvement rate* is defined as

$$I(x) = \mathbb{E}\left\{\frac{dq(\frac{x}{w})}{dx} \,|\, w \geq x\right\} = \int_x^{\hat{w}} \frac{dq(\frac{x}{w})}{dx} f(w \,|\, w \geq x) dw. \quad (5)$$

[6]Although one CPU may consume more energy using FPS than using other scheduling algorithms (as fast cores are always used whenever there are requests), the objective of FPS is to maximize the overall response quality. Therefore, as shown later, FPS saves energy by processing more requests (i.e., higher throughput) on each server subject to the average quality requirement, which effectively reduces the number of required servers as well as the corresponding energy consumption to sustain the total workload.

The expected quality improvement rate $I(x)$ denotes the expected quality improvement with a unit of additional work if the job has completed x amount of work. We plot the expected quality improvement rate for the search server workload in Fig. 3, which shows that the more a request is processed, the smaller its expected quality improvement rate is. The decreasing expected quality improvement rate is also true for a wide range of service demand distributions (e.g., exponential, Pareto distribution) when the quality profile is linear or concave, which therefore motivates early termination of long requests.

FPS-ET finds the optimal "early termination" time to maximize the total (expected) response quality, which can be formulated as a convex optimization problem. Suppose that there are M available jobs, indexed by $1, 2, \cdots, M$, following an increasing order of their arrival times. In particular, job 1 is processed by the fastest core N, job 2 by core $(N-1)$, $\cdots$, job N, if any, by core 1, while the remaining jobs are waiting in a queue. Next, we introduce a set of decision variables, namely, $\mathbf{y} = (y_1, y_2, \cdots, y_M)$, where y_i represents the reserved time for which job i is scheduled to be processed on the fastest core N. Hence, for job 1, the reserved (additional) work to be processed before termination is simply $y_1 s_N$. While job 1 is processed by core N, job 2 is processed by core $(N-1)$ for a duration of y_1, since it can only be scheduled to core N after job 1 is terminated. Overall, the reserved work to be processed for job 2 before termination is $y_1 s_{N-1} + y_2 s_N$, where y_1 and y_2 are the reserved time for which job 2 is processed by core $(N-1)$ and core N, respectively. Similarly, we can express the reserved work to be processed for job i before termination as

$$\tilde{w}_i = \sum_{j=\min\{1, i-N\}}^{i} y_j \cdot s_{(N-i+j)}. \qquad (6)$$

Thus, if job i has been processed for w_i unit of work, the expected quality of job i is

$$\bar{q}_i(\mathbf{y}) = \int_{w_i}^{\hat{w}} q\left(\min\left\{1, \frac{w_i + \tilde{w}_i}{w}\right\}\right) f(w \mid w > w_i) dw, \qquad (7)$$

which is concave in $\tilde{w}_i = \sum_{j=\min\{1,i-N\}}^{i} y_j \cdot s_{(N-i+j)}$ and hence also in the decision variable $\mathbf{y}$, as the composition preserves concavity [22]. Let r_i be the available remaining lifespan for job i, for $i = 1, 2, \cdots, M$, with deadline constraints : $y_1 \le r_1$, $y_2 + y_1 \le r_2$, $\cdots$, $\sum_{j=\min\{1,M-N\}}^{M} y_j \le r_M$. Thus, we provide the formulation of finding the optimal termination time as following:

$$\textit{OptTerm}: \quad \max_{\mathbf{y}=(y_1, y_2, \cdots, y_M)} \sum_{i=1}^{M} \bar{q}_i(\mathbf{y}), \qquad (8)$$

$$s.t., \quad \mathbf{y} \succeq \mathbf{0}, \qquad (9)$$

$$\sum_{j=\min\{1,i-N\}}^{i} y_j \le r_i, \ \forall i = 1, 2, \cdots, M, \qquad (10)$$

where the objective function is given in Eqn. (7), "$\succeq$" in constraint (9) represents the element-wise inequality, (9) specifies that the reserved time for each job on the fastest core must be greater than or equal to zero, and constraint (10) specifies the deadline constraint. With affine constraints and a concave objective function, the OptTerm formulation is convex programming, to which efficient algorithms (e.g., interior-point methods) are available [22]. FPS-ET calls OptTerm whenever the set of active jobs changes (e.g., a new job arrives, an existing job is completed or job 1 is early terminated).

To summarize, FPS possesses many desirable properties. FPS saves energy by migrating requests from slower to faster cores; it improves the response quality by running more urgent requests on faster cores, meeting the deadlines of more requests. FPS is computationally efficient: It does not require the request service demand information and it bounds the number of job migrations. Each job may migrate only up to $K-1$ times in the worst case, where K is the number of different core speeds, regardless of the server load. In practice, the number of migrations per job is much less than $K-1$, as we will show in Section IV.

TABLE I: Core Specification: Speed and power.

Core	Speed	Normalized speed	Power	Normalized power
Fast	3.33 GHz	1.0	32.5W	1.0
Medium	1.67 GHz	1/2	6.5W	1/5
Slow	1.11 GHz	1/3	2.7W	1/12

TABLE II: Server CPU Configuration: CPUs use the same power, but use different cores.

Name	Fast	Medium	Slow	CPU power
Hom-02f	2	0	0	65W
Hom-10m	0	10	0	65W
Hom-24s	0	0	24	64.8W
Het-1f-3m-5s	1	3	5	65.5W

IV. Performance Evaluation

This section presents simulation studies to evaluate the performance of FPS and demonstrates the advantage of heterogeneous CPUs over homogeneous CPUs. We perform five sets of experiments:

- Heterogeneous versus Homogeneous CPUs: Compare three homogeneous CPUs with heterogeneous CPUs on average quality, variance and 95%-quality.
- Algorithms comparison: we compare FPS with two well-known scheduling algorithms for homogeneous CPUs and two algorithms for heterogeneous CPUs.
- Impact of early termination in FPS: we show the advantage of using the additional early termination feature of FPS.
- Impact of migration overhead: we evaluate the performance of FPS in the presence of migration overhead.
- Sensitivity study: we evaluate FPS under different hardware configurations, workload and performance characteristics.

The first four sets are driven by the web search workload, and in the last experiment, we perform sensitivity study using various synthetic workloads.

A. Setup

Web search server: We model Bing, a large commercial web search engine, which accepts users' search queries and returns the top L webpages. The search and ranking process is an essential part of the web search engine and is CPU-intensive, making web search a CPU-bound application [29]. Our experiments use the following parameters, as used in the production environment: request delay deadline 120 ms and average request quality requirement 0.995. Service demand distribution is measured from production servers and shown in Fig. 1(b). Our experiments use a Poisson process to model request arrivals. We vary the mean query arrival rate which is measured in queries per second (QPS).

We approximate the measured quality profile of Bing in Fig. 1(a) using the smooth function in Eqn. (11) for the convenience of analysis:

$$q\left(\frac{x}{w}\right) = \frac{1 - e^{-b\cdot\min\left(1,\frac{x}{w}\right)}}{1 - e^{-b}}, \tag{11}$$

where $b = 3.53$ is the constant governing the curvature of $q(x)$, and x is the completed work for a job with service demand w. As shown in Fig. 4(a), the approximated profile is close to the measured profile (with a mean square error of 0.0093).

Hardware configuration: We consider three heterogeneous cores as specified in Table I. The fast core represents a conventional server core, such as Intel Core i7 [26], a medium core represents an energy-efficient core such as Atom N455 [27], and a slow core is typically used in embedded systems, such as Atom Z510 [28]. We use the fast core (3.33 GHz, 32.5W) as a reference to normalize the actual core speeds and power consumption in our experimental results. We assume that the request processing speed is linear to the clock rate in the first four experiments, while we evaluate the impact of nonlinear speed-up in the sensitivity study (Section IV-F).

We fix the maximum power budget for each CPU. The normalized power budget for each CPU is 2, which corresponds to 32.5x2=65W.[7] Thus, the available homogenous CPUs are: (1) Homogeneous CPU with 2 fast cores (Hom-02f); (2) Homogeneous CPU with 10 medium cores (Hom-10m); and (3) Homogeneous CPU with 24 slow cores (Hom-24s). Our default heterogeneous CPU uses 1 fast core, 3 medium and 5 slow cores (denoted as Het-1f-3m-5s). These configurations are summarized in Table II. Other possible choices for heterogeneous CPUs are discussed later.

B. Heterogeneous versus Homogeneous CPUs

We compare the three homogeneous CPUs (using a FIFO scheduler) to the heterogeneous CPU Het-1f-3m-5s (using FPS). In a FIFO scheduler, all the jobs wait in a single queue and an idle core pulls the job from the queue head and processes it until completion or expiration of the deadline. The FIFO scheduler is used in Bing servers, and it is commonly used by other servers as well. Notice that FPS and FIFO are equivalent on a homogeneous CPU.

Improved average quality: Fig. 4(b) depicts the average response quality on the y-axis against load on the x-axis. It shows that by applying FPS, the heterogeneous CPU outperforms the three homogeneous CPUs in the average response quality for a wide range of loads, which translates into higher throughput subject to a fixed quality requirements. For instance, we focus on the throughput at the target quality 0.995 for web search, which we call the 0.995-throughput. It significantly increases (by nearly 60%) on Het-1f-3m-5s compared to Hom-02f. The core utilization at quality 0.995 is as follows: for Hom-02f at 44 QPS, it is 59%, and for Het-1f-3m-5s at 71 QPS it is 90% fast, 55% medium, 8% slow cores. Hom-10m and Hom-24s can only support a maximum average quality of approximately 0.985 and 0.965. Fig. 4(b) also shows that Hom-02f can yield a high quality when the throughput is low (e.g., < 40), whereas Hom-10m and Hom-24s cannot achieve sufficiently high quality. Therefore, by integrating the high processing capability of fast cores with the low power consumption of medium and slow cores, a heterogeneous CPU can satisfy the stringent quality requirement (e.g., 0.995) while supporting a high throughput.

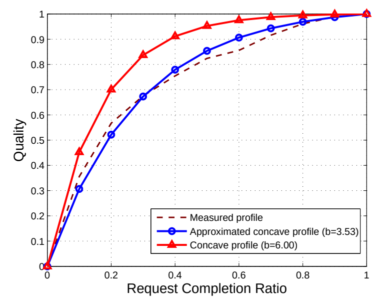
Reduced quality variance: Fig. 4(c) shows the variance of the response quality: the heterogeneous CPU using FPS has the lowest variance. Under Hom-24s and Hom-10m, long jobs may have very low quality, resulting in large variances. Hom-02f has little quality variance when QPS < 30, since it can almost complete every job when the throughput is low. However, when the load increases, the queue length and queuing time increase; long requests may get insufficient service before their deadline, thereby resulting in a reduced quality and an increased variance. When using a Het-1f-3m-5s, a long request that cannot get a fast core immediately can still be processed on one of many slow and medium cores and migrates to the fast core later, resulting in an improved quality and a reduced variance.

Improved 95%-quality: High-percentile quality is of considerable interest since many commercial services specify their service level agreement (SLA) using both high-percentile quality and average quality of responses [24]. For example, a web search engine can target to offer an average quality of 0.995 and a quality of 0.90 for at least 95% requests, which we call 95-percentile quality. The high-percentile quality depends on the response quality distribution. Fig. 4(d) shows that a heterogeneous CPU improves the 95%-quality over homogeneous CPUs on moderate and heavy loads, which is the result of improved average quality and reduced variance.

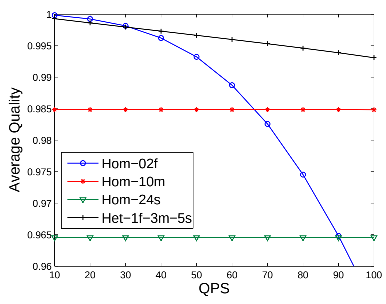
Reducing number of servers: To highlight the hardware and energy cost reduction by using a heterogeneous CPU with FPS scheduling, we consider a total load of 10,000 QPS and compute how many servers are needed to serve the workload subject to various average quality requirements. As discussed in the simulation setup, our CPUs are configured under (approximately) the same power budget. Fig. 4(e) shows the number of servers, for three CPUs, Het-1f-3m-5s, Hom-02f, and Hom-10m. Hom-24s is not shown as it does not meet the depicted quality range. For an average quality of 0.995, using the heterogeneous CPU reduces the number of servers by approximately 40%, which can also be translated into a significant reduction in the energy usage in large data centers.

Small number of job migrations: Given K different core speeds available, a job may migrate up to $K - 1$ times in the worst case. Het-1f-3m-5s has cores with three different speeds and thus, the upper bound on the number of job migrations is 2. Nevertheless, we observe that job migration does not occur frequently. The average number of migrations per job is 0.35

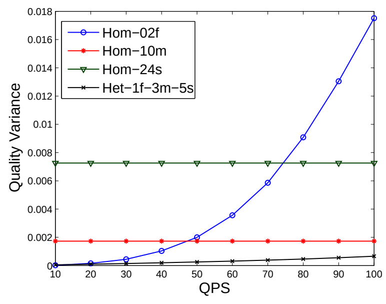
[7]Considering other power budgets gives us similar results.



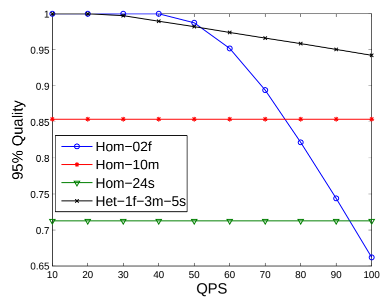
(a) Quality profiles.



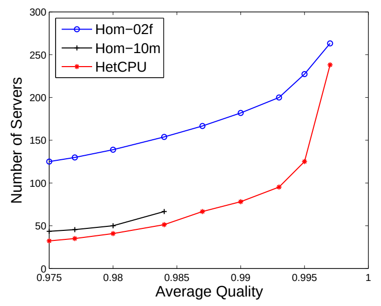
(b) Average quality.



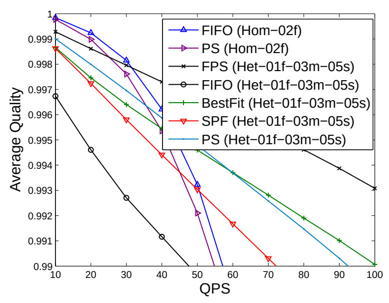
(c) Variance.



(d) 95%-Quality.



(e) Number of servers.



(f) Different scheduling algorithms.

Fig. 4: Figure (a) shows the request quality profiles. Figure (b), (c), (d) and (e) compare heterogeneous to homogeneous CPUs under different performance metrics. Figure (f) compares different scheduling algorithms on the heterogeneous CPU.

at 40 QPS and increases to 0.75 at 100 QPS. The small number of job migrations makes FPS appealing in practice.

C. Scheduling Algorithms Comparison

This section compares FPS with four other scheduling algorithms. First, we compare FPS to two well-known scheduling algorithms — FIFO and processor sharing (PS). We then look at a scheduling algorithm (Slow Preempt Fast, or SPF) which, as the exact opposite of FPS, migrates job from faster cores to slower cores. Our results show that FPS outperforms all three of them.

The above three algorithms as well as FPS are nonclairvoyant, which do not require the knowledge of request service demand. On the other hand, a clairvoyant scheduler knows the request service demand. Intuitively, a clairvoyant scheduler wants to run each request on the slowest core that can complete the job before its deadline, which we call BestFit. Our results show that, even without knowing the request service demand, FPS still outperforms BestFit which knows the request service demand.

Comparing FPS with FIFO and PS: Fig. 4(f) shows the performance of FPS and FIFO on Het-1f-3m-5s. FPS achieves a significantly higher quality than FIFO which cannot support a 0.995-throughput higher than 20 QPS. FPS outperforms FIFO because FPS migrates requests from slower to faster cores. FPS completes many short requests on slower cores and spares faster cores for long requests. However, using FIFO, the assignment of cores does not depend on the request service demand, and processing long requests using slower cores inevitably degrades the total response quality.

PS is another well-known scheduler for homogeneous CPUs, where processors are assigned to requests in a round-robin fashion. We extend PS to heterogeneous CPUs similarly: when the number of jobs are larger than or equal to the number of cores, the jobs equally share all the available cores; when the number of jobs M are smaller than the number of cores N, the jobs equally share the M fastest cores in the round-robin fashion.[8] Fig. 4(f) shows that FPS achieves a higher quality than PS. This is because the migration policy of FPS gives long requests a higher chance to use fast cores. Nevertheless, when using PS, fast cores are equally shared among short and long requests and hence, those long requests which really need the fast cores may not get enough share.

The results of FIFO and PS indicate that the classic scheduling algorithms on homogeneous CPUs do not consider the matching of request service demands and the heterogeneous power-performance characteristics of cores. Therefore, new scheduling algorithms are required to efficiently exploit the benefits of heterogeneous CPUs. Moreover, since the request service demand is unknown, we cannot find the most appropriate core for a request before its execution. However, during the execution, we progressively have more information on the request (e.g., request urgency) such that we may refine our scheduling decision. Therefore, *job migration among cores is important in order to improve the scheduling decision when request service demand is unknown*. Next, we examine the impact of different migration orders.

Comparing different migration orders. We consider a

[8] We assume that the context switch and migration overhead of PS is 0 and therefore the quality result is an upper bound on PS performance.

scheduler, SPF, in which job migration follows the reverse order of FPS and jobs are scheduled from fast cores to slow ones. Each job is processed until completion or expiration. If the number of jobs is smaller than the number of cores, all the jobs are processed by the fastest available cores. We show in Fig. 4(f) that FPS achieves a much higher quality than SPF. By using FPS to migrate jobs from slower to faster cores, short jobs are likely to be completed on slower cores, saving faster cores to process long jobs. In contrast, in SPF, short jobs are completed by faster cores whereas long jobs, which have higher "urgency", are processed by slower cores. Thus, it is likely that long jobs do not get fully completed before their respective deadlines. *This comparison shows the importance of job migration order from slower to faster cores.*

Comparing FPS with a clairvoyant scheduler: Even with known service demands, scheduling multiple jobs on a heterogeneous CPU is a challenging task. Here, we propose a clairvoyant scheduler, BestFit, which tries to schedule each job with the minimum energy in a greedy fashion. Specifically, each core maintains a separate queue and, a new job joins the queue served by the slowest core that can complete the job before its deadline. If none of the cores can complete the job because of a large number of waiting jobs, the job will be scheduled to the queue that produces the highest quality for the job. Hence, BestFit is a greedy clairvoyant scheduler with known service demands but without job migration. It is similar to scheduling algorithms in prior work [12], [13] in that jobs are mapped to the most "appropriate" cores.

Fig. 4(f) shows that, rather surprisingly, FPS without knowing request service demand outperforms BestFit that knows the service demand. We find that the major problem of BestFit is that it does not consider job migration. For example, consider the following scenario: A long job arrives and the best core to run the job is the fast one but the fast core is running another job. BestFit will let the job wait for the fast core to finish even when there are other medium or slow cores available. A better strategy is to use the medium or slow cores to run the job first and migrate it to the fast core when it becomes available. This is a general problem for schedulers that do not consider job migration, which lose the flexibility to use all available resources efficiently. In other words, *even when the request service demand is known, migration is important: it allows using available system resources and, in particular, long requests can make progress on slower cores first before migrating to available faster cores.*

The above comparison shows that classic algorithms are inefficient on heterogeneous CPUs and FPS exploits the benefits of core-level heterogeneity.

D. Impact of early termination in FPS

We investigate the performance improvement resulting from FPS-ET, which is FPS with early termination. Fig. 5 presents the average quality of FPS and FPS-ET using the quality profile in Eqn. (11) with two different values for parameter b: (1) $b = 3.53$, the approximated profile we have used for the previous experiment and (2) $b = 6.00$, a more concave profile as shown in Fig. 4(a). The results show that FPS-ET improves response quality over FPS because FPS-ET evicts long jobs processed by the fast core to improve the average quality. For example, at QPS=180 and $b = 6$, 23.4% of jobs are processed by the fast core and 97.9% of them get early terminated by FPS-ET. Next, when b is larger, the quality profile is "more concave" and the performance improvement by using FPS-ET becomes more substantial. The reason is that with the increase in concavity, processing a long job at its later stage brings only a negligible return, and thus, evicting it using FPS-ET saves the resource to other jobs with a higher potential gain in the overall quality.

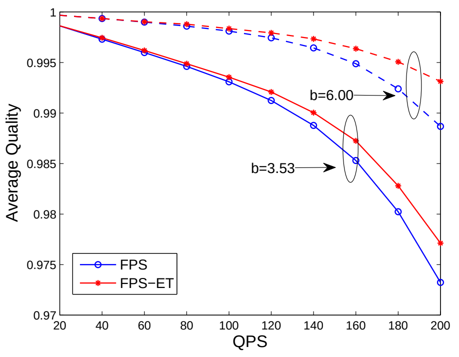


Fig. 5: FPS and FPS-ET with the early termination feature.

E. Impact of migration overhead

The previous discussion assumes zero overhead for job migrating from one core to another. This section presents the performance of FPS with different migration overheads. Job migration between cores requires a context switch for both cores. In practice, when the size of a job's working set is larger, a context switch is more costly because of cache warm-up, which may take tens of microseconds to a millisecond [31]. In our experiment, we model three migration overhead values: 0ms, 1ms and 2ms in Fig. 6, which also shows the curves of the three homogenous CPUs using FIFO (which does not incure any migration) for reference. Fig. 6 shows that, even with a rather high migration overhead of 2ms, FPS performs well with little performance degradation compared to that with zero overhead.

This is mainly due to two factors: (1) the migration overhead, which is typically at the range of tens of microseconds to a couple of milliseconds, is much smaller compared to the deadline which is often a few hundred milliseconds for interactive services; and (2) since most jobs are short and likely to be completed by slow cores without migration, the average number of job migrations produced by FPS is rather small. For example, the average number of migrations per job is 0.35 at 40 QPS and increases to 0.75 at 100 QPS. Therefore, even in the presence of migration overhead, the heterogeneous CPUs using FPS still outperforms homogeneous CPUs with higher response quality.

F. Sensitivity Study

This section evaluates the sensitivity of FPS by considering different hardware configurations, service demand distributions, quality profile functions and job completion times. The

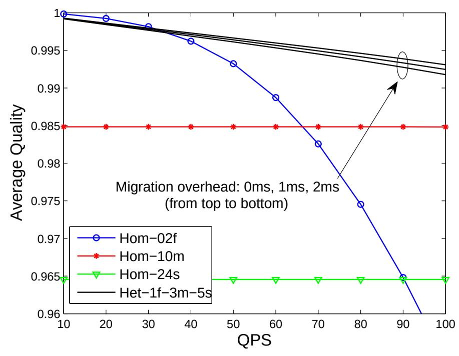


Fig. 6: Impact of different migration overheads.

simulation results show that heterogeneous CPUs with FPS are robust against these factors and outperform homogeneous CPUs and heterogeneous CPUs with other scheduling algorithms.

Different core configurations: We use all heterogeneous core configurations containing one fast core. In Fig. 7, we show that, using FPS, all heterogeneous CPU configurations (except for the one with 1 fast and 12 slow cores) support a higher 0.995-throughput than Hom-02f. Beyond 50 QPS, all heterogeneous CPU configurations perform better than the homogeneous one. For example, the best available heterogeneous CPU Het-1f-3m-5s can sustain a 0.995-throughput of approximately 80 QPS, and Het-1f-5m using only fast and medium cores can achieve a 0.995-throughput very close to 80 QPS. This is because, many jobs are short and thus, given the same power budget, having multiple energy-efficient but slow cores is preferable to having a single fast but energy-inefficient core, which is only used to process long jobs. We also perform experiments with different speed selection of the cores, and the results show that heterogeneous CPUs are not very sensitive to the core speeds. In other words, there is a large set of heterogeneous configurations meeting the quality requirement with a high throughput.

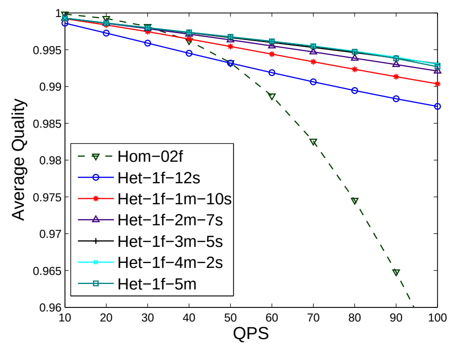


Fig. 7: Sensitivity study on different core configurations.

Service demand distributions: We consider synthetic workloads with three common service demand distributions: (1) exponential, (2) Pareto, and (3) bipolar distributions. All distributions have approximately the same mean, 28 ms, as in the search server workload. The Pareto distribution is as follows:

$$f(x) = \frac{1.2x_m^{1.2}}{x^{2.2}}, \text{ for } x \geq x_m, \tag{12}$$

where $x_m = 9.33$. For both exponential and Pareto distributions, we cap the maximum service demand at 120ms. For the bipolar distribution, we assume that the jobs have only two possible service demands: 10ms with a probability of 0.8 and 100 ms with a probability of 0.2. The variances of the measured search workload, capped exponential, capped Pareto and bipolar distributions are 1196.9, 692.4, 765.4, 1300.1, respectively.

Fig. 8(a), 8(b) and 8(c) evaluate FPS on heterogeneous CPUs with capped exponential, capped Pareto, and bipolar distributions as described in the setup subsection. The results show that applying FPS on heterogeneous CPUs consistently achieves a higher 0.995-throughput than that achieved by other algorithms on heterogenous CPUs and that achieved by homogenous CPUs under various service distributions. The benefit of using FPS on heterogeneous CPUs in terms of quality improvement mainly stems from variances in service demands: many short jobs can be efficiently executed on slow cores whereas long jobs will migrate to the fast core only if they are not completed by slow/medium cores.

Quality profile functions: We consider three additional quality profiles as shown in Fig. 9. The setup and staircase profiles are not concave. The linear is concave without diminishing returns.

(1) **Setup profile.** The setup profile mimics the effect of a setup cost with non-productive processing equal to 20% of the service demand. The setup phase could, for example, be used to initialize data.

(2) **Staircase profile.** The staircase profile mimics a situation in which improvement in response quality is discrete. For example, a web server can adapt its responses based on system load among (a) web pages with high-resolution images, (b) web pages with low-resolution images, and (c) text-only web pages, where different responses have corresponding qualities.

(3) **Linear profile.** The linear profile mimics applications that perform a linear scan or iterative processing of random (unsorted) data, and therefore may not exhibit the effect of diminishing returns.

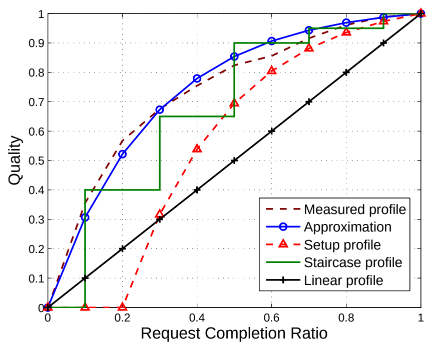
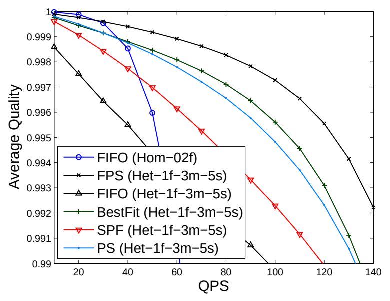
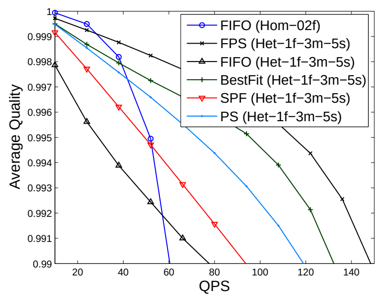


Fig. 9: Quality profiles.

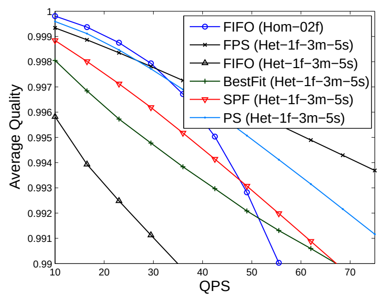
The results in Fig. 10(a), 10(b), and 10(c) show that the average quality produced by all the algorithms using these



(a) Exponential.

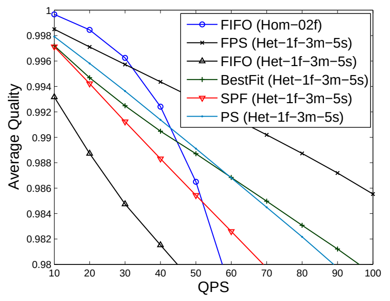


(b) Pareto.

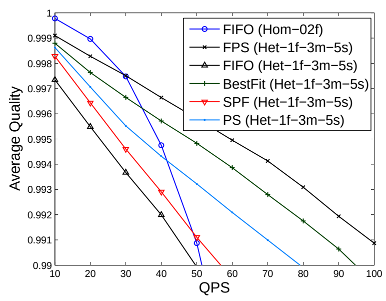


(c) Bipolar.

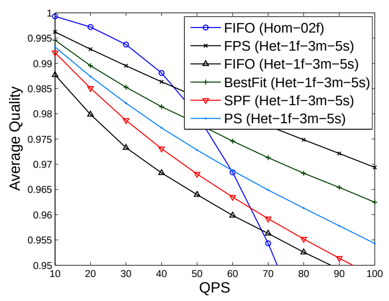
Fig. 8: Sensitivity study for different service demand distributions.



(a) Setup.



(b) Staircase.



(c) Linear.

Fig. 10: Sensitivity study for different quality profiles.

quality profiles is lower than that of using the approximated web search quality profile. The average quality is lower because for the majority of completion ratios in Fig. 9, the corresponding quality for the three profiles are lower than the measured quality profile. However, FPS consistently outperforms the other scheduling algorithms on heterogeneous CPUs and outperforms homogeneous CPUs at a moderate or high throughput for all quality profiles.

Completion time non-proportional to the inverse of core speed: In Table I, we assume that the completion time of a job on a core is in proportion to the inverse of the core speed. In practice, however, the actual completion time also depends on other factors such as processor architecture. As a consequence, the completion time may differ from the inverse of a core speed. To examine the applicability of FPS, we perform a sensitivity study of different job completion time relative to the core speed. We still normalize the fast core speed to one and use the job completion time on a fast core as a reference. On the one hand, Fig. 11(a) presents a case where a slower core is even slower: the completion of a unit work requires 4 units and 2.5 units of time on a slow core and medium core, respectively.[9] On the other hand, Fig. 11(b) presents a case where a slower core performs better than its nominal core speed: the completion of a unit work requires 2 units and 1.5 units of time on a slow core and medium core, respectively. Both results show that heterogeneous CPUs using FPS consistently outperforms homogeneous CPUs under moderate and high loads in terms of the average quality. In particular, for our quality of interest (i.e., 0.995), heterogeneous CPUs using FPS achieve a greater throughput than homogeneous CPUs, and the improvement is more significant when a slower core performs better than its nominal core speed. Therefore, FPS is applicable to applications and systems with different processor architecture/performance characteristics.

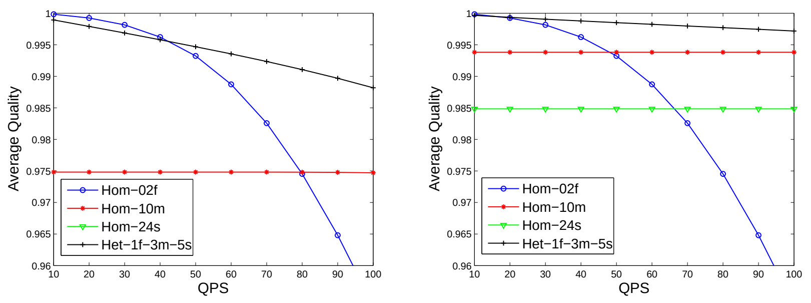
[9]The homogeneous CPU with 24 slow cores (i.e., Hom-24s) can only produce an average quality of approximately 0.9445, which is too low to be shown in Fig. 11(a).

V. Related Work

A. Heterogeneous CPUs

There are several proposals [2]–[4], [12]–[14], [23] for heterogeneous CPUs. ARM recently announced their big.LITTLE system for production [1], which combines high-performance and energy-efficient cores.

Prior work argues for the benefits of heterogeneous CPUs compared to homogeneous CPUs in two main scenarios. (1) A single job has different phases [3]–[5], such as parallel phases and sequential phases. Using a heterogeneous CPU, the sequential phase is executed on a high-performance core, and the parallel phase is executed on a number of energy-efficient cores. In contrast, an equivalent homogeneous CPU with a few fast cores has high-energy consumption during the parallel phase, while a homogenous system with many slow cores suffers a long delay during the sequential phase. Similarly, prior work [5] proposes to use high-performance but energy-inefficient fast cores to process critical phases of a job. (2) A heterogeneous CPU is more suitable for multiprogramming environments with diverse application demands [1], [12]–[14].



(a) Slow cores are slower than their nominal speed. (b) Slow cores are faster than their nominal speed.

Fig. 11: Sensitivity study on job completion time nonproportional to the inverse of core speed.

For instance, users run delay-sensitive tasks such as gaming and web surfing using fast cores, while running background services such as indexing and spell-checking using slow cores [1]. Another way to exploit the benefits in a multiprogramming environment is to assign threads with high instruction-per-cycle ratio to fast cores [14].

Unlike the existing research on heterogeneous CPUs, our work focuses on energy saving for interactive applications with adaptive execution subject to response quality and deadline constraints.

B. DVFS

DVFS trades performance for power consumption by appropriately adjusting the voltage and/or frequency, which has been studied extensively [7], [9], [10], [15]–[17]. Much prior work such as [16], [17] that investigates energy saving while meeting deadlines assumes known service demand, which, however, is not applicable in our environment. Some related work on DVFS assumes unknown service demand [7], [9], [10]. Two proposals [7], [9] progressively accelerate the processor speed during job execution to minimize the expected energy based on the service demand distribution, which is consistent with the findings of Theorem 1. However, neither of them considers scheduling multiple requests that share and compete for CPU resources. Another approach [10] minimizes the energy consumption for multiple types of periodically-arriving jobs, which are not applicable for our considered interactive applications.

DVFS and heterogeneous CPUs are complementary technology to save energy. The actual power-performance characteristics of a core depends on many factors (e.g., pipeline structure, type of transistors, etc.) in addition to its voltage and frequency. These factors limit the energy efficiency of DVFS at lower speeds and frequencies [3], [20]. In contrast, heterogeneous CPUs have an advantage: They address such deficiency by using cores with different micro-architectures to achieve better tradeoff between performance and energy [1]. On the other hand, DVFS is a relatively mature technology that does not require major changes of the existing software to exploit its benefits, while heterogeneous CPUs, as a very new technology, may require support from the OS, compiler, and libraries before it is used effectively by real-world services and applications. We leave it to future work to compare and combine the two technologies.

VI. Conclusion

This paper shows that heterogeneous CPUs boost energy efficiency of serving interactive applications. We propose an online scheduling algorithm, FPS, to improve the throughput of an interactive server subject to the quality and deadline constraints. FPS can effectively schedule long requests to fast cores and short requests to slow cores without knowing the actual service demands. We conduct extensive simulations to validate FPS, using measured workloads from Bing. Our results demonstrate significant benefits for using the FPS on heterogeneous CPUs in terms of quality improvement and energy saving.

References

[1] P. Greenhalgh, "Big.LITTLE processing with ARM Cortex.-A15 & Cortex-A7," *ARM Whitepaper*, Sep. 2011.
[2] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai, "The impact of performance asymmetry in emerging multicore architectures," *ISCA*, 2005.
[3] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-ISA heterogeneous multi-core architectures: the potential for processor power reduction," *Microarchitecture,* 2003.
[4] M. A. Suleman, Y. N. Patt, E. Sprangle, A. Rohillah, A. Ghuloum, and D. Carmean, "Asymmetric chip multiprocessors: Balancing hardware efficiency and programmer efficiency," *HPS Tech. Report,* 2007.
[5] M. A. Suleman, O. Mutlu, M. K. Qureshi, and Y. N. Patt, "Accelerating critical section execution with asymmetric multi-core architectures," *ASPLOS*, 2009.
[6] R. Raghhavendra, P. Ranganathan, V. Talwaz, Z. Wang, and X. Zhu, "No 'power' struggles: Coordinated multi-level power management for the data center," *ASPLOS*, 2008.
[7] J. R. Lorch and A. J. Smith, "Improving dynamic voltage scaling algorithms with PACE," *Sigmetrics* 2001.
[8] Y. He, S. Elnikety, and H. Sun, "Tians scheduling: Using partial processing in best-effort applications," *ICDCS*, 2001.
[9] R. Xu, C. Xi, R. Melhem, and D. Moss, "Practical PACE for embedded systems," *Embedded Software*, 2004.
[10] W. Yuan and K. Nahrstedt, "Energy-efficient CPU scheduling for multimedia applications," *ACM Trans. Computer Systems*, vol. 24, no. 3, pp. 292-331, 2006.
[11] W. Kim, M. S. Gupta, G. Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," *HPCA*, 2008.
[12] J. Chen and L. K. John, "Efficient program scheduling for heterogeneous multi-core processors," *DAC*, 2009.

[13] J. C. Saez, D. Shelepov, A. Fedorova, and M. Prieto, “Leveraging workload diversity through OS scheduling to maximize performance on single-ISA heterogeneous multicore systems,” *JPDC*, vol. 71, no. 1, pp. 114-131, 2011.

[14] M. Becchi and P. Crowley, “Dynamic thread assignment on heterogeneous multiprocessor architectures,” *ACM Computing Frontiers*, 2006.

[15] V. W. Freeh, N. Kappiah, D. K. Lowenthal, and T. Bletsch, “Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in MPI programs,” *Journal of Parallel and Distributed Computing*, vol. 68, no. 9, pp. 1175-1185, Sep. 2008.

[16] F. F. Yao, A. J. Demers, and S. J. Shenker, “A Scheduling Model for Reduced CPU Energy,” *FOCS*, 1995.

[17] S. Albers, F. Muller and S. Schmelzer, “Speed scaling on parallel processors,” *SPAA*, 2007.

[18] M. Harchol-Balter, “Task assignment with unknown duration,” *J. of ACM*, vol. 49, no. 2, pp. 260-288, 2002.

[19] M. Harchol-Balter, “The Effect of Heavy-Tailed Job Size Distributions on Computer System Design,” *Applications of Heavy Tailed Distributions in Economics*, 1999

[20] M. Bi, I. Crk, and C. Gniady, “IADVS: On-demand performance for interactive applications,” *HPCA*, 2010.

[21] D. von Seggern, *CRC Standard Curves and Surfaces*, 1993.

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

[23] M. D. Hill and M. R. Marty, “Amdahl’s Law in the Multicore Era”, *Computer*, vol. 41, pp. 33-38, 2004.

[24] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. “Dynamo: amazon’s highly available key-value store”, *SOSP*, 2007.

[25] J. Hamilton. Blog article: Perspectives. 2009.

[26] “Intel Core i7-975 Processor Extreme Edition Specification”, http://ark.intel.com/products/37153.

[27] “Intel Atom processor N455 Specification”, http://ark.intel.com/products/49491.

[28] “List of CPU power dissipation”, http://en.wikipedia.org/wiki/List_of_CPU_power_dissipation.

[29] V. J. Reddi, B. C. Lee, T. M. Chilimbi, K. Vaid, “Web search using mobile cores: quantifying and mitigating the price of efficiency”, *ISCA*, 2010.

[30] C. Huang, P. A. Chou, and A. Klemets. “Optimal control of multiple bit rates for streaming media”, *PCS*, 2004.

[31] C. Li, C. Ding, and K. Shen, “Quantifying the cost of context switch,” *ECS*, 2007.