Exploiting Processor Heterogeneity for Interactive Services

Shaolei Ren[‡*] Yuxiong He[†] Sameh Elnikety[†] Kathryn S. McKinley[†]
[‡]*Florida International University* [†]*Microsoft Research*

Abstract

To add processing power under power constraints, emerging heterogeneous processors include fast and slow cores on the same chip. This paper demonstrates that this heterogeneity is well suited to interactive data center workloads (e.g., web search, online gaming, and financial trading) by observing and exploiting two workload properties. (1) These workloads may trade response quality for responsiveness. (2) The request service demand is unknown and varies widely with both short and long requests. Subject to per-server power constraints, traditional homogeneous processors either include a few high-power fast cores that deliver high quality responses or many low-power slow cores that deliver high throughput, but not both.

This paper shows heterogeneous processors deliver both high quality and throughput by executing short requests on slow cores and long requests on fast cores with Fast Old and First (FOF), a new scheduling algorithm. FOF schedules new requests *with unknown service demands* on the fastest idle core and migrates requests from slower to faster cores. We simulate and implement FOF. In simulations modeling Microsoft's Bing index search, FOF on heterogeneous processors improves response quality and increases throughput by up to 50% compared to homogeneous processors. We confirm simulation improvements with an implementation of an interactive finance server using Simultaneous Multithreading (SMT), configured as a dynamic heterogeneous processor. Both simulation and experimental results indicate processor heterogeneity offers a lot of potential for interactive workloads.

1 Introduction

A heterogeneous processor contains cores with differentiated power and performance characteristics. All cores execute the same instruction set (ISA), but they run at different speeds and/or use different microarchitectures so that the faster the core, the more power it consumes. Since power consumption increases faster than speed, a *fast* core executes a request in less time than a *slow* core, but consumes more energy. We show that a homogeneous processor under a fixed power constraint can only deliver either high performance with a few fast cores or high throughput with many more slow cores for interactive data center workloads, whereas heterogeneous processors deliver both with substantial benefits.

Interactive services require high quality results and timely responses to satisfy users and generate revenue [23]. For example, Bing search servers are provisioned to execute requests within 120 ms with an average quality of 0.99. This *quality* metric compares returned search results within a time limit to an off-line search with unlimited time and resources. Interactive services have two properties. First, many interactive services — including web search, financial trading, and online gaming — are *adaptive*, i.e., processing a request for more time improves response quality. Adaptive execution may return lower-quality responses (or partial results) for responsiveness. Second, the service demand of a request is unknown a priori, and it varies widely with both short and long requests.

The main contribution of this paper is to demonstrate that exploiting processor heterogeneity delivers substantial benefits to interactive workloads in data centers as compared to using homogeneous processors. More concretely, when building a data center to support interactive services, power budgets constrain the overall system as well as individual servers. Such design-time power constraints limit the core speed and number of cores on a chip. With a fixed design-time power budget, system designers can deploy a homogeneous processor with either fewer *fast* high-performance power-hungry cores that are less energy-efficient or a larger number of *slow* cores that are more energy-efficient. For example, one fast core may burn as much power as 8 to 16 low power cores [38]. Fast cores offer high service quality and fast response at a light load, but when the load increases, both quality and throughput degrade quickly since requests significantly outnumber cores. In contrast, more slow energy-efficient

*This work was partially done while Shaolei Ren was visiting Microsoft Research.

cores increase throughput by executing more requests, but the quality of responses drops when the slow cores do not execute fast enough to fully process long requests before their respective deadlines. We show how to achieve both high quality and throughput with a heterogeneous design provisioned with both fast and slow cores. The key idea is to execute short requests on slow cores, so that they complete within their deadline with low energy consumption, and to execute long requests on fast cores to obtain high response quality. Towards this end, there are two challenges. (1) When a request arrives, we do not know its service demand, and predicting service demand is difficult for many workloads [34]. (2) Even if we know the service demand, there are multiple requests but only a limited number of cores, and therefore the most appropriate core on which to execute a request may not be available.

This paper address these challenges by introducing a new online algorithm for scheduling requests of interactive services on heterogeneous processors, called Fast Old and First (FOF). When a new request arrives, FOF assigns it the fastest available core (*Fast First*). When a request completes, FOF promotes the oldest request on a slower core to the fastest, newly available core (*Fast Old*). FOF achieves high throughput because it completes many short requests on energy-efficient slow cores. FOF achieves high response quality since requests are processed on fast cores whenever possible and long requests execute with a higher probability on fast cores and thus all requests are likely to complete before their deadlines.

We model Microsoft Bing, a commercial web search engine. We measure Bing workload distributions and quality profiles. We find that the request service demand has high variance, and response quality is monotonically increasing in processing time (before the deadline). Our simulation results of web search show that FOF on heterogeneous processors achieves a significantly higher response quality and up to 50% improvement in throughput compared to homogeneous processors with the same design-time power budget as well as compared to traditional scheduling algorithms. We also show that FOF improves throughput and quality for a variety of heterogeneous hardware configurations and workloads with various service demand distributions. Moreover, FOF improves average quality, reduces quality variance, improves high-percentile quality, and improves throughput.

We further show how to configure a Simultaneous Multithreading (SMT) core as a dynamic heterogeneous processor and modify FOF for it. A core executing one thread acts as a fast core, while a core executing $M > 1$ SMT hardware threads acts as M slow cores. Even with the limited heterogeneity of a 2-way 6 core SMT processor, FOF improves the *measured performance* of an interactive finance server by up to 16% compared to default round-robin OS scheduling and by 27% when SMT is turned off. We validate our simulator against these measurements. FOF in implementation performs the same or better than in simulation. These results indicate that FOF offers performance benefits for heterogeneity present in data centers today.

We show how to compute the number of slow and fast cores in the hardware configurations using the request service demand distribution. In general, more long requests require more fast cores, and more slow cores are required for sustaining a higher throughput. Future data centers can exploit these results either to substantially reduce the number of servers, or to increase the capacity without compromising quality or responsiveness.

2 Scheduling Model

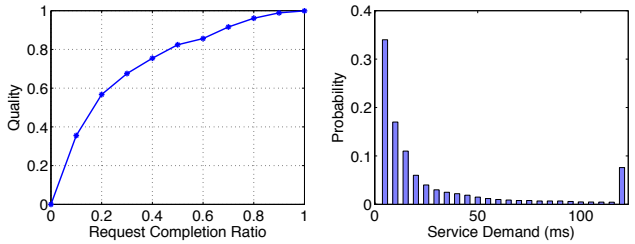
This section measures request characteristics in interactive services for a commercial web search engine, and formalizes our job and hardware model.

The literature establishes the following characteristics of interactive services [48, 35, 38, 26]. **(1) Adaptive execution.** Adaptive execution partially evaluates a request and returns a response before completion; more computation yields better quality. **(2) Concave quality profile.** If a request executes fully, it receives quality 1 (measured off-line). A concave function exhibits diminishing returns and captures the relationship between quality and computational resources (see Section 2.1). **(3) Deadline.** For online interactive services, users expect timely responses. Long response times are not acceptable because they cause user dissatisfaction and loss of revenue [23]. We express timing constraints as deadlines.

Recommendation systems, ad services, and video streaming have similar characteristics. For example, in scalable video coding, basic layers are more important than refinement layers, and the quality of received video streams improves monotonically with the number of layers but exhibits diminishing returns [28]. This paper focuses on web search and finance, but our results apply more broadly.

2.1 Measurement Study of Bing Search

We confirm these characteristics with measurements of Bing. Bing's web index serving system receives user queries and returns the response. This system is a distributed interactive search service. The web index contains billions of documents and thus, the index is partitioned and managed across thousands of servers. To meet responsiveness goals, Bing and other modern search engines [48, 35, 38] design and configure search such that the index for a server fits in main memory of the server with virtually no I/O or system calls, creating a CPU bound process. Other interactive services, such as video streaming, have similar goals and requirements. When a request arrives, the system assigns it to an aggregator, which sends the request to servers. Each server returns its matched results to the aggregator. The aggregator collects them and returns the top L webpages to the user.



(a) Quality profile. (b) Service demand distribution.

Figure 1: Measured workload of Bing search.

Search servers support adaptive execution with response deadlines. Each web search query contains a set of keywords. A server scans an inverted index looking for webpages that match the keywords and ranks the matching webpages. The response is links to the top documents that match the keywords. The more time the server spends in matching and ranking the documents, the better quality (i.e., more relevant) the search results. If the search server does not finish searching the entire index within the deadline, it returns the best matches so far. The server responds to the aggregator within 120 ms. The aggregator returns its collected results to users without waiting for any delayed responses. Ranking involves complex calculations and search servers are computationally intensive [38].

Quality Profile We obtained 200K queries from a production trace of Bing queries. We execute them multiple times with different completion deadlines using Bing in a controlled setting. Response quality compares the documents returned to the *golden* results (Quality = 1), without any deadline, such that Bing fully processes each request. The x-axis of Figure 1(a) is the *request time completion ratio* which is calculated as the actual measured processing time divided by its full processing time. The y-axis is the average response quality. Figure 1(a) shows that Web search has a monotonically increasing and (roughly) concave response quality profile.The concavity is because the inverted index searches popular webpages first, which are more likely to rank higher and contribute more to the response quality.

Service Demand Figure 1(b) presents the measured service demand distribution for 200K queries. Request service demand varies with many short requests, less than 40 ms, and under the 120 ms deadline, over 10% have a demand greater than or equal to 100 ms. This diversity in request service demands has been observed in many workloads [10, 48, 24, 25].

2.2 Job Model

An individual request is processed *sequentially*, while multiple requests can be processed on different cores *concurrently*. A job (request) is specified by a tuple (t_a, d, w), where t_a is arrival time, d is lifespan, and w is service demand. The job deadline is $t_a + d$. We assume d is the same for all jobs without loss of generality. We denote the maximum service demand by $\hat{w}$. The service demand is the total work (i.e., CPU cycles) required to complete a request and is unknown until the job completes. However, the service demand distribution is available by using online or offline profiling [34, 26, 25]. 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 can alternatively use discrete values of service demands (e.g., measured values shown in Figure 1(b)) and all the analysis still applies, where PDF $f(w)$ is the probability mass function and w takes values from a finite set.

The server can fully process a request, returning a complete result, or terminate with a partial result. We measure the actual *quality* $q(r) : \mathbb{R}^+ \to \mathbb{R}$ off-line, comparing processed work to demand. While each request may have a unique quality profile that is unavailable to an online scheduler, we use $q(r)$, as shown in Figure 1(a), to represent expected quality profile of a request.

2.3 Hardware Model

Limited by power constraints, architects are turning to parallelism and heterogeneity in search of power-efficient performance [3, 31, 36, 8, 33]. A heterogeneous processor consists of $N > 1$ heterogeneous cores, indexed by $1, 2, \cdots, N$, which offer non-uniform performance and power consumption due to processing speeds or microarchitecture or both. Without loss of generality, we assume that the i-th core performance (i.e., effective speed) s_i and power p_i satisfy $0 < s_1 \le s_2 \le \cdots \le s_N$ and $0 < p_1 \le p_2 \le \cdots \le p_N$ [34]. Moreover, we assume that a core with higher performance speed burns more power to process a unit of work, i.e., forall $1 \le i \le j \le N$, $p_i/s_i \le p_j/s_j$, 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 space. To fairly compare heterogeneous to homogeneous processors, we give them all the same design-time power budget.

3 Scheduling Algorithm

The scheduling objective of FOF is to improve the average response quality of all requests and thereby increase throughput. In practice, data center designers may exploit throughput improvements either by generating and servicing more load per server or by supporting the same load with fewer servers.

3.1 Key Insights

Intuitively, we want to schedule long requests on fast cores (since only fast cores are sufficient to ensure timely and

high-quality responses for long requests) and schedule short requests on slow cores (since they are sufficient to ensure timely and high-quality responses). An ideal scheduler will thus maximize throughput and quality by executing every request on the slowest core that can meet the request deadline and quality requirement. However, there are two challenges. (1) *Assignment*: since the request service demand is unknown, how do we assign short requests to slow cores and long requests to fast cores? (2) *Availability*: given multiple requests and a limited number of cores, the most appropriate core may not always be available.

(1) Assignment FOF *migrates requests from slow to fast cores during its execution*. This policy increases the probability that short requests complete on slow cores and when a fast core becomes available, it processes longer requests. Given a deadline, this policy improves total response quality of requests, sustaining higher throughput while satisfying the target quality.

Theorem 1 explains why "slow to fast" improves throughput. The theorem formally establishes that migrating a request from slower to faster cores during its execution is the most energy-efficient schedule. Given the server's design-time power budget, dynamic energy per time unit is bounded. Thus, when each individual request consumes less energy, the server can serve more requests, improving throughput. Theorem 1 assumes *the desired core(s) are always available*, and later we address multiple requests competing for cores.

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

Proof. We first meet the quality requirement. Request quality is a function of the quality profile q and work completed before the deadline. Let us define the target work $\bar{x}$, which specifies the maximum amount of work completed prior to the deadline regardless of the actual service demand. 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 expected average 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 energy-minimization problem for scheduling a request as follows:

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

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

where $\mathscr{X}$ is a schedule that specifies the order and cores that process the single request, $s_{\mathscr{X}}(x) \in \{s_1, s_2 \cdots s_N\}$, and $p_{\mathscr{X}}(x) = p_i$, if $s_{\mathscr{X}}(x) = s_i$. Constraint (2) guarantees that the schedule $\mathscr{X}$ satisfies the deadline. We now prove the theorem by contradiction. Suppose that $s_{\mathscr{X}'}(x)$ minimizes (1) while, under the schedule $\mathscr{X}'$, 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 $s_{\mathscr{X}'}(x_1') > s_{\mathscr{X}'}(x_2')$, 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 we assume faster cores consume more energy to process one unit of work than slower ones, the following inequality holds:

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

Thus, we have $[1 - F(x_1')] \cdot \frac{p_{\mathscr{X}'}(x_1')}{s_{\mathscr{X}'}(x_1')} + [1 - F(x_2')] \cdot \frac{p_{\mathscr{X}'}(x_2')}{s_{\mathscr{X}'}(x_2')} > [1 - F(x_1')] \cdot \frac{p_{\mathscr{X}'}(x_2')}{s_{\mathscr{X}'}(x_2')} + [1 - F(x_2')] \cdot \frac{p_{\mathscr{X}'}(x_1')}{s_{\mathscr{X}'}(x_1')}$. By evaluating the integral in (1), the expected energy consumption is further reduced by exchanging 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 $\mathscr{X}'$ unchanged. This contradicts the assumption that $\mathscr{X}'$ minimizes (1) and hence, proves Theorem 1. ■

The most obvious application of this theorem is optimizing dynamic run-time energy of service requests. However, we leave dynamic energy to future work and focus on design-time, because interactive service providers must *first* determine whether or not a heterogeneous processor can improve throughput and quality given design-time power constraints, and if it can, what combinations of fast and slow processors to use. We next leverage the insight of migrating a request from slower to faster cores to improve quality and throughput.

(2) Availability Given multiple requests and a limited number of cores, the most appropriate core to execute a request may not be available. FOF solves this problem by assigning the most *urgent* request to the fastest core, which is also the request that has already been executed for the longest time. Intuitively, the longer the system executes the job, the higher probability that the job requires faster cores to complete prior to the deadline. More formally, we define urgency as follows.

Definition 1: Request *urgency* is defined as the expected minimum core speed to complete the request prior to its deadline. Mathematically, we express urgency as follows:

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

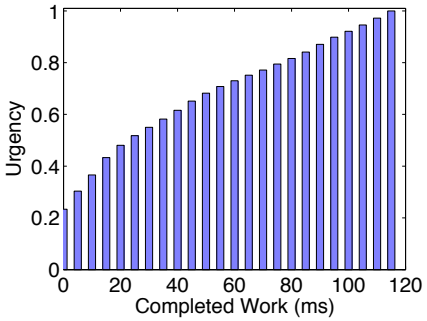


Figure 2: Urgency versus completed work. With more processing, request urgency increases: an older request has higher urgency.

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

Urgency indicates the (expected) necessary core speed to complete a job upon its deadline. By assigning faster cores to jobs with higher urgency, jobs have a greater chance to complete prior to their deadlines. Figure 2 shows a lower bound on the urgency value for the Bing service demand distribution (Figure 1(b)), where x-axis is the amount of work that a job has completed and y-axis is urgency. During actual processing, request urgency is impacted by its waiting time in the queue and its execution history, making the urgency in Figure 2 a lower bound.

A key observation is that as the request is processed, its urgency increases. When a request is processed more, its available time before the deadline decreases whereas its expected service demand increases. The general urgency trend is similar for other widely used service demand distributions such as exponential and Pareto. This observation motivates FOF to use faster cores to run the oldest request because that request has high urgency.

3.2 FOF Algorithm

Algorithm 1 shows the pseudo-code of FOF. When a job enters the system, FOF assigns it to the fastest available core. When a job completes or early terminates at its deadline, a core is idle and FOF promotes the oldest job on a slower core to this faster core. No job migrates between cores that have the same speed. Consider the following cases.

All cores are idle FOF assigns the job to the fastest idle core N.

Only the fastest core is busy FOF assigns the new job to idle core $N-1$, the second fastest core in the system.

All cores are busy: When an existing job completes or its deadline expires, a core becomes available. FOF promotes the oldest (longest running) job on any *slower* core to this faster core. It repeats this process until the slowest core becomes idle. At this point, FOF schedules the job at the head of the wait queue on the slowest core.

Algorithm 1 FOF

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: older jobs have higher urgency.
2:  i ← N
3:  while i ≥ 1 do
4:      if core i is idle then
5:          job J = job being processed on a slower core than core
            i (or waiting in the queue) with the highest urgency
6:          if job J is not null then
7:              schedule job J to core i
8:          end if
9:      end if
10:     i − −
11: end while
12: return
```

The FOF algorithm has the following key properties:

- A faster core always runs a request with higher or equal urgency than all jobs on slower cores, increasing response quality and the probability of completing all requests before their deadlines.
- When there are $1 \leq k < N$ requests where N is the number of cores, the fastest k cores process these k requests, increasing response quality.
- If a request migrates, it always migrates from a slower to a faster core. This choice increases the probability that short jobs will complete on a slow core and that long jobs will execute on fast cores.

Note that FOF is designed to improve quality and throughput on a heterogeneous processor, rather than attain the lowest possible dynamic run-time energy. For example, with only one request in the system, FOF will execute it on the fastest core, whereas executing it on a slower core first will consume less dynamic energy (as shown in Theorem 1). However, by improving throughput while satisfying response quality, the data center can buy and use fewer servers and consume less total energy. Thus, server provisioning and consolidation is the means by which FOF optimizes server and energy cost.

FOF is computationally efficient. It does not require a priori information on each request's *actual* service demand. It also bounds job migrations to $K-1$ times in the worst case, where K is the number of different core speeds, regardless of the server load. In practice, migrations per job are much less than $K-1$, as many short requests are processed and completed on one core.

4 Simulation Study

This section evaluates FOF with a simulation study using Bing web search measurements and various workload distributions. We show heterogeneous processors with FOF improve over homogeneous processors in terms of average quality, quality variance, 95%-quality, and number of servers required to support the workload. Furthermore,

Name	Processor	(C Cores T SMT)	technology	LLC	Speed	1 Core Power	Normalized Performance	Normalized Power
A	i7-2600 Sandy Bridge	(4C 2T)	32 nm	4 M	3.4 GHz	21 W	1.0	1.0
B	i5-670 Nehalem	(2C 2T)	32 nm	8 M	3.4 GHz	16 W	0.92	0.81
C	i7-920 Nehalem	(4C 2T)	45 nm	8 M	2.7 GHz	15 W	0.72	0.73
D	AtomD Bonnell	(2C 2T)	45 nm	1 M	1.7 GHz	4 W	0.45	0.19

Table 1: Core specification, measured and normalized PARSEC performance and power.

FOF achieves a higher quality than various alternative scheduling algorithms, including a clairvoyant scheduler. We explore how to select a good heterogeneous processor configuration based on workload characteristics. We observe that the quality improvement obtained from using FOF on heterogeneous processors translates directly into a throughput increase, thereby reducing the required number of servers. We also explore sensitivity to hardware choice and workload, showing FOF improves throughput at high quality in many scenarios. Section 5 shows how to configure processors with Simultaneous Multithreading (SMT) to mimic heterogeneous processors with FOF and attain benefits in today's data center servers.

4.1 Methodology

As heterogeneous servers are not yet available, we perform a simulation study using DESMO-J, a Java-based discrete-event simulator. The simulator models cores, scheduling, jobs, migration, completion, and other events. Although simulation cannot capture every detail of system implementation, it demonstrates the first-order impact of using FOF on heterogeneous processors. We however validate these simulation results using a finance server executing on an SMT multicore processor.

Workload We use the Bing index search measurements from Section 2.1. We model a server that accepts users' search queries and returns the top L webpages as a CPU intensive process [38]. Our simulator considers a request delay deadline of 120 ms. We model service demand distribution using measurements from production servers and shown in Figure 1(b). We model request arrivals as a Poisson process. To change system load, we control the mean query arrival rate as queries per second (QPS). We use the measured quality profile of Bing web search as shown in Figure 1(a). All these characteristics match other search engines in the literature [48, 35, 38].

Hardware performance and power We approximate heterogeneous core performance and power based on measurements of existing Intel processors. We use the performance and power data reported by Esmaeilzadeh et al. [19] executing PARSEC [7] on four architectures. We use PARSEC because Reddi et al. show that they exhibit similar performance characteristics to interactive services [38]. Table 1 presents core speed, and normalized single core performance and power for four architectures.

Name	A	B	C	D	E	Power
Hom-4A	4	0	0	0	0	82 W
Hom-5B	0	5	0	0	0	81 W
Hom-6C	0	0	6	0	0	88 W
Hom-22D	0	0	0	22	0	82 W
Het-3B-9D	0	3	0	9	0	82 W

Table 2: Processor configurations with design-time power budget.

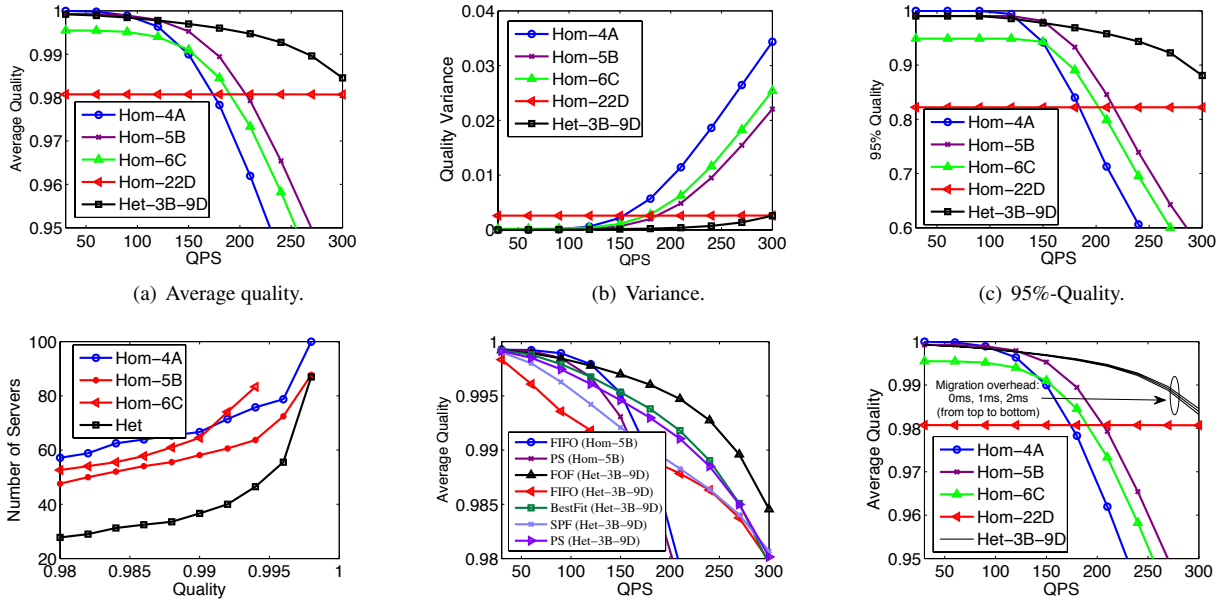
We model homogeneous and heterogeneous processors with a design-time power budget between 80 and 88 W shown in Table 2. Homogeneous configurations include as many individual cores as possible. We simulate a heterogeneous processor composed of three i5 Nehalem cores (3B) and nine AtomD cores (9D), called Het-3B-9D. Section 4.5 explores other heterogeneous configurations.

4.2 Heterogeneous versus Homogeneous

This section shows the benefit of FOF by comparing our default heterogeneous processor (Het-3B-9D) using FOF with the four homogeneous processors from Table 2 using FIFO scheduling in terms of average quality, quality variance, 95%-quality, and number of servers to support a given workload subject to the quality requirement. The widely-used FIFO scheduler places all jobs in a single queue and an idle core pulls the job from the head of the queue and processes it until completion or expiration of the deadline. FOF and FIFO are equivalent on a homogeneous processor.

Improved average quality Figure 3(a) plots average response quality (y-axis) against load measured in QPS (x-axis). A heterogeneous processor with FOF outperforms all homogeneous configurations in terms of average response quality on a wide range of loads, translating into a higher throughput subject to a fixed quality requirement. We focus on throughput at the target quality of 0.99.

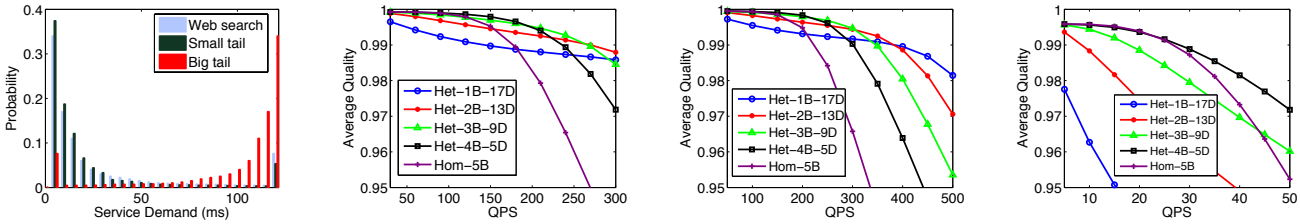
FOF increases throughput significantly, by 50%, on Het-3B-9D compared to Hom-5B, the best 0.99-throughput homogeneous processor. The core utilization at quality 0.99 is as follows: Hom-4A at 150 queries per second (QPS) is 91%; Hom-5B at 180 QPS is 92%; Hom-6C at 155 QPS is 81%; and Het-3B-9D at 270 QPS is 99%. Hom-22D only supports an average quality of approximately 0.9808. At



(a) Average quality. (b) Variance. (c) 95%-Quality.

(d) Number of servers. (e) Different scheduling algorithms. (f) Different migration overheads.

Figure 3: Figures (a), (b), (c) and (d) compare heterogeneous to homogeneous processors under different performance metrics. Figure (e) compares different heterogeneous scheduling algorithms. Figure (f) shows the impact of migration overheads.



(a) Service demand. (b) Measured distribution. (c) Small tail. (d) Big tail.

Figure 4: Heterogeneous core configurations for a range of service demand distributions.

higher load (not shown), quality drops off further. Figure 3(a) shows that both Hom-4A and Hom-5B, which are fast cores, produce high quality when the throughput is low (e.g., 90 QPS), whereas neither Hom-6C nor Hom-22D are fast enough to achieve sufficiently high quality.

With a fixed power budget, a heterogeneous design satisfies stringent quality requirements (e.g., 0.99) by combining the high processing capabilities of fast cores and the high throughput of multiple low-power slower cores. Key to this result is that one fast core consumes more power than 3 or more slow cores but a fast core has only about 2 times of the processing speed of a slow core.

Reduced quality variance Figure 3(b) shows response quality variance. The heterogeneous processor using FOF has the lowest variance. With Hom-22D, there are enough cores to serve (almost) every incoming job without delay and hence, the quality variance remains relatively constant throughout until the QPS exceeds its capabilities. Nonetheless, long jobs cannot complete prior to their respective deadlines, resulting in a quality variance among short and long jobs. Hom-4A, Hom-5B and Hom-6C have little quality variance with light load (QPS < 150), since they complete almost every job. When load increases, queuing time increases and some long requests get insufficient service before their deadline, which reduces quality and increases variance. When using a Het-3B-9D, a long request that cannot get a fast core immediately can still be processed on one of many slow and medium cores and migrate later to the fast core, which improves quality and reduces variance even at high load.

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 response quality [17]. Remember we compute quality off-line, comparing to a search with no limit on time or resources. For example, a web search engine may target a quality of 0.99 for average quality and 0.90 for at least 95% requests. The high-percentile quality depends on the response quality distribution. Figure 3(c) shows that a heterogeneous

processor improves the 95%-quality over homogeneous processors on moderate and heavy loads by improving average quality and reducing variance.

Reducing number of servers To highlight the hardware and energy reductions due to heterogeneous processors with FOF scheduling, we consider a total workload of 10,000 QPS and compute how many servers are needed with a given design-time power budget, subject to various average quality requirements. Figure 3(d) shows the number of servers, for four systems, Het-3B-9D, Hom-4A, Hom-5B and Hom-6C. For average quality of 0.99, a heterogeneous processor reduces the number of servers by approximately 35% compared with homogeneous processors with the same power budget, and may significantly reduce costs and energy usage in large data centers.

4.3 Comparing Scheduling Algorithms

This section compares FOF with four alternative scheduling algorithms: FIFO, Processor Sharing (PS), Slow Preempt Fast (SPF), and BestFit. FIFO, PS, SPF, and FOF are non-clairvoyant because when jobs arrive, their service demand is unknown in practice. BestFit is a clairvoyant scheduler; it knows each request's service demand but lacks migration. Figure 3(e), shows that even without knowledge of the request service demand, FOF outperforms BestFit and all the other schedulers because its migration policy takes advantage of core heterogeneity.

FIFO and EDF Since requests have the same maximum delay, FIFO and Earliest Deadline First (EDF), an optimal real-time scheduler, behave the same. FOF achieves a significantly higher quality than FIFO which cannot support a 0.99-throughput higher than 160 QPS. FOF outperforms FIFO because FOF completes many short requests on slower cores and migrates long requests to faster cores. In 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.

Processor sharing (PS) Processor sharing is the well-known round-robin scheduler for homogeneous processors. We extend PS to heterogeneous processors. When the number of jobs M is greater than the number of cores N, the jobs equally share all the available cores. When M is smaller than N, the jobs equally share the M fastest cores in a round-robin fashion. We assume that the context switch and migration overhead of PS is 0 and therefore the PS quality result is an upper bound. Figure 3(e) shows that FOF achieves a higher quality than PS because the FOF migration policy gives long requests a higher chance to use fast cores. PS shares fast cores equally among short and long requests and hence, long requests that really need fast cores may not get them.

These classic scheduling algorithms for homogeneous processors are insufficient because they do not consider how to match request service demands to heterogeneous core characteristics. Moreover, since the request service demand is unknown, it is not possible to determine the most appropriate core for a request before its execution. During execution, however, the scheduler progressively has more information about requests (i.e., urgency) such that the scheduler may refine its decision. Thus, using FOF, *job migration among cores refines and improves scheduling decisions when request service demand is unknown*.

Migration policies We consider the SPF scheduler which migrates jobs in reverse order to FOF, 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 Figure 3(e) that FOF achieves a much higher quality than SPF. By migrating jobs from slower to faster cores, FOF is more likely to complete short jobs on slower cores, saving faster cores to process long jobs. In contrast, SPF completes short jobs on 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 that the job migration order from slower to faster cores is critical to exploit processor heterogeneity.*

Clairvoyant without migration Even with known service demands, scheduling multiple jobs on a heterogeneous processor is a challenging task. BestFit 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. Combining job migration with clairvoyance requires solving an integer programming problem that we leave for future work. This algorithm is similar to scheduling algorithms in prior work [14, 40, 44], which map jobs to the most "appropriate" cores.

Figure 3(e) shows that, rather surprisingly, FOF without knowing request service demand outperforms BestFit with known service demand. Because BestFit does not consider job migration, it cannot fully exploit heterogeneous cores. For example, if a long job arrives and the best core to run the job is a fast core but all fast cores are running another job. BestFit will let the job wait for the fast core to finish even when there are other cores available. FOF is better because it uses the slow cores to run the job first and then migrates it to the fast core when it becomes available. FOF also outperforms the Shortest Job First (SJF) algorithm (a widely-used algorithm for improving the response time in

homogeneous processors) in terms of response quality. We omit these results due to space limitations. *Even when the request service demand is known, migration is critical to exploiting heterogeneous core resources as they become available. In particular, long requests make progress on slow cores first before migrating when fast cores become available.*

4.4 Migration Overhead

This subsection shows FOF is not sensitive to migration overheads, since actual migration overheads are less than 1%. Even modeling higher migration overheads does not diminish FOF's performance benefits.

Job migration requires a context switch, whose time is proportional to the size of a job's working set due to cache warm-up time, which may take tens of microseconds to a millisecond [18]. We model three migration overhead values: 0 ms (prior section), 1 ms, and 2 ms. Figure 3(f) shows migration overheads hardly have any effect on quality, which is mainly due to the following two factors. (1) Migration overhead is typically at the range of tens of microseconds to a couple of milliseconds and thus is much smaller compared to the deadline of a few hundred milliseconds. (2) The number of migrations is small. Given K different core speeds available, a job may migrate up to $K-1$ times in the worst case, and K is a small number in general. Moreover, as many short jobs complete on slow cores, they do not need to migrate at all. Het-3B-9D has two core types and thus, the upper bound on the number of job migrations is 1 while the average number of migrations per job is 0.45 at 150 QPS and increases to 0.56 at 300 QPS.

4.5 Heterogeneous Core Configuration

This subsection explores how to select good heterogeneous processor configurations based on workload characteristics. We find that more long jobs need more fast cores for a given quality target but some small cores are always required.

We consider three representative workloads as illustrated in Figure 4(a): (1) *measured* distribution of web search; (2) *small tail* reduces the probability of long jobs (>30 ms) and increases the probability of short jobs (≤ 30 ms) in the measured distribution; and (3) *big tail* reverses the measured distribution (most jobs are long). Figure 4 shows their average quality. We explored more configurations, but for brevity, only show combinations of B and D cores. We compare against a homogeneous processor with B cores, since D cores cannot deliver the desired quality target.

Under all three service demand distributions, a heterogeneous processor outperforms a homogeneous one (Hom-5B) on 0.99-throughput. In particular, Het-1B-17D is the best heterogeneous processor with a small tail, whereas Het-4B-5D is the best under a big tail. Figure 4 confirms our intuition that the best core configurations depend on the service demand distribution and that the more long running jobs the system expects, the more fast cores it should include, because the slow cores cannot produce sufficiently high quality. However, there is always a point where one or more fast cores should be traded for slow cores to gracefully degrade quality and improve throughput.

4.6 Other Workloads

We performed sensitivity studies on synthetic workloads (exponential, Pareto, and bipolar service demand distributions), quality profiles (linear profile, discrete staircase profile), and different deadlines. These results consistently show the benefits of using heterogeneous processors with FOF to meet the desired quality with high throughput. We omit the details of the results due to space constraint.

5 Exploiting Heterogeneity in SMT Systems

This section describes: (a) How to configure an Simultaneous Multithreaded (SMT) multicore processor to act as a heterogeneous processor. (b) How to modify FOF for it. (c) An implementation of a finance server for SMT that delivers improvements in throughput and quality. The experimental results on a 2-way SMT 4 core machine show that FOF-SMT improves throughput by up to 16% compared to the default OS scheduler and 27% compared to no SMT. (d) A comparison of implementation and simulation results confirms the accuracy of our simulator. SMT, as a form of heterogeneity, is already present in data centers, and thus FOF can have an immediate impact.

5.1 SMT as a Heterogeneous Multicore

SMT adds heterogeneity to existing hardware and we use it to mimic heterogeneous processors. Intuitively, a core executing one thread acts as a fast core, and a core executing $N>1$ SMT hardware threads acts as N slow cores. The N slow cores exhibit higher throughput but each thread runs slower than the fast core, similar to heterogeneous processors.

5.2 FOF for SMT

We modify FOF to create FOF-SMT. We enable SMT for all cores on a processor and then FOF-SMT controls which cores execute as fast cores by executing only one thread or as slow cores by executing N threads on an N-way SMT. FOF-SMT works as follows.

Fast first When a request joins, if there is an idle core, FOF-SMT schedules the request on it.

Fast old Consider two cases. (1) When a request joins, if there are available SMT hardware threads but no idle cores, FOF-SMT schedules the current request on an SMT thread such that it shares the same core as the

youngest executing request. (2) At the point a request completes, its core may become idle. In this case, if other cores are sharing, FOF-SMT finds the sharing core with the oldest job. Rather than moving the oldest of the jobs sharing a core, it moves the other job. This other job is less urgent, which minimizes the impact on the oldest job and with 2-way SMT, gives both jobs a core to themselves, i.e., fast cores. Both cases leave the old request running on a fast core as they are likely more urgent.

The implementation uses thread pools, affinity, and request queues on each core prioritized by age to schedule jobs in constant time. To study FOF in an implementation and validate our simulator, we use an interactive finance server.

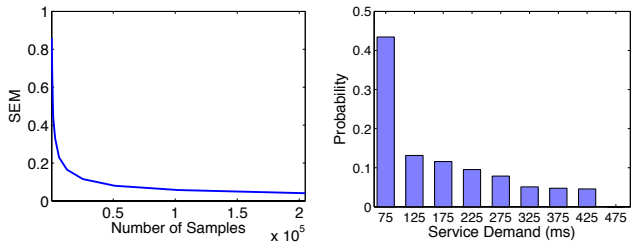
5.3 Option Pricing Finance Server

Finance servers are another example of interactive online services. Banks and fund management companies evaluate thousands of financial derivatives every day, submitting requests that value derivatives and then making immediate trading decisions. Many of these calculations use Monte Carlo methods, which are computationally intensive and rely on repeated random sampling to compute results. We implement an option pricing server that uses Monte Carlo methods for complex path-dependent Asian options [9, 16].

Each request is a Monte Carlo task that estimates an option price under various economic scenarios with different interest rates, strike prices, dividend yields, and volatility. The tasks are time bounded, since traders use them to perform online trading. With more samples, the processing time is longer and the estimated price gets closer to the real price. The system is adaptive and supports returning partial results. The result quality is measured by a well-known statistical metric called standard error of mean (SEM). SEM is the standard deviation of the sample mean of the population mean. SEM is computed by sampling the standard deviation s divided by the square root of the sample size n, i.e., SEM $= s/\sqrt{n}$. The smaller the SEM is, the closer the estimated price to the real price. The goal is to minimize the average and high-percentile SEM value for all requests so the estimated prices are closer to the real prices.

Figure 5(a) shows an error profile with increasing sample sizes. When the number of samples increases along with the processing time, SEM decreases, which indicates increasing quality and the error profile is convex. When the sample size is large, the change in sample standard deviation is small and the square root of the sample size dominates. The convexity of $1/\sqrt{n}$ leads to the convexity of SEM. Minimizing SEM with a convex error profile is equivalent to maximize quality with a concave quality profile.

Figure 5(b) shows the service demand distribution. Each request incurs different processing time to compute a sample, therefore service demand varies. Similar to web search (Figure 1(a)), this workload is non-uniform with a mix of mostly short and some long requests. However, the finance server does not have as heavy of a tail.



(a) Error profile. (b) Service demand distribution.

Figure 5: Measured quality of finance server workloads.

5.4 Methodology

We use a server with 6-core 2-way SMT 3.33 GHz Intel Xeon X5680 processor with 24 GB of memory running Windows Server 2012. The requests follow a Poisson arrival process. When a request's SEM reaches the target of 0.05 or lower, we consider it fully evaluated and terminate the request. Otherwise, the request is partially evaluated and terminated when it reaches a 500 ms deadline. We compare three finance server configurations:

NoSMT SMT disabled.

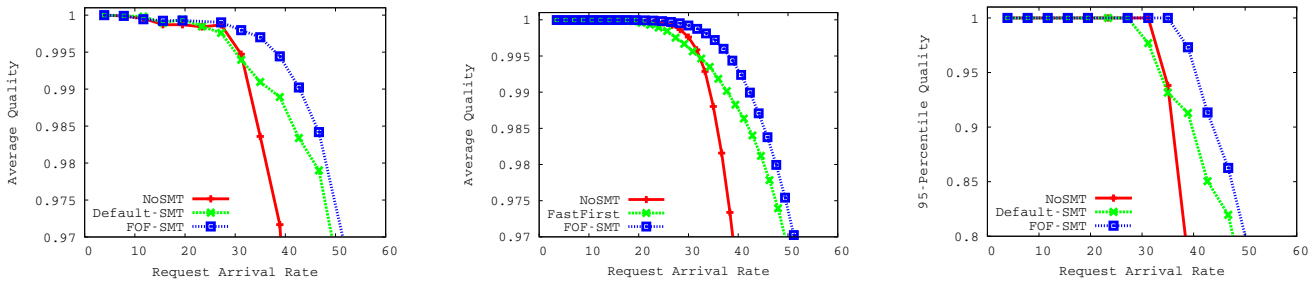
Default-SMT SMT enabled with default round-robin OS scheduling. After all cores are occupied with a single request, it shares with SMT, choosing the core in round-robin fashion [39].

FOF-SMT SMT enabled with FOF-SMT scheduling.

5.5 Implementation Results

We compare FOF with NoSMT and Default-SMT with respect to average and high-percentile quality. Our results show that FOF improves response quality of requests and improves throughput by 27% over NoSMT and 16% over SMT using default round-robin OS scheduling.

Figure 6(a) presents the average quality of requests with varying load. The x-axis is load, expressed as request arrival rate in requests per second (RPS), and the y-axis is the average quality of all requests, where request quality is computed by how far the result is from the target accuracy. The request quality is $1-(SEM_m - SEM_t)/SEM_t$ where SEM_m is the measured SEM value and SEM_t is the target SEM value. The results show that, at light load, all of NoSMT, FOF-SMT and Default-SMT achieve high quality because no core needs to share. With increasing load, FOF-SMT outperforms both NoSMT and Default-SMT with improved quality. For example, at quality target 0.99, NoSMT sustains a throughput of 33 RPS, Default-SMT 37 RPS, and FOF-SMT improves it to 42 RPS, achieving a 27% improvement over NoSMT and 16% over Default-SMT. FOF-SMT reduces quality variance (not shown) and



(a) Implementation: average quality. (b) Simulation: average quality. (c) Implementation: 95-% quality.

Figure 6: Implementation results with 2-way SMT dynamic heterogeneity on 6 cores for a finance server. Implementation matches simulation results. FOF-SMT delivers higher average and 95-percentile quality at higher load.

Figure 6(c) shows that FOF-SMT improves high-percentile quality.

These improvements come from two sources: (1) capacity due to adding hardware parallelism in the form of SMT, and (2) better scheduling choices by FOF. To put capacity improvements in context, we measure a request X alone on a core with processing time T_X and two identical requests on two SMT threads sharing the same core concurrently, which takes time $1.582T_X$. In theory, SMT could improve performance by $2T_X$, but in practice SMT delivers less because it shares hardware resources (issue queues, caches, etc.). In other words, given a core with speed 1, each 2-way SMT hardware context has speed 0.632. 2-way SMT thus provides a $0.632 \times 2 - 1 = 26.4\%$ increase of computational capacity. Thus the capacity improvements of FOF-SMT are close to optimal for this workload. Adding a larger number of slow cores to replace one or more fast cores on heterogeneous processors increases throughput. FOF-SMT makes better scheduling decisions as witnessed by the gap between Default-SMT and FOF-SMT. Smart scheduling algorithms are necessary to fully exploit the benefits of SMT. FOF-SMT outperforms Default-SMT because FOF-SMT shares cores among new requests, which are both likely short and complete on shared (slow) cores, leaving long requests to run on unshared (fast) cores, where they are more likely to complete before the deadline.

5.6 Validating Simulation with Implementation

We validate the simulation results with the finance server implementation. The simulator uses measured service demand, error profile, raw performance on one core (1), and the relative performance of SMT (0.632 of one core) using the 6-core 2-way SMT processor measurements. Comparing Figure 6(b) with the simulation results to the implementation results in Figure 6(a) for average quality shows that the SMT performance reported by the simulator is very close to the SMT implementation results, both with respect to the trends and absolute performance. This result increases our confidence in the accuracy of our simulator and the results in Section 4.

6 Related Work

Heterogeneous processors There are several proposals for heterogeneous processors [27, 4, 31, 43, 5, 40, 14]. ARM recently announced their big.LITTLE processor for production [22], which combines high-performance and energy-efficient cores. Recent work argues for the benefits of heterogeneous processors compared to homogeneous processors in two main scenarios.

(1) A single job has different phases [43, 31, 42], such as parallel phases and sequential phases. This work is grounded in applying Amdahl's law to a parallel program to accelerate its sequential bottleneck [2, 27, 46]. Using a heterogeneous processor, the sequential phase is executed on a high-performance core, and the parallel phase is executed on a number of energy-efficient cores. Our work considers a stream of independent jobs that execute in parallel instead of a single parallel program to which Amdahl's law was applied. We show more than one fast core is often necessary and we furthermore show how to use workload characteristics to choose the mix and variety of fast and slow cores.

(2) A heterogeneous processor is more suitable for multiprogramming environments with diverse application demands [22, 5, 40, 14, 30, 33, 44]. Suleman et al. use high-performance but energy-inefficient fast cores to process critical phases of a job [42]. Others try to match program phases to the appropriate core such that the part of the program that benefits most from the high power core executes on it [5, 40, 44]. They either improve performance or save dynamic energy while being performance neutral. Users run delay-sensitive tasks such as gaming and web surfing using fast cores, while background services such as indexing and spell-checking use slow cores [22]. Lakshminarayana et al. schedule the thread in a parallel job with a larger remaining execution time on a fast core [32]. They predict the remaining time based on thread creation or dynamic profiling. FOF achieves a much more substantial throughput improvement because it leverages the diversity (short versus long jobs) and adaptivity in the workload

demand, since it terminiates a job early if it exceeds it deadline. A long running job cannot monopolize the fast core indefinitely. Moreover, instead of using information about each job, our work exploits the service demand distribution of all jobs.

Real-time scheduling Prior work on real-time scheduling that investigates saving energy while meeting deadlines [47, 1] assumes known service demands, which is not applicable in our environment. Other related work assumes unknown service demand [34, 45, 49] and uses Dynamic voltage/frequency scaling (DVFS) to save energy. None of the prior work considers scheduling multiple requests that both support partial execution as well as share and compete for CPU resources. Trading resource consumption for service quality has also been explored in other contexts (e.g., wireless networks) using stochastic control techniques, but only average queue length or average response time is addressed [37]. Embedded and real-time systems did not explore partial execution, and hence their algorithms are not applicable to interactive services we study. Our objective is to improve total quality with deadline constraints, rather than meeting the deadline of each job.

Next, we discuss scheduling algorithms for SMT and DVFS systems.

SMT In a multiprogrammed environment, prior work on SMT scheduling improves fairness and throughput by coscheduling jobs according to their performance characteristics and interference [41, 20, 12] . They all consider workloads without deadlines. SMT scheduling on real-time and soft real-time systems considers deadlines, but it focuses on periodic tasks such as multimedia applications and partitions resources to meet deadlines [29, 11]. In contrast, we use SMT to emulate a heterogeneous processor and develop an SMT-aware scheduler for interactive workloads, where jobs arrive aperiodically and can be partially evaluated.

DVFS DVFS trades performance for power consumption by adjusting voltage or frequency [47, 1, 21, 34, 45, 49, 13]. Instead of optimizing for dynamic energy, our scheduling improves response quality and therefore supports higher throughput per server under design-time power constraints. Two proposals [34, 45] 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, those studies do not address multiple jobs that compete for resources. Another approach [49] minimizes the energy consumption for multiple types of periodically-arriving jobs, which is not applicable for interactive applications. Similarly, others [15] propose a dynamic voltage scaling algorithm for multimedia applications based on the service demand information provided by content providers, but such information is not available for our applications. Other related work exploits partial execution (referred as differential service level) and proposes an algorithm based on Markov decision process to maximize total response quality given a mean response time constraint [13]. Their algorithm is applicable to server systems with different speeds. They consider mean response time of jobs as constraint but our jobs need to meet response deadlines.

DVFS and heterogeneous processors are complementary technologies. The actual power-performance characteristics of a core depends factors such as pipeline structure, type of transistors, degree of speculation, voltage and frequency. These factors limit the energy efficiency of DVFS at lower speeds and frequencies [6, 31]. In contrast, heterogeneous processors address such inefficiencies by using cores with different microarchitectures to achieve a better tradeoff between performance and energy [22].

While DVFS and SMT are relatively mature technologies that do not require major changes to existing software to exploit their benefits, heterogeneous processors are an emerging technology that will require additional support from the OS, compiler, and libraries before it is practical to use by real-world services and applications.

7 Conclusion

We propose an online scheduling algorithm, FOF, to improve the quality and throughput of an interactive service on a heterogeneous processor. FOF effectively schedules long requests to fast cores and short requests to slow cores without knowing the actual service demands. Extensive simulations evaluate FOF based on workloads from Bing search. The results show that using FOF on heterogeneous processors improves throughput by up to 50% compared to using homogeneous processors. We also show how to use an SMT processor as a dynamic heterogeneous processor. We implement the scheduling algorithm for a financial server. Our experimental results show up to 16% higher throughput by using FOF on an SMT processor compared to a default round-robin OS scheduler. A comparison of the implementation results to our simulator configured with similar features show that they closely match. These results point to significant opportunities for heterogeneous processors in data centers.

8 Acknowledgments

We thank Gregg McKnight and James Larus from Microsoft for the insightful discussions and feedback.

References

[1] S. Albers, F. Muller, and S. Schmelzer. Speed scaling on parallel processors. In *SPAA*, 2007.

[2] G. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. pages 483–485, 1967.

[3] ARM Corporation. big.LITTLE processing, 2011.

[4] S. Balakrishnan, M. Upton, and K. Lai. The impact of performance asymmetry in emerging multicore architectures. In *ISCA*, 2005.

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

[6] M. Bi, I. Crk, and C. Gniady. Iadvs: On-demand performance for interactive applications. In *HPCA*, 2010.

[7] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. Technical Report TR-811-08, Princeton University, January 2008.

[8] S. Borkar and A. Chien. The future of microprocessors. *Communications of the ACM*, 54(5):67–77, 2011.

[9] M. Broadie and P. Glasserman. Estimating security price derivatives using simulation. *Manage. Sci.*, 42:269–285, February 1996.

[10] B. Cahoon and K. S. McKinley. Performance evaluation of a distributed architecture for information retrieval. In *SIGIR*, pages 110–118, Geneva, Switzerland, Aug. 1996.

[11] F. J. Cazorla, P. M. W. Knijnenburg, R. Sakellariou, E. Fernández, A. Ramirez, and M. Valero. Architectural support for real-time task scheduling in smt processors. In *CASES*, 2005.

[12] F. J. Cazorla, A. Ramirez, M. Valero, P. M. W. Knijnenburg, R. Sakellariou, and E. Fernández. Qos for high-performance smt processors in embedded systems. In *Micro*, 2004.

[13] S. Chaitanya, B. Urgaonkar, and A. Sivasubramaniam. QDSL: QoS-aware systems with differential service levels. *ACM SIGMETRICS*, 2008.

[14] J. Chen and L. K. John. Efficient program scheduling for heterogeneous multi-core processors. In *DAC*, 2009.

[15] E. Y. Chung, L. Benini, and G. D. Micheli. Contents provider assisted dynamic voltage scaling for low energy multimedia applications. In *IEEE Symposium on Low Power Electronics and Design*, ISPLED'02, 2002.

[16] G. Cortazar, M. Gravet, and J. Urzua. The valuation of multidimensional american real options using the LSM simulation method. *Computers and Operations Research.*, 2006.

[17] 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. In *SOSP*, 2007.

[18] C. L. C. Ding and K. Shen. Quantifying the cost of context switch. In *ECS*, 2007.

[19] H. Esmaeilzadeh, T. Cao, Y. Xi, S. M. Blackburn, and K. S. McKinley. Looking back on the language and hardware revolutions: Measured power, performance, and scaling. In *ASPLOS*, pages 319–332, 2011.

[20] S. Eyerman and L. Eeckhout. Probabilistic job symbiosis modeling for SMT processor scheduling. In *ASPLOS '10*, 2010.

[21] 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*, 68(9):1175–1185, Sep. 2008.

[22] P. Greenhalgh. Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7. *ARM Whitepaper*, 2011.

[23] J. Hamilton. Blog article: Perspectives, 2009.

[24] M. Harchol-Balter. The effect of heavy-tailed job size distributions on computer system design. In *Applications of Heavy Tailed Distributions in Economics*, 1999.

[25] M. Harchol-Balter. Task assignment with unknown duration. *J. of ACM*, 49(2):260–288, 2002.

[26] Y. He, S. Elnikety, and H. Sun. Tians scheduling: Using partial processing in best-effort applications. In *ICDCS*, 20011.

[27] M. D. Hill and M. R. Marty. Amdahl's law in the multicore era. *Computer*, 41:33–38, 2008.

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

[29] R. Jain, C. J. Hughes, and S. V. Adve. Soft real-time scheduling on simultaneous multithreaded processors. In *RTSS*, 2002.

[30] D. Koufaty, D. Reddy, and S. Hahn. Bias scheduling in heterogeneous multi-core architectures. In *EuroSys*, 2010.

[31] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-ISA heterogeneous multicore architectures: The potential for processor power reduction. In *MICRO*, 2003.

[32] N. B. Lakshminarayana, J. Lee, and H. Kim. Age based scheduling for asymmetric multiprocessors. In *ACM/IEEE Conference on High Performance Computing (SC)*, 2009.

[33] T. Li, P. Brett, R. C. Knauerhase, D. A. Koufaty, D. Reddy, and S. Hahn. Operating system support for overlapping-ISA heterogeneous multi-core architectures. In *HPCA*, pages 1–12, 2010.

[34] J. R. Lorch and A. J. Smit. Improving dynamic voltage scaling algorithms with pace. In *SIGMETRICS*, 2001.

[35] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. Power management of online data-intensive services. In *ACM/IEEE International Symposium on Computer Architecture*, ISCA '11, pages 319–330, 2011.

[36] T. Y. Morad, U. C. Weiser, A. Kolodnyt, M. Valero, and E. Ayguadé. Performance, power efficiency and scalability of asymmetric cluster chip multiprocessors. *Computer Architecture Letters*, 5(1):14–17, 2006.

[37] M. J. Neely. *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan & Claypool, 2010.

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

[39] M. E. Russinovich, D. A. Solomon, and A. Lonescu. *Microsoft Windows Internals, Fifth Edition: Covering Windows Server 2008, Windows Vista*. Microsoft Press, 2009.

[40] 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. *Journal of Parallel and Distributed Computing*, 71(1):114–131, 2011.

[41] A. Snavely and D. M. Tullsen. Symbiotic job scheduling for a simultaneous multithreaded processor. In *ASPLOS*, 2000.

[42] M. A. Suleman, O. Mutlu, M. K. Qureshi, and Y. N. Patt. Accelerating critical section execution with asymmetric multi-core architectures. In *ASPLOS*, 2009.

[43] M. A. Suleman, Y. N. Patt, E. Sprangle, A. Rohillah, A. Ghuloum, and D. Carmean. Asymmetric chip multiprocessors: Balancing hardware efficiency and programmer efficiency. Technical report, HPS, 2007.

[44] K. Van Craeynest, A. Jalelle, L. Eeckhout, P. Narvaez, and J. Emer. Scheduling heterogeneous multi-cores through performance impact estimation (PIE). In *ISCA '12*, 2012.

[45] R. Xu, C. Xi, R. Melhem, and D. Moss. Practical pace for embedded systems. In *Embedded Software*, 2004.

[46] E. Yao, Y. Bao, G. Tan, and M. Chen. Extending Amdahl's law in the multicore era. *ACM SIGMETRICS Performance Evaluation Review*, 37(2):24–26, Oct. 2009.

[47] F. F. Yao, A. J. Demers, and S. J. Shenker. A scheduling model for reduced CPU energy. In *FOCS*, 1995.

[48] J. Yi, F. Maghoul, and J. Pedersen. Deciphering mobile search patterns: a study of yahoo! mobile search queries. In *ACM International Conference on World Wide Web*, WWW '08, pages 257–266, 2008.

[49] W. Yuan and K. Nahrstedt. Energy-efficient CPU scheduling for multimedia applications. *ACM Trans. Computer Systems*, 24(3):292–331, 2006.