Navigating Heterogeneous Processors with Market Mechanisms

Marisabel Guevara
Duke University
mg@cs.duke.edu

Benjamin Lubin
Boston University
blubin@bu.edu

Benjamin C. Lee
Duke University
benjamin.c.lee@duke.edu

Abstract

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.

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 [28].

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 [3, 59]. 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 [32]. 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 [61] and more recent markets [44, 10] 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 (§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 (§3).** We develop a market that manages resources and navigates performance-efficiency trade-offs 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 (§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 welfare and reduces energy. Moreover, 30% of tasks incur response time violations in homogeneous systems but not in heterogeneous ones.
- **Application to Further Heterogeneity (§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× relative to a homogeneous system.

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 §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 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 paper we trade-off efficiency and performance by considering two existing processors: the mobile-class Atom and the server-class Xeon (§4). Additionally, we design and evaluate up to twelve cores that lie along the efficiency-vs-performance spectrum (§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.

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

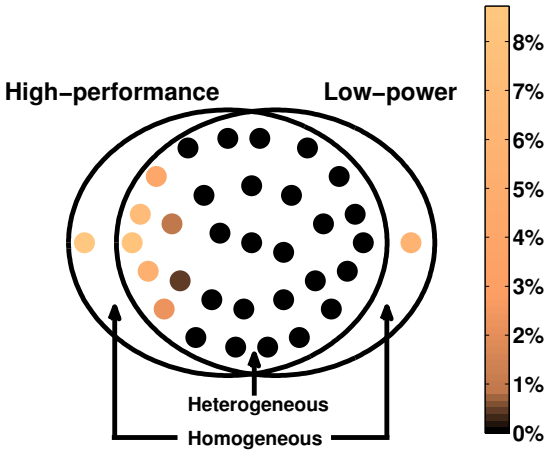


Figure 1: Venn diagram that illustrates a datacenter design space for low-power and high-performance processors; the intersection harbors heterogeneous design options. Colored points depict QoS violations.

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 trade-off 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 [32]. Small cores are well balanced for the modest computational intensity of simple web search queries, distributed memory caching, and key-value stores [3, 32, 53]. Such research in

unconventional datacenter hardware has spurred broader commercial interest [4, 12] and analogous research in other technologies, such as DRAM [45, 66].

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 [61], 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 in 1992 [64], by Chase in 2001 [10], and Lubin in 2009 [44]. 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:

$$\frac{\texttt{Tasks}}{\texttt{Sec}} = \frac{\texttt{Cycles}}{\texttt{Sec}} \times \frac{\texttt{Insts}}{\texttt{Cycle}} \times \frac{\texttt{Tasks}}{\texttt{Inst}} \qquad (1)$$

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 re-compiled, 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× 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.

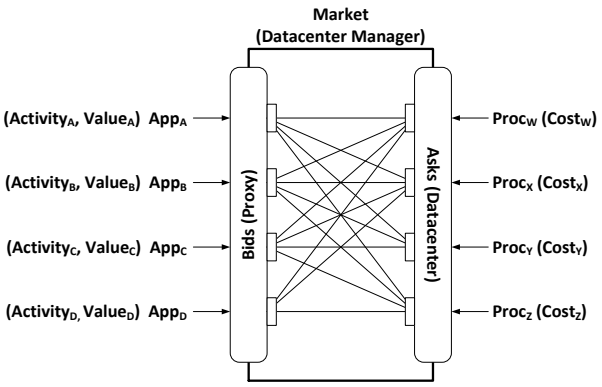


Figure 2: Market Overview.

To calculate scaling factors, we rely on application profiling data. In this paper, we assume that existing profilers provide this data (see §7 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. manage homogeneous servers by asking users to bid on performance [10]. Lubin et al. extend this formulation with processor frequency scaling, a novel modeling and bidding language, and a mixed integer program to clear the market [44]. We start from the latter market, which assumes fungible processor cycles, and extend it to account for architectural heterogeneity.

Figure 2 illustrates such market mechanism with 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 non-fungible 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.

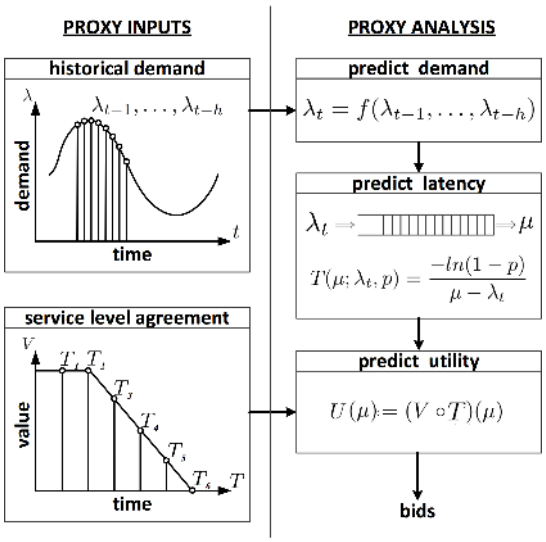


Figure 3: Proxy Bids

3.1. Proxies and Value Analysis

In this paper, we present extensions for our novel setting, embedding 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 [44].

Estimate Task Service Rate. To serve these arriving tasks, an optimizer searches an allocation space of heterogeneous cores. Prior efforts assume fungible processor cycles [10, 44], 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, sub-optimal 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 inter-arrival and service times are low; see §6 for details. We estimate latency percentiles with Equation (2) and use the 95$^{\text{th}}$ percentile as the figure of merit, $p = 0.95$.

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

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

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 (Equations (4)–(6)).

$$\text{datacenter profiler} \quad | \quad \mathbf{P_a} : \{hw_a\} \rightarrow \{service\ rate\} \qquad (4)$$

$$\text{datacenter queues} \quad | \quad \mathbf{T} : \{service\ rate\} \rightarrow \{latency\} \qquad (5)$$

$$\text{user value} \quad | \quad \mathbf{V} : \{latency\} \rightarrow \{dollars\} \qquad (6)$$

$$\text{market welfare} \quad | \quad \mathbf{W} = \sum_{a \in A} \Big(\mathbf{V} \circ \mathbf{T} \circ \mathbf{P_a}(hw_a) \Big) - \mathbf{C}(hw) \qquad (7)$$

A profile $\mathbf{P_a}$ maps proxy $\mathbf{a}$'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 (Equation (3)). 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 [47, 48]. For example, we model server power modes as three possible states: active, idle (but in an active power mode), and sleep.

In Equation (8), the datacenter accounts for the number of servers (n^*) in each mode and power (P^*) dissipated over the allocation time period (Δ) [44]. Servers that transition between

	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 1: Architecture parameters with for Xeons, Atoms [20, 21, 31].

	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	

Table 2: Power modes and parameters [32].

	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	$\kappa_{\mathbf{X}} = 1.0$ $\kappa_{\mathbf{A}} = 0.33$	$\kappa_{\mathbf{X}} = 1.0$ $\kappa_{\mathbf{A}} = 1.0$

Table 3: Application Characteristics. For a task stream, T is 95^{th} percentile queueing time.

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

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 [6].

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 (Equation (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 [44]. 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

Our evaluation uses an in-house datacenter simulator. A proxy predicts demand from history, predicts latency using a closed-form response time model, and constructs a bid. The framework then clears the market, identifying welfare-maximizing allocations by invoking CPLEX to solve a MIP. The MIP solution is an allocation for the next 10-minute interval. For this interval, the simulator uses response time models, cost models, application demand, and the allocation to compute value produced and energy consumed. The simulator does exactly what a real cluster manager would do, providing hints at future prototype performance. The simulator does not perform per-task microarchitectural simulation, which is prohibitively expensive.

Tables 1–2 summarize platform parameters. 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 [32, 59]. Xeon core power is 10× Atom core power. Servers transition from active to sleep mode in 6 secs and from sleep to active in 8 secs, powering off everything but the memory and network interface [1, 18]. Power usage effectiveness (PUE) for the datacenter is 1.6, an average of industry standards [15, 62]. Energy costs are $0.07 per kWh, an average of surveyed energy costs from prior work [56].

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 3 and illustrated in

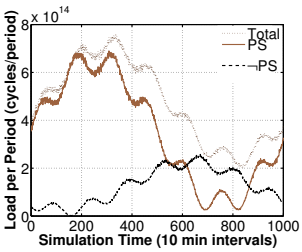
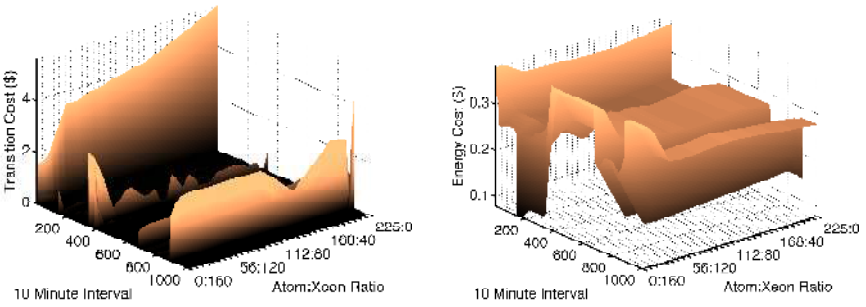
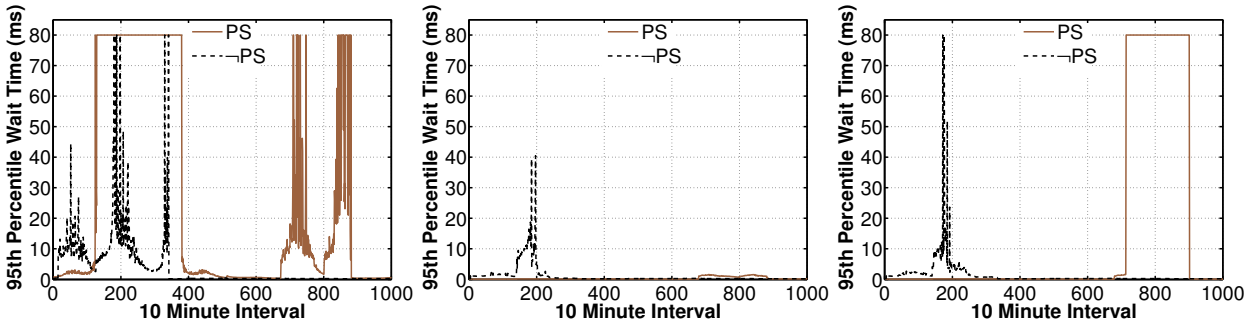


Figure 4: Demand for processor sensitive (PS) and insensitive (¬PS) applications.



(a) **Transition Cost** (b) **Energy**

Figure 5: Seller costs due to (a) energy and (b) transition penalty, as the ratio of Atom:Xeon processors varies. Energy cost corresponds closely to application behavior across datacenter configurations. Ridges in transition cost are due to a $0.05 penalty per transition that accounts for increased system wear-out [24].



(a) **0 Atoms::160 Xeons** (b) **147 Atoms::55 Xeons** (c) **225 Atoms::0 Xeons**

Figure 6: 95^{th} percentile waiting time for (a) only Xeons, (b) mix of Atoms and Xeons, and (c) only Atoms. Heterogeneous system (b) violates performance targets less often than homogeneous configurations (a), (c).

Figure 4. 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. 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 [47].

Applications specify value (SLA) as a function of 95^{th} 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.

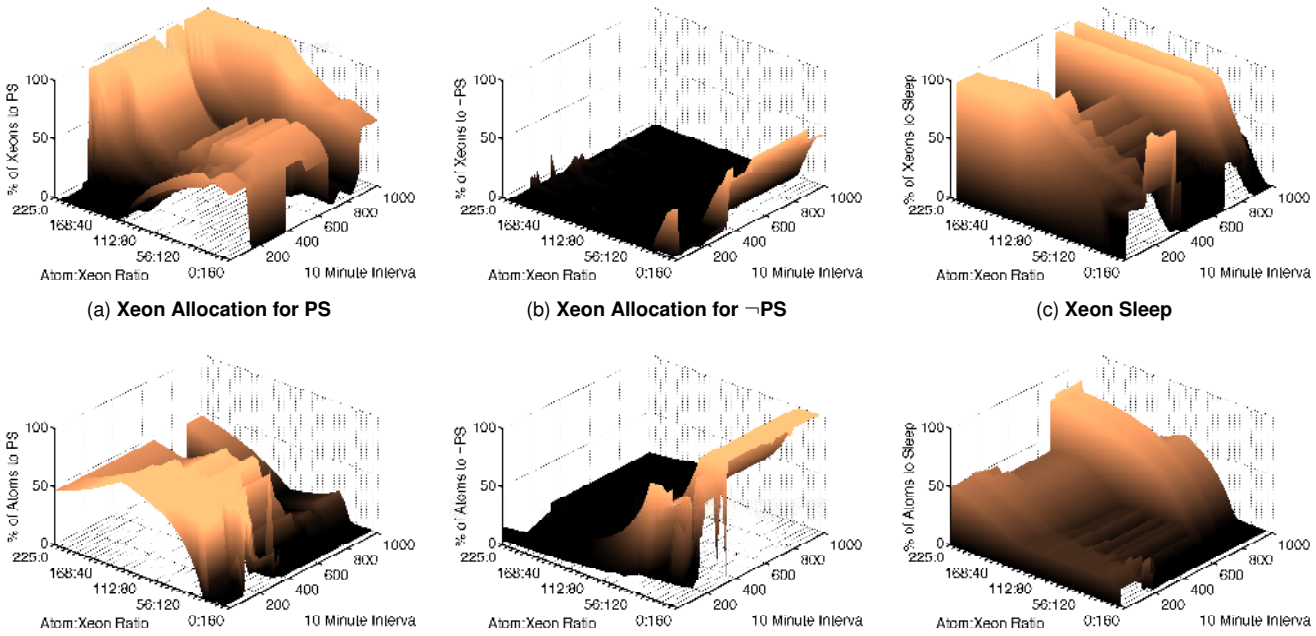
The architecture scaling factors κ are consistent with prior datacenter workload characterizations. Reddi et al. find $\frac{\texttt{Inst}}{\texttt{Cycle}}$ on Atom is 33% of that on Xeon for Microsoft Bing web search [32]. Lim et al. find performance on the mobile-class Intel Core 2 Turion is 34% of that on the Intel Xeon [42]. 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 [13, 34]. Andersen et al. propose embedded processors for distributed key-value store servers [3]. Servers that deliver Youtube-like traffic can run on small cores with negligible performance penalties [42]. Higher processor performance does not benefit such workloads. ¬PS applications are indifferent to Xeons and Atoms: $\kappa_X = \kappa_A = 1$.

4.3. Improving Welfare

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 6a see periods 130-380.



(a) **Xeon Allocation for PS** (b) **Xeon Allocation for ¬PS** (c) **Xeon Sleep**

(d) **Atom Allocation for PS** (e) **Atom Allocation for ¬PS** (f) **Atom Sleep**

Figure 7: Allocation measured in fraction of configured nodes as Atom:Xeon ratio varies. For example, a 50% Atom allocation in a A:X=168:40 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.

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 7a shows the market allocating a larger fraction of the remaining Xeons to PS thus improving its response time. Simultaneously, Figure 7b 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 5a, a charge is imposed for every transition to account for increased wear and reduced mean-time-to-failure as machines power-cycle [24].

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 causes a response time spike at the first load peak (period 200) but not the second (Figure 6c).

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 7f) and the fraction of Atoms allocated to PS and ¬PS decline (Figure 7d and Figure 7e, 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 5a, 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 7a and Figure 7d, respectively), decreasing 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 7a and Figure 7c, 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.

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 6b, this mix of Xeons and Atoms provide queueing times that are stable and far below the 80ms cut-off.

For contrast, consider Figure 6a and Figure 6c. 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 6). 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 [7]. 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 5b). 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.

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.

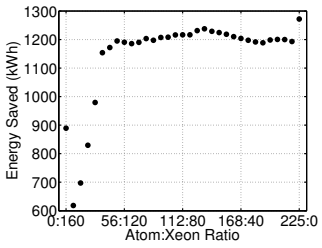


Figure 8: Energy saved from sleep modes.

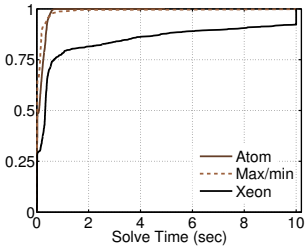


Figure 9: Solve time CDF across all periods.

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 [44].

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.

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 4. We simulate these designs with the gem5 cycle-accurate simulator in syscall emulation mode [8].

For this experiment, we consider the preferences of SPEC CPU2006 applications on heterogeneous processors. These

	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 4: Parameters for the twelve cores simulated in gem5.

	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

Table 5: Area and power estimates for four core types.

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

Table 6: Application characteristics.

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 6, which shares parameters from §4 where possible.

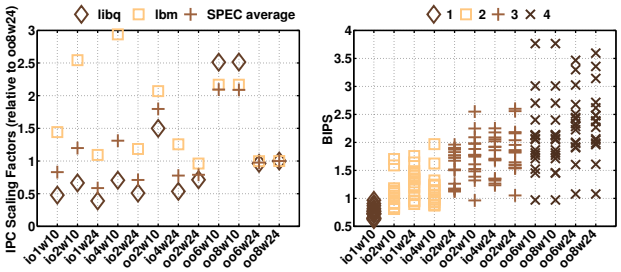
5.2. Architectural Preferences

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 specifing its performance for n applications. We cluster these vectors with multi-dimensional, hierarchical clustering [55]. 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 10b 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 5). 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; area and power are estimated with McPAT models [40], and calibrated to real Xeon and Atom measurements. We normalize silicon area since it is the primary determinant of a processor's marginal cost. We align server power with estimates from related work [32].

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.



(a) **IPC scaling factors** (b) **Core clustering k=4**

Figure 10: gem5 simulation results, cores on the horizontal axis are in order of increasing peak dynamic power.

5.3. Improving Service Quality

Increased processor diversity benefits service quality. Figure 11 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 10a).

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.

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

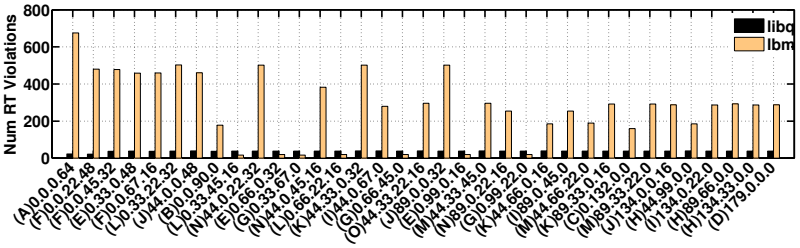


Figure 11: Number of 95th percentile waiting time violations. The horizontal axis indicates the number of servers of each type as the tuple [io1w10].[io1w24].[io4w24].[oo6w24]. Prepended letters mark the corresponding region in Figure 12.

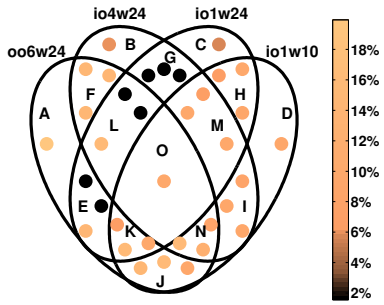


Figure 12: Sum of libq and lbm waiting time violations, shown on a Venn diagram.

lbm. For example, a system with 33 io1w24 cores and 67 io4w24 cores (00.33.67.0 in Figure 11) 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 12 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.

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 12 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 trade-offs. But related work in heterogeneous microarchitectures have illustrated diminishing marginal returns, which coincidentally arise as heterogeneity increases beyond four designs [38]. Moreover, datacenters with more than four core types may produce impractical capital and maintenance costs.

This complex design space and its sophisticated trade-offs 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. 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. [46] 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 straight-forward. 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) inter-arrival 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 inter-arrival times. Although C_v

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

increases with job and task heterogeneity, our framework uses different queues for different jobs to limit task heterogeneity. Thus, $C_v \approx 1$ for inter-arrival times. Moreover, inter-arrival times for university datacenter services and Google queries follow a near-exponential distribution [49, 47].

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) [25]. 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 [10, 44].

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.

7. Related Work

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 [5, 13, 34, 43]. Server efficiency also benefits from re-purposing processors originally designed for mobile platforms [32, 33, 42]. 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 [16]. Our market is a step toward managing heterogeneous microarchitectures in datacenters.

Heterogeneity. Our treatment of heterogeneity focuses on diverse core microarchitectures and their mix in datacenters. Prior work studied core heterogeneity in chip multiprocessors [11, 35, 36, 38, 41] 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 [26, 60] or devote an efficient core to the operating system [50]. 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 [46, 51]. 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 [2, 9, 63]. And prior work matched heterogeneous software demands (e.g., from Hadoop tasks) with heterogeneous VMs [22, 39]. Such work occupies a different abstraction layer, neglects the processor microarchitecture, and complements this work.

Resource Allocation. Early computational economies focused on maximizing performance in shared, distributed systems [17, 29, 61, 64]. Chase et al. extended these mechanisms to account for energy costs [10]. Lubin et al. further accommodated dynamic voltage/frequency scaling in datacenter markets [44]. 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 non-fungible processor cycles.

Prior studies 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) [19, 51, 58]. Both Chase and Lubin show greedy solvers are less effective than markets for improving service time and reducing cost. Like Lubin [44], we use a mixed integer program to find exactly optimal allocations, but approximate methods like gradient ascent [10, 46] 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 [22]. However, maximizing welfare and fairness in this setting are mutually exclusive [54]. Navigating conflicting optimization objectives is important future work.

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 [2]. Similarly, recent efforts in datacenter management assume that users explicitly request processors and memory [22, 27].

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 [23], VTune [30], or OProfile [52]. 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 [57].

Given samples, inferred statistical machine learning models might predict scaling factors as a function of software characteristics and hardware parameters [65]. Such models might be trained with profile databases, like Google's, to produce

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

scaling factors. Such a capability requires integrating two bodies of related work in microarchitecturally-independent software characteristics and statistical inference [14, 37].

8. Conclusion

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.

Acknowledgements

This work is supported, in part, by NSF grant CCF-1149252 (CAREER), a Google Faculty Research Award, and the Duke Wannamaker Foundation. This work is also supported by STARnet, a Semiconductor Research Corporation program, sponsored by MARCO and DARPA. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of these sponsors.

References

[1] Y. Agarwal et al. Somniloquy : Augmenting Network Interfaces to Reduce PC Energy Usage. In *NSDI*, 2009.
[2] Amazon. Elastic cloud computing. http://aws.amazon.com/ec2/.
[3] D. G. Andersen et al. FAWN : A Fast Array of Wimpy Nodes. In *SOSP*, 2009.
[4] Anonymous. Space invaders. *The Economist*, 2012.
[5] L. Barroso et al. Piranha: A Scalable Architecture Based On Single-Chip Multiprocessing. *ISCA*, 2000.
[6] L. A. Barroso and U. Hölzle. The Case for Energy-Proportional Computing. *Computer*, Dec. 2007.
[7] L. A. Barroso and U. Hölzle. The Datacenter as a Computer. *Synthesis Lectures on Computer Architecture*, Jan. 2009.
[8] N. Binkert et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1, Aug. 2011.
[9] R. Calheiros et al. CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 2011.
[10] J. S. Chase et al. Managing Energy and Server Resources in Hosting Centers. *SIGOPS Operating Systems Review*, 2001.
[11] N. K. Choudhary et al. FabScalar. In *ISCA*, 2011.
[12] R. Courtland. The battle between ARM and Intel gets real. *IEEE Spectrum*, 2012.
[13] J. Davis, J. Laudon, and K. Olukotun. Maximizing CMP Throughput With Mediocre Cores. *PACT*, 2005.
[14] L. Eeckhout, S. Nussbaum, J. Smith, and K. D. Bosschere. Statistical simulation: Adding efficiency to the computer designer's toolbox. *IEEE Micro*, 2003.
[15] Facebook. More Effective Computing. Technical report, 2011.
[16] M. Ferdman et al. Clearing the clouds. In *ASPLOS*, 2012.
[17] D. F. Ferguson et al. Economic models for allocating resources in computer systems. In *Market Based Control of Distributed Systems*. 1996.
[18] A. Gandhi et al. The Case for Sleep States in Servers. In *4th Workshop on Power-Aware Computing and Systems*, 2011.
[19] S. Garg, S. Sundaram, and H. Patel. Robust heterogeneous data center design: A principled approach. In *SIGMETRICS*, 2011.
[20] V. George et al. Penryn : 45-nm Next Generation Intel ® Core ™ 2 Processor. In *ISSCC*, 2007.
[21] G. Gerosa et al. A Sub-2 W Low Power IA Processor for Mobile Internet Devices in 45 nm High-k Metal Gate CMOS. *IEEE Journal of Solid-State Circuits*, Jan. 2009.
[22] A. Ghodsi et al. Dominant resource fairness: Fair allocation of multiple resource types. In *NSDI*, 2011.
[23] S. Graham et al. Gprof: A call graph execution profiler. In *CC*, 1982.
[24] B. Guenter, N. Jain, and C. Williams. Managing Cost, Performance, and Reliability Tradeoffs for Energy-Aware Server Provisioning. In *Information communications*, 2011.
[25] V. Gupta et al. On the inapproximability of M/G/k. *Queueing Systems: Theory and Applications*, 2010.
[26] M. Hill and M. Marty. Amdahl's Law in the multi-core era. *IEEE Computer*, 2008.
[27] B. Hindman et al. Mesos: A platform for fine-grained resource sharing in the data center. In *NSDI*, 2011.
[28] M. Horowitz et al. Scaling, power, and the future of CMOS. In *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*, pages 7 pp. –15, dec. 2005.
[29] T. Ibaraki and N. Katoh. *Resource allocation problems: Algorithmic Approaches*, volume 45. MIT Press, Cambridge, MA, USA, Jan. 1988.
[30] Intel. Vtune. http://software.intel.com/en-us/intel-vtune.
[31] Intel. Intel ® 64 and IA-32 Architectures Software Developer's Manual. Technical Report 326018, 2011.
[32] V. Janapa Reddi et al. Web Search Using Mobile Cores. In *ISCA*, 2010.
[33] L. Keys, S. Rivoire, and J. D. Davis. The Search for Energy-Efficient Building Blocks for the Data Center System. In *WEED*, 2010.
[34] P. Kongetira and K. Aingaran. Niagara: A 32-way multithreaded sparc processor. *Micro, IEEE*, 2005.
[35] R. Kumar et al. Single-ISA Heterogeneous Multi-Core Architectures. In *MICRO*, 2003.
[36] R. Kumar, D. M. Tullsen, and N. P. Jouppi. Core architecture optimization for heterogeneous chip multiprocessors. In *PACT*, page 23, 2006.
[37] B. Lee and D. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *ASPLOS*, 2006.
[38] B. C. Lee and D. M. Brooks. Illustrative design space studies with microarchitectural regression models. In *HPCA*, 2007.
[39] G. Lee, B.-G. Chun, and R. Katz. Heterogeneity-aware resource allocation and scheduling in the cloud. In *HotCloud*, 2011.
[40] S. Li et al. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*, 2009.
[41] S. Li et al. System-level integrated server architectures for scale-out datacenters. In *MICRO*, 2011.
[42] K. Lim et al. Understanding and Designing New Server Architectures for Emerging Warehouse-Computing Environments. In *ISCA*, 2008.
[43] P. Lotfi-Kamran et al. Scale-out processors. In *ISCA*, 2012.
[44] B. Lubin et al. Expressive power-based resource allocation for data centers. In *IJCAI*, 2009.
[45] K. Malladi et al. Towards energy-proportional datacenter memory with mobile DRAM. In *ISCA*, 2012.
[46] J. Mars, L. Tang, and R. Hundt. Heterogeneity in "homogeneous" warehouse-scale computers: A performance opportunity. *CAL*, 2011.
[47] D. Meisner et al. Power Management of Online Data-Intensive Services. In *ISCA*, 2011.
[48] D. Meisner, B. Gold, and T. Wenisch. PowerNap: eliminating server idle power. *ACM SIGPLAN Notices*, 44, 2009.
[49] D. Meisner and T. F. Wenisch. Stochastic Queuing Simulation for Data Center Workloads. In *WEERT*, 2010.
[50] J. Mogul et al. Using asymmetric Single-ISA CMPs to save energy on operating systems. *IEEE Computer*, 2008.
[51] R. Nathuji, C. Isci, and E. Gorbatov. Exploiting platform heterogeneity for power efficient data centers. In *ICAC*, 2007.
[52] Open Source. OProfile. http://oprofile.sourceforge.net.
[53] J. Ousterhout et al. The case for RAMCloud. *CACM*, 2011.
[54] D. Parkes, A. Procaccia, and N. Shah. Beyond dominant resource fairness: Extensions, limitations, and indivisibilities. In *EC*, 2012.
[55] A. Phansalkar, A. Joshi, and L. K. John. Analysis of redundancy and application balance in SPEC CPU 2006. In *ISCA*, 2007.
[56] A. Qureshi et al. Cutting the Electric Bill for Internet-Scale Systems. *SIGCOMM*, 2009.
[57] G. Ren et al. Google-wide profiling: A continuous profiling infrastructure for data centers. *IEEE Micro*, 2010.
[58] C. Rusu et al. Energy-efficient real-time heterogeneous server clusters. In *RTAS*, 2006.
[59] Seamicro. SeaMicro Introduces the SM10000-64HD, 2011.
[60] M. Suleman et al. Accelerating critical section execution with asymmetric multi-core architectures. In *ASPLOS*, 2009.
[61] I. Sutherland. A futures market in computer time. *CACM*, 1968.
[62] U.S. Environmental Protection Agency. Report to Congress on Server and Data Center Energy Efficiency. 2007.
[63] C. Vecchiola et al. Deadline-driven provisioning of resources for scientific applications in hybrid clouds with aneka. *FGCS*, 2012.
[64] C. Waldspurger et al. Spawn: A Distributed Computational Economy. *IEEE Transactions on Software Engineering*, 18, 1992.
[65] W. Wu and B. Lee. Inferred models for dynamic and sparse hardware-software spaces. In *MICRO*, 2012.
[66] D. Yoon et al. BOOM: Enabling mobile memory based low-power server DIMMs. In *ISCA*, 2012.