Market Mechanisms for Managing Datacenters with Heterogeneous Microarchitectures

MARISABEL GUEVARA, Duke University
BENJAMIN LUBIN, Boston University
BENJAMIN C. LEE, Duke University

Specialization of datacenter resources brings performance and energy improvements in response to the growing scale and diversity of cloud applications. Yet heterogeneous hardware adds complexity and volatility to latency-sensitive applications. A resource allocation mechanism that leverages architectural principles can overcome both of these obstacles.

We integrate research in heterogeneous architectures with recent advances in multi-agent systems. Embedding architectural insight into proxies that bid on behalf of applications, a market effectively allocates hardware to applications with diverse preferences and valuations. Exploring a space of heterogeneous datacenter configurations, which mix server-class Xeon and mobile-class Atom processors, we find an optimal heterogeneous balance that improves both welfare and energy-efficiency. We further design and evaluate twelve design points along the Xeon-to-Atom spectrum, and find that a mix of three processor architectures achieves a 12× reduction in response time violations relative to equal-power homogeneous systems.

Categories and Subject Descriptors: C.0 [**Computer Systems Organization**]: General—*Systems architectures*

General Terms: Economics, Management, Performance

Additional Key Words and Phrases: Energy-efficient datacenters, resource allocation, economic mechanisms

ACM Reference Format:
Marisabel Guevara, Benjamin Lubin, and Benjamin C. Lee. 2014. Market mechanisms for managing datacenters with heterogeneous microarchitectures. ACM Trans. Comput. Syst. 32, 1, Article 3 (February 2014), 31 pages.
DOI: http://dx.doi.org/10.1145/2541258

1. INTRODUCTION

As datacenters proliferate and access to them is democratized, increasingly diverse cloud applications will demand computation. To accommodate the rise in demand, traditional datacenter servers have relied on Moore's Law. Yet this strategy is insufficient as Dennard scaling ends and constrains the power efficiency of processor servers [Horowitz et al. 2005].

This article extends "Navigating heterogeneous processors with market mechanisms" in *Proceedings of the 19th International Symposium on High Performance Computer Architecture* [Guevara et al. 2013].
This work was supported, in part, by STARnet, a Semiconductor Research Corporation program, sponsored by MARCO and DARPA. Further support was provided by the National Science Foundation, under CAREER CCF-1149252. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the view of these sponsors.
Authors' addresses: M. Guevara, Computer Science Department, Duke University; email: mg@cs.duke.edu; B. Lubin, Information Systems Department, Boston University; email: blubin@bu.edu; B. C. Lee, Electrical and Computer Engineering Department, Duke University; email: benjamin.c.lee@duke.edu.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2014 ACM 0734-2071/2014/02-ART3 $15.00
DOI: http://dx.doi.org/10.1145/2541258

Instead of relying on process technology for datacenter efficiency, we turn to new system architectures and microarchitectures. Recent research and industry trends highlight opportunities for building servers with lightweight processors that were originally designed for mobile and embedded platforms [Andersen et al. 2009; Lim et al. 2008; Moor Insights and Strategy 2013; Seamicro 2011]. These small cores are several times more energy-efficient than high performance processors.

However, lightweight cores have limited applicability. While memory- or IO-intensive applications benefit from small core efficiency, the era of big data is introducing more sophisticated computation into datacenters. Tasks may launch complex analytical or machine learning algorithms with strict targets for service quality [Janapa Reddi et al. 2010]. To guarantee service, high-performance cores must continue to play a role. To this end, heterogeneous datacenter servers can balance big core performance and small core efficiency.

Not only must we design heterogeneous hardware, we must deploy it in large, dynamic systems. Doing so successfully requires mitigating performance risk and uncertainty as diverse applications contend for heterogeneous hardware. Additionally, datacenters must shield increasingly non-expert users from the complexity of underlying heterogeneity.

To address these challenges, we coordinate the design of heterogeneous architectures with recent advances in multi-agent systems. We present a market where diverse applications bid for heterogeneous architectures. On behalf of users, a proxy profiles hardware-software interactions, infers preferences for heterogeneous hardware, and translates preferences into bids.

Both early research [Sutherland 1968] and more recent markets [Lubin et al. 2009; Chase et al. 2001] assume fungible processor cycles, an assumption that no longer holds given processor heterogeneity. Ours is the first to incorporate microarchitectural preferences of the applications into an economic mechanism for hardware allocation. In particular, we make the following contributions.

—*Processor Heterogeneity in the Datacenter* (Section 2). We identify a new design space where heterogeneous processor architectures allow a datacenter to combine the benefits of specialization with the performance guarantees of traditional high-performance servers.
—*Economic Mechanisms and Optimization* (Section 3). We develop a market that manages resources and navigates performance-efficiency tradeoffs due to microarchitectural heterogeneity. Inferring application preferences for hardware, proxies compose bids on behalf of applications within the market. A mixed integer program allocates resources to maximize welfare, which is user value net datacenter cost.
—*Application to Big/Small Cores* (Section 4). We apply the economic mechanism to explore a space of heterogeneous datacenters, varying the mix of server- and mobile-class processors. We find an optimal heterogeneous balance that improves user value and reduces energy. Moreover, 30% of tasks incur response time violations in homogeneous systems but not in heterogeneous ones.
—*Application to Further Heterogeneity* (Section 5). We further explore the microarchitectural design space and tailor processor cores to application mixes. With processors that differ in pipeline depth, superscalar width, and in-order versus out-of-order execution, we find that a combination of three processor architectures can reduce response time violations by $12\times$ relative to a homogeneous system.
—*Analytical Model for Quality-of-Service* (Section 6). We evaluate our choice of queueing models, describing the interactions between waiting and service times. The market optimizes for waiting time, which in turn improves system utilization.

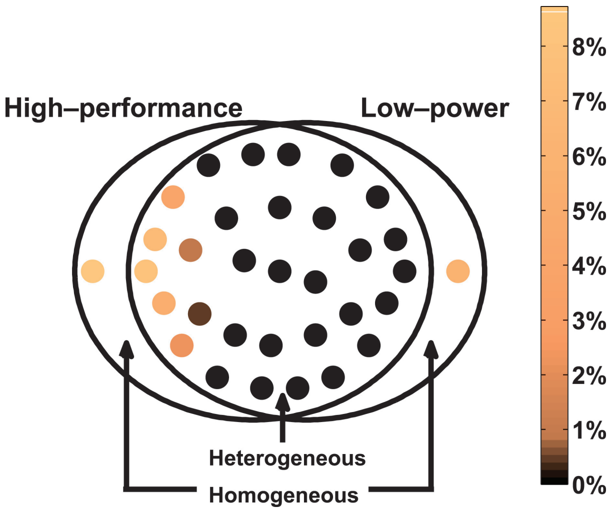


Fig. 1. Venn diagram that illustrates a datacenter design space for low-power and high-performance processors; the intersection harbors heterogeneous design options. Colored points depict QoS violations.

Thus, we present a management framework that allows datacenters to exploit the efficiency of heterogeneous processors while mitigating its performance risk.

2. HETEROGENEITY – PRINCIPLES AND STRATEGIES

The largest datacenters today are equipped with high-performance processors. Despite diversity due to process technology or generations, these cores all reside at the high-performance end of the design spectrum. Thus, we refer to the processors in state-of-the-art datacenters as homogeneous by design. While such homogeneity can provide near-uniform performance, it also keeps datacenters from exploiting recent advances in energy-efficient hardware. For example, small processor cores are far more power efficient than conventional, high-performance ones. Since only certain tasks are amenable to small core execution, big cores must also remain as guarantors of service quality.

2.1. Heterogeneity as a Design Space

Server heterogeneity is efficient but requires sophisticated resource managers to balance performance risk and reward. This balance requires a novel type of design space exploration to survey and appraise a variety of datacenter configurations. To illustrate the challenge, Figure 1 depicts the design space for two core types: a high-performance, server-class core and its low-power, mobile-class counterpart. Combinations of these two processor types fall into three regions shown in the Venn diagram. Two regions represent homogeneous configurations, where the datacenter is comprised of only server or mobile cores. Heterogeneous mixes lie in the third region, the intersection of the sets.

The colorbar shows the percentage of allocation intervals that suffered a quality-of-service degradation for a pair of task streams; this data is collected through simulation with parameters found in Section 4. For the workloads in this experiment, the two homogeneous configurations violate quality-of-service agreements at least 6% of the time.[1] As some high-performance, power-hungry nodes are replaced by a larger

[1]These are equal power datacenters, and there are more than five times more mobile than server processors in the homogeneous configurations.

number of low-power processors, datacenter heterogeneity improves quality-of-service and reduces the frequency of violations to $<1\%$.

Indeed, ensuring service quality poses the greatest challenge to heterogeneity in datacenters. Several design questions arise when we consider how to populate a datacenter with diverse processor types. First, what are the right core types for a given set of applications? In this article, we tradeoff efficiency and performance by considering two existing processors: the mobile-class Atom and the server-class Xeon (Section 4). Additionally, we design and evaluate up to twelve cores that lie along the efficiency-vs-performance spectrum (Section 5).

Second, how many of each processor type do we provision in the datacenter? Using microarchitectural and datacenter simulation, we evaluate performance and energy consumption for mixes of Xeons and Atoms, and mixes of the twelve cores.

Third and equally important is the resource management of heterogeneous components. How do we allocate heterogeneous processing resources to diverse applications? It turns out that we cannot answer the first two questions without first designing a solution to the third. A policy for matching applications to processing resources is vital to ensuring quality-of-service guarantees for datacenter applications.

Our effort to differentiate preferences for heterogeneous cycles is driven by a desire to exploit low-power cores when possible. Small cores are efficient but exact a task-specific performance penalty. Thus, we encounter a tension between design and management in heterogeneous systems. When designing for efficiency, we would prefer to tailor processor mix to task mix. Each task would run only on the processor that is most efficient for its computation, but datacenter dynamics preclude such extreme heterogeneity and its brittle performance guarantees. In contrast, when managing for performance, we would favor today's homogeneous systems and suffer their inefficiencies.

We strike a balance by moderating heterogeneity and increasing manager sophistication. Using the market as a management mechanism, we explore types and ratios of heterogeneous processors as a coordinated study of this novel design space. Balancing allocative efficiency loss against computational speed, our approach approximates complex heterogeneous hardware allocations by simpler, canonical ones. Doing this well requires microarchitectural insight that properly captures software preferences for hardware. With such insight, the market can quickly tradeoff performance and efficiency across heterogeneous processors.

2.2. Accommodating Architectural Heterogeneity

Up to $5\times$ more efficient than big ones, small processor cores are increasingly popular for datacenter computation [Janapa Reddi et al. 2010]. Small cores are well balanced for the modest computational intensity of simple web search queries, distributed memory caching, and key-value stores [Andersen et al. 2009; Janapa Reddi et al. 2010; Ousterhout et al. 2010]. Such research in unconventional datacenter hardware has spurred broader commercial interest [Anonymous 2012; Courtland 2012] and analogous research in other technologies, such as DRAM [Malladi et al. 2012; Yoon et al. 2012].

Performance variations across processor types are well-studied in architecture, yet such detail is abstracted away in markets for systems. Since Sutherland's market for a shared PDP-1 [Sutherland 1968], allocators have considered simple, exchangeable slots of computer or network time. This limited model of the architecture has persisted despite large strides in computational economies during the past two decades, most notably by Waldspurger et al. [1992], by Chase et al. [2001], and Lubin et al. [2009]. Simply counting cycles is insufficient when the value of each hardware cycle depends on software-specific preferences.

The heterogeneity required for the largest efficiency gains demands sophisticated architectural insight. For heterogeneous processors, performance differences depend on computer architecture's classical equation:

$$\frac{\texttt{Tasks}}{\texttt{Sec}} = \frac{\texttt{Cycles}}{\texttt{Sec}} \times \frac{\texttt{Insts}}{\texttt{Cycle}} \times \frac{\texttt{Tasks}}{\texttt{Inst}} \tag{1}$$

To scale $\frac{\texttt{Cycles}}{\texttt{Sec}}$, we must consider software compute-memory ratios and sensitivity to processor frequency. To scale $\frac{\texttt{Insts}}{\texttt{Cycle}}$, we must consider software instruction-level parallelism and its exploitation by hardware datapaths. And, if code is tuned or recompiled, we must also scale $\frac{\texttt{Tasks}}{\texttt{Inst}}$.

Heterogeneous Processors and Hard Constraints. Some processors may be incapable of providing the desired service. By obtaining application performance characteristics, a resource manager can account for machine restrictions. For example, the manager might determine the suitability of small cores based on memory, network, or I/O activity. The market uses profiling information to determine if an application derives no value from certain processors. These hard restrictions are enforced by constraints when we clear the market by solving a mixed integer program.

Heterogeneous Cycles and Soft Constraints. Suppose a processor is suited to execute a task. Then service rate and queueing delay are determined by core microarchitecture. For compute-bound workloads, a cycle on a superscalar, out-of-order core is worth more than one from an in-order core. How much more depends on the task's instruction-level parallelism. Memory-bound tasks are indifferent to heterogeneous cycles.

To account for cycles that are not fungible, we introduce scaling factors that translate task performance on heterogeneous cores into its performance on a canonical one. Applications constrained by memory or I/O will not necessarily benefit from the additional compute resources of a big, out-of-order core. On the other hand, a big core might commit $3\times$ more instructions per cycle than a small core for applications with high instruction-level parallelism.

We differentiate cycles from each core type with a vector of scaling factors, $\kappa = (\kappa_{big}, \kappa_{small})$, that accounts for the application-specific performance variation of the two core types. For example, an agent sets $\kappa = (1, \frac{1}{3})$ for the application with high ILP, and $\kappa = (1, 1)$ for the memory-intensive job.

To calculate scaling factors, we rely on application profiling data. In this paper, we assume that existing profilers provide this data (see Section 8 for a survey of related work). Although more advances are needed, existing profilers are sophisticated and allow us to focus on the allocation mechanism.

3. THE MARKET MECHANISM

To ensure quality-of-service, we introduce a novel market in which proxies, acting on behalf of applications, possess microarchitectural insight. Heterogeneous system design allows us to tailor resources to task mixes for efficiency. Yet specialization increases performance risk and demands sophisticated resource allocation. In this work, we balance efficiency and risk by identifying datacenter designs that provide robust performance guarantees within the market framework.

We present a market for heterogeneous processors that builds on two prior efforts. Chase et al. [2001] manage homogeneous servers by asking users to bid on performance. Lubin et al. [2009] extend this formulation with processor frequency scaling, a novel modeling and bidding language, and a mixed integer program to clear the market. We start from the latter market, which assumes fungible processor cycles, and extend it to account for architectural heterogeneity.

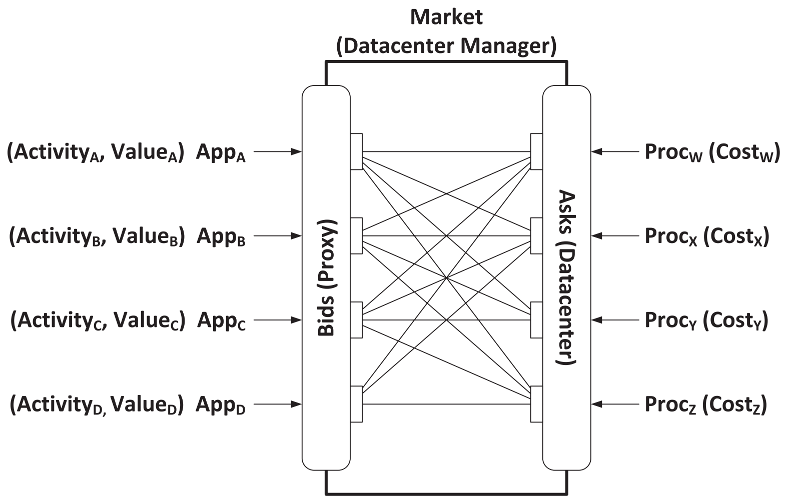

Market
(Datacenter Manager)
(Activity$_A$, Value$_A$) App$_A$
(Activity$_B$, Value$_B$) App$_B$
(Activity$_C$, Value$_C$) App$_C$
(Activity$_D$, Value$_D$) App$_D$
Bids (Proxy)
Asks (Datacenter)
Proc$_W$ (Cost$_W$)
Proc$_X$ (Cost$_X$)
Proc$_Y$ (Cost$_Y$)
Proc$_Z$ (Cost$_Z$)

Fig. 2. Market overview.

The Market and User Value. Figure 2 illustrates the market mechanism which performs three operations: (i) hardware performance is evaluated to calculate bids for each user application (buyer proxy), (ii) hardware efficiency is used to calculate costs (seller proxy), (iii) a welfare maximizing allocation is found (mixed integer program).

This approach has several advantages in our setting with nonfungible cycles. First, proxies are made to account for performance variation across heterogeneous cycles based on instruction-level parallelism in the datapath. Second, proxies will bid for complex, heterogeneous combinations of cores, while hiding the complexity of the heterogeneous hardware from users who are ill-equipped to reason about it. Lastly, an optimizer maximizes welfare according to the submitted bids when clearing the market and allocating resources.

Proxies require users who can express value for computation. This valuation includes a response time target and a payment if the target is met. From this information, the proxy defines a basic service-level agreement. Proxies translate the agreement into bids for hardware, thereby assuming much of the burden for navigating heterogeneous processors. Yet, in our market formulation, proxies cannot operate beyond parameters set by the user.

We assume users express their true value for performance. Users have incentives to deviate from their true value in two settings. In the first, the datacenter is underutilized. Since the datacenter has spare capacity, a user might report lower value for performance yet receive her desired allocation. Taken to an extreme, a single user in the datacenter could receive her desired hardware allocation by valuing performance at the cost incurred by the datacenter to provide it.

Alternatively, suppose the datacenter is over-subscribed. In a datacenter with too many users, some users may be denied service. Value for performance implicitly measures priority. Users that report lower value may starve as the market allocates hardware to maximize welfare. Starving users must increase their reported value, wait for periods of lower demand, or realize that the datacenter cannot provide the desired service.

To act strategically in these two settings, users would require information on datacenter utilization and outcomes from prior market allocations. In our framework, proxies are best positioned to collect and communicate this information to users. Doing

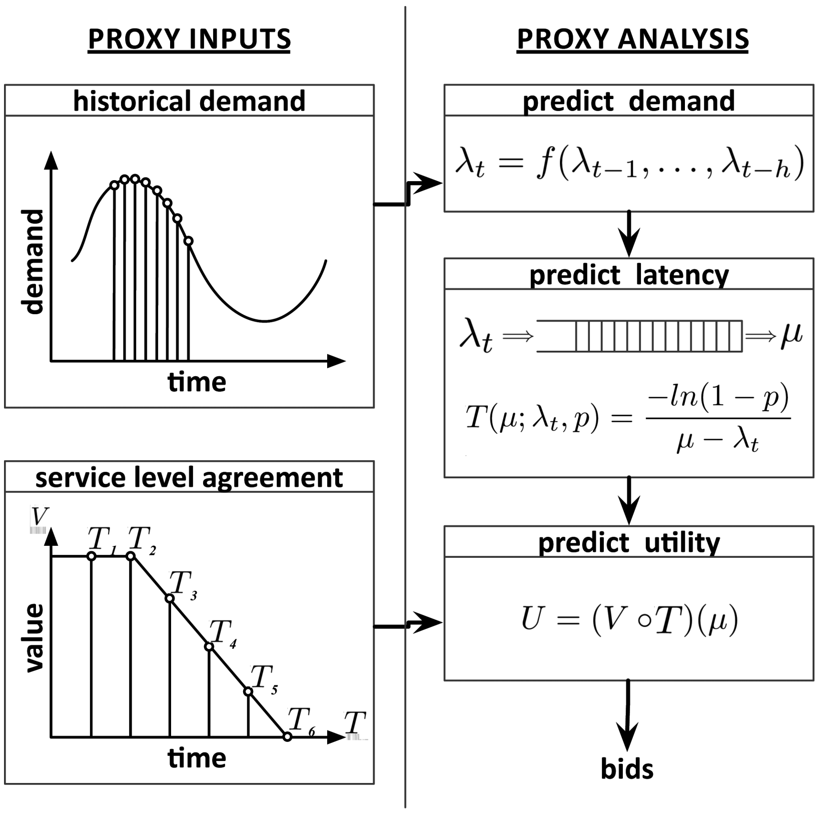

PROXY INPUTS
PROXY ANALYSIS
historical demand
demand
time
predict demand
$\lambda_t = f(\lambda_{t-1}, \ldots, \lambda_{t-h})$
predict latency
$\lambda_t \Rightarrow \mu$
$T(\mu; \lambda_t, p) = \frac{-ln(1-p)}{\mu - \lambda_t}$
service level agreement
V
T_1 T_2 T_3 T_4 T_5 T_6 T
value
time
predict utility
$U = (V \circ T)(\mu)$
bids

Fig. 3. Proxy bids.

so would require a cleanly defined feedback loop that continues to hide system complexity from users. Extending proxies in this manner is an avenue for future work.

3.1. Proxies and Value Analysis

We extend previously proposed markets to accommodate heterogeneous processors. These extensions embed greater hardware insight into the market. Buyers are task streams with diverse requirements and valuations. Sellers are datacenters with processors that differ in performance and energy efficiency. Proxies infer hardware preferences and bid for candidate hardware allocations. Figure 3 summarizes the role of the proxy.

Resource allocations are optimized periodically. Prior to each period, each application's proxy anticipates task arrivals and estimates the value of candidate hardware assignments. The bidding process has several steps: (i) estimate task arrival distribution, (ii) estimate task service rates, (iii) estimate task latency, and (iv) translate latency into bid.

Estimate Task Arrival Distribution. At the start of an allocation period t, the proxy has historical task arrival rates for h prior periods: $\lambda_H = (\lambda_{t-1}, \ldots, \lambda_{t-h})$. To estimate the current period's rate λ_t, the proxy fits a Gaussian distribution to the history and estimates task arrival rate by sampling from $N(E[\lambda_H], Var(\lambda_H))$. Thus, we drive the market with a predicted distribution of arrivals as in prior work [Lubin et al. 2009].

Estimate Task Service Rate. To serve these arriving tasks, an optimizer searches an allocation space of heterogeneous cores. Prior efforts assume fungible processor cycles [Chase et al. 2001; Lubin et al. 2009], an assumption that breaks under microarchitectural heterogeneity. In contrast, we scale each candidate allocation into a canonical one based on application-architecture interactions.

Suppose we have n core types. Let $q = (q_1, \ldots, q_n)$ denote a heterogeneous allocation of those cores and let $\kappa = (\kappa_1, \ldots, \kappa_n)$ denote their task-specific performance relative to a canonical core. Let Q denote an equivalent, homogeneous allocation of canonical cores. Finally, P denotes task performance (i.e., throughput) on the canonical core. In this notation, the canonical allocation is $Q = \kappa^T q$, which provides task service rate $\mu = PQ$.

The system can determine P and κ with little effect on performance. The proxy profiles a new task on the canonical core to determine P and initializes $\kappa_i = 1,\ i \in [1, n]$ to reflect initial indifference to heterogeneity. As allocations are made and as tasks are run, the proxies accrue insight and update κ. In steady state, κ will reflect task preferences for hardware. With many tasks, suboptimal hardware allocations to a few tasks for the sake of profiling have no appreciable impact on latency percentiles.

Estimate Task Latency. Service rate determines task latency. Agents estimate M/M/1 queueing effects, which is sufficiently accurate in our setting because the coefficients of variation for interarrival and service times are low; see Section 7 for details. We estimate latency percentiles with Eq. (2) and use the 95th percentile as the figure of merit, $p = 0.95$.

$$\text{p-th latency percentile} \mid T = -ln(1-p)/(\mu - \lambda) \tag{2}$$

$$\text{service rate inflections} \mid \hat{\mu}_t = \lambda_t - ln(1-p)/\hat{T} \tag{3}$$

Translate Latency into Bid. Latency determines user value. To faithfully represent their users, proxies must create a chain of relationships between hardware allocation, service rate, response time, and dollar value (Eqs. (4)–(6)).

$$\text{datacenter profiler} \mid \mathbf{P_a} : \{hw_a\} \rightarrow \{service\ rate\} \tag{4}$$

$$\text{datacenter queues} \mid \mathbf{T} : \{service\ rate\} \rightarrow \{latency\} \tag{5}$$

$$\text{user value} \mid \mathbf{V} : \{latency\} \rightarrow \{dollars\} \tag{6}$$

$$\text{market welfare} \mid \mathbf{W} = \sum_{a \in A} \big(\mathbf{V} \circ \mathbf{T} \circ \mathbf{P_a}(hw_a)\big) - \mathbf{C}(hw) \tag{7}$$

A profile $\mathbf{P_a}$ maps proxy **a**'s hardware allocation to an application-specific service rate. A queueing model $\mathbf{T}$ maps service rate to latency. Finally, the user provides a value function $\mathbf{V}$, mapping latency to dollars. Note that only $\mathbf{V}$ requires explicit user input.

These functions are composed when proxy a bids for a candidate hardware allocation: $\mathbf{V} \circ \mathbf{T} \circ \mathbf{P_a}(hw_a)$. To compose $\mathbf{V} \circ \mathbf{T}$, the proxy identifies inflections in the piecewise-linear value function $\mathbf{V}$. Then, the proxy translates each inflection in time $\hat{T}$ into an inflection in service rate $\hat{\mu}$ by inverting the queueing time equation (Eq. (3)).

Suppose the user derives some value given queueing time $\leq \hat{T}$ and less value given queueing time $> \hat{T}$. Equivalently, the user derives some value given service rate $\geq \hat{\mu}$ and less value given service rate $< \hat{\mu}$. Thus, service rate maps to dollar value. Note that service rate inflections depend on the arrival rate λ_t of tasks. To accommodate load changes, the proxy determines new inflection points for each period.

3.2. Seller Cost Analysis

For an accurate estimate of electricity use, the market requires information about server and processor power modes from the datacenter [Meisner et al. 2011, 2009]. For example, we model server power modes as three possible states: active, idle (but in an active power mode), and sleep.

In Eq. (8), the datacenter accounts for the number of servers (n^*) in each mode and power (P^*) dissipated over the allocation time period (Δ) [Lubin et al. 2009]. Servers that transition between modes incur a latency (δ^*). For example, a server that enters a sleep mode will dissipate P^{idle} over δ^{is} as it transitions and dissipate P^{sleep} for the remaining $\Delta - \delta^{is}$. Similarly, a server that wakes from sleep will require δ^{sa} during which P^{act} is dissipated but no useful work is done. Energy is multiplied by datacenter power usage effectiveness (PUE) and then by electricity costs [Barroso and Hölzle 2007].

$$E = \underbrace{\left(n^a P^{act} + n^i P^{idle} + n^s P^{sleep}\right)\Delta}_{\text{no power transition}} + \underbrace{n^{is}\left(P^{idle}\delta^{is} + P^{sleep}(\Delta - \delta^{is})\right)}_{\text{idle}\rightarrow\text{sleep}} + \underbrace{n^{sa}\left(P^{act}\delta^{sa} + P^{act}(\Delta - \delta^{sa})\right)}_{\text{sleep}\rightarrow\text{active}}. \tag{8}$$

3.3. Welfare Optimization

Proxies submit complex bids for candidate hardware allocations on behalf of users. Sellers submit machine profiles and their cost structure. The market then allocates processor cores to maximize welfare, or buyer value minus seller cost (Eq. (7)). Welfare optimization is formulated as a mixed integer program (MIP), which determines the number and type of cores each user receives. For MIP details, see Lubin's formulation [Lubin et al. 2009]. Allocations are optimized at core granularity but each core is ultimately mapped to processors and servers in post-processing. For example, active and sleeping cores cannot map to the same server if machines implement server-level sleep modes.

Heterogeneity increases optimization difficulty. In a naïve approach, value is a multi-dimensional function of heterogeneous quantities $q = (q_1, \ldots, q_n)$. However, the proxies would need to construct piecewise approximations for multi-dimensional bids, which is increasingly difficult as n grows. Each new core type would add a dimension to the problem.

Scaling to a canonical resource type improves tractability by imposing an abstraction between user proxies and datacenter hardware. By encapsulating this complexity, the proxy determines the relative performance of heterogeneous quantities $\kappa = (\kappa_1, \ldots, \kappa_n)$ and computes $Q = \kappa^T q$. Bids for Q are one-dimensional.

4. MANAGING HETEROGENEOUS PROCESSORS

For a heterogeneous datacenter with big Xeon and small Atom cores, we exercise three key aspects of the economic mechanism. First, heterogeneous microarchitectures are well represented by Xeons and Atoms. Cycles from in-order and out-of-order datapaths are not fungible. Second, heterogeneous tasks contend for these cycles with different preferences and valuations. Third, large processor power differences are representative of trends in heterogeneity and specialization.

4.1. Experimental Setup

Market Simulator. The evaluation uses an in-house simulator for the datacenter market. Simulator inputs include task-arrival patterns and hardware parameters, which are drawn from previously validated studies and detailed here. Task arrivals follow known diurnal patterns, which are accurately captured by sinusoids. Hardware performance and power parameters are either measured from physical systems or simulated using validated cycle-accurate models.

Table I. Architecture Parameters for Xeons, Atoms.
[George et al. 2007; Gerosa et al. 2009; Intel 2011]

	Xeon	Atom
Number of Nodes	0–160	0–225
Number of Cores	4	16
Frequency	2.5 GHz	1.6 GHz
Pipeline	14 stages	16 stages
Superscalar	4 inst issue	2 inst issue
Execution	out-of-order	in-order
L1 I/D Cache	32/32KB	32/24KB
L2 Cache	12MB, 24-way	4MB, 8-way

Table II. Power Modes and Parameters.
[Janapa Reddi et al. 2010]

	Xeon	Atom
Core sleep	0 W	
Core idle	7.8 W	0.8 W
Core active	15.6 W	1.6 W
Platform sleep	25 W	
Platform idle	65 W	
Platform active	65 W	
Sleep → Active	8 secs, $0.05	
Active → Sleep	6 secs, $0.05	

With these inputs, we simulate proxy responsibilities. The simulator does exactly what a real cluster manager would do. Indeed, to produce a system prototype, the implementation would only need software interfaces to receive proxy inputs and to transmit allocation decisions across the datacenter network. The computational costs of the simulator accurately reflect management overheads of a deployed system.

Much of the current implementation focuses on proxy functions and welfare optimization. The proxy predicts demand from history, predicts latency using a closed-form response time model, and constructs a bid. These bids are compared against hardware costs, identifying welfare-maximizing hardware allocations by invoking CPLEX to solve a mixed integer linear program (MIP). The MIP solution is the allocation for the next 10-minute interval.

To evaluate an allocation, the simulator computes the sum of users' values and subtracts hardware cost. An allocation provides computational cycles, which are scaled to reflect potential performance penalties from heterogeneous processors. Collectively, cycles determine task service rate. Given task arrival and service rates, queueing models estimate waiting time, which determines user value. Costs simply reflect energy consumed by allocated hardware.

Processor Parameters. Tables I–II summarize platform parameters [George et al. 2007; Gerosa et al. 2009; Intel 2011]. Xeon core power is 10× Atom core power. The hypothetical sixteen-core Atom integrates many cores per chip to balance the server organization and amortize platform components (e.g., motherboard, memory) over more compute resources [Grot et al. 2012; Janapa Reddi et al. 2010; Seamicro 2011]. We estimate server power by increasing core power in proportion to the number of cores per processor while leaving platform power unchanged.

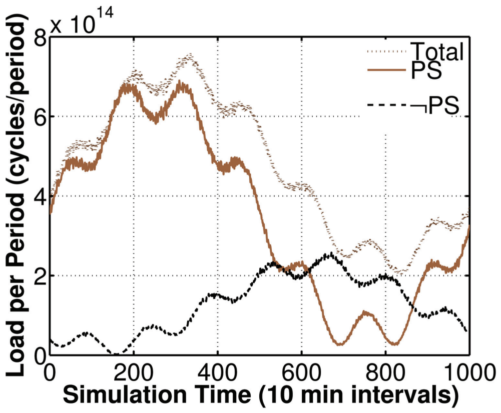
Servers transition from active to sleep mode in 6 seconds and from sleep to active in 8 secs, powering off everything but the memory and network interface [Agarwal et al. 2009; Gandhi et al. 2011]. Power usage effectiveness (PUE) for the datacenter is 1.6, an average of industry standards [Facebook 2011; U.S. Environmental Protection Agency 2007]. Energy costs are $0.07 per kWh, an average of surveyed energy costs from prior work [Qureshi et al. 2009].

We explore a range of heterogeneous configurations, varying the ratio of Xeons and Atoms. The initial system has 160 Xeon servers, a number determined experimentally to accommodate the load of the evaluated applications. We sweep the Atom to Xeon ratio by progressively replacing a Xeon with the number of Atom servers that fit within a Xeon power budget. A 20kW datacenter accommodates 160 Xeons, 225 Atoms, or some combination thereof.

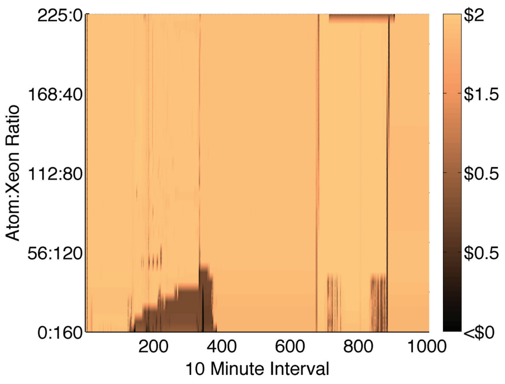
Workloads. We study tasks that are generated to follow a time series, which is detailed in Table III and illustrated in Figure 4(a). We simulate a week of task load that is a composite of two sinusoids, one with a week-long period and one with a day-long period.

Table III. Application Characteristics. For a task stream, T is 95th percentile queueing time

	Proc Sensitive (PS)	Proc Insensitive (¬PS)
P–task profile (Mcycles/task)	70	50
λ–peak load (Ktasks/min)	1000	500
V–value ($/month)	$5000 if $T \leq$ 10ms $0 if $T \geq$ 80ms	$4500 if $T \leq$ 10ms $0 if $T \geq$ 80ms
κ–scaling factor	$\kappa_{\mathbf{X}} = 1.0$ $\kappa_{\mathbf{A}} = 0.33$	$\kappa_{\mathbf{X}} = 1.0$ $\kappa_{\mathbf{A}} = 1.0$



(a) **Request Rate**



(b) **Datacenter Welfare**

Fig. 4. (a) Demand for processor sensitive (PS) and insensitive (¬PS) applications. (b) Increasing the number of Atoms improves welfare. Both applications are better serviced during traffic peaks (e.g., intervals 130–380).

The sinusoid determines the average arrival rate around which we specify a Gaussian distribution to reflect load randomness. Such patterns are representative of real-world web services [Meisner et al. 2011].

Applications specify value (SLA) as a function of 95th percentile response time. Value degrades linearly up to a cut-off of 80ms, after which computation has no value. The value functions express priorities for applications. Since the market maximizes welfare, users with higher value per requested cycle are more likely to receive hardware. The economic mechanism does not accommodate under-bidding and valuations must at least cover the cost of computation.

4.2. Architectural Preferences

We consider two workloads that contend for Xeons and Atom servers, yet value the cores differently. The first is a processor sensitive (PS) application that prefers cycles from the high-throughput Xeon and values an Atom cycle less, scaling its value down by $\kappa = \frac{1}{3}$. The second, on the other hand, is a processor insensitive (¬PS) application indifferent between the two processor types.

The architecture scaling factors κ are consistent with prior datacenter workload characterizations. Reddi et al. find $\frac{\texttt{Inst}}{\texttt{Cycle}}$ on Atom is 33% of that on Xeon for Microsoft Bing web search [Janapa Reddi et al. 2010]. Lim et al. find performance on the mobile-class Intel Core 2 Turion is 34% of that on the Intel Xeon [Lim et al. 2008]. These applications exhibit instruction-level parallelism, which benefits from wider pipelines and out-of-order execution in server-class cores: $\kappa_X = 1$, $\kappa_A = \frac{1}{3}$.

In contrast, ¬PS does not benefit from extra capabilities in server-class processors and is representative of web, file, or database servers [Davis et al. 2005; Kongetira and Aingaran 2005]. Andersen et al. [2009] propose embedded processors for distributed key-value store servers. Servers that deliver YouTube-like traffic can run on small cores with negligible performance penalties [Lim et al. 2008]. Higher processor performance does not benefit such workloads. ¬PS applications are indifferent to Xeons and Atoms: $\kappa_X = \kappa_A = 1$.

PS and ¬PS computation is valued at \$1.16 and \$1.04 per period, respectively, assuming performance targets are met. The cost of allocating all resources in the most expensive Xeon-only datacenter configuration is less than \$0.30. Thus, valuations cover the cost of computation.

4.3. Improving Welfare

A heterogeneous mix of Xeons and Atoms enhances welfare, which is value minus cost. Figure 4(b) illustrates welfare for a variety of datacenter configurations. The vertical axis presents various datacenter configurations that differ in the number of Xeons and Atoms that fit within a given power budget. For example, the datacenter can accommodate 160 Xeons, 225 Atoms, or some other configuration in between.

The market periodically allocates datacenter cores. The horizontal axis shows time intervals that span one week of activity. Each interval demands a different level of activity as shown in Figure 4(a). For these varying ratios of processor cores and activity levels, Figure 4(b) quantifies system welfare delivered by the market. We find that a heterogeneous mix of Xeons and Atoms enhances welfare. To understand this advantage, we study homogeneity's limitations on both sides of the ledger: value and cost.

Value. A Xeon-only system provides less value because it cannot meet performance targets during traffic spikes. Users derive no value when latencies violate the target (waiting time $\leq$ 80ms), which happens in more than a third of the allocation periods. Periods of low welfare arise directly from periods of poor service quality; in Figure 5(a) see periods 130–380.

As we replace Xeons with Atoms, we increase system capacity within the 20kW budget. Each four-core Xeon server can be replaced by 1.4 sixteen-core Atom servers. Equivalently, each Xeon core is replaced by 5.6 Atom cores. And if we account for the frequency difference, each Xeon cycle is replaced by 3.6 Atom cycles. This extra capacity enhances value and welfare during traffic peaks even after scaling down core capability by κ.

Moreover, applications successfully bid for preferred architectures. As Xeons become scarce, PS receives more of its preferred Xeons at the expense of ¬PS, which is indifferent between Xeons and Atoms. As Xeons are replaced by Atoms, Figure 6(a) shows the market allocating a larger fraction of the remaining Xeons to PS thus improving its response time. Simultaneously, Figure 6(c) shows the market allocating fewer Xeons to ¬PS.

Cost. On the other side of the ledger, energy costs degrade welfare. Cores incur transition costs when they change power modes. During a transition, cores dissipate power but do not add value. As shown in Figure 7(a), a charge is imposed for every transition to account for increased wear and reduced mean-time-to-failure as machines power-cycle [Guenter et al. 2011].

Per server, Xeons and Atoms incur the same transition cost. Yet the Atom-only system incurs larger transition costs than alternate systems as it manages more servers. Since an Atom system contributes fewer cycles and less value than a Xeon server, such costs reduce allocation responsiveness. This inertia of the Atom-only system

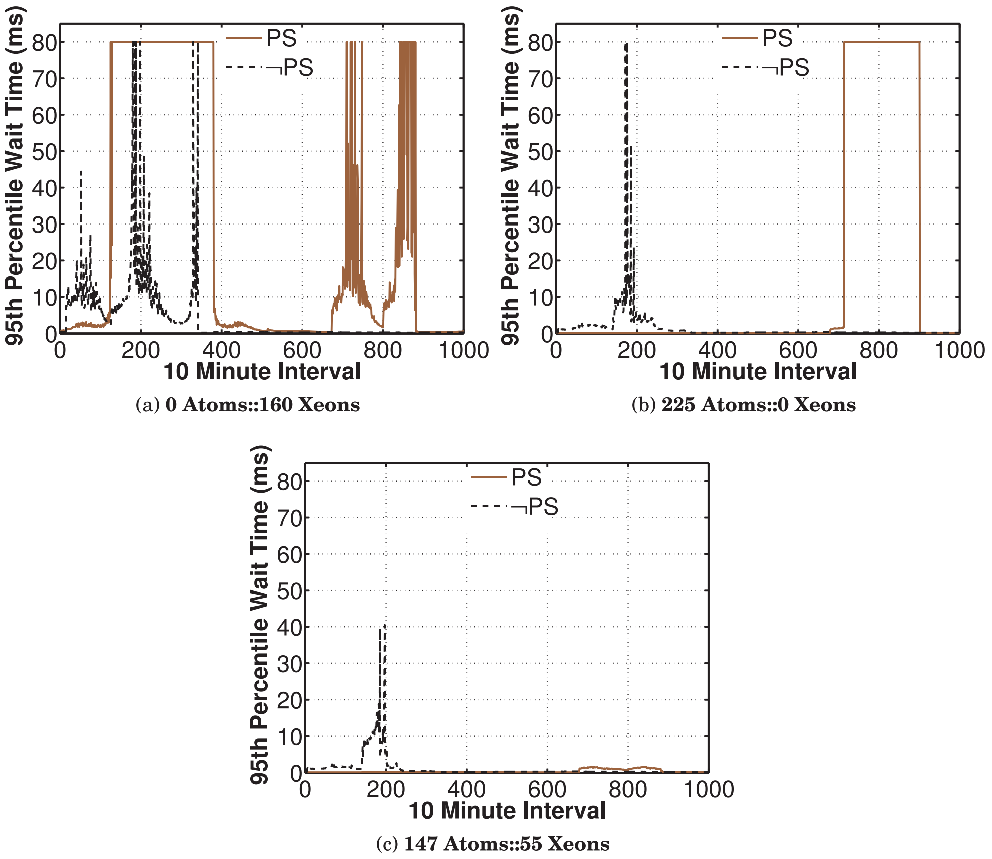


Fig. 5. 95th percentile waiting time for (a) only Xeons, (b) only Atoms, and (c) a mix of Xeons and Atoms.

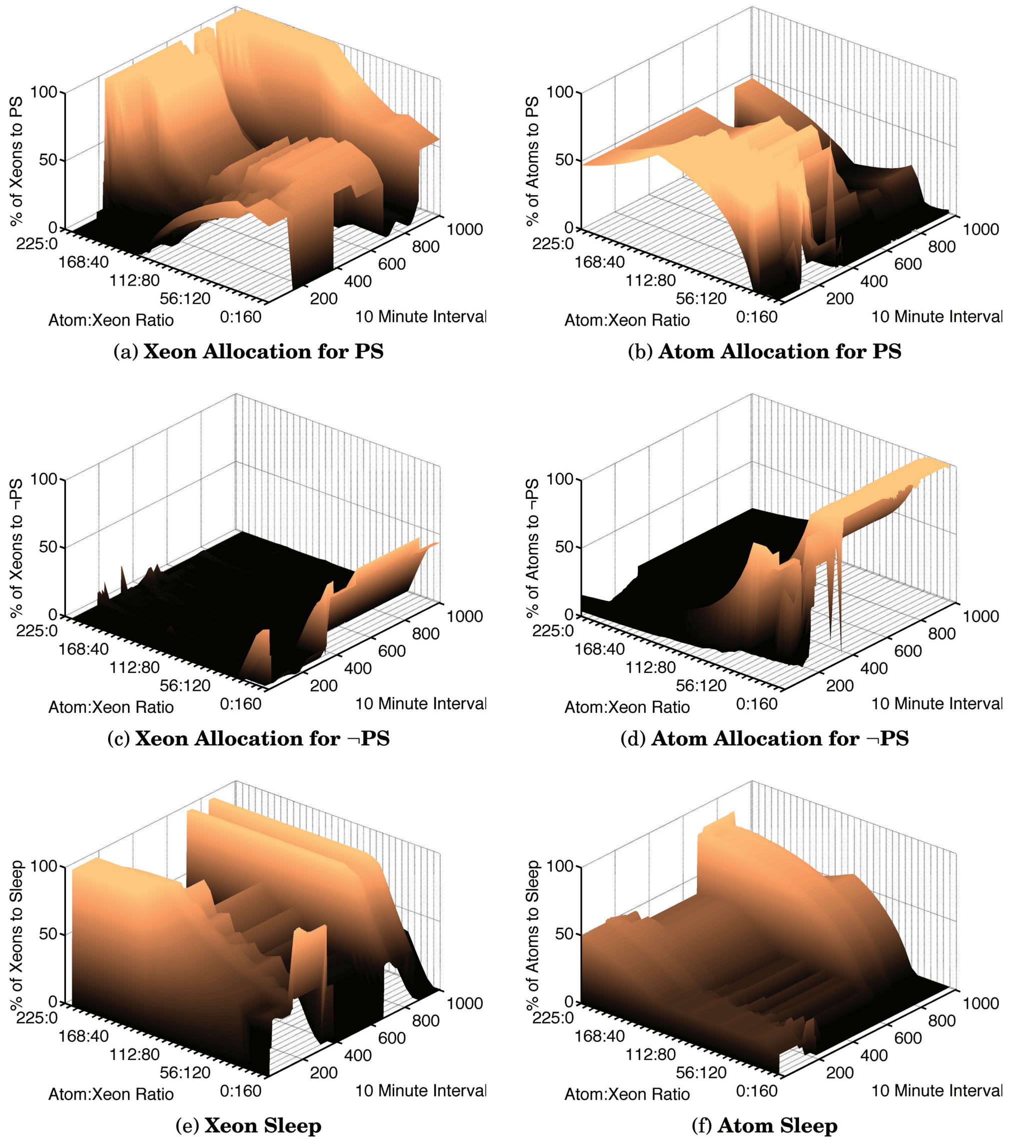
causes a response time spike at the first load peak (period 200) but not the second (Figure 5(b)).

4.4. Balancing Atoms and Xeons

Number of Atoms. Given a mix of applications and hardware preferences, there exists a maximum number of Atoms that can be usefully substituted into the system. Beyond this number, additional Atoms are not useful to either application, leaving the absolute number of allocated Atoms unchanged.

In our datacenter, the maximum number of useful Atom servers is 147. This maximum marks a point of diminishing marginal returns for substituting Atoms. Beyond this point, additional Atoms are put to sleep (Figure 6(f)) and the fraction of Atoms allocated to PS and ¬PS decline (Figure 6(b) and Figure 6(d), respectively). In fact, adding Atoms beyond this point can harm welfare as transition costs are incurred to turn them off. This cost produces the highest ridge of Figure 7(a), where spare Atom servers are transitioned to sleep.

Number of Xeons. A related conclusion can be made for Xeons: there exists a minimum number of Xeons necessary to provide the PS application adequate performance. Beyond this point, as Atoms are added and Xeons are removed, the number of Xeons available to be allocated to PS steadily decreases – Atoms are used for part of the processor-sensitive computation (Figure 6(a) and Figure 6(b), respectively), decreasing



(a) **Xeon Allocation for PS** (b) **Atom Allocation for PS**

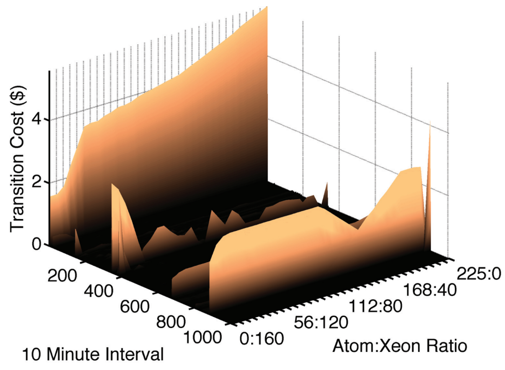
(c) **Xeon Allocation for ¬PS** (d) **Atom Allocation for ¬PS**

(e) **Xeon Sleep** (f) **Atom Sleep**

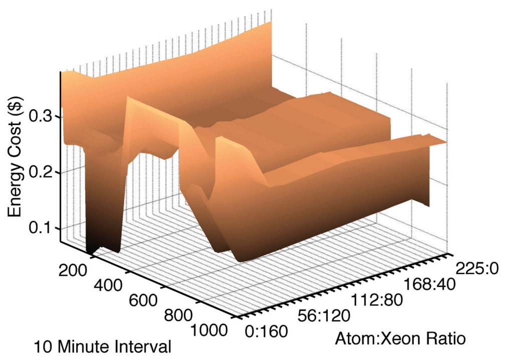
Fig. 6. Allocation measured in fraction of configured nodes as Atom:Xeon ratio varies. For example, a 50% Atom allocation in a A:X = 168:48 configuration maps to 84 Atom nodes. Atom and Xeon cores at each datacenter configuration may be allocated to the processor sensitive (PS), processor insensitive (¬PS), or neither (sleep) application.

performance. As we replace most of the Xeons in the system with Atoms, the few Xeons remaining in the system are either allocated to PS or put to sleep during PS activity troughs (Figure 6(a) and Figure 6(e), respectively). Clearly, as they become scarce, the remaining Xeons are increasingly precious to PS.

Based on this, our datacenter should have at least 55 Xeon servers. This minimum marks a point of increasing marginal penalties incurred when removing Xeons. Strikingly, this minimum occurs in a heterogeneous configuration with 55 Xeons and 147 Atoms, which coincides with our analysis for the maximum number of Atoms.



(a) **Transition Cost**



(b) **Energy**

Fig. 7. Seller costs due to (a) energy and (b) transition penalty, as the ratio of Atom:Xeon processors varies. Energy costs correspond to application behavior. Ridges in transition cost are due to $0.05 penalty per transition that accounts for increased system wear.

Max/Min Heterogeneity. We refer to this balance of 147 Atom and 55 Xeon servers as the max/min configuration for the datacenter. This heterogeneous configuration provides better service quality and fewer SLA violations. As seen in Figure 5(c), this mix of Xeons and Atoms provide queueing times that are stable and far below the 80ms cut-off.

For contrast, consider Figure 5(a) and Figure 5(b). 38% and 19% of allocation periods violate the 80ms cut-off for PS queueing time in Xeon- and Atom-only systems, respectively. In the Xeon-only system, ¬PS suffers longer waiting times due to contention with PS for limited computational capacity. In the Atom-only system, ¬PS experiences volatile waiting times during time periods 147–217.

Thus, by replacing Xeon with Atom nodes within a fixed power budget, the mixed configurations increase the system's computational capacity. This clear benefit of specialization will play an important role towards sustaining the rapidly growing demand on datacenters.

4.5. Saving Energy

The allocation mechanism activates servers only in response to demand. The datacenter saves energy by putting unneeded servers to sleep. As shown in Figure 8, a homogeneous Xeon-only datacenter saves 900kWh over a week of simulated time.

When Atoms are first introduced, cycles become scarce and fewer servers exploit sleep modes; the datacenter saves only 600kWh. Note, however, that the heterogeneous datacenter saves this energy while simultaneously improving service quality (Figure 5). Energy savings from server activation plateau at 1200kWh for most heterogeneous systems, including the max/min configuration. While an Atom-only system could save up to 1280kWh, it would sacrifice service quality and violate performance targets during the PS activity trough.

Datacenters may prefer to schedule low-priority batch jobs rather than exploit sleep states [Barroso and Hölzle 2009]. Presumably, the value of batch computation exceeds servers' operating and amortized capital costs. Spot prices for Amazon EC2 are one measure of these costs. Given batch jobs with sufficient value, a policy that replaces sleep modes with active batch computing will only increase welfare.

Even in such datacenters, heterogeneity improves efficiency. A mix of active Xeons and Atoms consumes less energy (Figure 7(b)). The max/min configuration consumes 4.0kWh per allocation period. In contrast, the Xeon-only system consumes 5.4kWh yet exhibits more volatile service quality.

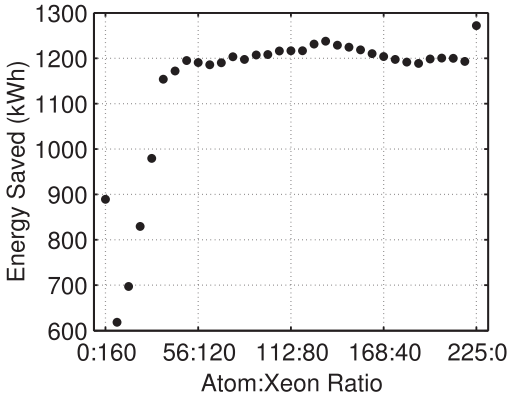


Fig. 8. Energy saved due to server activation.

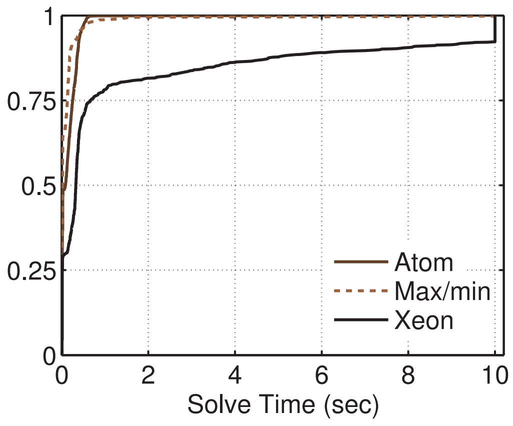


Fig. 9. CDF of time to solve MIP across all periods.

4.6. Evaluating Optimization Time

Given that the market and proxy implementation include all the elements required in a real system, market clearing performance is important. Across allocation periods in all datacenter configurations, solve time (wall clock) varies but is less than 800ms for 98% of allocation periods. All other periods clear the market in less than 10s as shown in Figure 9. Solve time increases when resources are scarce and contention is high. And contention is highest in a Xeon-only system, which provides the worst service quality.

We implement the market and proxy in Java as it would be in a real distributed system. Inputs are task arrival history and user value functions. Outputs are resource allocations, which maximize welfare. Welfare is optimized with a mixed integer program, which is quickly solved to exact optimality by commercial CPLEX 12.1 MIP solver codes despite the formally NP-Hard nature of the problem.

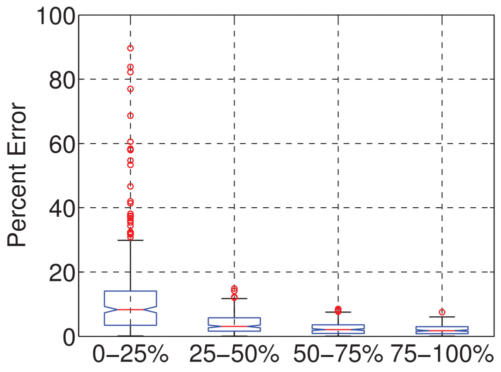
We obtain this computational speed by representing heterogeneous resources as scaled canonical ones, and thus keeping the MIP tractable. Further, in our MIP formulation the task arrival rate and the number of servers in the system simply affect coefficients, not MIP computational complexity. However, the number of user applications and heterogeneous resource types does impact MIP complexity, and for sufficiently complex data centers it is possible that CPLEX might solve only to approximate optimality within time allotted for computation. Fortunately, previous work has shown that such approximate solutions are efficient with high probability [Lubin et al. 2009].

4.7. Assessing Demand Prediction

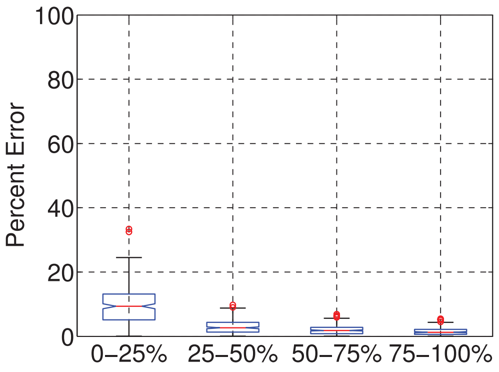
Figure 5 illustrates a response time spike, which corresponds to a trough in $\neg$PS load. Although adding mobile-class Atoms to the datacenter improves PS waiting time, the queueing time for $\neg$PS suffers during periods 165–185 across all mixes of Xeon and Atom cores. These performance effects may be counter-intuitive as one might expect low response times during periods of low activity.

However, demand prediction is particularly difficult to predict in this trough. And poor predictions affect an agent's value estimates and bids for resources. If an agent underpredicts load during an interval, the market provides too few resources. Even with low demand, the allocated cores may be insufficient for low response times.

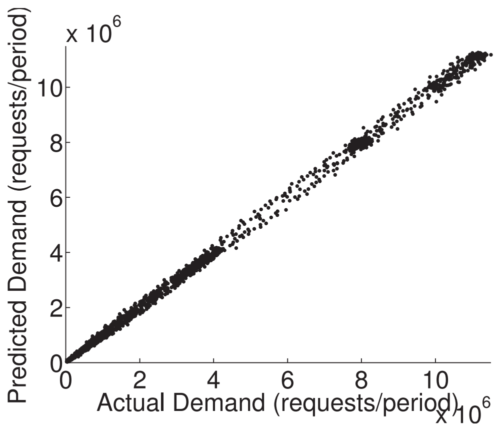
Figure 10(a)–10(b) illustrates prediction errors broken into demand quartiles. For example, 0–25% examines prediction errors for intervals with computational demand in the lowest quartile. The boxplot illustrates the error distribution for predictions



(a) **Processor Insensitive Application**



(b) **Processor Sensitive Application**



(c) **Relative Error**

Fig. 10. Prediction error in demand: (a, b) percent error for each application at different quartiles of total load. Using 6 time periods of load traffic history, the predictions are sufficiently accurate; (c) relative error.

made in that quartile. The horizontal lines indicate error quartiles. For example, the red horizontal line shows the median error.

These figures indicate errors are larger at low levels of demand. This effect is most noticeable in for ¬PS. At lowest quartile of load, Figure 10(a) indicates prediction errors for ¬PS can be as high as 90%. Figure 10(b) indicates PS prediction error follows a similar trend, yet the errors are not nearly as large as those for ¬PS. Thus, we attribute the ridge in ¬PS waiting time to prediction error during time periods 165–185.

More generally, however, prediction errors are low. Figure 10(c) plots predicted demand against actual demand, illustrating a strong correlation between the two. Larger levels of demand are predicted to be larger and smaller levels of demand are predicted to be smaller.

5. INCREASED PROCESSOR HETEROGENEITY

Increasing processor diversity allows for tailoring datacenter resources to the application mix. In this section, we investigate the design space of *sets* of diverse processor types, when the goal is to obtain an effective mix within a datacenter. To do so, we cluster processor/core designs and identify representative individuals. We then study combinations of these cores for datacenters that span a spectrum of heterogeneity.

Table IV. Parameters for the Twelve Cores Simulated in gem5

	Out-of-order			InOrder		
Clock	1.0GHz	2.4GHz		1.0	2.4	
Width	2	6	8	1	2	4
ROB	192	320	342	–		
RF	80	120	160	–		
L1 I-$	64 KB 4-way			32 KB 4-way		
L1 D-$	64 KB 4-way			32 KB 4-way		
L2 $	4 MB 8-way			1 MB 8-way		

Table V. Application Characteristics

	libq	lbm
P – task profile (Mcycles/task)	67	149
λ – peak load (Ktasks/min)	480	80
V – value ($/month)		
if $T \leq 20$ms	$5000	$2500
if $T \geq 160$ms	$0	$0
κ – scaling factor		
$\kappa_{\mathbf{io1w10}}$	0.50	1.45
$\kappa_{\mathbf{io1w24}}$	0.40	1.09
$\kappa_{\mathbf{io4w24}}$	0.56	1.26
$\kappa_{\mathbf{oo6w24}}$	1.00	1.00

5.1. Experimental Setup

Atom efficiency is derived from three key design elements: static instruction scheduling, narrower issue width, and lower frequency. We define a space around these elements, producing twelve designs with parameters in Table IV. We simulate these designs with the gem5 cycle-accurate simulator in syscall emulation mode [Binkert et al. 2011].

For this experiment, we consider the preferences of SPEC CPU2006 applications on heterogeneous processors. These benchmarks are sensitive to processor choice, and we study the opportunity of using low-power cores even for applications with high instruction-level parallelism. We simulate 100M instructions from gobmk, hmmer, h264ref, mcf, libquantum, bzip2, sjeng, gcc, xalancbmk, milc, gromacs, namd, calculix, deallII, soplex, and lbm. Applications are cross-compiled into ALPHA with level -O2 optimizations.

These architectures and applications offer a wide range of performance scaling factors for evaluating heterogeneity. The market allocates resources for streams of computational tasks. We define a stream for each SPEC application with service-level agreements defined in Table V which shares parameters from Section 4 where possible.

5.2. Architectural Preferences

Only a subset of the twelve cores is necessary to reap the efficiency of heterogeneity. To identify this subset, we cluster cores with similar performance for the application suite. For each core, we define an n-element vector specifying its performance for n applications. We cluster these vectors with multidimensional, hierarchical clustering [Phansalkar et al. 2007]. In this formulation, each application adds a dimension. Hierarchical clustering constructs a dendrogram that quantifies similarity using Euclidean distance. By examining this tree at different levels, we choose results for a particular number of clusters k.

Figure 11(b) shows $k = 4$ clusters. The twelve original cores are ordered by increasing power on the x-axis. For each core, we plot the performance for various applications. Across the application suite, cores in the same cluster provide similar performance. From each cluster, we select the core with the lowest variation in performance (Table VI). We refer to cores with the tuple: [IO/OO][width][frequency]. For example, io1w10 denotes a 1-wide, in-order core with a 1.0GHz clock.

We organize these cores into servers that use equal-area processors. We normalize processor designs by silicon area since it is the primary determinant of a processor's marginal cost. Area is estimated with McPAT models [Li et al. 2009]. Power is also estimated with McPAT and calibrated to real Xeon and Atom measurements. In addition to processor core power, we consider power dissipated by platform components (e.g.,

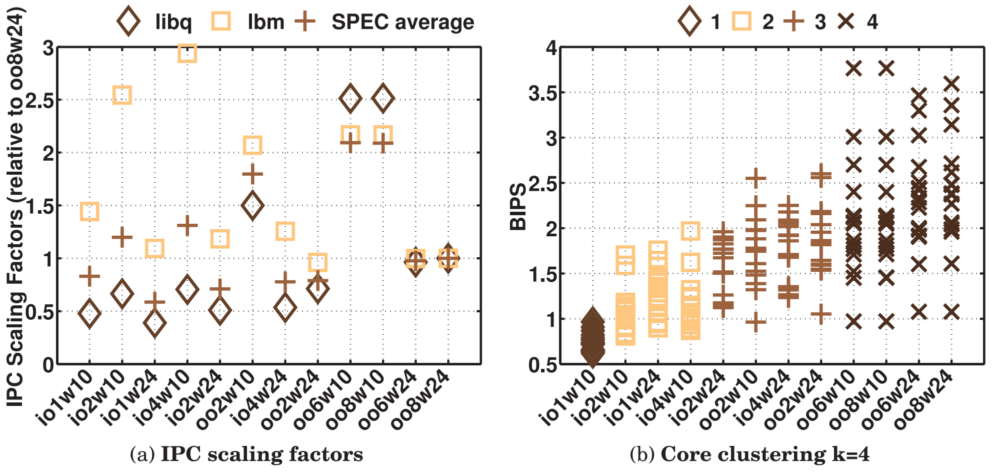


Fig. 11. gem5 simulation results, cores on the horizontal axis are in order of increasing peak dynamic power.

Table VI. Area and Power Estimates for Four Core Types

	io1w10	io1w24	io4w24	oo6w24
Num	18	18	12	6
Area (mm^2)				
Core	12.31	12.31	17.31	36.89
Die	<225			
Power (W)				
Core	1.10	2.63	8.40	28.10
Sys	65.00			
Tot	85	114	168	235

motherboard, memory), drawing on numbers reported in prior work [Janapa Reddi et al. 2010].

Finally, we determine the number of servers that fit in a 15KW datacenter. We explore a mix of heterogeneous processors and servers. Because a full sweep of heterogeneous combinations is prohibitively expensive for more than two core types, we simulate datacenters comprised of $\frac{1}{4}$, $\frac{1}{2}$, $\frac{3}{4}$, or entirely of each core type within the power budget.

5.3. Improving Service Quality

Increased processor diversity benefits service quality. Figure 12 compares the number of allocation periods where response time exceeds target cutoffs on each datacenter configuration, which are ordered by increasing computational capacity on the x-axis. Data is shown for libquantum and lbm, which are representative of diversity in the broader application suite (Figure 11(a)).

As in the Xeon and Atom case, a homogeneous system that uses the highest performing core provides the fewest number of these cores within a limited power budget. In fact, homogeneous systems of any core type violate performance targets for 20% or more of the allocation periods.

Replacing the oo6w24 core with io*w** cores produces a configuration with strictly more compute cycles available per unit time. However, these cycles do not necessarily translate into better performance. Cycles are scaled by diverse factors that reflect heterogeneous preferences for hardware.

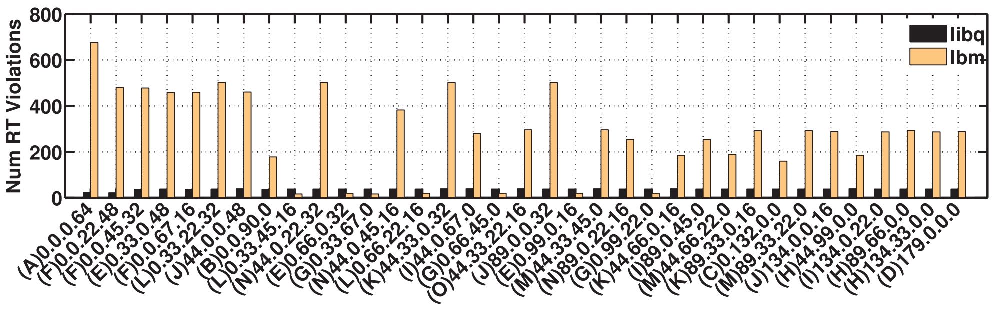


Fig. 12. Number of 95th percentile waiting time violations. The horizontal axis indicates the number of servers of each type as the tuple [io1w10].[io1w24].[io4w24].[oo6w24]. Prepended letters mark the corresponding region in Figure 13.

On its own, each core type is inadequate. But as part of a heterogeneous mix, diverse cores can improve service quality. Specifically, the worst of the homogeneous systems uses only oo6w24 cores. Yet oo6w24 cores are included in more than half of the most effective heterogeneous mixes, which produce the fewest service violations.

This observation showcases the complexity of navigating a heterogeneous design space. Had oo6w24 been discarded as a candidate design due to its poor performance in a homogeneous setting, several heterogeneous systems that include this core type would remain undiscovered.

More generally, combinations of io1w24, io4w24, and oo6w24 provide the best service quality for libquantum and lbm. For example, a system with 33 io1w24 cores and 67 io4w24 cores (00.33.67.0 in Figure 12) has the fewest response time violations. Our applications prefer designs with deeper pipelines and higher frequencies. However, if applications had exhibited complex control flow and poorly predicted branches, shallower pipelines would have been preferred.

5.4. Balancing Core Types

Figure 13 depicts the datacenter design space for four processor types. Colored dots show the percentage of allocation intervals that incurred waiting time violations for a system servicing libquantum and lbm task streams. Configurations in regions A–D are homogeneous. And those in regions E–J, K–N, and O are heterogeneous combinations of two, three, and four core types, respectively.

Microarchitectural Heterogeneity. Various combinations of io1w24, io4w24, and oo6w24 provide attractive service quality. Heterogeneity with design elements that span instruction scheduling and superscalar width are best suited to accommodate the diversity of libquantum and lbm. In contrast, despite the power savings, the decreased performance of a shallower pipeline is unattractive for these applications.

The design space has a few unambiguous conclusions. A mix of io4w24 and io1w24 cores performs well. This intersection, region G, contains the configuration with the best service quality, incurring quality-of-service violations for 1.6% of the time intervals. The two other points in this region are almost as good at 1.7%.

Also clear, configurations that include io1w10 unanimously provide poor service quality. Its ellipse is solely populated by light colored points, marking waiting time violations for up to 15.5% of the experiment. Datacenter configurations within this ellipse can likely be trimmed from a subsequent, fine-grained sweep of remaining regions. In general, discarding core combinations is not straightforward because of inconsistent trends like those in regions E and L.

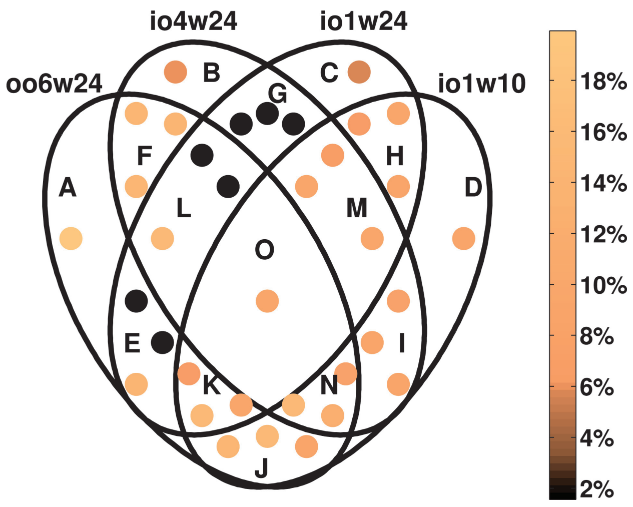


Fig. 13. Sum of libq and lbm waiting time violations, shown on a Venn diagram.

Number of Heterogeneous Microarchitectures. Heterogeneous design space exploration is iterative and expensive. For tractability, this study has assumed four heterogeneous core types but this choice might also be parameterized to produce subtle effects.

If we had chosen $k = 3$ clusters, io1w10 would have been absorbed into the io1w24 cluster. Moreover, io1w10 would have replaced io1w24 as the representative core from this cluster since we select cores to minimize performance variation.[2] In this scenario, regions E, G and L of Figure 13 would not have been explored. Missing the opportunity to explore G is particularly unfortunate since its heterogeneous configurations produced the best service quality.

Choosing more clusters $k > 4$ might have produced other tradeoffs. But related work in heterogeneous microarchitectures have illustrated diminishing marginal returns, which coincidentally arise as heterogeneity increases beyond four designs [Lee and Brooks 2007]. Moreover, datacenters with more than four core types may produce impractical capital and maintenance costs.

This complex design space and its sophisticated tradeoffs call for further innovation in the heuristics and metrics that guide optimization. The benefits to specialization of datacenter resources are manifold, and the market mechanism provides necessary abstractions and management capabilities.

6. MODELING QUALITY-OF-SERVICE

The market mechanism acts on predictions from M/M/1 queueing models. Queueing dynamics affect quality-of-service via waiting time. The time a task spends in the queue determines an allocation's value. But user-perceived response time also includes service time, which begins when a task exits the queue and ends when the task completes. We analytically consider the interaction between waiting and service times for Xeons and Atoms. In doing so, we acquire intuition and justify the use of waiting time as a quality-of-service measure.

[2]Alternatively, we could limit the clustering methodology to microarchitecture alone and apply dynamic frequency scaling to include both designs.

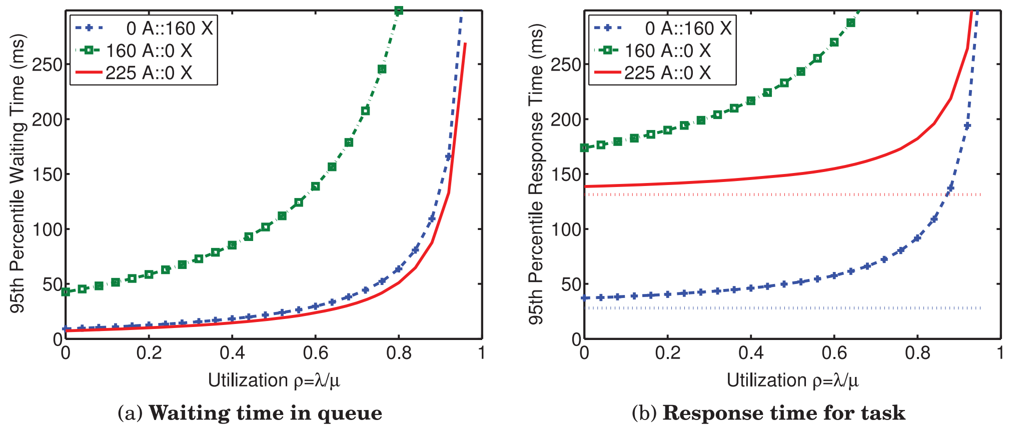


Fig. 14. For three datacenter configurations, response time as a function of utilization, broken into (a) queuing time and (b) service time.

Amdahl's Law. Prior applications of the M/M/1 queue assume an application accumulates its allocated resources, such as cores and $\frac{\texttt{Cycle}}{\texttt{Sec}}$ [Chase et al. 2001; Lubin et al. 2009]. For a fixed task arrival rate λ, M/M/1 would estimate lower response time T as service rate μ increases—precisely the effect we desire. But response time has two components: wait time T_W and service time T_S. According to queuing theory, expected service time is $1/\mu$, assuming a single server that accumulates all allocated hardware to provide service rate μ.

Yet we know from Amdahl's Law that execution time, T_S, will eventually be bound by the non-parallel fraction of execution. For large μ, the M/M/1 model alone may optimistically estimate lower T_S. For this reason, we use queuing models for T_W and instrument the proxy to enforce T_S in Section 3. Allocation decisions that seek to guarantee a given response time should foremost be wary of T_W, which is affected by load variability.

Optimizing for Waiting Time. Utilization, $\rho = \frac{\mu}{/}\lambda$, is a measure of the traffic intensity. Given a fixed size datacenter, ρ increases with λ. Figure 14 shows the 95th percentile response time for three systems as utilization varies. The baseline (A:X = 0:160) is comprised of Xeons only. The second system replaces all Xeon nodes with an equivalent number of Atom nodes (A:X=160:0). Atom's microarchitecture retires $3\times$ fewer $\frac{\texttt{Inst}}{\texttt{Cycle}}$. Combined with Atom's lower frequency, the second system causes an overall slowdown of $4.7\times$. The third system has 225 Atom nodes, which fits in the first system's Xeon-only power budget.

Figure 14(a) compares the 95th percentile waiting time computed analytically for a M/M/1 queue serviced by the three systems described here. Replacing 160 Xeons with 160 Atoms increases waiting time; tasks queue longer when served by Atoms that provide less throughput. In fact, the second system violates waiting time constraints in our service-level agreements. More alarming, Atom adoption shifts the knee of the curve to lower values of ρ, making response time more vulnerable to sharp increases in waiting time if ρ were to increase beyond the knee due to load variability.

The third configuration, where the number of Atoms is scaled to fit the power budget of the Xeon system, improves waiting time and the utilization knee. Waiting time improves beyond that of the Xeon system given the larger number of available Atoms. The knee of the curve shifts outward as well; the system can provide low waiting times at higher ρ. These are the effects we observe in our design space for heterogeneous systems as we sweep combinations of Xeons and Atoms.

Accounting for Service Time. Figure 14(b) provides a comparison of the 95th percentile response time for the three systems, adding service time T_S and accounting for single-threaded tasks that suffer a 4.7× slowdown on Atoms (3× from $\frac{\texttt{Inst}}{\texttt{Cycle}}$ and 1.56× from $\frac{\texttt{Cycle}}{\texttt{Sec}}$). T_S for tasks exiting the queue is shown by horizontal dotted lines. A PS task that requires 70M canonical cycles has a T_S of 28ms on the Xeon and a T_S of 132ms on the Atom. Each system's curve sums fixed T_S given the processor type and T_W from Figure 14(a).

T_S is the larger component of response time at low utilization; in Figure 14(b), T_S is the vertical difference between the curves. However, a good datacenter resource allocator would seek to maximize utilization subject to meeting waiting time targets T_S by shutting down under-utilized nodes and increasing utilization. At higher ρ, T_W dominates response time and load spikes may push ρ beyond the knee. At such high ρ, almost any service-level agreement would be violated given the steep, almost vertical, growth in response time.

Any mechanism aiming to right-size a datacenter for incoming load aims to operate active servers at high ρ. Therefore, by guiding our allocator with a queuing time estimate, our resulting allocations will be effective. In other words, these allocations seek high system utilization while striving to ensure that response time lies before a knee specified by the service-level agreement.

Waiting and Service Time Interactions. If the allocator were to emphasize waiting time T_W and neglect service time T_S, a low-throughput core might violate the SLA even with a near-zero T_W. Our approach guards against this case as the proxy communicates with the seller's side of the market to determine which architectures can run a given application with acceptable T_S. The same proxy mechanisms that profile machine scaling factors κ can profile application service time T_S.

If a given application could not possibly meet response times on an Atom, even with zero waiting time, the proxy will communicate this information to the seller side, thereby restricting the optimization and allocation when the market is cleared. For the PS and ¬PS applications in Section 4, the SLA allows for the 4.7x slowdown in T_S that PS experiences on the Atom, therefore Atoms can be allocated to PS.

By allocating to optimize waiting time and neglecting service time, the market precludes certain effects. End-to-end task response time is the sum of waiting time in the queue and service time at the machine. In theory, an allocation might reduce service time (e.g., faster machines) and consume that reduction with higher waiting time (e.g., fewer machines) without affecting response time.

These effects are modest. Consider two machines with service time $T_{S2} = kT_{S1}, k > 0$; machine 2 is slower. For equal response time, allocations need to provide service rates μ_1, μ_2 so that $\frac{1}{\mu_1 - \lambda} + T_{S1} = \frac{1}{\mu_2 - \lambda} + kT_{S1}$. Experimentally, we find accounting for waiting and service in an integrated way does not materially affect allocations μ_1, μ_2 for values of k found in diverse, general-purpose processor design spaces. Moreover, accounting for these second-order effects would force approximations elsewhere that would limit the market (i.e., MIP solver) and its ability to quickly and comprehensively search the space of heterogeneous hardware allocations.

7. QUALIFICATIONS AND ASSUMPTIONS

We assume users submit jobs that are comprised of tasks. For these tasks, we assume the 95th percentile response time determines service quality. This task stream model does not extend naturally to batch jobs with deadlines. Accommodating such workloads requires further research, especially since a single job offers no representative task to profile.

In our case studies, the k vectors collected from simulation do not account for performance degradation due to task co-location. Mars et al. [2011] propose a technique for mapping applications to machine groups such that co-located tasks incur minimal interference. With such schemes, contention is modest and profiling k vectors is straightforward. Without such schemes, more sophisticated profilers to accommodate contention effects will be needed.

We also assume M/M/1 queues are sufficient approximations for datacenter dynamics. M/M/1 models make three assumptions: (i) interarrival times are distributed exponentially; (ii) service times are distributed exponentially; (iii) a single server executes tasks. The first two assumptions break when the coefficient of variation $C_v = \sigma/\mu$ is large. However, we find C_v to be small for interarrival times. Although C_v increases with job and task heterogeneity, our framework uses different queues for different jobs to limit task heterogeneity. Thus, $C_v \approx 1$ for interarrival times. Moreover, interarrival times for university datacenter services and Google queries follow a near-exponential distribution [Meisner and Wenisch 2010; Meisner et al. 2011].

For service times, we compare an exponential distribution (M) against a general distribution (G). A standard queueing time approximation indicates that M/M/1 is close to M/G/1 when $C_v \approx 1$.[3] Assumptions of exponential distributions break when C_v is large (e.g., 20 or 100) [Gupta et al. 2010]. However, in our simulations of heterogeneous processor cores with more realistic hyperexponential distributions, we find that C_v for service times is often near 1 and well below 2, indicating M/M/1 is a good approximation for M/G/1, at least in expectation. Moreover, exponentially distributed service times have been applied in prior computing markets [Chase et al. 2001; Lubin et al. 2009].

Finally, the number of parallel servers (M/M/*k* versus M/M/1) affects the probability that a task must wait in the queue. We assume a single server whose capability (i.e., throughput) increases with the hardware allocation. However, with only one server, tasks queue with high probability. This assumption means our queueing time estimates are pessimistic, which lead to conservative hardware allocations where the market may over-provision resources. A more accurate model with parallel servers would only reduce queueing times and further improve our market's efficiency.

8. RELATED WORK

8.1. Power-Efficient Processors

Since the advent of chip multiprocessors, small and efficient processor cores have been studied for datacenters. Piranha, Niagara, and scale-out processors integrate many small cores for throughput [Barroso et al. 2000; Davis et al. 2005; Kongetira and Aingaran 2005; Lim et al. 2008; Lotfi-Kamran et al. 2012]. Server efficiency also benefits from re-purposing processors originally designed for mobile platforms.

Lim et al. [2008] and Janapa Reddi et al. [2010] deploy mobile-class processors for web search and other datacenter workloads to quantify performance and efficiency tradeoffs. Davis et al. [2005] evaluate the benefit of lightweight cores using equal area as the cost constraint. Keys et al. [2012] evaluate the energy improvement of using mobile, embedded, and low-power processors on single systems, and extrapolate these findings to a data center. Our work is driven by these developments in lightweight processors as our aim is to improve the energy proportionality of distributed systems through power-efficient cores.

The SM10000 server from Seamicro [2011] is a production system that uses 512 Intel Atom cores to reduce power utilization while maintaining support for all x86 applications. This system is an example of the importance of power-efficient systems,

[3] $E[W^{M/G/1}] \approx \frac{C_v^2+1}{2} E[W^{M/M/1}]$.

yet our work identifies the need for true energy efficiency (which includes performance) and thus makes use of heterogeneous processing cores.

The ability to satisfy complex requests using mobile cores is explored by Janapa Reddi et al. [2010] using Microsoft Bing search as an example of upcoming data center workloads in which the requests themselves require a certain degree of single-thread performance. One take-away from the work by Janapa Reddi et al. is that lower throughput cores, in this case Atoms, will still be able to maintain quality of service guarantees for simple tasks in future server applications.

But Janapa Reddi et al. also find some tasks may be more computationally intensive than today's typical workloads, ensuring a continuing role for server-class processors. These efforts illustrate small-core efficiency for memory- and I/O-bound tasks, and warn about performance penalties for more complex computation. Indeed, microarchitecture increasingly affects datacenter computation [Ferdman et al. 2012]. Our market is a step toward managing heterogeneous microarchitectures in datacenters.

8.2. Heterogeneity

Our treatment of heterogeneity focuses on diverse core microarchitectures and their mix in datacenters. Prior work studied core heterogeneity in chip multiprocessors [Choudhary et al. 2011; Kumar et al. 2003, 2006; Lee and Brooks 2007; Li et al. 2011] but does not identify the optimal number of cores for each type in a large system as we do. Other studies accommodate differences in serial and parallel code portions [Hill and Marty 2008; Suleman et al. 2009] or devote an efficient core to the operating system [Mogul et al. 2008]. In contrast, we consider a more general mix of datacenter computation.

Prior work in heterogeneous datacenters studied high-performance processors from different design generations or running at different clock frequencies [Mars et al. 2011; Nathuji et al. 2007]. In contrast, our heterogeneous cores occupy very different corners of the design space. Efficiency gains are larger but so is performance risk. Mitigating risk, we make novel contributions in coordinating core design, core mix, and resource allocation.

In distributed systems and grid/cloud computing, prior work emphasized virtual machine (VM) and/or software heterogeneity. CloudSim simulates federated datacenters with local, shared, and public VMs that might differ in core count or memory capacity [Amazon 2009; Calheiros and Buyya 2011; Vecchiola et al. 2012]. And prior work matched heterogeneous software demands (e.g., from Hadoop tasks) with heterogeneous VMs [Ghodsi et al. 2011; Lee et al. 2011]. Such work occupies a different abstraction layer, neglects the processor microarchitecture, and complements this work.

This article draws on prior measurements in energy proportionality and power modes, which are subjects of on-going research. Barroso et al. motivate component-level energy proportionality [Barroso and Hölzle 2007] while Meisner et al. [2011, 2009] study server-level sleep modes. Continuing research in power modes would benefit economic mechanisms for computational resource management.

8.3. Resource Allocation

Research in allocating resources for a data center, grid, or other form of distributed system has dealt with satisfying demand within the bounds of different constraints, such as response time, execution time of the resource manager, heterogeneity in the resources, and most recently power efficiency.

Early computational economies focused on maximizing performance in shared, distributed systems [Ferguson et al. 1996; Ibaraki and Katoh 1988; Sutherland 1968; Waldspurger et al. 1992]. Resource allocators based on market mechanisms are attractive because they can respond to varying levels of demand and provide a bidding mechanism that specifies resource requirements and/or preferences. The early work of Ferguson et al. [1996] and Ibaraki and Katoh [1988] introduce the relationship between computational resources and economic models. Stonebraker et al. [1996] implement a resource allocator that uses economic principles for a distributed database system in which traditional cost-based models encounter scalability challenges.

Lai et al. [2005] propose Tycoon, a market-based resource allocator, for a distributed system due to its low-latency decisions and its ability to grant users the ability to differentiate the importance of jobs. As opposed to previous work where market mechanisms were used for resource allocation, Tycoon does not require the users to participate in the actual bidding of resources, and instead users provide a priority for work that is used by the system to automate the bidding process. Our work does not address the problem of defining priorities using economic principles.

There is also research in observing bidding behavior in market-inspired resource management. Byde [2006] explores bidding strategies in a distributed computation environment where compute agents bid for resources on behalf of a CGI application, as well as applying genetic algorithms to the development of these bidding agents [Byde et al. 2003]. Senevirante et al. show the opportunity for improving bidding decisions based on the quality of predicting the computational cost of a job [Seneviratne and Levy 2010]. Our work does not focus as closely on the actual bidding mechanism and would thus benefit from research that improves the fairness or complexity of an auctioning system.

These systems projects all exploit utility mechanisms that can adapt to incoming computational demands. Broberg et al. [2007] provide a survey of market-inspired resource allocation techniques, and identify the benefits and limitations that much of the research encounters. Broberg et al. identifies the growing popularity of flexible market mechanisms over traditional resource allocators. But managing the resources becomes more difficult when resources are scarce or allocations are made at too fine/coarse time intervals. Byde [2002] discuss the importance of utility modeling for the effectiveness of any market mechanism.

On the other hand, utility based systems are subject to the truthfulness of the bidding agents. Bellagio attempts to address this limitation with a federated, market-based resource allocator for a distributed system of heterogeneous resources [Auyoung et al. 2004]. Bellagio provides better decisions during peak utilization by providing motivation for users to be truthful when prioritizing their jobs. Our work does not evaluate fairness guarantees nor identify phenomena that result in undesirable allocation, but these are directions of interest within the goal of allocating for energy efficiency.

Chase et al. [2001] extended many economic mechanisms to account for energy costs. Lubin et al. [2009] further accommodated dynamic voltage/frequency scaling in datacenter markets. This prior work is agnostic of microarchitectural differences and their effect on instruction-level parallelism. Addressing this limitation, we present a multi-agent market that navigates nonfungible processor cycles.

Early mechanisms relied on greedy solvers, allocating cores to tasks in their queued order and provisioning heterogeneous cores in a deterministic fashion (e.g., low-power cores first) [Garg et al. 2011; Nathuji et al. 2007; Rusu et al. 2006]. Both Chase and Lubin show greedy solvers are less effective than markets for improving service time and reducing cost. Like Lubin et al. [2009], we use a mixed integer program to find exactly optimal allocations, but approximate methods like gradient ascent [Chase et al. 2001; Mars et al. 2011] may also apply.

We optimize welfare and neglect fairness, which is increasingly important in federated clouds. Dominant resource fairness accommodates heterogeneous demands for multiple, complementary resources (e.g., processors and memory) in a shared datacenter [Ghodsi et al. 2011]. However, maximizing welfare and fairness in this setting are mutually exclusive [Parkes et al. 2012]. Navigating conflicting optimization objectives is important future work.

8.4. Profiling

Obtaining application preferences is trivial if users explicitly request particular hardware resources. Clouds offer a menu of heterogeneous virtual machine types, which differ in the number of compute units and memory capacity [Amazon 2009]. Similarly, recent efforts in datacenter management assume that users explicitly request processors and memory [Ghodsi et al. 2011; Hindman et al. 2011].

As heterogeneity increases, users or agents acting on their behalf rely on profiling tools that measure software sensitivity to hardware differences. These tools include gprof [Graham et al. 1982], VTune [Intel 2009], or OProfile [Open Source 2010]. With a combined analysis of software call graphs and hardware performance counters, agents can scale factors in the architecture performance equation. At datacenter scale, profiling every application on every node is infeasible and sampling is required. For example, the Google-Wide Profiling infrastructure periodically activates profilers on randomly selected machines and collects results for integrated analysis [Ren et al. 2010]. If these samples are drawn from diverse hardware-software pairings, more sophisticated profile analysis is required.

Given samples, inferred statistical machine learning models might predict scaling factors as a function of software characteristics and hardware parameters [Wu and Lee 2012]. Such models might be trained with profile databases, like Google's, to produce scaling factors. Such a capability requires integrating two bodies of related work in microarchitecture-independent software characteristics and statistical inference [Eeckhout et al. 2003; Lee and Brooks 2006].

9. CONCLUSIONS AND FUTURE DIRECTIONS

Collectively, our results motivate new directions in heterogeneous system design and management. Within datacenters, we find opportunities to mix server- and mobile-class processors to increase welfare while reducing energy cost. Architects may design heterogeneous systems but they cannot ignore their deployment. Market mechanisms are well suited to allocating heterogeneous resources to diverse users. As we continue to build bridges between computer architecture and economic and multi-agent systems, enhancing allocation procedures with greater architectural insight is imperative.

REFERENCES

Yuvraj Agarwal, Steve Hodges, Ranveer Chandra, James Scott, Paramvir Bahl, and Rajesh Gupta. 2009. Somniloquy: Augmenting network interfaces to reduce PC energy usage. In *Proceedings of the 6th Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, Berkeley, CA, 365–380.

Amazon. 2009. Elastic cloud computing. http://aws.amazon.com/ec2/.

David G. Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, and Vijay Vasudevan. 2009. FAWN: A fast array of wimpy nodes. In *Proceedings of the 22nd Symposium on Operating Systems Principles (SOSP)*. ACM, New York, 1–14.

Anonymous. 2012. Space Invaders. *The Economist*.

Alvin Auyoung, Brent N. Chun, Alex C. Snoeren, and Amin Vahdat. 2004. Resource allocation in federated distributed computing infrastructures. In *Proceedings of the 1st Workshop on Operating System and Architectural Support for the On-Demand IT Infrastructure*. 1–10.

Luiz André Barroso, Kourosh Gharachorloo, Robert McNamara, Andreas Nowatzyk, Shaz Qadeer, Barton Sano, Scott Smith, Robert Stets, and Ben Verghese. 2000. Piranha: A scalable architecture based on single-chip multiprocessing. In *Proceedings of the 27th International Symposium on Computer Architecture (ISCA)*. ACM, New York, 282–293.

Luiz André Barroso and Urs Hölzle. 2007. The case for energy-proportional computing. *IEEE Comput.* 40, 12, 33–37.

Luiz André Barroso and Urs Hölzle. 2009. The datacenter as a computer. In *Synthesis Lectures on Computer Architecture*.

Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 simulator. *SIGARCH Comput. Archit. News* 39, 2, 1–7.

James Broberg, Srikumar Venugopal, and Rajkumar Buyya. 2007. Market-oriented grids and utility computing: The state-of-the-art and future directions. *J. Grid Comput.* 6, 3, 255–270.

Andrew Byde. 2002. Applying evolutionary game theory to auction mechanism design. In *Proceedings of the 4th ACM Conference on Electronic Commerce*. ACM, New York, 192–193.

Andrew Byde. 2006. A comparison between mechanisms for sequential compute resource auctions. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. ACM, New York, 1199–1201.

Andrew Byde, Mathias Sallé, and Claudio Bartolini. 2003. Market-based resource allocation for utility data centers. Tech. Rep.

Rodrigo N. Calhieros, Rajid Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. 2011. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw.: Practice and Exper.* 41, 23–50.

Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin M. Vahdat, and Ronald P. Doyle. 2001. Managing energy and server resources in hosting centers. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*. ACM, New York, 103–116.

Niket K. Choudhary, Salil V. Wadhavkar, Tanmay A. Shah, Hiran Mayukh, Jayneel Gandhi, Brandon H. Dwiel, Sandeep Navada, Hashem H. Najaf-abadi, and Eric Rotenberg. 2011. FabScalar: Composing synthesizable RTL designs of arbitrary cores within a canonical superscalar template. In *Proceedings of the 38th International Symposium on Computer Architecture (ISCA)*. ACM, New York, 11–22.

Rachel Courtland. 2012. The battle between ARM and Intel gets real. *IEEE Spectrum*.

John D. Davis, James Laudon, and Kunle Olukotun. 2005. Maximizing CMP throughput with mediocre cores. In *Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE Computer Society, Los Alamitos, CA, 51–62.

Lieven Eeckhout, Sebastien Nussbaum, James E. Smith, and Koen DeBosschere. 2003. Statistical simulation: Adding efficiency to the computer designer's toolbox. *IEEE Micro* 23, 5, 26–38.

Facebook. 2011. *More effective computing*. Tech. Rep.

Michael Ferdman, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, Anastasia Ailamaki, and Babak Falsafi. 2012. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, New York, 37–48.

Donald F. Ferguson, Christos Nikolaou, Jakka Sairamesh, and Yechiam Yemini. 1996. Economic models for allocating resources in computer systems. In *Market-Based Control*, World Scientific Publishing Co., Inc., River Edge, NJ, 156–183.

Anshul Gandhi, Mor Harchol-Balter, and Michael A. Kozuch. 2011. The case for sleep states in servers. In *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*. ACM, New York, 2:1–2:5.

Siddharth Garg, Shreyas Sundaram, and Hiren D. Patel. 2011. Robust heterogeneous data center design: A principled approach. *SIGMETRICS Perform. Eval. Rev.* 39, 3, 28–30.

Varghese George, Sanjeev Jahagirdar, Chao Tong, K. Smits, Satish Damaraju, Scott Siers, Ves Naydenov, Tanveer Khondker, Sanjib Sarkar, and Puneet Singh. 2007. Penryn: 45-nm next generation intel core 2 processor. In *Proceedings of the Asian Solid-State Circuits Conference (ASSCC)*. IEEE, Los Alamitos, CA, 14–17.

Gianfranco Gerosa, Steve Curtis, Micahel D'Addeo, Bo Jiang, Belliappa Kuttanna, Feroze Merchant, Bina Patel, Mohammed Taufique, and Haytham Samarchi. 2009. A sub-2 W low power IA processor for mobile internet devices in 45 nm high-k metal gate CMOS. *IEEE J. Solid-State Circ.* 44, 1, 73–82.

Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant resource fairness: Fair allocation of multiple resource types. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI)*. USENIX Association, Berkeley, CA, 24.

Susan L. Graham, Peter B. Kessler, and Marshall K. Mckusick. 1982. Gprof: A call graph execution profiler. In *Proceedings of the SIGPLAN Symposium on Compiler Construction (CC)*. ACM, New York, 120–126.

Boris Grot, Damien Hardy, Pejman Lotfi-Kamran, and Babak Falsafi. 2012. Optimizing datacenter TCO with scale-out processors. *IEEE Micro* 32, 5, 52–63.

Brian Guenter, Navendu Jain, and Charles Williams. 2011. Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. In *Proceedings of the 30th International Conference on Computer Communications (INFOCOM)*. 1332–1340.

Marisabel Guevara, Benjamin Lubin, and Benjamin C. Lee. 2013. Navigating heterogeneous processors with market mechanisms. In *Proceedings of the 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Los Alamitos, CA, 95–106.

Varun Gupta, Mor Harchol-Balter, J. G. Dai, and B. Zwart. 2010. On the inapproximability of M/G/k. *Queue. Syst. Theory Appl.* 64, 1, 5–48.

Mark Hill and Michael Marty. 2008. Amdahl's Law in the multi-core era. *IEEE Computer* 41, 7, 33–38.

Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI)*. USENIX Association, Berkeley, CA, 22.

Mark Horowitz, Elad Alon, Dinesh Patil, Samuel Naffziger, Rajesh Kumar, and Kerry Bernstein. 2005. Scaling, power, and the future of CMOS. In *International Electron Devices Meeting Technical Digest (IEDM)*. IEEE, Los Alamitos, CA, 7–15.

Toshihide Ibaraki and Naoki Katoh. 1988. *Resource allocation problems: Algorithmic Approaches*. Vol. 45, MIT Press, Cambridge, MA.

Intel. 2009. VTune. http://software.intel.com/en-us/intel-vtune.

Intel. 2011. *Intel 64 and IA-32 Architectures Software Developers Manual*. Intel.

Vijay Janapa Reddi, Benjamin C. Lee, Trishul Chilimbi, and Kushagra Vaid. 2010. Web search using mobile cores: quantifying and mitigating the price of efficiency. In *Proceedings of the 37th International Symposium on Computer Architecture (ISCA)*. ACM, New York, 314–325.

Laura Keys, Suzanne Rivoire, and John D. Davis. 2012. The search for energy-efficient building blocks for the data center. In *Proceedings of the International Conference on Computer Architecture*. Springer-Verlag, Berlin, 172–182.

Poonacha Kongetira and Kathirgamar Aingaran. 2005. Niagara: A 32-way multithreaded sparc processor. *IEEE Micro* 25, 2, 21–29.

Rakesh Kumar, Keith I. Farkas, Norman P. Jouppi, Parthasarathy Ranganathan, and Dean M. Tullsen. 2003. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *Proceedings of the 36th International Symposium on Microarchitecture (MICRO)*. IEEE Computer Society, Los Alamitos, CA, 81.

Rakesh Kumar, Dean M. Tullsen, and Norman P. Jouppi. 2006. Core architecture optimization for heterogeneous chip multiprocessors. In *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. ACM, New York, 23–32.

Kevin Lai, Lars Rasmusson, Eytan Adar, Li Zhang, and Bernardo A. Huberman. 2005. Tycoon: An implementation of a distributed, market-based resource allocation system. *Multiagent Grid Syst.* 1, 3, 169–182.

Benjamin C. Lee and David M. Brooks. 2006. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, New York, 185–194.

Benjamin C. Lee and David M. Brooks. 2007. In *Proceedings of the 13th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, Los Alamitos, CA, 340–351.

Gunho Lee, Byung-Gon Chun, and H. Katz. 2011. Heterogeneity-aware resource allocation and scheduling in the cloud. In *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing*. USENIX Association, Berkeley, CA, 4–4.

Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd International Symposium on Microarchitecture (MICRO)*. ACM, New York, 469–480.

Sheng Li, Kevin Lim, Paolo Faraboschi, Jichuan Chang, Parthasarathy Ranganathan, and Norman P. Jouppi. 2011. System-level integrated server architectures for scale-out datacenters. In *Proceedings of the 44th International Symposium on Microarchitecture (MICRO)*. ACM, New York, 260–271.

Kevin Lim, Parthasarathy Ranganathan, Jichuan Chang, Chandrakant Patel, Trevor Mudge, and Steven Reinhardt. 2008. Understanding and designing new server architectures for emerging warehouse-computing environments. In *Proceedings of the 35th International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, Los Almitos, CA, 315–326.

Pejman Lotfi-Kamran, Boris Grot, Michael Ferdman, Stavros Volos, Onur Kocberber, Javier Picorel, Almutaz Adileh, Djordje Jevdjic, Sachin Idgunji, Emre Ozer, and Babak Falsafi. 2012. Scale-out processors. In *Proceedings of the 39th International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, Los Alamitos, CA, 500–511.

Benjamin Lubin, Jeffrey O. Kephart, Rajarshi Das, and David C. Parkes. 2009. Expressive power-based resource allocation for data centers. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence (IJCAI)*. Morgan-Kaufmann Publishers Inc., San Francisco, CA, 1451–1456.

Krishna T. Malladi, Benjamin C. Lee, Frank A. Nothaft, Christos Kozyrakis, Karthika Periyathambi, and Mark Horowitz. 2012. Towards energy-proportional datacenter memory with mobile DRAM. In *Proceedings of the 39th International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, Los Alamitos, CA, 37–48.

Jason Mars, Lingjia Tang, and Robert Hundt. 2011. Heterogeneity in Homogeneous Warehouse-Scale Computers: A Performance Opportunity. *IEEE Comput. Archit. Lett.* 10, 2, 29–32.

David Meisner, Brian T. Gold, and Thomas F. Wenisch. 2009. PowerNap: Eliminating server idle power. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, New York, 205–216.

David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. 2011. Power management of online data-intensive services. In *Proceedings of the 38th International Symposium on Computer Architecture (ISCA)*. ACM, New York, 319–330.

David Meisner and Thomas F Wenisch. 2010. Stochastic queuing simulation for data center workloads. In *Proceedings of the Workshop on Energy-Efficient Design*.

Jeffrey Mogul, Jayaram Mudigonda, Nathan Binkert, Parthasarathy Ranganathan, and Vanish Talwar. 2008. Using asymmetric single-ISA CMPs to save energy on operating systems. *IEEE Computer* 28, 3, 26–41.

Moor Insights and Strategy. 2013. *HP Moonshot: An accelerator for hyperscale workloads*. Tech. Rep.

Ripal Nathuji, Canturk Isci, and Eugene Gorbatov. 2007. Exploiting platform heterogeneity for power efficient data centers. In *Proceedings of the 4th International Conference on Autonomous Computing (ICAC)*. IEEE, Los Alamitos, CA, 5.

Open Source. 2010. OProfile. http://oprofile.sourceforge.net.

John Ousterhout, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, Guru Parulkar, Mendel Rosenblum, Stephen M. Rumble, Eric Stratmann, and Ryan Stutsman. 2010. The case for RAMClouds: Scalable high-performance storage entirely in DRAM. *SIGOPS Oper. Syst. Rev.* 43, 4, 92–105.

David C. Parkes, Ariel D. Procaccia, and Nisarg Shah. 2012. Beyond dominant resource fairness: Extensions, limitations, and indivisibilities. In *Proceedings of the 13th Conference on Electronic Commerce (EC)*. ACM, New York, 808–825.

Aashish Phansalkar, Ajay Joshi, and Lizy K. John. 2007. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite. In *Proceedings of the 34th International Symposium on Computer Architecture (ISCA)*. ACM, New York, 412–423.

Asfandyar Qureshi, Rick Weber, Hari Balakrishnan, John Guttag, and Bruce Maggs. 2009. Cutting the electric bill for internet-scale systems. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication (SIGCOMM)*. ACM, New York, 123–134.

Gang Ren, Eric Tune, Tipp Moseley, Yixin Shi, Silvius Rus, and Robert Hundt. 2010. Google-wide profiling: A continuous profiling infrastructure for data centers. *IEEE Micro* 30, 4, 65–79.

Cosmin Rusu, Alexandre Ferreira, Claudio Scordino, and Aaron Watson. 2006. Energy-efficient real-time heterogeneous server clusters. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE Computer Society, Los Alamitos, CA, 418–428.

Seamicro. 2011. SeaMicro Introduces the SM10000-64HD.

Sena Seneviratne and David C Levy. 2010. Cost profile prediction for grid computing. *Concurr. Computat. Practice Experi.* 22, 1 107–142.

Michael Stonebraker, Paul M. Aoki, Witold Litwin, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Staelin, and Andrew Yu. 1996. Mariposa: A wide-area distributed database system. *VLDB J.* 5, 1, 048–063.

M. Aater Suleman, Onur Mutlu, Moinuddin K. Qureshi, and Yale N. Patt. 2009. Accelerating critical section execution with asymmetric multi-core architectures. In *Proceedings of the 14th International Conference* *on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, New York, 253–264.

Ivan E. Sutherland. 1968. A futures market in computer time. *Commun. ACM* 11, 6, 449–451.

U.S. Environmental Protection Agency. 2007. Report to Congress on Server and Data Center Energy Efficiency.

Christian Vecchiola, Rodrigo N. Calheiros, Dileban Karunamoorthy, and Rajkumar Buyya. 2012. Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka. *Future Gen. Comput. Syst.* 28, 1, 58–65.

Carl A. Waldspurger, Tad Hogg, Bernardo A. Huberman, Jeffrey O. Kephart, and W. Scott Stornetta. 1992. Spawn: A Distributed Computational Economy. *IEEE Trans. Softw. Eng.* 18, 2, 103–117.

Weidan Wu and Benjamin C. Lee. 2012. Inferred Models for Dynamic and Sparse Hardware-Software Spaces. In *Proceedings of the 45th International Symposium on Microarchitecture (MICRO)*. IEEE Computer Society, Los Alamitos, CA, 413–424.

Doe Hyun Yoon, Jichuan Chang, Naveen Muralimanohar, and Parthasarathy Ranganathan. 2012. BOOM: enabling mobile memory based low-power server DIMMs. In *Proceedings of the 39th International Symposium on Computer Architecture (ISCA)*. IEEE Computer Society, Los Alamitos, CA, 25–36.

Received January 2013; accepted May 2013