Coordinating the Design and Management of Heterogeneous Datacenter Resources

by

Marisabel Guevara

Department of Computer Science
Duke University

Date: ______________________

Approved:

____
Benjamin C. Lee, Supervisor

____
Jeffrey S. Chase

____
Benjamin Lubin

____
Bruce M. Maggs

____
Daniel J. Sorin

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2014

Abstract

Coordinating the Design and Management of Heterogeneous Datacenter Resources

by

Marisabel Guevara

Department of Computer Science
Duke University

Date: ______________________

Approved:

____
Benjamin C. Lee, Supervisor

____
Jeffrey S. Chase

____
Benjamin Lubin

____
Bruce M. Maggs

____
Daniel J. Sorin

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University
2014

Abstract

Heterogeneous design presents an opportunity to improve energy efficiency but raises a challenge in management. Whereas prior work separates the two, we coordinate heterogeneous design and management. We present a market-based resource allocation mechanism that navigates the performance and power trade-offs of heterogeneous architectures. Given this management framework, we explore a design space of heterogeneous processors and show a 12x reduction in response time violations when equipping a datacenter with three processor types over a homogeneous system that consumes the same power. To better understand trade-offs in large heterogeneous design spaces, we explore dozens of design strategies and present a risk taxonomy that classifies the reasons why a deployed system may underperform relative to design targets. We propose design strategies that explicitly mitigate risk, such as a strategy that minimizes the coefficient of variation in performance. In our experiments, we find that risk-aware design accounts for more than 70% of the strategies that produce systems with the best service quality. We also present a new datacenter management mechanism that fairly allocates processors to latency-sensitive applications. Tasks express value for performance using sophisticated piecewise-linear utility functions. With fairness in market allocations, we show how datacenters can mitigate envy amongst latency-sensitive users. We quantify the price of fairness and detail efficiency-fairness trade-offs. Finally, we extend the market to fairly allocate heterogeneous processors.

For Irene, Nelson, and Oscar – the greatest gifts in my life.

Contents

List of Tables

List of Figures

Acknowledgements

It is appropriate for the word "fulfillment" to be on the cover of this dissertation, as that is a word I would choose to describe my graduate school experience. Verbatim, the text on the cover reads: *Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University.* Though its appearance in that sentence is meant to describe this thesis as a requirement, my use of the word fulfillment to describe my graduate school experience is to express gratification. Fulfillment of this degree is a realization of a challenge, where intellect and dedication were both equally necessary to attain this goal. Many additional elements went into this dissertation, one of which was key to make this experience fulfilling. That is the element of human relationships, a combination of professional and personal connections that I wish to acknowledge.

I want to thank my research advisor, Dr. Benjamin Lee, for having been a great example of dedication. In the years that we have worked together, I have only seen him make full, never partial, investments in projects and students. This is a great trait, and I appreciate that I am one of the students that he invested in. I am also thankful to Dr. Lee for the time, encouragement, challenges, and the freedom he gave me in my work. Dr. Daniel Sorin contributed to my experience at Duke as the voice of wisdom, and I am deeply grateful for his advice. He served as a much-needed external opinion and it has elevated the quality of my work. I am also thankful to Dr.

Benjamin Lubin at Boston University for his enthusiasm and advice that helped me pursue the research reflected in all of the chapters of this thesis. Also elemental to this work has been Dr. Kevin Skadron at the University of Virginia, his mentorship formed the foundation of my approach to research. Another mentor to whom I am thankful is Dr. Thomas Schweiger, who ignited my curiosity in research before I even considered attending graduate school.

My experience at Duke also benefited from the support and enthusiasm of my colleagues. I am thankful for the tireless support provided by Dr. Blake Hechtman, who believed in each of my ideas and my work long enough to convince me that they were solid. I found a valuable friend and confidant in Luwa Matthews, to whom I am particularly thankful to for having made this experience a memorable one. Qiuyun Wang is another extraordinary friend and supporter, and I am thankful for each time that she asked for my advice as it pushed my professional growth. I thank all my labmates and friends at Duke for sharing the good times and the hard times of graduate school. I am also grateful for my ongoing friendship with Dr. Chris Gregg and Dr. Yousef Shaksheer, whose advice and solidarity have been unwavering since the moment we met. Lastly, the most important element to my rewarding experience in graduate school has been my family. Irene, Nelson, and Oscar are my lifelong blessings; I am indebted to each of you for pushing me to maintain grace, kindness, and passion throughout this journey.

1
Introduction

Energy efficiency has typically been a concern for embedded systems that are constrained by battery life. Since the end of Dennard scaling, however, peak power has become a primary constraint to the performance of general purpose systems [45]. Datacenters are warehouse-sized systems made up of hundreds of racks of servers. They power numerous web services including internet search engines, remote data storage, media streaming, and social networks. The compute load on datacenters is rapidly growing due to the recent trend towards ubiquitous computing and cloud services. This thesis aims to improve the energy efficiency of datacenters so that more computation can be achieved within a fixed power envelope.

Datacenters have traditionally relied on Moore's Law to increase compute capability across generations of processors. Given that improvements in single core performance have reached a stand-still, datacenters can instead scale-out the number of servers and processors in the system. This option is unattractive as it increases the power required by the datacenter. Recent studies have shown that datacenters already dissipate a non-trivial amount of power. Datacenters consume approximately 1.5% of total United States energy, the equivalent of 5.8M households in 2006 [97].

Global datacenter energy usage increased by 2$\times$ from 2000 to 2005, and an additional 1.6$\times$ from 2005 to 2010 [51]. If we could halve cloud computing energy, we would save the equivalent of the electricity consumption of the United Kingdom [42].

Recent work has shown that mobile technologies can be deployed in a datacenter setting to service cloud applications. Mobile processors and memory technologies have been optimized for power efficiency, whereas their server-class counterparts aim to maximize peak performance. For this reason, mobile cores use a fraction of the power of a server core, and are likely to degrade performance. Fortunately, the reduction in power far outweighs the performance loss. For example, web search can execute on Intel Atom cores that consume $\frac{1}{10}$th the power of Intel Xeon cores, yet only incur a 3$\times$ reduction in query throughput [84]. Mobile memory is another opportunity for efficiency. Recent work has shown that LPDDR2 consumes $\frac{1}{10}$th the power of DDR3 and reduces bandwidth by 2$\times$ when executing web search [67]. The power savings will remain constant regardless of the application. The performance degradation, however, varies based on the compute and memory intensity of each application. There are applications where the performance degradation of mobile technologies is trivial, yet there are also applications where the slow-downs are unacceptable. For this reason, deploying mobile technologies as a substitute to high-performance components is not a panacea.

Datacenters should be more energy efficient, yet application performance remains the foremost design target. Performance targets are the most strict for latency-sensitive applications, where the duration of each query has an impact on user experience. Web search is an example of this class of applications. A search engine specifies a service level agreement (SLA) that states a latency target for P^{th} percentile of its queries, where $P \geqslant 95\%$. A datacenter provides good service quality to web search if $P\%$ of search queries are serviced within the latency target specified by the SLA. If the datacenter cannot meet the SLA, the application reaps no benefit and is at risk of losing its user base. In contrast, throughput applications benefit from any cycles that a datacenter can provide. Even if this best-effort service is not ideal, throughput applications will make forward progress. In this thesis, we focus on datacenters that are designed to service latency-sensitive applications.

In this thesis, we demonstrate an alternative approach to increase the compute capability of a datacenter within a fixed power budget. Mobile and server technologies can be deployed side-by-side in a single heterogeneous system. Designing systems with heterogeneous components can qualitatively improve the energy efficiency of datacenters. By tailoring hardware architectures to the application mix, a datacenter can perform the same amount of work with less power, or it can service more applications within the same power budget.

Along with the opportunity to build a more energy efficient system, however, heterogeneity also introduces a challenge to resource management and design. Applications benefit from each of the heterogeneous resources in a unique way. A resource manager must consider these preferences in order to provide good service quality. Thus far, efforts to exploit the opportunity and address the challenge have been uncoordinated. This separation exposes software to performance risk or leaves hardware energy efficiency unexploited.

Coordinated design and management is crucial for the adoption of heterogeneity in systems research. Prior work has been conservative in its treatment of hardware diversity by studying modest heterogeneity across hardware generations [69, 76]. In contrast, our work provides solutions to the challenge of managing and designing datacenters that are *heterogeneous by design.* Aggressively heterogeneous systems require robust resource management frameworks that avoid catastrophic allocations. Without greater coordination between the design and management of heterogeneous systems, there is a risk that datacenters become prohibitively complicated to manage, or allocators may be ineffective for the most efficient forms of heterogeneity.

To address these challenges, we coordinate the design of heterogeneous architectures with recent advances in multi-agent systems. In Chapter 2, we embed microarchitectural insight into a market mechanism that serves as a resource allocator. We propose a novel approach to selecting heterogeneous processors in Chapter 3 from a design space such that the performance uncertainty of the system is minimized. In Chapter 4, we enforce fairness in the allocations produced by the market and experimentally show that the reduction in throughput when fairness is guaranteed is most significant when resources are scarce. Next, we describe the granularity of heterogeneity that we consider throughout this work, and then detail each of the three contributions listed above.

1.1 System-Level Heterogeneity

The driving force for datacenter resource heterogeneity is application diversity in the cloud. But even though the various task streams are heterogeneous, the many tasks within a stream are homogeneous. Further, any given stream typically has a volume of tasks high enough to require several rack's worth of computation. Given such coarse-grained diversity, we envision heterogeneity deployed at the level of servers (or racks thereof). Consequently, a datacenter need not deploy heterogeneous chip multiprocessors (CMPs) since these CMPs will likely be allocated to serve a stream of many, homogeneous tasks. Thus we propose to use system-level, and not core-level, heterogeneity as the most effective way of deploying and managing diverse processors.

System-level heterogeneity also grants architects the ability to deploy a heterogeneous mix of resources in a ratio that matches an expected mix of applications. For example, a datacenter may prefer one big core for every two small cores. Achieving a desired heterogeneous ratio across servers and racks using homogeneous CMPs is easy. If instead we were to focus on fine-grained, intra-chip heterogeneity, the core ratios would be pre-determined by CMP design decisions and as a consequence, likely result in a poor match to the application mix.

Finally, system-level heterogeneity facilitates global datacenter resource management. An allocator can assign resources to serve application tasks, precisely tuning a heterogeneous allocation with its choice of servers and racks. On the other hand, heterogeneous CMPs would be more difficult to manage. A system-wide allocator would assign tasks to a particular node, yet the exact mapping of heterogeneous CMP cores to tasks would be determined by a local scheduler. To ensure that performance is determined by global managers, not local schedulers, we advocate system-level heterogeneity.

1.2 Market Mechanism to Allocate Heterogeneous Processors

Determining an allocation of resources in response to application preferences and system dynamics is difficult in the presence of heterogeneity. Heterogeneous system design allows us to tailor resources to task mixes for efficiency. Yet specialization increases performance risk and demands sophisticated resource allocation. To ensure quality-of-service, we introduce a novel market in which proxies, acting on behalf of applications, possess microarchitectural insight.

The market we deploy to manage heterogeneous processors builds on two prior efforts. Chase et al. manage homogeneous servers by asking users to bid on performance [19]. 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 [66]. We start from the latter market, which assumes fungible processor cycles, and extend it to account for architectural heterogeneity. 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, on the other hand, are indifferent to heterogeneous cycles. We rely on profiling to capture this application-specific preference for processor type, and embed the profiles into the market.

Given the market as a mechanism to match applications to resources, our experiments show the efficiency trade-offs as we vary the types and number of processors that populate the datacenter. We vary the mix of server- and mobile-class processors and find an optimal heterogeneous balance that improves welfare and reduces energy. By increasing the heterogeneity in the system, we find that a combination of three processor architectures reduces response time violations by 12$\times$ relative to a homogeneous system. The results in Chapter 2 demonstrate that an architect can balance efficiency and risk at design-time, and the next chapter formalizes the design decisions to identify strategies that are more likely to produce heterogeneous systems that meet a service quality target.

1.3 Strategies for Heterogeneous Design

Designing heterogeneous systems is difficult. In fact, design methodologies from prior work provide no insight into the performance of heterogeneity under real-world conditions [54, 59]. Prior efforts tailor heterogeneity to diverse software by assuming ideal scheduling and allocation. However, datacenters are large systems with complex dynamics. It is likely, in fact almost inevitable, that an application will not execute on its ideal hardware due to run-time effects, such as contention.

We propose a novel approach to datacenter design that aims for manageability, accounting for the performance uncertainty and management risk introduced by heterogeneity. As an example of run-time risk, consider a datacenter equipped with two resource types such as server and mobile processors. Due to resource availability, an application ill suited to a mobile core may still be forced to use one, and as a result incur a catastrophic slowdown. The penalties of a sub-optimal allocation increase with hardware diversity. Consequently, run-time effects should influence design-time decisions to mitigate this effect.

We need metrics to quantify manageability and to penalize extreme heterogeneity, which may potentially provide high efficiency that is, in practice, difficult to attain. We also need to understand disparate sources of management risk. A heterogeneous system may perform poorly if hardware choices are tailored for other applications, if hardware contention is severe, or if hardware mixes are not matched to software arrival rates. To this end, Chapter 3 contributes a taxonomy of risk, and the evaluation of design strategies uses quantitative metrics to compare the risk across many heterogeneous designs.

1.4 Appraising Fairness in the Market

Compute resources in a datacenter are inevitably shared by a large number of users. To ensure user happiness, datacenters must take a holistic view of performance and fairness. The market mechanism described in Chapter 2 allocates resources to maximize efficiency, yet does not consider whether an allocation is fair. Performance has been the primary objective, whether the manager is navigating heterogeneity or mitigating contention in the shared datacenter. Beyond performance, however, shared datacenters require new policies and mechanisms for fairness.

In Chapter 4, we present a new datacenter management mechanism that fairly allocates processors to tasks with sophisticated performance objectives. Many definitions of fairness exist in systems and economics research. Rather than equate fairness to equal slowdowns amongst applications in a shared system [75], we define fairness in game-theoretic terms [99]:

An allocation is fair when each user weakly prefers her own allocation to that of every other user. In other words, no user envies the allocation of another.

Envy-free systems encourage user participation in shared systems and can thus be deemed fair.

The market allocates resources to latency-sensitive applications, and the novelty of our approach is its ability to mitigate envy for users and tasks with strict performance objectives. Prior efforts have examined fairness for throughput-oriented tasks with simple Leontief utilities [36]. In contrast, we examine latency-sensitive tasks with articulate piecewise-linear utilities. These expressive utility functions more accurately represent realistic service-level agreements. In this setting, reducing envy poses new challenges in resource allocation.

Envy-freeness is a strict definition of fairness, which inevitably has a price. If a management mechanism neglects envy, it can optimize performance by searching an unconstrained space of allocations. However, if the mechanism instead constrains envy, efficiency falls. By comparing welfare-maximizing and envy-minimizing mechanisms, we find that the price of fairness is prohibitively high when a datacenter system is highly loaded. For such settings, we present an alternative to envy-free allocation – ϵ-envy-freeness, which is parameterized by the amount of envy permitted in datacenter allocations. Our work also shows that the envy-free mechanism can allocate heterogeneous resources by using the solution from Chapter 2 to express the preferences of the applications to the market.

1.5 Key Contributions

This thesis presents design and management strategies for datacenters equipped with heterogeneous processors. In Chapter 2 we allocate heterogeneous processors to applications by embedding microarchitectural information into a market mechanism. We demonstrate novel design strategies in Chapter 3 that are more likely to produce manageable heterogeneous datacenters than traditional methodologies. In Chapter 4 we evaluate the trade-off between efficiency and fairness in allocations made by a modified version of the market that restricts the amount of envy allowed in the system. The key contributions of this thesis include:

- **Processor Heterogeneity in the Datacenter.** We define a novel design space where heterogeneous processors are deployed in datacenters to increase the compute capability of the system within a fixed power budget. (§2.1)

- **Economic Mechanisms and Optimization.** We allocate processors to applications with a market that navigates performance-efficiency trade-offs of heterogeneity. (§2.2)

- **Application to Big/Small Cores.** We vary the number of server- and mobile-class processors in a datacenter managed by the market mechanism. Experimentally we observe that 30% of tasks incur response time violations in homogeneous systems but not in heterogeneous ones that use the same power. (§2.3)

- **Application to Further Heterogeneity.** Out of a larger design space that varies frequency, superscalar width, and dynamic execution, we find that a combination of three processor types reduces response time violations by 12× relative to a homogeneous baseline. (§2.4)

- **Anticipating Risk in Heterogeneous Design.** We consider resource management at design-time and ask whether a deployed heterogeneous system is likely to meet design objectives using non-ideal resource allocation. (§3.1)

- **Formalizing Heterogeneous Design Strategies.** We construct a framework of strategies for heterogeneous design, and propose strategies that minimize performance uncertainty. (§3.2)

- **Designing for Manageability.** We explore tens of design strategies and rank the resulting systems based on service quality. Risk-aware design accounts for more than 70% of the top-ranked strategies. (§3.4)

- **Incurring Risk to Increase Reward.** We classify reasons for a deployed system to deviate from expected efficiency, and find that aggressively heterogeneous systems exhibit more risk yet reduce violations of response time targets by 50% compared to less diverse systems. (§3.5)

- **Examine Fairness for Latency-Sensitive Tasks.** We demonstrate that prior algorithms for fair allocations do not support expressive piecewise-linear utility functions, which are necessary to describe the performance targets of latency-sensitive tasks. (§4.2)

- **Introduce Fairness to Markets.** We add constraints to a market mechanism so that the resulting resource allocations are fair. (§4.2)

- **Enforcing Fairness for Heterogeneous Tasks.** We find that web search queries have vastly different runtimes and can be classified upon arrival based on the number of search terms. (§4.3)

- **Quantifying the Price of Fairness.** We observe that the efficiency of a welfare-maximizing market is 1.5× that of fair allocations, though we can improve efficiency by parametrizing the amount of envy allowed in the system. (§4.4)

- **Extending Fairness to Heterogeneous Processors.** We show that the microarchitectural differences in heterogeneous processors can be embedded into the market when the fairness constraints are present. (§4.5)

2

Market Mechanism to Allocate Heterogeneous Processors

Datacenter energy efficiency can benefit from new system architectures and microarchitectures that are designed for energy-constrained systems. Recent research and industry trends highlight opportunities for building servers with lightweight processors that were originally designed for mobile and embedded platforms [3, 88]. 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 [84]. 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. This chapter is organized as follows:

- **Processor Heterogeneity in the Datacenter (§2.1).** We identify a new design space where heterogeneous processor microarchitectures allow a datacenter to combine the benefits of specialization with the performance guarantees of traditional high-performance servers.

- **Economic Mechanisms and Optimization (§2.2).** 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 (§2.3).** 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 (§2.4).** 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.

2.1 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.

2.1.1 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 2.1 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 §2.3. 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 $< 1\%$.

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

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 work, we trade-off efficiency and performance by considering two existing processors: the mobile-class Atom and the server-class Xeon (§2.3). Additionally, we design and evaluate up to twelve cores that lie along the efficiency-vs-performance spectrum (§2.4).

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. Balanc-

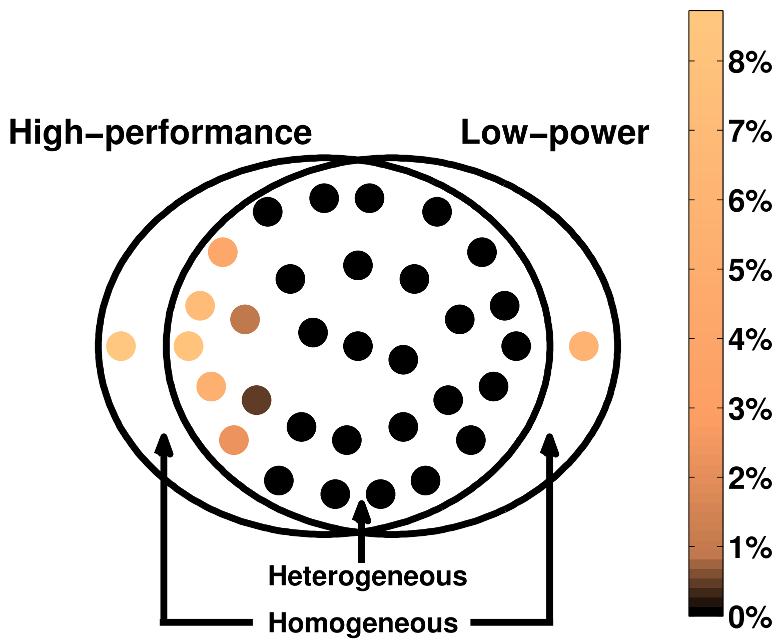


Figure 2.1: Venn diagram that illustrates a datacenter design space for low-power and high-performance processors; the intersection harbors heterogeneous design options. Colored points depict QoS violations.

ing 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.

2.1.2 Accommodating Architectural Heterogeneity

Up to 5× more efficient than big ones, small processor cores are increasingly popular for datacenter computation [84]. Small cores are well balanced for the modest computational intensity of simple web search queries, distributed memory caching, and key-value stores [3, 78, 84]. Such research in unconventional datacenter hardware has spurred broader commercial interest [4, 25] and analogous research in other technologies, such as DRAM [67, 106].

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 [96], 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 [101], by Chase in 2001 [19], and Lubin in 2009 [66]. 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:

$$\frac{\texttt{Tasks}}{\texttt{Sec}} = \frac{\texttt{Cycles}}{\texttt{Sec}} \times \frac{\texttt{Insts}}{\texttt{Cycle}} \times \frac{\texttt{Tasks}}{\texttt{Inst}} \tag{2.1}$$

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.

To calculate scaling factors, we rely on application profiling data. In this thesis, we assume that existing profilers provide this data (see §2.6 for a survey of related work). Although more advances are needed, existing profilers are sophisticated and allow us to focus on the allocation mechanism.

2.2 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 [19].

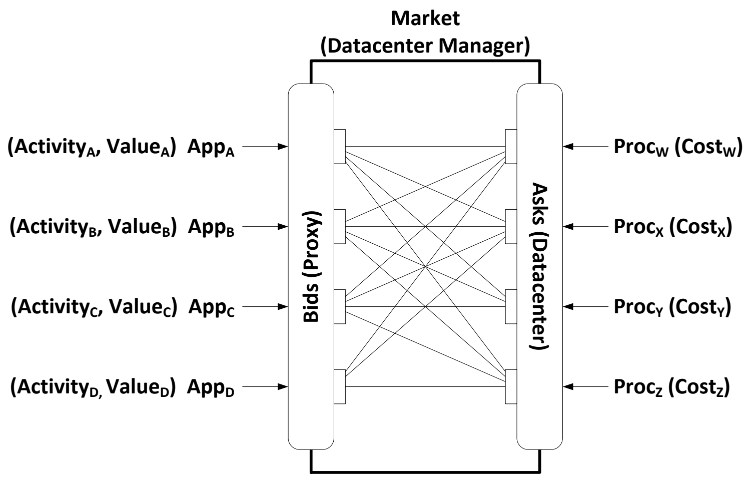

Market
(Datacenter Manager)
(Activity$_A$, Value$_A$) App$_A$
(Activity$_B$, Value$_B$) App$_B$
(Activity$_C$, Value$_C$) App$_C$
(Activity$_D$, Value$_D$) App$_D$
Bids (Proxy)
Asks (Datacenter)
Proc$_W$ (Cost$_W$)
Proc$_X$ (Cost$_X$)
Proc$_Y$ (Cost$_Y$)
Proc$_Z$ (Cost$_Z$)

FIGURE 2.2: Market Overview.

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 [66]. We start from the latter market, which assumes fungible processor cycles, and extend it to account for architectural heterogeneity.

Figure 4.4 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.

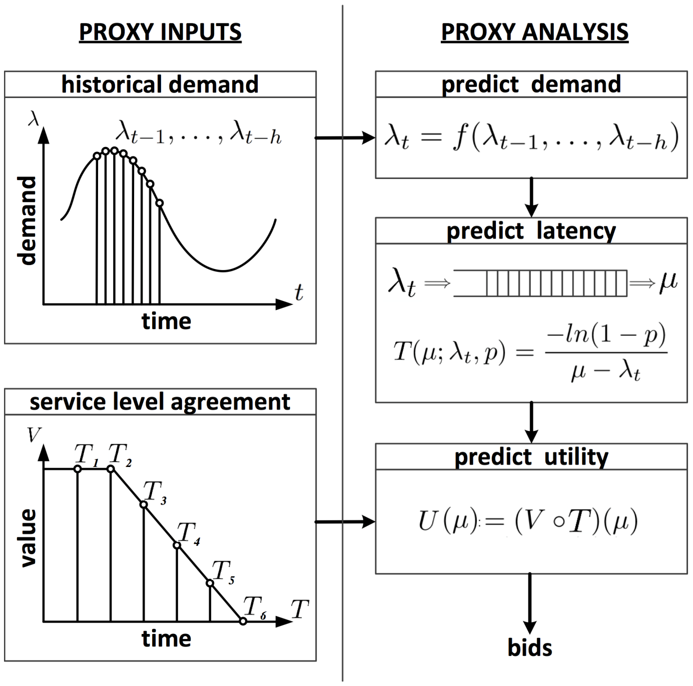

PROXY INPUTS
PROXY ANALYSIS
historical demand
λ
$\lambda_{t-1}, \ldots, \lambda_{t-h}$
demand
time
t
service level agreement
V
T_1 T_2 T_3 T_4 T_5 T_6 T
value
time
predict demand
$\lambda_t = f(\lambda_{t-1}, \ldots, \lambda_{t-h})$
predict latency
$\lambda_t \Rightarrow \Rightarrow \mu$
$T(\mu; \lambda_t,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

FIGURE 2.3: Proxy Bids

2.2.1 Proxies and Value Analysis

In this chapter, 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 2.3 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 [66].

Estimate Task Service Rate. To serve these arriving tasks, an optimizer searches an allocation space of heterogeneous cores. Prior efforts assume fungible processor cycles [19, 66], 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 §2.5 for details. We estimate latency percentiles with Equation (2.2) and use the 95th percentile as the figure of merit, $p = 0.95$.

$$\text{p-th latency percentile} \quad \Big| \quad T = -ln(1-p)/(\mu - \lambda) \tag{2.2}$$

$$\text{service rate inflections} \quad \Big| \quad \hat{\mu}_t = \lambda_t - ln(1-p)/\hat{T} \tag{2.3}$$

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 (2.4)–(2.6)).

$$\text{datacenter profiler} \quad | \quad \mathbf{P_a} : \{hw_a\} \rightarrow \{service\ rate\} \tag{2.4}$$

$$\text{datacenter queues} \quad | \quad \mathbf{T} : \{service\ rate\} \rightarrow \{latency\} \tag{2.5}$$

$$\text{user value} \quad | \quad \mathbf{V} : \{latency\} \rightarrow \{dollars\} \tag{2.6}$$

$$\text{market welfare} \quad | \quad \mathbf{W} = \sum_{a \in A} \Big(\mathbf{V} \circ \mathbf{T} \circ \mathbf{P_a}(hw_a) \Big) - \mathbf{C}(hw) \tag{2.7}$$

$$\begin{aligned} E = & \underbrace{\left(n^a P^{act} + n^i P^{idle} + n^s P^{sleep}\right) \Delta}_{\text{no power transition}} \\ & + \underbrace{n^{is} \left(P^{idle} \delta^{is} + P^{sleep}(\Delta - \delta^{is})\right)}_{\text{idle}\rightarrow\text{sleep}} \\ & + \underbrace{n^{sa} \left(P^{act} \delta^{sa} + P^{act} \left(\Delta - \delta^{sa}\right)\right)}_{\text{sleep}\rightarrow\text{active}} \end{aligned} \tag{2.8}$$

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 (Equation (2.3)). 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.

2.2.2 Seller Cost Analysis

For an accurate estimate of electricity use, the market requires information about server and processor power modes from the datacenter [71, 72]. For example, we model server power modes as three possible states: active, idle (but in an active power mode), and sleep.

In Equation (2.8), the datacenter accounts for the number of servers (n^*) in each mode and power (P^*) dissipated over the allocation time period (Δ) [66]. 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 [8].

2.2.3 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 (2.7)). 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 [66]. 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,

Table 2.1: Architecture parameters for Xeons and Atoms [34, 35, 48].

	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 2.2: Power modes and parameters for Xeons and Atoms [84].

	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 $\rightarrow$ Active	8 secs, $0.05	
Active $\rightarrow$ Sleep	6 secs, $0.05	

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.

2.3 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.

2.3.1 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

Table 2.3: Characteristics of the processor sensitive (PS) and insensitive (¬PS) applications. For a task stream, T is 95th percentile queueing time.

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

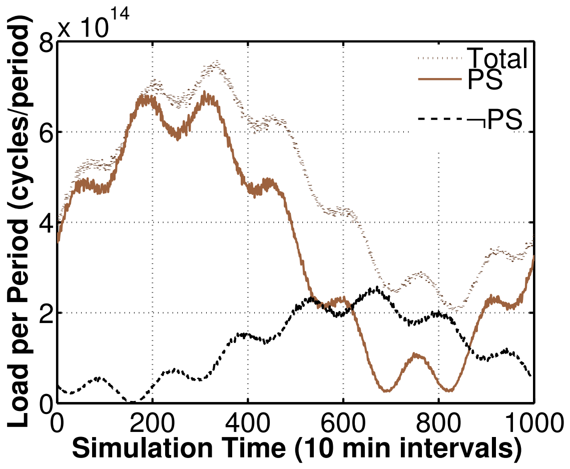
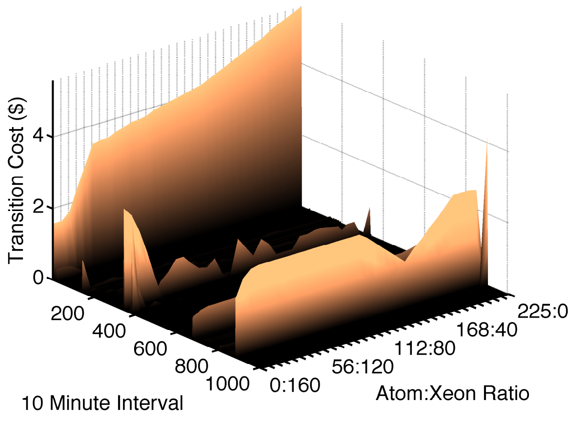


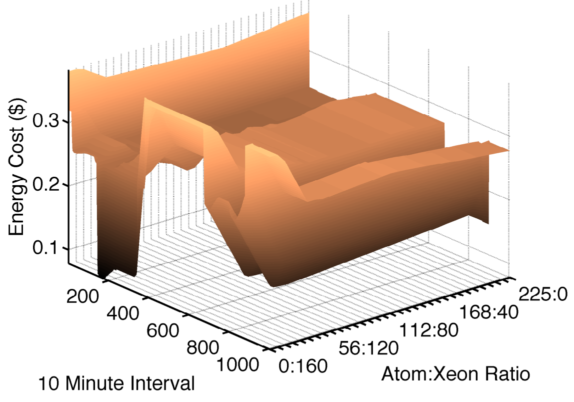
Figure 2.4: Demand for processor sensitive (PS) and insensitive (¬PS) applications.

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 2.1–2.2 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 [84, 88]. 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



(a) **Transition Cost**



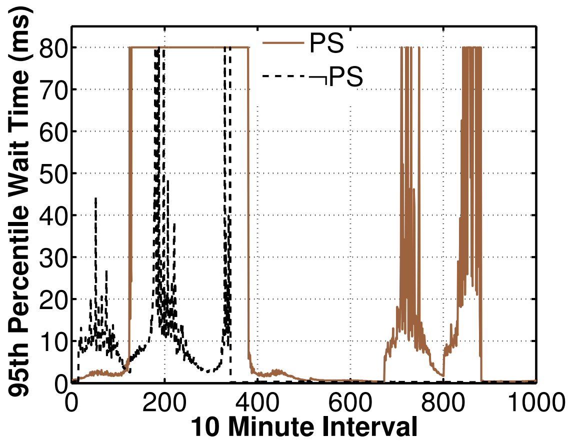
(b) **Energy**

FIGURE 2.5: 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 [38].

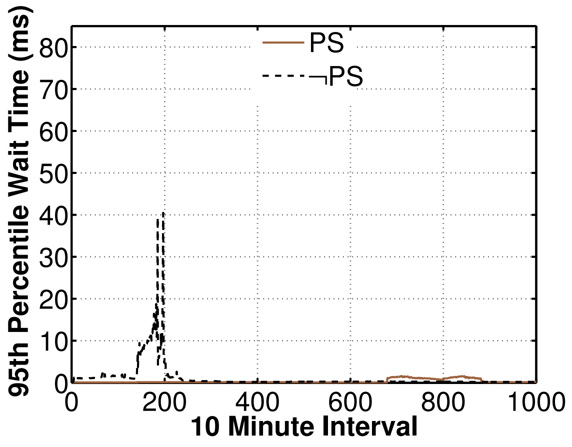
the memory and network interface [1, 32]. Power usage effectiveness (PUE) for the datacenter is 1.6, an average of industry standards [29, 97]. Energy costs are \$0.07 per kWh, an average of surveyed energy costs from prior work [83].

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.

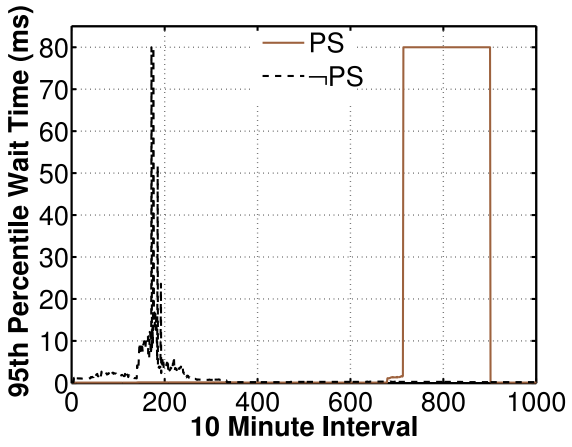
Workloads. We study tasks that are generated to follow a time series, which is detailed in Table 2.3 and illustrated in Figure 2.4. We simulate a week of task load that is a composite of two sinusoids, one with a week-long period and one with a day-long period. The sinusoid determines the average arrival rate around which we specify a Gaussian distribution to reflect load randomness. Such patterns are representative of real-world web services [72].



(a) **0 Atoms::160 Xeons**



(b) **147 Atoms::55 Xeons**



(c) **225 Atoms::0 Xeons**

FIGURE 2.6: 95th 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).

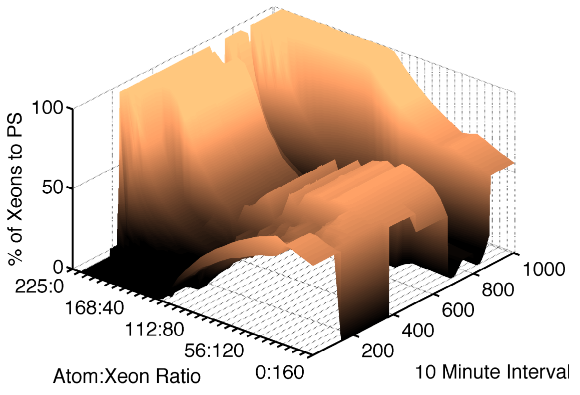
Applications specify value (SLA) as a function of 95^{th} percentile response time. Value degrades linearly up to a cut-off of 80ms, after which computation has no value. The value functions express priorities for applications. Since the market maximizes welfare, users with higher value per requested 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.

2.3.2 Architectural Preferences

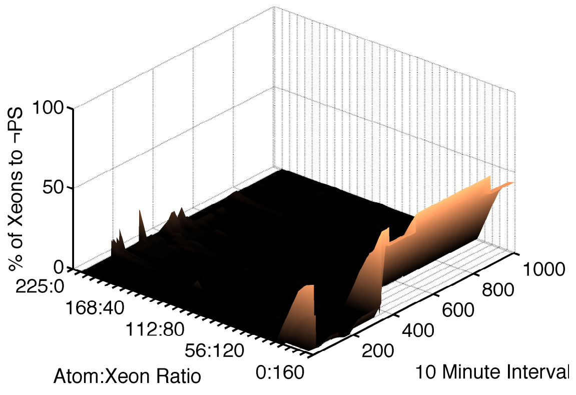
We consider two workloads that contend for Xeons and Atom servers, yet value the cores differently. The first is a processor sensitive (PS) application that prefers cycles from the high-throughput Xeon and values an Atom cycle less, scaling its value down by $\kappa = \frac{1}{3}$. The second, on the other hand, is a processor insensitive ($\neg$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 [84]. Lim et al. find performance on the mobile-class Intel Core 2 Turion is 34% of that on the Intel Xeon [64]. These applications exhibit instruction-level parallelism, which benefits from wider pipelines and out-of-order execution in server-class cores: $\kappa_X = 1, \kappa_A = \frac{1}{3}$.

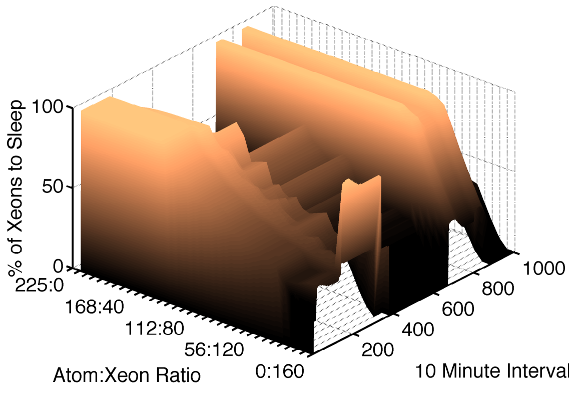
In contrast, $\neg$PS does not benefit from extra capabilities in server-class processors and is representative of web, file, or database servers [26, 50]. Andersen et al. propose embedded processors for distributed key-value store servers [3]. Servers that deliver Youtube-like traffic can run on small cores with negligible performance penalties [64]. Higher processor performance does not benefit such workloads. $\neg$PS applications are indifferent to Xeons and Atoms: $\kappa_X = \kappa_A = 1$.



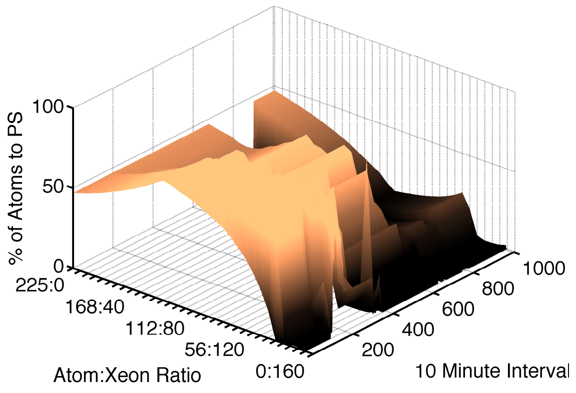
(a) **Xeon Allocation for PS**



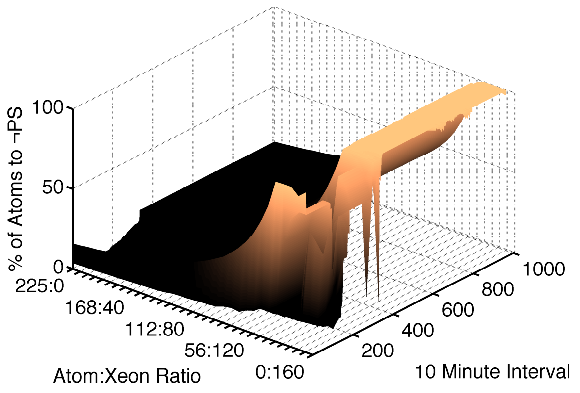
(b) **Xeon Allocation for ¬PS**



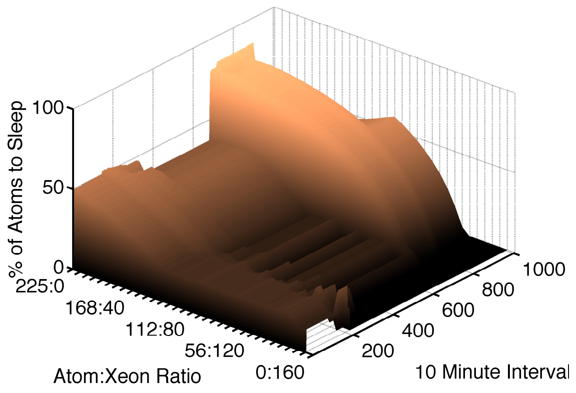
(c) **Xeon Sleep**



(d) **Atom Allocation for PS**



(e) **Atom Allocation for ¬PS**



(f) **Atom Sleep**

Figure 2.7: 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.

2.3.3 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 $\leqslant$ 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 2.6a 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 $\neg$PS, which is indifferent between Xeons and Atoms. As Xeons are replaced by Atoms, Figure 2.7a shows the market allocating a larger fraction of the remaining Xeons to PS thus improving its response time. Simultaneously, Figure 2.7b shows the market allocating fewer Xeons to $\neg$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 2.5a, a charge is imposed for every transition to account for increased wear and reduced mean-time-to-failure as machines power-cycle [38].

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 2.6c).

2.3.4 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 2.7f) and the fraction of Atoms allocated to PS and $\neg$PS decline (Figure 2.7d and Figure 2.7e, 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 2.5a, 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 2.7a and Figure 2.7d, 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 2.7a and Figure 2.7c, 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 2.6b, this mix of Xeons and Atoms provide queueing times that are stable and far below the 80ms cut-off.

For contrast, consider Figure 2.6a and Figure 2.6c. 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, $\neg$PS suffers longer waiting times due to contention with PS for limited computational capacity. In the Atom-only system, $\neg$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.

2.3.5 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 2.8, 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

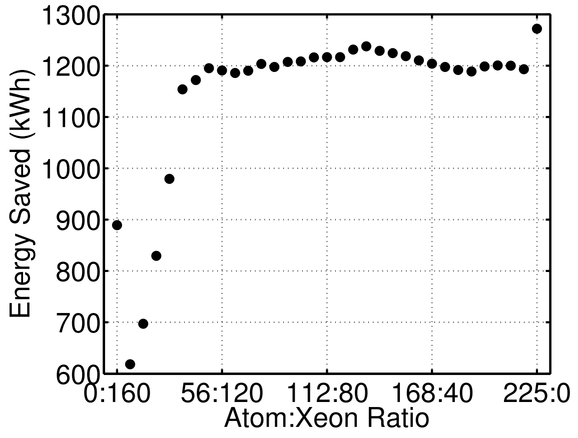


FIGURE 2.8: Energy saved from sleep modes.

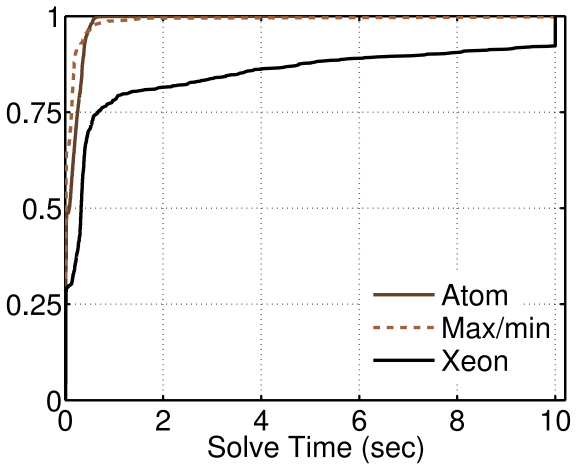


FIGURE 2.9: Solve time CDF across all periods.

(Figure 2.6). 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 [9]. 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 2.5b). 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.

2.3.6 Evaluating Optimization Time

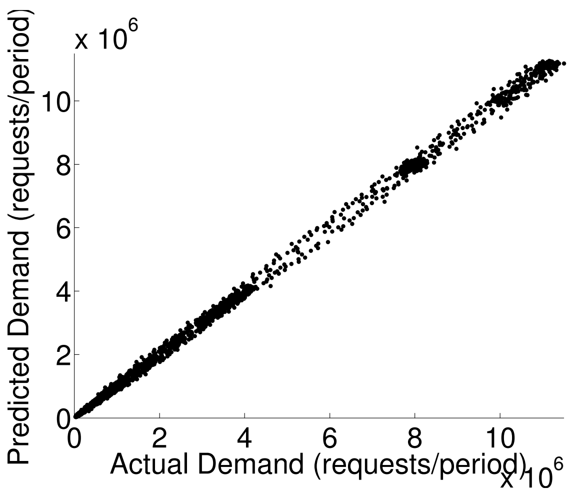
Given that the market and proxy implementation include all the elements required in a real system, market clearing performance is important. Across allocation periods in all datacenter configurations, solve time (wall clock) varies but is less than 800ms for 98% of allocation periods. All other periods clear the market in less than 10s as shown in Figure 4.12. 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.

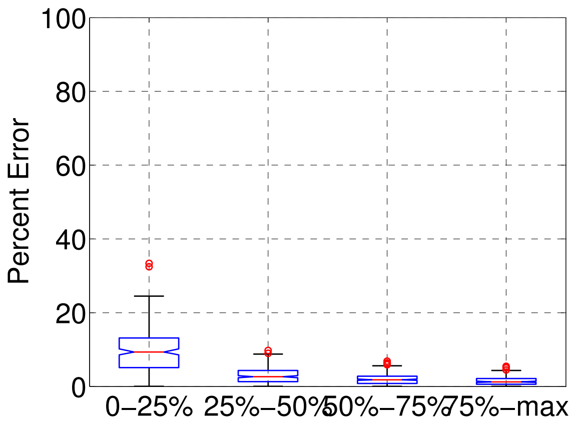
We obtain this computational speed by representing heterogeneous resources as scaled canonical ones, and thus keeping the MIP tractable. Further, in our MIP formulation the task arrival rate and the number of servers in the system simply affect coefficients, not MIP computational complexity. However, 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 [66].

2.3.7 Assessing Demand Prediction

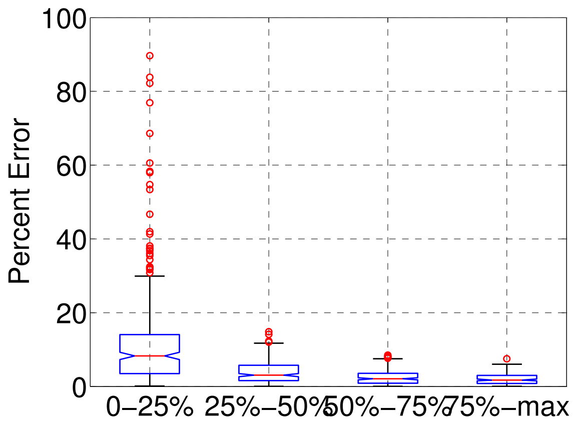
Although adding mobile-class Atoms to the datacenter improves PS waiting time, the queuing time for $\neg$PS suffers during time periods 165-185 across all mixes of Xeon and Atom cores. Figure 2.6 illustrates the response time spike. These intervals correspond to a trough in $\neg$PS load. Demand is particularly difficult to predict in this trough.



(a) Relative Error



(b) Processor Sensitive Application



(c) Processor Insensitive Application

FIGURE 2.10: Prediction error in demand: (a) relative error, (b,c) percent error for each application at different quartiles of total load. Using 6 time periods of load traffic history, the predictions are sufficiently accurate.

Figure 2.10a indicates that predicted demand is strongly correlated with actual demand. However, if we consider prediction error broken into demand quartiles, we find errors are larger at low levels of demand. This effect is most noticeable for ¬PS. At lowest quartile of load, Figure 2.10c indicates that prediction errors for ¬PS can be as high as 90%. Figure 2.10b 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.

Table 2.4: 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 2.5: 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

Table 2.6: Characteristics for the task streams made of two applications from SPEC CPU 2006, libquantum and lbm.

	libq	lbm
P – task profile (Mcycles/task)	67	149
λ – peak load (Ktasks/min)	480	80
V – value ($/month)		
if $T \leqslant 20$ms	$5000	$2500
if $T \geqslant 160$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

2.4 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.

2.4.1 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 2.4. We simulate these designs with the gem5 cycle-accurate simulator in syscall emulation mode [12].

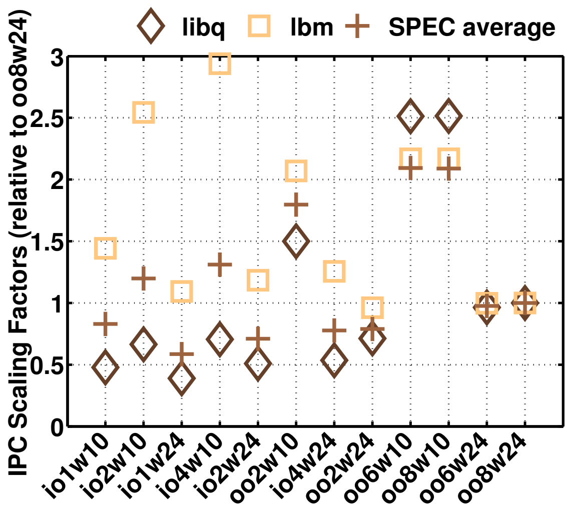
For this experiment, we consider the preferences of SPEC CPU2006 applications on heterogeneous processors. These benchmarks are sensitive to processor choice, and we study the opportunity of using low-power cores even for applications with high instruction-level parallelism. We simulate 100M instructions from gobmk, hmmer, h264ref, mcf, libquantum, bzip2, sjeng, gcc, xalancbmk, milc, gromacs, namd, calculix, dealII, 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 2.6, which shares parameters from §2.3 where possible.

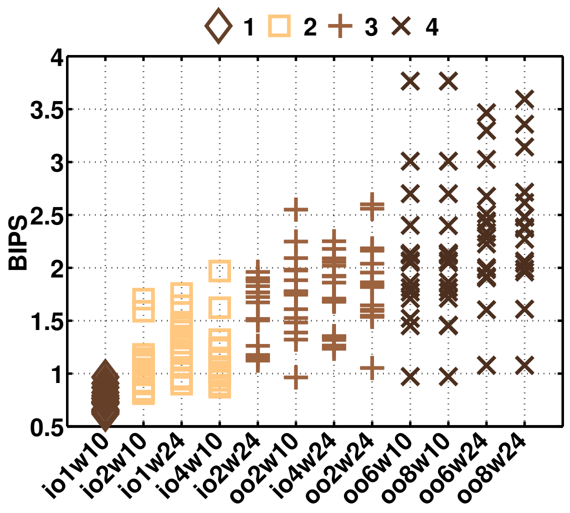
2.4.2 Architectural Preferences

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 [82]. 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 2.11b 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 2.5). 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.



(a) **IPC scaling factors**



(b) **Core clustering k=4**

FIGURE 2.11: gem5 simulation results, cores on the horizontal axis are in order of increasing peak dynamic power.

We organize these cores into servers that use equal-area processors; area and power are estimated with McPAT models [61], 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 [84].

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.

2.4.3 Improving Service Quality

Increased processor diversity benefits service quality. Figure 2.12 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

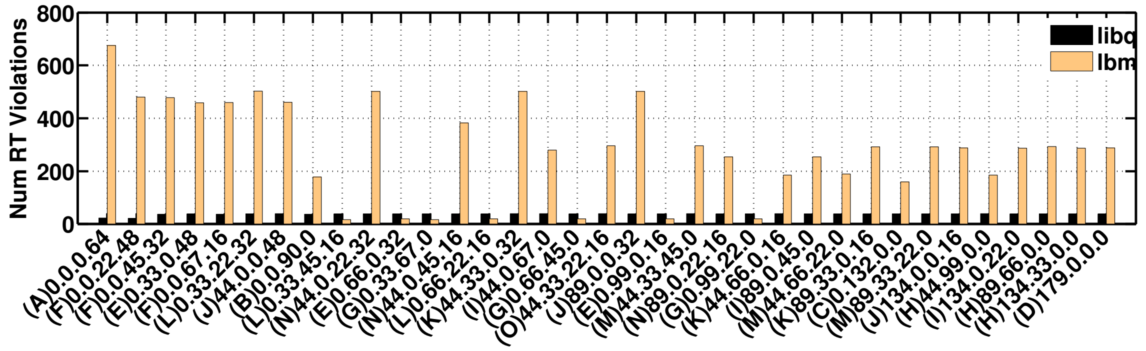


FIGURE 2.12: Number of 95th percentile waiting time violations. The horizontal axis indicates the number of servers of each of the four core types: io1w10, io1w24, io4w24, and oo6w24, in that order. Prepended letters mark the corresponding region in Figure 2.13.

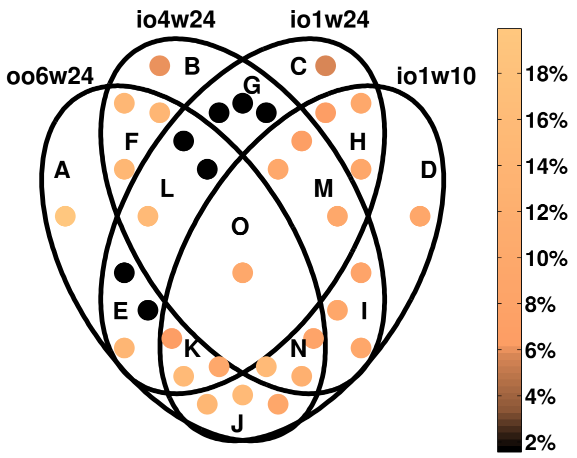


FIGURE 2.13: Sum of libq and lbm waiting time violations, shown on a Venn diagram.

x-axis. Data is shown for libquantum and lbm, which are representative of diversity in the broader application suite (Figure 2.11a).

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 lbm. For example, a system with 33 io1w24 cores and 67 io4w24 cores (00.33.67.0 in Figure 2.12) 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.

2.4.4 Balancing Core Types

Figure 2.13 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.

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 2.13 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 trade-offs. But related work in heterogeneous microarchitectures have illustrated diminishing marginal returns, which coincidentally arise as heterogeneity increases beyond four designs [59].

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

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.

2.5 Qualifications and Assumptions

We assume users submit jobs that are comprised of tasks. For these tasks, we assume the 95^{th} 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. [70] 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 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 [72, 73].

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) [41]. 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 [19, 66].

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.

2.6 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 [7, 26, 50, 65]. Server efficiency also benefits from re-purposing processors originally designed for mobile platforms [49, 64, 84]. These efforts illustrate small-core efficiency for memory- and I/O-bound tasks, and warn about performance penalties for more complex computation. Indeed, microarchitec-

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

ture increasingly affects datacenter computation [30]. 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 [23, 53, 54, 59, 62] 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 [43, 95] or devote an efficient core to the operating system [74]. 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 [70, 76]. 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, 16, 100]. And prior work matched heterogeneous software demands (e.g., from Hadoop tasks) with heterogeneous VMs [36, 60]. 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 [31, 46, 96, 101]. Chase et al. extended these mechanisms to account for energy costs [19]. Lubin et al. further accommodated dynamic voltage/frequency scaling in datacenter markets [66]. 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) [33, 76, 87]. Both Chase and Lubin show greedy solvers are less effective than markets for improving service time and reducing cost. Like Lubin [66], we use a mixed integer program to find exactly optimal allocations, but approximate methods like gradient ascent [19, 70] 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 [36]. However, maximizing welfare and fairness in this setting are mutually exclusive [79]. 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 [36, 44].

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 [37], VTune [47], or OProfile [77]. 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 [85].

Given samples, inferred statistical machine learning models might predict scaling factors as a function of software characteristics and hardware parameters [104]. 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 microarchitecturally-independent software characteristics and statistical inference [28, 58].

2.7 Summary

Results in this chapter 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 that design heterogeneous systems cannot ignore their deployment. Market mechanisms are well suited to allocating heterogeneous resources to diverse users.

3

Strategies for Heterogeneous Design

The previous chapter applied a market mechanism to manage heterogeneous processors by feeding it application-specific information to estimate the performance difference between different microarchitectures. Experimentally, we observed a large variation in the service quality of applications running on different heterogeneous systems made up of the same two or four cores. In this chapter, we dissect the heterogeneous design process to study the effect of design decisions on the run-time behavior of a system using a realistic resource manager.

Designing heterogeneous systems is difficult. and we find that design methodology from prior work provides no insight into the performance of heterogeneity under real-world conditions [54, 59]. Prior efforts tailor heterogeneity to diverse software by assuming ideal scheduling and allocation. However, datacenters are large systems with complex dynamics. It is likely, in fact almost inevitable, that an application will not execute on its ideal hardware due to run-time effects, such as contention.

In particular, we propose a novel approach to datacenter design that aims for manageability, accounting for the performance uncertainty and management risk introduced by heterogeneity. Effective resource allocation is more difficult in systems with diverse hardware. As an example of run-time risk, consider a datacenter equipped with two resource types such as server and mobile processors. Due to resource availability, an application ill suited to a mobile core may still be forced to use one, and as a result incur a catastrophic slowdown. The penalties of a sub-optimal allocation increase with hardware diversity. Consequently, run-time effects should influence design-time decisions to mitigate this effect.

We need metrics to quantify manageability and to penalize extreme heterogeneity, which may potentially provide high efficiency that is, in practice, difficult to attain. We also need to understand disparate sources of management risk. A heterogeneous system may perform poorly if hardware choices are tailored for other applications, if hardware contention is severe, or if hardware mixes are not matched to software arrival rates.

This chapter makes the following contributions towards a design flow for heterogeneous systems that anticipates run-time risk:

- **Anticipating Risk in Heterogeneous Design.** We propose a novel approach to heterogeneous design that accounts for system management. Unlike prior efforts, we ask whether a deployed heterogeneous system is likely to meet design objectives using non-ideal resource allocation (§3.1).

- **Formalizing Heterogeneous Design Strategies.** We set forth a holistic framework of design strategies, and propose strategies that minimize performance variation. In particular, we consider the coefficient of variation for each application on all processors in the heterogeneous system (§3.2).

- **Designing for Manageability.** From the tens of design strategies in our framework, we identify those that produce systems with the best service quality. We find that risk-aware design accounts for more than 70% of these desirable strategies (§3.4).

- **Incurring Risk to Increase Reward.** We show that the additional diversity of aggressively heterogeneous systems reduces violations of response time targets by 50% compared to less diverse systems. Having formalized the notion of risk, we enumerate reasons why a heterogeneous datacenter might deviate from expected efficiency (§3.5).

Rather than consider a single strategy, we define a framework of design strategies (§3.2). Each strategy produces a heterogeneous system, which is managed by a market mechanism (§3.3). We identify strategies that are likely to produce designs with higher performance, higher energy efficiency, and lower run-time risk (§3.4–3.5). Collectively, our findings make the case for rethinking heterogeneous design strategies to account for run-time risk.

3.1 Anticipating Risk in Heterogeneous Design

One of the greatest challenges to heterogeneous system design is resource management. A deployed system may not realize the performance opportunity of the design effort due to the difficulty of mapping applications to diverse hardware. Prior approaches aim for performance and/or efficiency targets based on ideal mappings of workloads to resources. In contrast, our work presents a novel approach to heterogeneous design that provisions for the management of such systems in the design flow.

3.1.1 Anticipating Run-time Effects

The state-of-the-art in heterogeneous processor design focuses on tractable analysis and optimization. Heterogeneity significantly expands the design space, especially given all permutations of application-to-core pairings. Prior studies explore design spaces for processor cores [23, 54, 59] and datacenter organizations [39]. These methodologies find a subset of cores from a design space that satisfy diverse application behavior.

This state-of-the-art strategy is rather limited. It focuses on maximizing best-case performance and/or efficiency. Such performance guarantees may collapse when an application cannot execute on its best-matched architecture due to run-time effects, such as contention, which are prevalent in datacenters.

Yet heterogeneity in datacenter hardware is desirable as it presents an opportunity for energy efficiency. Web search executes on small processors at $\frac{1}{5}\times$ the energy of big cores, and degrades throughput by $\frac{1}{3}\times$ [64, 84]. Similarly, web search and memcached will transfer data across low-bandwidth memory channels at $\frac{1}{5}\times$ the energy of high-bandwidth channels with negligible performance penalty [67]. Energy-efficient technologies cannot provide uniform performance guarantees to all applications; thus, a heterogeneous hardware mix is needed to balance performance and efficiency.

Datacenters that are *heterogeneous by design* will use hardware that better matches application diversity to increase efficiency. For a given a power budget, a heterogeneous datacenter's quality-of-service is greater than that of homogeneous datacenters [39]. Achieving this improvement in service quality depends on effectively managing heterogeneous resources. In this chapter, we present a design flow to anticipate resource management challenges during heterogeneous system design.

We propose alternative strategies that consider metrics beyond best-case performance and efficiency. We introduce the notion of anticipating run-time effects during the design process. Our new design strategies seek energy efficiency while improving worst-case performance and mitigating performance variation. Such optimization criteria are particularly relevant for datacenters, which aim for strict service quality guarantees and seek to avoid run-time variations.

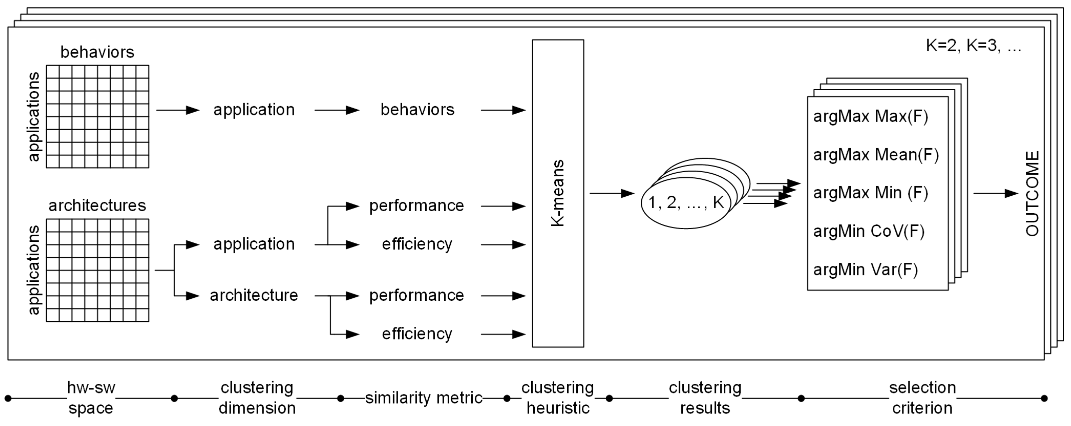


FIGURE 3.1: A Strategy Framework shows the numerous approaches to heterogeneous design.

3.1.2 Understanding Sources of Risk

As we introduce new heterogeneous design strategies, we evaluate their ability to mitigate risk. To illustrate the importance of risk analysis, consider the classical problem of reducing application diversity through basic block clustering [90] and benchmark redundancy analysis [81]. Suppose X_i is performance for application i and X_i's are identically and independently distributed:

$$\mathrm{Var}\left(\mathrm{E[X]}\right) = \mathrm{Var}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{1}{n^2}\sum_{i=1}^{n}\mathrm{Var}(X_i) = \frac{1}{n}\mathrm{Var}(X_i)$$

Redundant applications may be removed from a suite while preserving the mean (i.e., expected performance). But variance is affected, which is unfortunate since understanding performance uncertainty is critical to heterogeneous design.

If risk is defined as uncertainty, then performance and efficiency risk increases with heterogeneity. Next, we consider three types of risk that may prevent a heterogeneous system from realizing best-case performance and efficiency: (i) application risk, (ii) contention risk, and (iii) system risk.

Application Risk. Processor architects design product families using benchmark suites. But system architects demand performance for only a subset of these applications. By using only a few representative benchmarks, architects risk designing for benchmarks that are dissimilar to run-time software. Note that this definition of risk excludes applications that lie in the superset of the benchmark suite. Heterogeneity exacerbates application risk by more tightly tying benchmark mixes to processor optimization.

Contention Risk. Architects select core types with the intent that each application executes on the core that maximizes efficiency. Prior studies in heterogeneous processors consider only this optimal matching of applications to cores [23, 53, 59, 94]. Contention risk occurs when the preferred core is present in the heterogeneous system, yet allocated to another application. When a task uses an alternative to its preferred core, design-time decisions determine performance and efficiency penalties. Mitigating contention risk requires accounting for substitution effects during design-time.

System Risk. Datacenter procurement invests a fraction of the power budget to each core type. System risk is the uncertainty that the fractional share of deployed cores will match the run-time application mix. Prior work considers only one core of each type or fixed shares [23, 53, 59, 94]. Fractional shares matter most as core types become increasingly diverse. Equally dividing a system's power budget to big and small cores may work well for a particular application mix. Yet a different mix may demand another fractional share of big and small cores. Mitigating system risk requires a coordinated decision between the design of the heterogeneous core types and the fractional share of each.

3.2 Formalizing Heterogeneous Design Strategies

Systems with heterogeneous processors aim to provide microarchitectures that more closely match diverse applications. The design flow requires a series of decisions to select cores from a design space: Do we cluster similar applications or architectures? How do we select a processor to match each cluster? Do we optimize for performance or efficiency? We refer to the answers to these design questions as a *strategy*. Strategies produce heterogeneous designs, with different performance characteristics when deployed and managed at scale.

We set forth a set of essential design decisions in a *strategy framework*, which is the scaffolding that contains all strategies. The framework includes strategies from prior work that aim to maximize performance or efficiency as long as applications run on the best-matched core. In addition, the framework includes novel strategies where cores are selected to reduce variation or minimize performance under imperfect allocations. We illustrate our framework in Figure 3.1, and detail each stage in which a decision is made (e.g., hardware-software space, clustering dimension, etc.).

3.2.1 Characterizing the Hardware-Software Space

A strategy begins with data from the *hardware-software space*, which we represent as two data matrices. In the first, we profile application behavior on many architectures. Microarchitecture-independent characteristics, such as instruction mix, branches taken, and basic block size, are the elements of the first matrix. Matrix rows represent applications (e.g., web search), and columns represent behaviors (e.g., basic block size).

In the second data matrix, we profile figures of merit for a variety of application-architecture pairs. Matrix rows still represent applications, and columns now represent architectures (e.g., out-of-order, six-wide superscalar, 1GHz). Matrix elements are measures of performance (BIPS) or efficiency (BIPS^3/W). These matrices are populated with data from cycle-accurate simulation for diverse applications and processors.

3.2.2 Formulating the Clustering Problem

The applications and architectures we study may have similar characteristics. To discard repetition from our design space, we select a *clustering dimension* by grouping together rows or columns of the matrices based on similarity. In the application dimension, clusters identify software that behave similarly across many architectures. In the architecture dimension, clusters identify hardware that perform similarly across many applications. Further, we choose a *similarity metric* for clustering, e.g. software behavior, performance, or efficiency.

Clustering Dimension. Only a subset of the cores in a design space need actually be deployed given that many of the cores provide similar performance or efficiency to similar applications, and are thus redundant. Clustering applications distills many applications into a few representative ones for which hardware can be tailored. For a particular application mix, this approach may produce a narrowly defined mix of cores. Clustering architectures identifies a few representative cores that span the full spectrum of performance and power trade-offs. Diversity is particularly useful at run-time as resource managers have the opportunity to maximize efficiency and/or meet performance targets with more types of cores. Next, we detail clustering implementations.

Application Clustering. We can identify similar applications based on their microarchitecture-independent behavior. The application-behavior matrix is split into row vectors, which are clustered so that applications with similar behavior belong in the same cluster.

Alternatively, we might identify similar applications based on performance or efficiency. If two applications exhibit similar performance across a broad spectrum of cores, we infer that microarchitectural mechanisms (e.g., dynamic instruction scheduling) affect both in similar ways. In this case, the application-architecture matrix is split into row vectors and clustered. Applications that prefer the same architectures will be assigned to the same cluster.

Architecture Clustering. In the architecture dimension, cores that deliver similar performance across a spectrum of applications might be expected to have similar microarchitectures. The application-architecture matrix is clustered by columns, distilling the large hardware design space into representative cores.

3.2.3 Invoking the Clustering Heuristic

We use the K-Means algorithm to cluster applications or architectures. As the number of clusters, K, increases, the heuristic identifies more classes of similar applications (respectively architectures). The data fed into the clustering heuristic is one of the following three *similarity metrics*: microarchitecture-independent behavior, performance (BIPS), or efficiency (BIPS^3/W). Each vector is a dimension to the heuristic, and similarity is defined by Euclidean distance between vector-elements. When application behavior is the similarity metric, the vector-elements are weighted by the correlation coefficient of each dimension to performance.

Empirically, we see that clustering by application behaviors or performance produces a small number of similar clusters. This approach leads to systems made up of big cores that deliver high performance. In contrast, clustering by efficiency exploits interesting performance and power trade-offs. The outcome is a mix of big and small cores, reflecting the fact that the performance advantage of big cores may not justify their power cost.

Table 3.1: Parameters of the processor design space made up of 42 cores.

Core ID	**Exe**	**Width (insns)**	**L2 (MB)**	**Freq (GHz)**	**Power (W)**	**Area (mm^2)**	**Num (per CMP)**
1	IO	1	1/4	1	2.01	8.30	18
2	IO	1	1/4	2	3.11	8.30	18
3	IO	1	1/2	1	2.33	8.96	17
4	IO	1	1/2	2	3.43	8.96	17
5	IO	1	1	1	2.24	9.99	15
6	IO	1	1	2	3.34	9.99	15
7	IO	2	1/4	1	2.69	9.78	15
8	IO	2	1/4	2	4.45	9.78	15
9	IO	2	1/2	1	3.00	10.44	14
10	IO	2	1/2	2	4.77	10.44	14
11	IO	2	1	1	2.91	11.47	13
12	IO	2	1	2	4.68	11.47	13
13	IO	4	1/4	1	4.42	13.29	11
14	IO	4	1/4	2	7.93	13.29	11
15	IO	4	1/2	1	4.74	13.96	11
16	IO	4	1/2	2	8.24	13.96	11
17	IO	4	1	1	4.65	14.99	10
18	IO	4	1	2	8.15	14.99	10
19	OOO	2	1	1	5.56	13.96	11
20	OOO	2	1	2	9.98	13.96	11
21	OOO	2	2	1	5.72	16.94	9
22	OOO	2	2	2	10.14	16.94	9
23	OOO	2	4	1	6.42	23.21	6
24	OOO	2	4	2	10.84	23.21	6
25	OOO	4	1	1	9.90	16.40	9
26	OOO	4	1	2	18.66	16.40	9
27	OOO	4	2	1	10.07	19.38	7
28	OOO	4	2	2	18.82	19.38	7
29	OOO	4	4	1	10.76	25.65	5
30	OOO	4	4	2	19.52	25.65	5
31	OOO	6	1	1	12.13	22.96	6
32	OOO	6	1	2	23.11	22.96	6
33	OOO	6	2	1	12.29	25.94	5
34	OOO	6	2	2	23.27	25.94	5
35	OOO	6	4	1	12.99	32.20	4
36	OOO	6	4	2	23.97	32.20	4
37	OOO	8	1	1	17.12	29.20	5
38	OOO	8	1	2	33.09	29.20	5
39	OOO	8	2	1	17.28	32.18	4
40	OOO	8	2	2	33.25	32.18	4
41	OOO	8	4	1	17.98	38.45	4
42	OOO	8	4	2	33.95	38.45	4

3.2.4 Selecting Designs from Clusters

The final heterogeneous system contains a core from each of the K clusters. To select a single, representative core from several in a cluster, we specify a *selection criterion.* For each cluster $k \in [1, K]$, we use a figure of merit F, either performance or efficiency, to evaluate its n_k cores. The selection criterion can either maximize reward or minimize risk; we describe each approach below.

Maximizing Reward. Selecting cores from each cluster can maximize performance or power assuming a best-case allocation, but this approach makes no provision for imperfect profiling. It also neglects contention that diverts a task from its preferred core to a sub-optimal alternative. While this criterion produces a system with the best potential performance or efficiency, it may perform poorly under typical, let alone adverse, conditions.

We describe this reward-maximizing selection criterion as argMax Max(F): from the n_k cores in cluster k, the argMax operator selects the core that maximizes the figure of merit (F) from within the cluster. In practice, the criterion typically produces extreme cores tailored for a particular application and is the approach taken by prior work [23, 54, 59].

To accommodate other applications, we can use selection criteria that reduce uncertainty albeit with lower rewards. We might select cores to target the moderate center of the application space, using argMax Mean(F) or argMax Median(F), which select a representative core to maximize the mean or median figure of merit across all applications.

Minimizing Uncertainty. Reward-centric selection criteria handle uncertainly only implicitly. These criteria assume that uncertainty in performance or efficiency may fall simply by optimizing these figures of merit less aggressively. As an extreme example of a reward-centric approach towards minimizing uncertainty, we include argMax Min(F). This conservative selection criterion opts for cores that accommodate worst-case allocations and the lowest-performance application in the suite.

In contrast, we propose selection criteria that handle uncertainty explicitly by using measures of variance. For each of n_k cores in cluster k, we calculate the variance of that core's figure of merit F across all applications. Low variance indicates that a core provides similar F across all applications. High variance suggests that improving F for one application is attained at the expense of others. We describe an uncertainty-minimizing selection criterion as argMin Var(F).

However, minimizing variance alone may sacrifice too much reward; one way to minimize variance is to select a core that provides equally bad performance for all applications. To strike a balance, a better selection criterion uses the coefficient of variation (CoV = σ/μ) and selects cores using argMin CoV(F). [1] The coefficient of variation is the standard deviation divided by the mean. Lower mean performance degrades this selection criterion, hence it minimizes uncertainty in a way that favors high-performance cores [68].

Heterogeneous Outcomes. In summary, we start with matrices that detail hardware-software interactions. We apply the *strategy framework* to make decisions that constitute design strategies. Each strategy first creates clusters of applications and cores. A representative core is selected from each cluster, creating a heterogeneous mix of cores that we refer to as the strategy's *outcome*. An outcome is a family of processor designs, which system architects can use to organize a large system (e.g., datacenter) tailored for diverse applications.

3.2.5 Ranking Heterogeneous Outcomes

The strategy framework yields many heterogeneous outcomes, but processor and datacenter architects choose only one of these to produce and procure. This choice depends on applications in the system, yet different applications prefer different outcomes. One application might prefer heterogeneous systems with at least one big core. Another might prefer systems with various small cores. The designer needs to navigate, not only the hardware design space, but also divergent application preferences for heterogeneous systems. We present a voting mechanism to reconcile these divergent preferences, and aid the designer in selecting the best heterogeneous

[1] Alternatives might also be used, for example the Sharpe ratio that is effectively the inverse of CoV, or the Sortino ratio to penalize downside risk.

outcome.

Ranked Voting for Heterogeneous Core Types. The following ranked voting (a.k.a. preferential voting) system allows a designer to reconcile divergent preferences. Outcomes are ranked based on the preferences of each application, and then aggregated by computing rank sums. This ranking mechanism is a design-time exercise to identify the outcomes that provide the best quality-of-service across many applications. The ranking balances competing application preferences, and the degree to which application preferences align determines a system's overall service quality.

Suppose the system runs requests from two applications, a_1 and a_2. A designer ranks heterogeneous systems based on the service quality of a_1 when competing with a_2 for shared resources. Thus, a ranking of outcomes based on the service quality of a_1 is different than one for a_2. We combine two sets of ranked preferences by computing rank sums. If a particular heterogeneous outcome is ranked 1st by a_1 and 10th by a_2, the outcome has rank sum 11. The mechanism behaves likewise for any a.

Applications that prefer the same heterogeneous systems will have rankings that align, and the resulting system will provide high service quality to all. However, if applications have different preferences, the rank sums will identify heterogeneous design compromises that avoid sacrificing one application more than others. We apply preferential voting to the outcomes of the strategy framework, and compare the service quality of the best-ranked outcomes in §3.4.

3.3 Experimental Methodology

To evaluate heterogeneous processor design strategies for datacenters, we deploy a comprehensive methodology. Cycle-accurate processor and memory simulations

Table 3.2: Parameters for the out-of-order cores in the design space.

Width	2	4	6	8
Phys RF	64	128	192	256
ROB	64	128	192	256
Fetch Q	24	48	72	96
Load Q	24	48	72	96
Store Q	24	48	72	96
L1 D-$	64 KB 4-way, wb			
L1 I-$	64 KB 4-way, wb			

Table 3.3: Parameters for the in-order in the design space.

Width	1	2	4
Dispatch Q	8	16	32
Store Buff	8	16	32
Forward Buff	16	32	64
Commit Buff	16	32	64
L1 D-$	32 KB 4-way, wb		
L1 I-$	32 KB 4-way, wb		

Table 3.4: Characteristics for the task streams made up of three applications from the benchmark suites: barnes, radix, and quadword (web search).

	Task Profile (Mcycles/task)	Daily Peak (tasks/min)	Weekly Peak (tasks/min)	Value (K$/month)	Scaling Factors
barnes	2972	420	1261	\$5 if $T \leqslant 200ms$ \$0 if $T \geqslant 1000ms$	$0.19 - 1.0$
radix	1326	1884	5652	\$5 if $T \leqslant 200ms$ \$0 if $T \geqslant 1000ms$	$0.79 - 1.0$
quadword	32	76827	230482	\$5 if $T \leqslant 20ms$ \$0 if $T \geqslant 160ms$	$0.25 - 1.0$

for datacenter workloads provide detailed, node-level analysis. For datacenter-level analysis, we add queueing models and allocation mechanisms for heterogeneous hardware. Collectively, this infrastructure allows us to study processor design strategies and their run-time effect on datacenters.

Processor Simulation. We use the Marssx86 full-system simulator [80], integrated with DRAMSIM2 [86], to simulate the 42 processors listed in Table 3.1. The design space is defined by four microarchitectural parameters: instruction scheduling, superscalar width, last-level cache size, and frequency. Additional structures scale in proportion to superscalar width, shown in Tables 3.2–3.3. We use McPAT/CACTI at 32nm to model power and area [61]. We set an area budget for chip multiprocessors and determine the number of cores that can fit within it.

Applications. We simulate web search queries using the open-source Nutch/-SOLR search engine. We crawl/index 50K Wikipedia documents and evaluate diverse types of queries [91]. A query type denotes whether page *content* or *title* is searched. Wildcards and searches for similar words (*near*) are supported. Negative queries

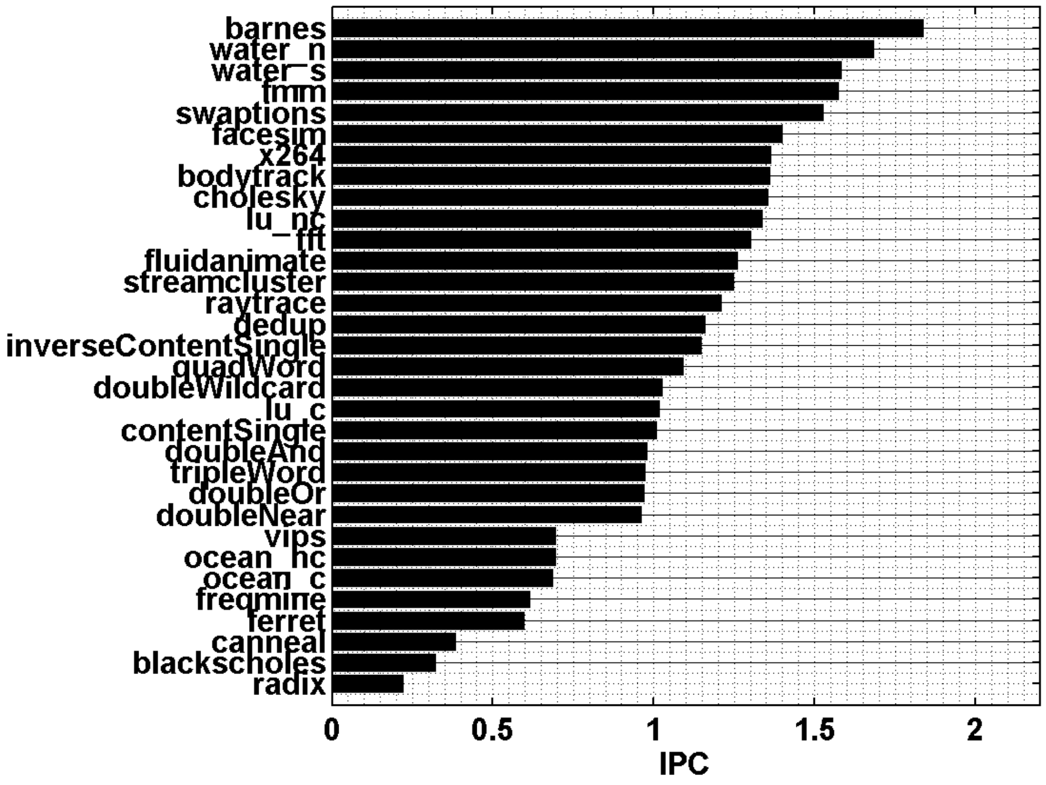


FIGURE 3.2: Performance diversity across applications (IPC on Core 42 of Table 3.1).

are denoted by *inverse*. Queries may contain multiple search terms (*single, double, triple, quad*) connected by logical operators (*and, or*).

We use checkpoints to simulate 100M instructions at regions of interest in PARSEC, SPLASH-2, and web search. Our checkpoints for web search are taken after the server has been initialized and warmed up with 100 queries.

These applications exhibit diverse performance as shown in Figure 3.2. Diverse search queries vary in computational complexity. The complex inverseContentSingle query incurs a larger performance penalty on a small core than the simple doubleOr query. Some workloads have a strong preference for high-performance cores (e.g., barnes). Others execute more efficiently on a small, low-power core (e.g., radix).

Application Task Streams. To evaluate applications in the datacenter, we organize workloads into task streams that follow known diurnal and sinusoidal activity [72]. Such patterns have been used to evaluate other datacenters [19, 39, 66].

The task arrival rate is the composite of a sinusoid with a week-long period and another with a day-long period. Amplitudes, shown in Table 4.2, are set such that load is no greater than the maximum computational capacity of a 20KW budget. We add Gaussian-distributed noise to the time series.

As is typical in elastic clouds [19, 39, 66], users specify service-level agreements (SLA) that define value as a function of response time. Without loss of generality, we use M/M/1 queues to estimate 95^{th} percentile response time. Value degrades linearly as response time increases up to some cut-off, after which computation has no value (Table 4.2). Users that derive higher value from computation are given higher priority.

Datacenter Management. To anticipate run-time effects during design, we must consider a particular management mechanism. For heterogeneous design, the mechanism must differentiate heterogeneous processing resources when allocating them to diverse applications. Any management mechanism may be used within our framework.

Without loss of generality, we use markets to allocate hardware in our evaluation. Markets have several compelling properties. First, markets provide an attractive interface as users express value for performance. Second, market agents use value functions to automatically bid for hardware on behalf of users, thus shielding users from the complexity of heterogeneous hardware. Finally, markets are cleared to maximize welfare via hardware allocations.

Our market allocates heterogeneous processors periodically (e.g., every 10 minutes) to serve diverse application tasks. An allocation might span many heterogeneous core types. The market invokes CPLEX, a mixed integer program solver, to determine an allocation that efficiently meets quality-of-service targets. We refer the reader to prior work for detail [19, 39, 66].

We apply the market to the heterogeneous outcome of a design strategy from Figure 3.1. We configure systems with a 20KW budget, which is approximately the power dissipated by two datacenter racks. Server power is processor power plus 65W, which accounts for memory, motherboard, network interface, and disk [84]. These servers are integrated into a heterogeneous chassis (e.g., IBM PureFlex). Datacenter power usage effectiveness (PUE) is 1.6. Energy costs $0.07 per kWh.

Scaling Factors. The market uses scaling factors to account for performance differences in heterogeneous systems[39]. For each application, these factors report performance for each core relative to that of the highest-performance core.

If scaling factors for an application are uniformly 1.0, its tasks are indifferent to core microarchitecture. In contrast, an application with greater diversity in scaling factors has stronger preferences for particular core types, perhaps its tasks are more compute intensive. Table 4.2 shows a broad range of scaling factors. Radix is least sensitive to core type whereas barnes and quadword search queries are more sensitive.

Scaling factors may be obtained from profiling tools, such as gprof [37], VTune [47], or OProfile [77]. 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 [85]. Given sparsely sampled profiles, statistical machine learning can fit models and predict scaling factors [104].

System Model. Our datacenter model envisions applications, made up of task streams, running on heterogeneous processors, where heterogeneity exists across racks. Rack-level heterogeneity allows for system-wide resource management. A resource allocator explicitly directs applications to compute resources by steering tasks to the right rack. In addition, system designers can deploy heterogeneous resources at a fractional share that matches an expected application mix. By fractional share we refer to the portion of the total power budget provisioned to each type of heterogeneous resource. In §3.5.1 we evaluate each heterogeneous outcome and its sensitivity to fractional shares.

3.4 Designing for Manageability

In this section, we compare the strategies that produced the 50 unique outcomes listed in Tables 3.5–3.6. An outcome is a set of heterogeneous core types, and it is the product of a design strategy that is applied to the benchmark suite and processor design space. We evaluate heterogeneous design strategies based on manageability, which we quantify as service quality across co-running applications using a realistic heterogeneous resource manager. We simulate equal-power datacenters equipped with systems that house heterogeneous processors across racks.

The framework of strategies detailed in §3.2 produces the outcomes in Tables 3.5–3.6, where each entry is a tuple of identifiers that map to the microarchitectures in Table 3.1. Identifiers in the `1 - 18` range designate in-order, power efficient cores, whereas `19 - 42` are out-of-order, high performance cores. For example, heterogeneous outcome `42|37` is made up of two types of out-of-order cores, and is the product of a MaxMax(BIPS) selection strategy from two clusters of applications grouped by behavior. The systems we evaluate range from homogeneous high-performance big cores, `42`, to highly heterogeneous datacenters composed of both big and small cores, `11|24|30|42`.

Allocation and run-time risks are a function of user competition, thus we study application pairs that exemplify three, distinct contention scenarios:

- **barnes|radix** exhibit complementary preferences

Table 3.5: Heterogeneous outcomes from BIPS strategies.

	K=2	**K=3**	**K=4**
Application Behavior			
MaxMax	42|37	42|37|40	41|42|37|40
MaxMean	41|42	41|42	41|42
MaxMin	42	42	42
MinVar	5|2	5|2	5|2
MinCoV	5|4	5|2|24	5|2|24
Application Performance			
MaxMax	41	41	41
MaxMean	41	41	41
MaxMin	35|41	41	29|35|41
MinVar	6|1	5|2	1|2||14|27
MinCoV	1|22	5|20|31	1|2|14|27
Architecture Performance			
MaxMax	17|42	12|17|42	17|24|30|42
MaxMean	17|41	12|17||41	17|23|41|38
MaxMin	11|42	5|11|42	11|24|30|42
MinVar	5|21	5|11|21	5|21|25|27
MinCoV	5|21	5|11|21	5|21|29|25

Table 3.6: Heterogeneous outcomes from BIPS^3/W strategies.

	K=2	**K=3**	**K=4**
Application Behavior			
MaxMax	29| 35	29|35	29|35
MaxMean	29	29	29|35
MaxMin	5|23	5|23|29	5|23|29
MinVar	14	14	14
MinCoV	14	14	14
Application Efficiency			
MaxMax	29	29|35	23|29
MaxMean	29	29|35	29|35
MaxMin	5|35	5|29	5|29|35
MinVar	14	14	14|38
MinCoV	14	14	14|42|38
Architecture Efficiency			
MaxMax	29|42	22|29|30	11|23|29|37
MaxMean	29|42	23|22|29	11|23|29|37
MaxMin	5|23	5|23|29	5|23|29|37
MinVar	14|21	14|24|37	14|22|38
MinCoV	14|21	14|24	14|22|38

- **barnes|quadword** contend for big, high-performance cores
- **radix|quadword** contend for small, low-power cores

Across all three types of contention, we find that risk-aware strategies at design-time improve service quality at run-time. Results also show that ranked voting at design-time reconciles competing user preferences for hardware at run-time.

Mitigating Risk During Design. Prior approaches to heterogeneous design

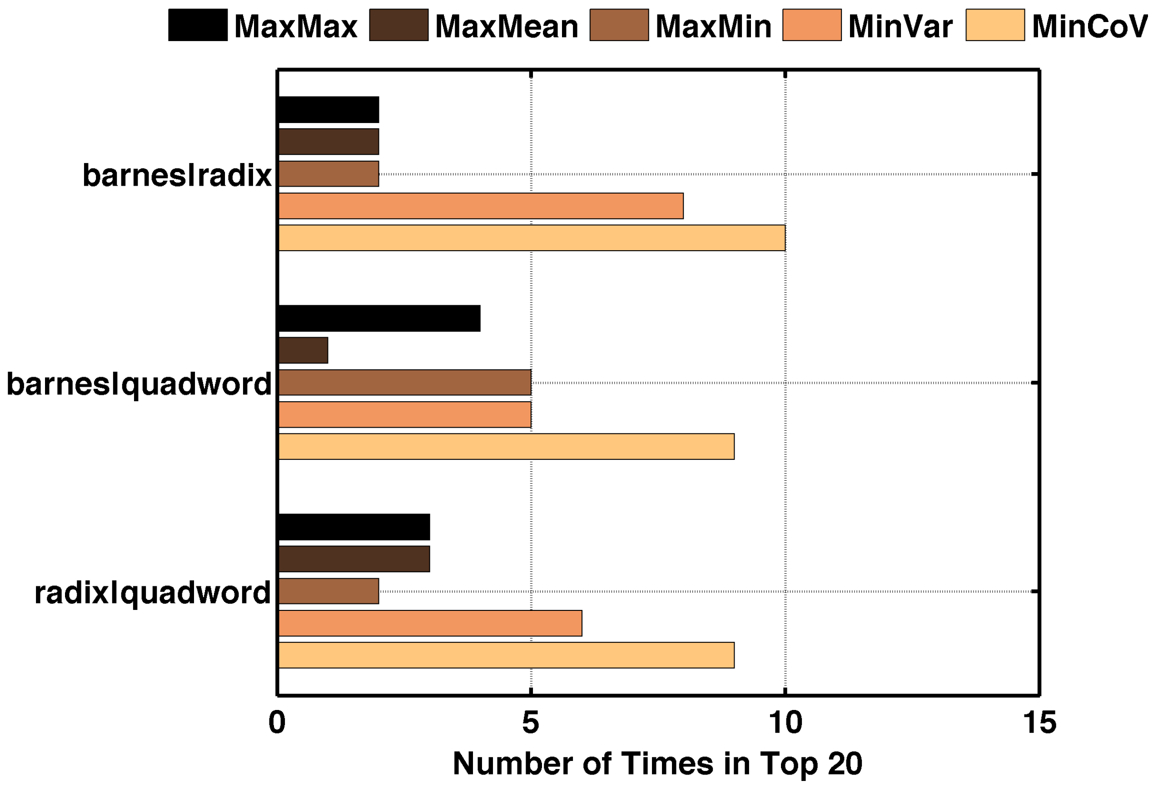


FIGURE 3.3: Risk-aware strategies are more likely to produce outcomes with the best service quality.

identify core types that maximize performance or efficiency for the benchmark suite, and hence aim only to maximize rewards. Such approaches correspond to MaxMax and MaxMean strategies in our framework. For these strategies to deliver expected performance and efficiency, users must receive the cores best suited to their applications. Yet, such ideal allocations are difficult to obtain in the real world, where applications compete for shared hardware.

In contrast, strategies that anticipate risk by measuring performance variance at design-time are more robust to dynamic hardware allocation at run-time. Selecting cores using MinVar or MinCoV criteria accounts for allocation risk in heterogeneous systems. Alternatively, the MaxMin criterion optimizes heterogeneity for worst case management scenarios when a user receives the least ideal processor type in the system.

To quantify these effects, we examine the 20 best outcomes (e.g., `14|22|38|39`, ...) based on service quality across applications. Figure 3.3 shows all the design strategies that produce the 20 most effective heterogeneous systems for barnes|radix. These strategies are desirable from the perspective of manageability. For example, there exist 24 strategies that provide the 20 heterogeneous outcomes with the best service quality when barnes and radix are co-runners.[2] Of these 24 strategies, 83% of them account for risk either by optimizing performance variance with MinVar or MinCov, or by optimizing for worst-case scenarios with MaxMin. Similarly, risk-aware strategies account for more than 70% of the strategies that produce the top 20 outcomes for barnes|quadword and radix|quadword.

Case for Risk-Aware Design. Three of the selection criteria form a part of risk-aware design strategies: MinCoV, MinVar, and MaxMin. These strategies balance rewards in performance and efficiency against risks in heterogeneous resource allocation. As Figure 3.3 shows for barnes|radix, the 20 outcomes with the best service quality are most likely the product of risk-aware design.

MinCoV is the best at balancing risk and reward as it selects cores to moderate variance but not at the expense of average performance. Thus, cores with higher performance are included in the heterogeneous outcome. For example, MinCoV BIPS produces `5|2|24` while MinVar BIPS more conservatively produces `5|2`.

MinVar minimizes performance variance, which tends to favor small cores that provide uniformly low performance. The advantage of small cores is power efficiency, which allows more servers to fit within a fixed power budget. More servers translate into greater throughput and fewer service quality violations. If bigger cores are

[2] Note that the number of good strategies exceed the number of good outcomes because multiple strategies may produce the same outcome.

needed for performance, MinVar provides them in highly heterogeneous systems (e.g., when $K = 4$, MinVar BIPS produces `1|2|14|27`).

Finally, MaxMin can be considered a risk-aware design strategy. This strategy favors big cores to ensure service quality in worst-case allocation scenarios by maximizing minimum performance. MaxMin accommodates the most demanding applications with a high-performance core (e.g., `42` for BIPS) or with a high-efficiency core (e..g, `5|23|29` for BIPS^3/W).

Limitations of Risk-Agnostic Design. In contrast to risk-aware strategies, MaxMax and MaxMean strategies rarely lead to a heterogeneous system with good service quality. These strategies identify the very best heterogeneous processor mixes for the complete application suite. When a subset of these applications actually use the resulting systems, their lack of flexibility degrades service quality. For example, radix|quadword prefers small, low-power cores. Yet MaxMax will produce several big, high-performance cores as it tries to maximize best-case performance for the original set of 32 applications.

3.5 Classifying Sources of Risk

Our evaluation has thus far determined that the best design strategies are risk-aware. Next, we consider the sources of risk in the top-ranked outcomes and observe that higher reward comes at higher risk (§3.5.1). Another metric of interest to system architects is the efficiency of top-ranked outcomes. We compare the efficiency of heterogeneous outcomes to an optimal case where each application runs on the efficiency-maximizing core in the design space. We find that heterogeneous systems are most efficient when running complementary applications (§3.5.2).

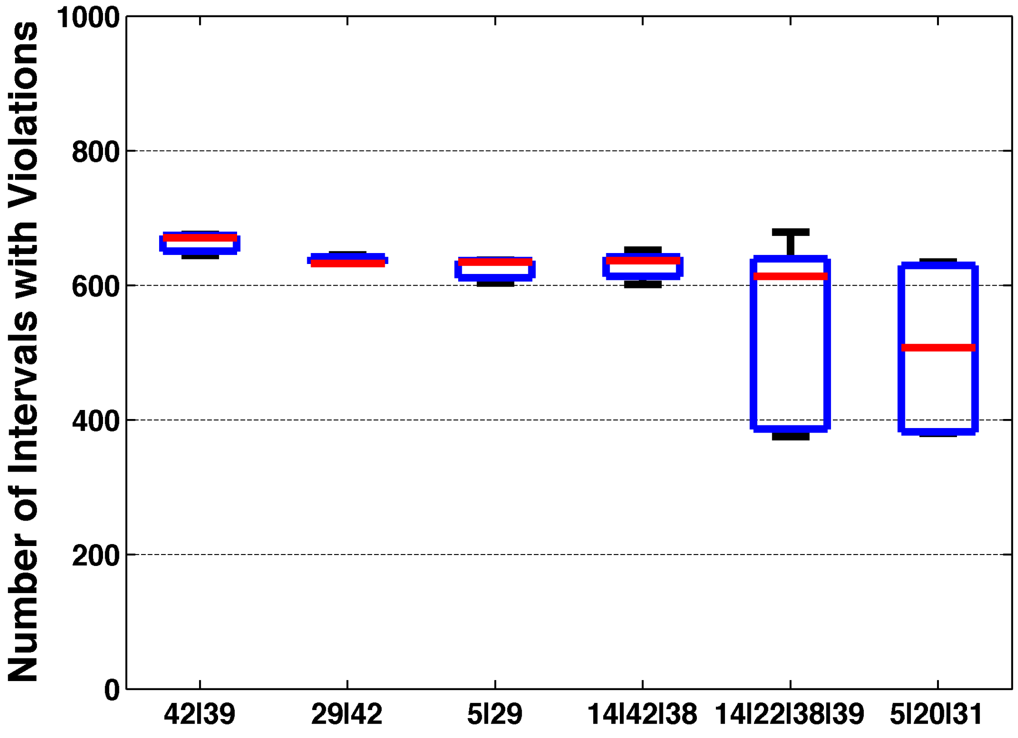
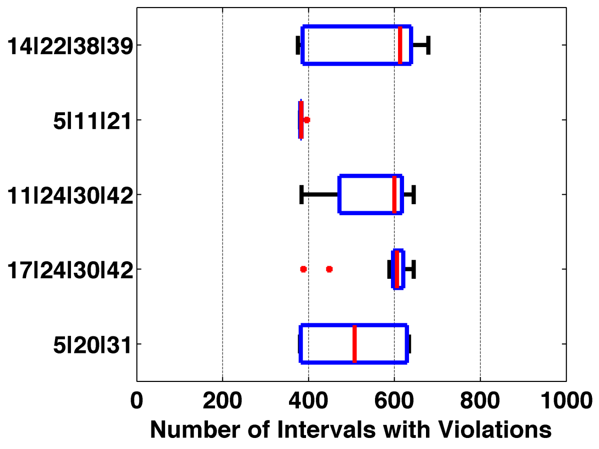


FIGURE 3.4: Heterogeneous system 14|22|38|39 exhibits more risk yet reduces response time violations by 50% relative to low-risk system 42|39 (barnes|radix).

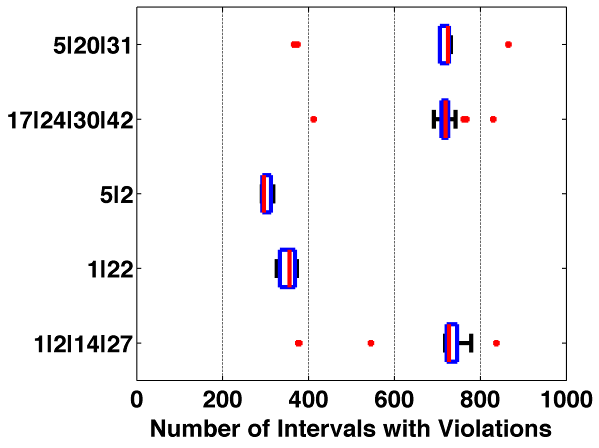
3.5.1 Incurring Risk to Increase Reward

Heterogeneity allows a system to provide specialized resources for subsets of applications, and thus more effectively invest a limited power budget than systems with low diversity. The reward of heterogeneity is an improvement in performance. In a datacenter, this reward is a reduction in the number of allocation periods that incur response time violations.

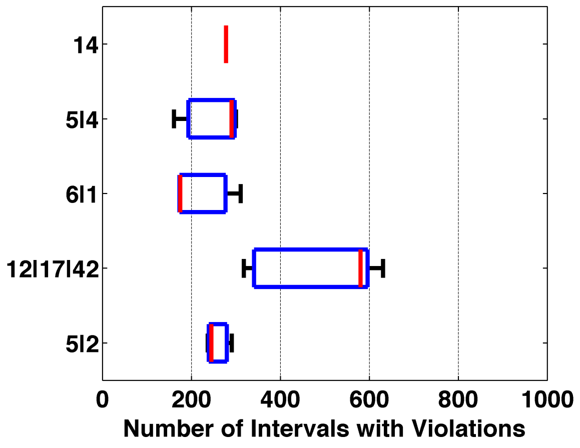
However, it is difficult to provision diverse resources in proportions that match the application mix. A strategyâĂŹs outcome defines a set of heterogeneous cores, but not their organization in the system. Our evaluation thus far has assumed that the systemâĂŹs power budget is divided amongst heterogeneous core types such that



(a) **barnes|radix**



(b) **barnes|quadword**



(c) **radix|quadword**

FIGURE 3.5: Quality-of-service as we vary fractional shares of cores for top 5 ranked outcomes.

service quality is optimized. Yet identifying each core typeâĂŹs share of the power budget is a design space of its own. We explore this space and assess service quality.

Risk. We quantify risk-reward trade-offs by varying the fractional share of the power budget that each processor type is allocated. For barnes|radix, Figure 3.4 illustrates service quality for different heterogeneous outcomes (x-axis) at different shares of those types (boxes). The x-axis spans varying degrees of heterogeneity, from the conservative system `42|39` to increasingly heterogeneous systems.

For an application mix, the box shows the effect of different fractional shares. Given K heterogeneous cores, we evaluate all combinations of $\frac{1}{K}$-sized fractions within a fixed power budget (e.g., 20KW). For example, when $K = 2$, core types can be organized into fractions of 1:0, $\frac{1}{2}$:$\frac{1}{2}$, and 0:1. Across these different shares, boxes illustrate the variance in service quality, measured by the number of allocation periods that violate response time targets (y-axis).

Greater heterogeneity leads to higher variability in service quality and system risk. Figure 3.4 shows increased risk for outcomes `14|22|38|39` and `5|20|31`. Note that heterogeneity and system risk is not simply a function of the number of core types. Although both outcomes have three core types, `5|20|31` exhibits lower variance than `14|42|38`. Cores `42` and `38` only differ in L2 cache size and hence the variation in application performance across the two cores is small.

Reward. Despite the increase in system risk, the reward is a significant improvement in service quality. The highest ranked outcome for barnes|radix is `14|22|38|39`, which is also most risky. If the deployed share is well-matched to barnes and radix load, we observe only 375 allocation periods in which an application violates the service target. This is a 50% reduction in intervals that suffer violations, compared to the 675 violations observed on a more conservative outcome `42|39`. We also see that in the worst case, when the fractional share of the aggressively heterogeneous outcome `14|22|38|39` is poorly suited to the application mix, the number of violations is no worse than that of the conservative outcomes in Figure 3.4.

barnes|radix. Most top ranked outcomes in Figure 3.5a pose significant system risk, as shown by the span of the boxes as we vary the shares at which each outcome may be deployed. The exception is `5|11|21`, which we prefer since it provides high service quality at low system risk. In contrast, `17|24|30|42` is a particularly poor option; the fractional share that provides the best service is an outlier.

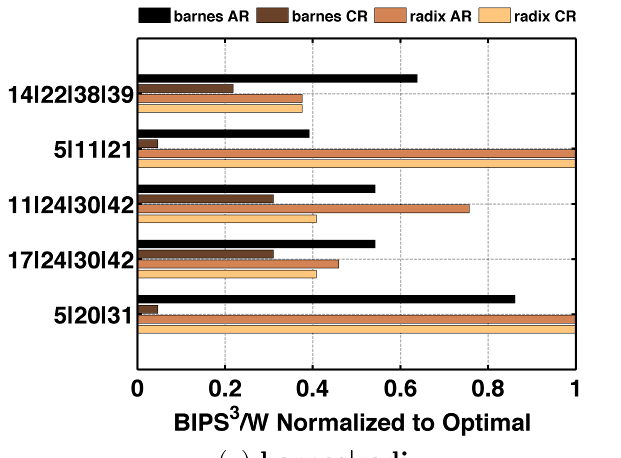
barnes|quadword. Most heterogeneous outcomes are capable of high service quality. But `5|2` or `1|22` clearly provide that service quality at lower risk. Selecting poor fractional shares is too likely in the other three outcomes.

radix|quadword. These applications contend for small, efficient cores and hence prefer outcomes with low heterogeneity. The top ranked outcome is, in fact, homogeneous (Figure 3.5c). However, outcomes `5|4` and `6|1` may be better choices. In the best case, if the fractional shares are well-matched to application mixes, heterogeneity improves service quality. Only 161 allocation periods see service violations, a 42% reduction compared to the 278 violations observed on the top-ranked homogeneous system. Moreover, despite system risk, the worst-case service on the heterogeneous system is no worse than that of the homogeneous one.

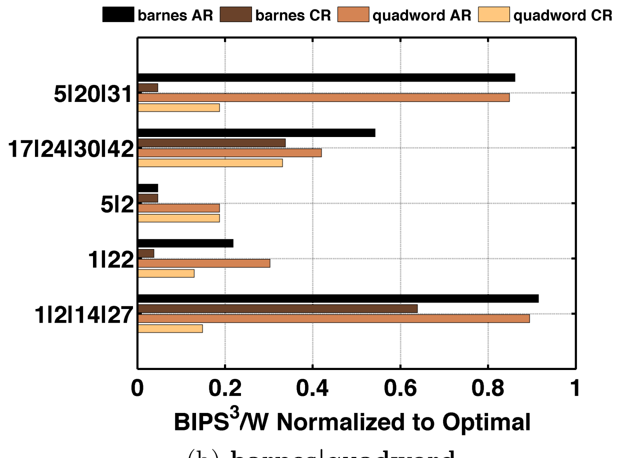
3.5.2 Quantifying Risks to Efficiency

The previous section studied performance sensitivity to the number of each processor type and task mix. Next, we consider variability in energy efficiency and two additional sources of risk. For this evaluation, we define the upper bound on efficiency as the application running on its BIPS^3/W-maximizing core from the complete design space.

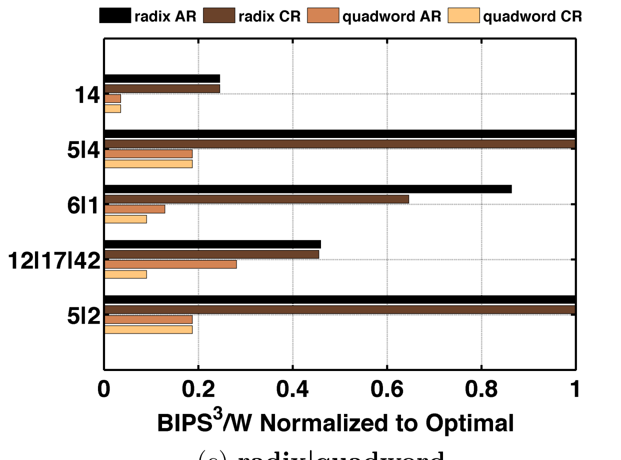
An application may not realize the upper bound on efficiency and instead run on a less efficient core for two reasons. First, an application's most efficient core may not



(a) **barnes|radix**



(b) **barnes|quadword**



(c) **radix|quadword**

FIGURE 3.6: Efficiency for top ranked outcomes of heterogeneous cores. The BIPS^3/W-maximizing core for each application may not occur in an outcome due to application risk (AR) or may not be allocated to that application due to contention risk (CR).

be available in the heterogeneous outcome chosen for the system. In this scenario, efficiency is lost due to application risk. Second, an application's most efficient core may be present but allocated to another application, which is contention risk.

In Figure 3.6, we report efficiency when the application runs on the best core in the heterogeneous outcome. Any efficiency lost to application risk is due to choices during design ("AR"). We also report efficiency based on the actual allocation of cores in the market mechanism. Any further efficiency loss is due to contention during allocation ("CR"). Carefully selected heterogeneous designs provide 80% of the BIPS^3/W upper-bound. On the other hand, contention can cause systems to realize only 20% of this potential.

barnes|radix. Figure 3.6a shows efficiency losses due to application and contention risk. Radix executes at near-optimal efficiency for a few of the top ranked outcomes (i.e., `5|11|21` and `5|20|31`). Reconciled rankings closely align with radix's ranking. If the second- and third-ranked configurations are chosen, efficiency losses are zero.

On the other hand, barnes loses efficiency to both application and contention risk. The heterogeneous configurations in the reconciled rankings do not represent barnes's preferences; radix introduces big cores into the system, which determines barnes's efficiency loss from application risk. Moreover, these big cores are most often allocated to barnes, which ensures service quality but degrades run-time efficiency.

Since the top-ranked configurations provide similar quality of service, the system architect can opt for the configuration that maximizes efficiency. Although not shown, the sixth-ranked configuration provides better efficiency for both applications without further harming service quality.

barnes|quadword. These applications illustrate efficiency losses primarily from application risk. When these two applications are contending for cycles, neither will likely execute at near-optimal efficiency. Figure 3.6b shows that several of the configurations most highly ranked for quality-of-service achieve performance by sacrificing at least 60% of the efficiency available in the design space. Although the first and fifth configurations retain most of the efficiency during design clustering, contention means that actual efficiency is often less than 20% of the upper bound. This mix is particularly difficult to accommodate within the fixed power budget given that both applications incur benefit from power-hungry cores.

radix|quadword. For both radix and quadword queries, Figure 3.6c indicates that allocated efficiency matches the best possible designed efficiency. Nearly all the cores in various configurations are from the low-power, in-order end of the design spectrum (cores `1-18`). Since these cores are similar, efficiency is not significantly impacted by allocation decisions. Only the fourth configuration includes a high-performance design (core `42`), and only this configuration suffers any significant efficiency loss from allocation decisions.

3.6 Related Work

Heterogeneous Chip Multiprocessors. Kumar et al. consider existing cores drawn from multiple design generations [53, 55]. Alternatively, Kumar et al. exhaustively simulate and search a space with hundreds of designs to maximize performance subject to power and area constraints [54]. Choudhary et al. use FabScalar to evaluate synthesizable core designs and evaluate cores in a heterogeneous mix [23]. This particular strategy maps approximately to our architecture-driven clustering with a MaxMean selection criterion on performance.

Lee and Brooks also explore a large design space, using regression models to explore performance and power trade-offs tractably, and use K-means clustering to optimize [59]. Strozek and Brooks similarly study clustering strategies for heterogeneous embedded systems [94]. This strategy maps approximately to our architecture-driven clustering with a maxMean selection criterion on BIPS^3/W efficiency.

Prior heterogeneous strategies do not produce designs that are robust to system integration, performance risk, and contention risk. These prior efforts take a particular strategy whereas we explore an broad space of strategies. Unlike prior work, we anticipate run-time manageability at design-time.

Heterogeneous Datacenters. While much prior work in distributed systems have considered diverse tasks and heterogeneous virtual machines, the underlying processors are often homogeneous by design. At present, heterogeneity in datacenters is modest and involves multiple generations of processors [70, 76] or processors operating at different frequencies [66]. However, studies of datacenter software on diverse hardware motivate greater heterogeneity due to the potential for efficiency [30, 64, 84].

Heterogeneous Management. To anticipate run-time manageability, we deploy a framework that uses a market mechanism to assign cores to tasks [19, 39, 66]. These mechanisms operate at datacenter scale, examining application preferences for hardware and allocating cores to task streams.

A much larger body of work studies scheduling in heterogeneous chip multiprocessors. Much of this work focuses on profiling and thread migration. Scheduling diverse software to heterogeneous hardware might account for memory-level parallelism [98], instruction-level parallelism [10, 52], resource demands [5, 20, 89, 92, 93], thread age [57], load balance [63], or hardware faults [13, 103]. Scheduling is simplified when big or small cores are used for a specific purpose. Big cores can accelerate critical sections in parallel computation [95] while small cores can efficiently support the operating system [74] or managed software [17].

3.7 Summary

Our work is the first to define a taxonomy of the risks that heterogeneous systems face due to the current divide between design and management. We present a framework of design strategies, and for the first time include risk-aware strategies that complement traditional performance or efficiency maximizing strategies. We evaluate these strategies under diverse datacenter contention scenarios, and find them to reduce service quality violations by 50% relative to traditional approaches to heterogeneous design.

4

Appraising Fairness in the Market

In the resource allocation mechanism of Chapter 2 we describe a market mechanism that maximizes welfare when dividing available processing resources to service incoming applications. Welfare, which is value net cost, is a measure of the efficiency of an allocation. In fact, similar resource allocators typically maximize for other measures of efficiency and performance. Whether the manager is navigating heterogeneity or mitigating contention in a shared datacenter, performance has been the primary objective. Beyond performance, however, shared datacenters require new policies and mechanisms for fairness.

In this chapter, we present a new datacenter management mechanism that *fairly* allocates processors to tasks with sophisticated performance objectives. Many definitions of fairness exist in systems and economics research. Rather than equate fairness to equal slowdowns amongst applications in a shared system [75], we define fairness in game-theoretic terms [99]:

An allocation is fair when each user weakly prefers her own allocation to that of every other user. In other words, no user envies the allocation of another.

Conceptually, a system that does not induce envy between users is fair. In such a system, resources have been allocated in an equitable manner so that no user has cause to complain. Envy-free systems encourage user participation in shared systems and promote allocation stability.

We seek to mitigate envy for users and tasks with strict performance objectives. Prior efforts have examined fairness for throughput-oriented tasks with simple Leontief utilities [36]. In contrast, we examine latency-sensitive tasks with articulate piecewise-linear utilities. These expressive utility functions more accurately represent realistic service-level agreements. In this setting, reducing envy poses new challenges in resource allocation.

Envy-freeness is a strict definition of fairness, which inevitably has a price. If a management mechanism neglects envy, it can optimize performance by searching an unconstrained space of allocations. Such a strategy efficiently deploys datacenter hardware to the software that needs it most. However, if the mechanism instead constrains envy, efficiency falls. By comparing welfare-maximizing and envy-minimizing mechanisms, we analyze the price of fairness.

We find that the price of fairness is prohibitively high when a datacenter system is highly loaded. For such settings, we present an alternative to envy-free allocation – ϵ-envy-freeness, which is parameterized by the amount of envy permitted in datacenter allocations. We vary ϵ to understand the trade-offs between fairness and efficiency.

We assess these trade-offs for specific datacenter applications and architectures. We deploy and characterize web search. The management mechanism first classifies latency-sensitive queries by their length. It then allocates processors to maximize welfare while mitigating envy across query types. This chapter makes the following contributions:

- **Examine Fairness for Latency-Sensitive Tasks.** To describe value for processors, latency-sensitive tasks require expressive piecewise-linear utility functions. In this setting, prior algorithms do not allocate fairly. (§4.2)

- **Introduce Fairness to Markets.** Market mechanisms typically allocate processors to maximize efficiency. We add fairness into the system, introducing constraints that account for envy when allocating processors. (§4.2)

- **Enforcing Fairness for Heterogeneous Tasks.** Fairness for end users require fairness for diverse application tasks. We take web search as a case study, classifying queries according to length and complexity. (§4.3)

- **Quantifying the Price of Fairness.** For search queries, welfare-maximizing processor allocations are 1.5$\times$ more efficient than envy-free ones. Allowing a tunable amount of envy ϵ improves efficiency. (§4.4)

- **Extending Fairness to Heterogeneous Processors.** A market mechanism that maximizes welfare subject to constraints on envy generalizes to heterogeneous processors. Big and small cores are allocated to mitigate envy. (§4.5)

4.1 Background

Resource allocation in datacenters must navigate several (often conflicting) targets, accounting for service quality, energy-efficiency, and application preferences for various resources. Fair resource allocations are important for systems that provide computation for a variety of applications. Datacenters are such systems, both in the private setting where many services that are of importance to a company must share resources, and in the public setting where a provider aims to guarantee fair access to resources that users are renting.

Even systems that optimize performance may wish to provide a measure of fairness. A fair system would allocate resources to maximize throughput while providing each user a minimum allocation. In other words, the system optimizes throughput with safeguards against starvation. These objectives are sensible even for market-based allocation in which users pay for datacenter resources [2, 19, 39, 66]. Fairness in this setting leads to a capitalist market with a social safety net, which is often desirable.

4.1.1 Efficiency versus Fairness

We define allocative efficiency as the sum of user utilities. Each user's utility depends on measures of performance, such as latency and throughput. Within a shared system, there exists an optimal allocation that maximizes efficiency. There also exist Pareto-efficient allocations for which increasing a user's utility necessarily reduces another's. Beyond these special cases, there exist many other allocations of varying efficiency.

Inevitably, efficiency is degraded by fairness. We define fairness using concepts from economic game theory, which consider sharing incentives and the degree of envy in multi-agent systems. First, fairness matters only if a system provides an incentive for users to share. Without such incentives, users would not participate in the system. An allocation mechanism provides sharing incentives when each user prefers her allocation to an equal division of resources; see condition 4.1.

$$\text{SI}: \quad V_A(X_A) \geqslant V_A(\mathbf{X}/2) \qquad V_A(X_A) \geqslant V_A(\mathbf{X}/2) \tag{4.1}$$

$$\text{EF}: \quad V_A(X_A) \geqslant V_A(X_B) \qquad V_B(X_B) \geqslant V_B(X_A) \tag{4.2}$$

$$V_* \leftarrow \text{value of allocation to } *$$

$$X_* \leftarrow \text{machines allocated to } *$$

$$\mathbf{X} \leftarrow \text{number of machines in the system}$$

Second, a fair system mitigates envy between users. In a special case, envy-free (EF) allocations are those for which no user prefers another's allocation; see condition 4.2. Beyond the intuitive links between envy and fairness [99], EF is desirable because it produces stable allocations in which no user wishes to trade. In practice, an EF allocation may be infeasible or may demand large trade-offs in efficiency. Yet systems that cannot eliminate envy may still find ways to reduce it.

These concepts of fairness pose interesting questions for datacenters and distributed systems. For example, dominant resource fairness (DRF) fairly divides multiple resource types to heterogeneous applications [36, 79]. Based on the observation that each application's performance is determined by a dominant resource, the DRF mechanism implements max-min fairness to equalize dominant shares. In this setting, max-min fairness guarantees sharing incentives and envy-freeness.

4.1.2 Expressive Utility Functions

The strength of a system's game-theoretic guarantees depends on users and their utility functions. In practice, throughput- and latency-oriented users employ fundamentally different utility functions. Strong and attractive game-theoretic properties have been demonstrated for throughput-oriented applications [36, 79]. But latency-sensitive applications and their expressive utility functions pose new challenges.

First, consider Figure 4.1 and utility functions for throughput-oriented applications. The x-axis is the number of allocated resources (e.g., processor cycles) and the y-axis expresses the value of possible allocations to an application. Throughput-oriented users experience better service as more resources are allocated. One example is the Leontief utility function [36]. Suppose each task or virtual machine requires 2 units of a resource. As a user receives more resources, she is able to launch more virtual machines.

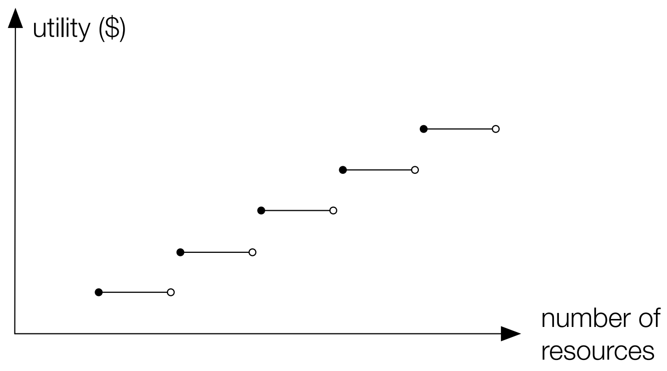


FIGURE 4.1: Piecewise-Uniform Utility. Utility increases with the number of indivisible resources.

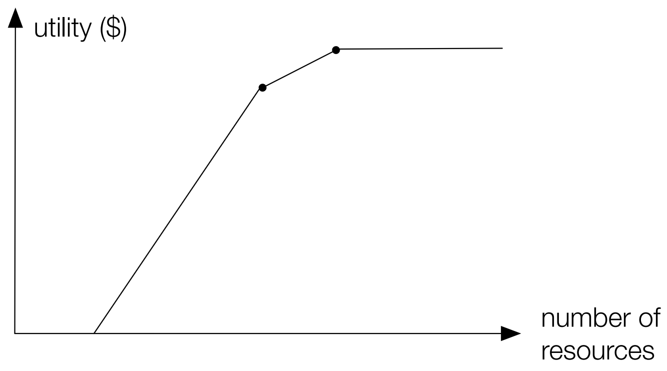


FIGURE 4.2: Piecewise-Linear Utility. Utility is zero when given insufficient resources. Utility does not increase beyond the resources required to meet a performance target.

On the other hand, the preferences of latency-sensitive applications are best modeled with piecewise-linear utility functions [19, 39, 66]; see Figure 4.2. An allocation is of no value unless it delivers some minimum performance. Value increases as the allocation grows and service improves. Eventually, the application is completely satisfied with its service and derives no value from additional resources. Piecewise-linear utilities can be used to approximate any valuation function [56].

The difference between the two types of utility functions is as important as the difference between throughput and latency. For a latency-sensitive application, allocating either an insufficient or an excessive amount of resources is undesirable.

Without an expressive piecewise-linear function, a user may be forced to pay for resources that are inadequate to meet its service-level agreement. Similarly, allocating more resources than necessary is inefficient and costly to the system. In contrast, throughput-oriented applications in the DRF model always benefit from more resources (i.e., there is always another task to execute).

4.2 Fair Market Mechanism

Max-min fairness applies naturally to throughput computing, which has a relatively simple utility function. Each software task requires a particular number of resources and task throughput increases with the allocation. Such linear or piecewise-uniform utility functions are amenable to max-min fair allocation and produce strong, desirable game-theoretic properties.

Beyond throughput computing, datacenters must accommodate latency-sensitive applications with strict service quality targets. Applications that receive an insufficient resource allocation will violate service targets and provide no user utility [84]. With expressive piecewise-linear utility functions, allocating resources fairly is challenging [15, 21, 24]. We compare throughput and latency-sensitive models of computation to describe the difficulties of fair allocation for the latter.

4.2.1 Expressive Utilities and Fairness

Using piecewise-linear utility functions has significant implications for fairness. In this setting, unfortunately, max-min solutions (e.g., DRF) no longer guarantee fairness properties, like envy-freeness and Pareto efficiency. The difficulty lies in the regions where the user's utility does not increase as the allocation increases. We provide examples to illustrate how max-min allocations no longer satisfy fairness criteria.

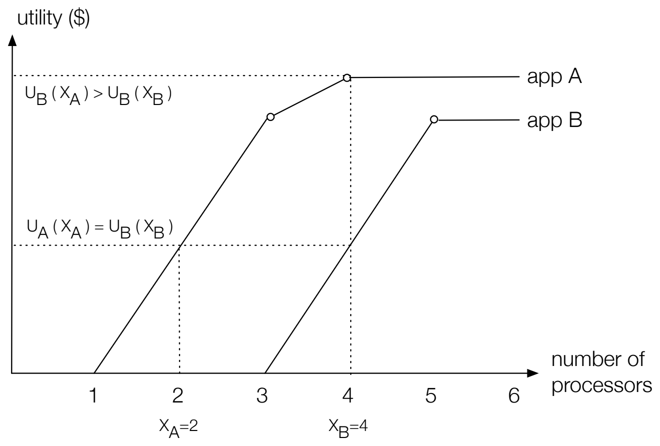


FIGURE 4.3: An example of violating Envy-Freeness with piecewise-Linear utility functions. Max-min fairness for utilities is not EF; U_A is higher with X_B.

Max-Min Allocation. To understand the limitations of max-min allocation, we first describe the underlying rationale and an algorithm for its implementation. Formally defined, a max-min fair mechanism maximizes the allocation to each user i subject to the constraint that an incremental increase in i's allocation does not cause a decrease in another user's allocation that is already smaller than i's [102]. Conceptually, such fairness is achieved when the most poorly treated user is prioritized when allocating an available resource [11]. These notions are drawn from much prior work in max-min control flow for communication networks.

Iterative mechanisms for max-min allocation assign resources to users incrementally. Note that the allocation mechanism can define max-min with respect to allocations or utilities. At the beginning of each iteration, the mechanism identifies the user with the smallest allocation or the lowest utility. This user then receives an incremental allocation. After this update, the next iteration begins. Such mechanisms have been applied to fairly allocate network bandwidth [11, 102] and have been generalized to multiple resources [36, 79].

Violating Envy-Freeness. Suppose the max-min algorithm allocates for fairness across agent utilities. We show that the resulting allocations cannot guarantee fairness when applied to piecewise-linear utility functions. Again, consider two applications, A and B, with utility functions in Figure 4.3. A max-min algorithm identifies X_A and X_B such that $V_A(X_A) = V_B(X_B)$. In this solution, $X_B > X_A$ and we see that A envies B's allocation. A would obtain more value from swapping the allocations. In other words, the allocation is not envy-free since $V_A(X_A) \not\succ V_A(X_B)$.

Although these examples are not exhaustive, they serve as clear indicators that algorithms for max-min fairness do not extend to expressive utility functions. In fact, guaranteeing fairness for more expressive utility functions is a hard problem. Theoretical challenges continue to motivate interesting work in computational economics and theory. Economist and theorists are considering this problem from an abstract perspective.

In contrast, our work is the first to link fairness for expressive utility function to an important class of applications in distributed systems: latency-sensitive tasks. We have shown how expressive utility functions are needed when applications have service quality targets, a real-world setting. We will now discuss our approach to guaranteeing fairness within a market-based resource allocator.

4.2.2 Fairness and Markets

In 1968, Sutherland proposed a market for time slots for a shared PDP-1 computer at Harvard [96]. Over the past 40 years, however, market dynamics for computing resources have become far more complex [2]. Despite recent advances in game-theoretic fairness [36] and welfare-maximizing markets for datacenters [19, 39, 66], the trade-offs between fairness and welfare are not yet understood in these settings.

Market Overview. We use markets to allocate hardware and maximize system welfare. Markets have several attractive properties. First, markets provide a clean abstraction to users who simply specify their utility for performance. Second, proxy

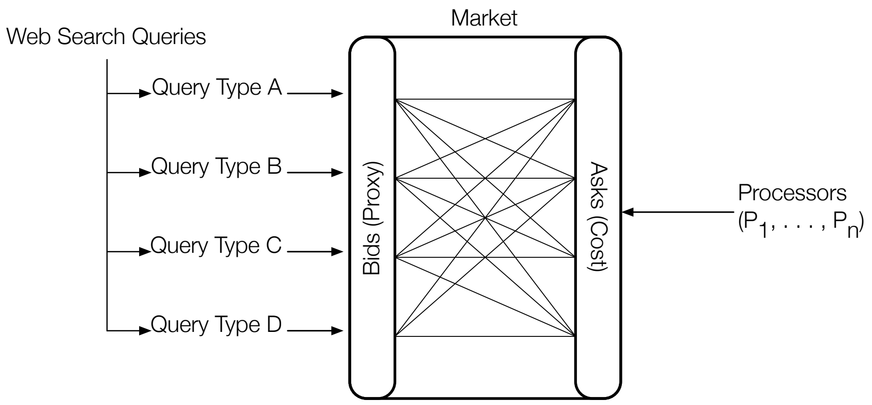


FIGURE 4.4: Market Overview: Users report value derived from performance. Proxy agents bid on behalf of users for computational resources. Market clears and allocates hardware.

agents represent users within the market, automatically profiling and bidding for hardware thereby absolving users of these burdens. Third, the market clears to maximize welfare, which corresponds to throughput in the datacenter setting.

We consider the market illustrated in Figure 4.4, examining fairness for latency-sensitive users. Users employ expressive utility functions to specify value for computation. Value increases with performance and performance increases with the hardware allocation. Proxy agents automatically profile performance to determine how to bid for a hardware allocation.

Resources are allocated periodically (e.g., every ten minutes). At the beginning of every period, bids are collected and compared against hardware costs, which include operational and amortized capital costs. Operational costs depend on hardware power and electricity prices. Capital costs include the price of servers depreciated over their lifetime. In the market, welfare is defined as aggregate user value minus datacenter hardware cost. We use welfare and efficiency interchangeably to describe the market's maximization objective.

Market Clearing Mechanism. Previously proposed market mechanisms maximize throughput while meeting latency objectives. Chase et al. prototype a market for allocating cycles to web servers [19]. Lubin et al. extend the market to accommodate dynamic voltage and frequency scaling [66]. Most recently, Guevara et al. embed microarchitectural insight into the market's proxy agents, allowing them to navigate heterogeneous processors within a datacenter [39].

Formally, a market clearing mechanism maximizes welfare subject to datacenter resource capacity. Expression 4.3 is the objective, which maximizes welfare summed across all agents minus datacenter cost. Constraint 4.4 ensures that the allocation is bound by the number of resources **X** in the system.

$$\max \quad \sum_{a \in A} V_a(X_a) - C \tag{4.3}$$

$$\text{s.t.} \quad \sum_{a \in A} X_a \leqslant \mathbf{X} \tag{4.4}$$

$$V_a(X_a) \geqslant V_a(X_b) \qquad \forall a, b \in A; a \neq b \tag{4.5}$$

$V_a \leftarrow$ value of allocation to application a

$X_a \leftarrow$ machines allocated to application a

$C \leftarrow$ cost of allocation

$A \leftarrow$ set of all applications

$\mathbf{X} \leftarrow$ number of machines in the system

In each of these prior mechanisms, the objective is welfare maximization. For the first time, we introduce the notion of fairness into the market clearing mechanism. Our definition of fairness is tied to the notion of envy. An agent is envious when she derives higher utility from another agent's allocation than from her own. A market clearing mechanism with fairness must avoid allocations that induce envy.

Constraint 4.5 enforces envy-freeness, ensuring that every agent a derives greater value from its allocation X_a than from another agent b's allocation. This constraint is enforced on every pair of agents. The market clearing mechanism performs this constrained optimization with a mixed integer linear program. The resulting allocations are envy-free and thus fair.

4.3 Fairness for Heterogeneous Tasks

To allocate resources effectively, we must understand datacenter applications. Studies often treat the application as a large, monolithic piece of software. In this setting, the resource manager provides quality-of-service (QoS) to each application as a whole. For an application with many latency-sensitive tasks, QoS is expressed as a percentile on response time (e.g., 95th). This QoS metric allows a few tasks to fall short of the performance target or fail altogether.

Tasks within an application are diverse. For example, memcached requests may vary in size. Map Reduce tasks may vary in complexity. Web search queries may vary in length. A single QoS metric expressed for each of these applications does not capture differences in task behavior. Neglecting these differences may produce unintended consequences. Yet addressing these differences raise hard questions about fairness for tasks of different types.

Fairness for Users Requires Fairness for Tasks. Whether tasks of a particular type get a fair allocation depends on the datacenter manager. To achieve a given response time, complex tasks require more time and hardware than simple ones. Hardware may be diverted from simple tasks to complex ones. Whether such task-level strategies are fair requires an understanding of user-level preferences and envy. Fairness between task types translates into fairness between the users that create those tasks.

For example, suppose two users or agents issue web search queries – agent A_s issues simple queries and agent A_c issues complex ones. Intuitively, fairness could be argued both ways. Agent A_s might argue that a datacenter is unfair to devote many resources for relatively few, complex queries, especially if those resources could make common, simple queries faster. To address this argument, the datacenter manager needs a strategy to mitigate envy.

On the other hand, agent A_c might argue that a datacenter is unfair to sacrifice the performance of complex queries even if they are rare. Moreover, the search engine may wish to build user trust in its algorithms by consistently performing well on "hard" queries. To address this argument, the datacenter manager must explicitly recognize task diversity and ensure no task type is victim to discriminatory service.

Table 4.1: System Parameters for Web Search performance measurements.

Component	**Specification**
Processor	3.3 GHz OOO cores, 6-wide issue (2 ld + 1 st + 3 int/fp)
L1 Cache	32 KB, 8-way SA, 64-byte blocks, 4-cycle (min)
L2 Cache	256 KB, 8-way SA, 64-byte blocks, 11-cycle (min)
L3 Cache	6 MB, 4-bank 8-way SA, 64-byte blocks, 26-31-cycle
DRAM Bandwidth	12.8 GB/s, dual channel

4.3.1 Agents for Heterogeneous Tasks in the Market

We classify an application's tasks by type and consider a datacenter manager that distributes resources across diverse tasks. We frame the fairness problem with a multi-agent system. Each agent is associated with a task type and requests resources on behalf of its queued tasks. We then make the case for mitigating envy and encouraging sharing.

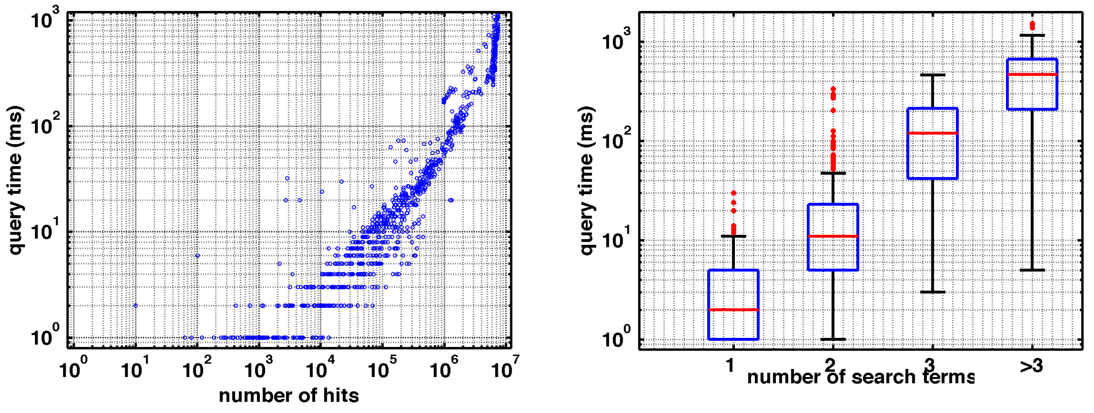
Envy-Freeness (EF). First, consider the benefits of envy-free (EF) allocation for tasks of varying complexity. An EF allocation is one in which no user envies the allocation of another. An EF mechanism might provide more resources to agents with complex tasks. This outcome is not obviously EF. One might think that an agent A_s with simple tasks would envy the larger allocation for an agent A_c with complex ones. However, additional resources may not necessarily benefit A_s.

Suppose agent A_s defines service quality with respect to some target latency (e.g., 100ms). Once this target is met, further latency reductions provide no additional utility. If A_s is already meeting her latency target with her allocation, she will not envy the larger allocation for A_c. Thus, EF allocations are fair yet differentiated for task complexity.

Differentiation is important. For fairly balanced waiting times, Little's Law says that queues with complex tasks need more resources than ones with simple tasks. By providing differentiated allocations, EF naturally produces desirable system behavior. In particular, agents communicate task complexity with expressive piecewise-linear utility functions. And the EF mechanism distributes resources to ensure agents who represent diverse tasks do not envy each other. As a consequence, the users who issue those tasks do not experience envy.

Sharing Incentives (SI). Datacenter resources might be divided statically or dynamically. Without sharing incentives, n agents would not participate in a shared system. Instead, they would prefer a static $1/n$ division of resources. Such division is inefficient because it neglects task diversity. Complex tasks would better utilize a $> 1/n$ allocation and simple tasks would see little harm from a $< 1/n$ allocation.

Sharing incentives (SI) allow an application to dynamically divide resources among its constituent agents and tasks, providing each agent the utility it would have received under a $1/n$ division. With dynamism, complex tasks will be served by more hardware and simple tasks will be served by less. Note that this outcome, SI, arises naturally from an EF mechanism. For any number of agents, sharing incentives follow from envy-freeness as long as all resources in the system are allocated (i.e., no



(a) **Number of Hits** (b) **Number of Terms**

FIGURE 4.5: Query runtime is (a) highly correlated to the number of hits and (b) increases with the number of search terms.

free disposal in cake cutting theory [22]).

Efficiency and the Price of Fairness. We define allocative efficiency with respect to system throughput. Given this definition, the price of fairness is intuitive. Without fairness, a datacenter manager could allocate resources to maximize throughput without regard to fairness. In the pursuit of fairness, the datacenter manager must allocate resources subject to constraints imposed by EF and SI. These constraints shift resource allocations and reduce throughput. The extent of this reduction constitutes the price of fairness.

4.3.2 Web Search and Heterogeneous Queries

Our approach to fair allocation relies on assigning an application's heterogeneous tasks to separate agents and queues. To do so, we need a method to distinguish task types before having executed it. If we can classify tasks as they arrive, we can forward them to different agents and queues.

We demonstrate this strategy for web search, focusing on latency-sensitive query processing. Search is an important datacenter application that exhibits significant diversity across queries. Moreover, we show that queries can be classified based on the number of terms. These queries have strict latency targets [84] and would benefit from fair resource allocation.

Creating an Index. We benchmark and characterize web search queries using an open-source search engine. Lucene is a library that powers many open-source search engines, including Solr which we use to index and search web pages. Search queries vary in sophistication and length [14, 91], making them an ideal benchmark for our system model. Note that while we focus on Wikipedia queries, Solr has a broad user base, which includes Netflix, LinkedIn, and Twitter.

We use Wikimedia's publicly available database of Wikipedia articles from June 2013. The database has over 13.5M articles, which produces a 12.3 GB index. Real world search engines execute on indexes of all sizes. Our Wikipedia index is relatively small compared to Google's reported 100 PB index from July 2012. While we could have used Nutch to crawl Wikipedia pages to create an index dynamically, we prefer a static dataset that provides experimental repeatability.

Constructing Queries. We construct a diverse query stream to exercise the search engine based on the 1,000 most popular Wikipedia English articles as reported by Wikitrends. Specifically, we construct one query from each of the top article titles. If the title is comprised of multiple words, we concatenate them with logical ORs. For example, the most popular article is "List of reporting software" and the corresponding query is "List+of+reporting+software". Queries constructed in this manner exhibit significant diversity. Long, complex queries have execution times that are orders of magnitude greater than those for short, simple queries.

Our approach produces diverse yet relevant queries. For contrast, suppose we were to construct queries with random terms selected independently of the indexed pages (e.g., words from an English dictionary). Such queries could be irrelevant to the indexed pages and produce too few results. For example, searching for jargon is unlikely to yield any results for indexed content in a different discipline. Alternatively, these queries could be too common (e.g., “the”) and produce too many results. These effects are exacerbated in multi-term queries.

Executing Search. We run the web search engine on the platform in Table 4.1 and execute the set of 1,000 queries. We use Solr’s built-in logging to measure the time required to execute each query. To ensure consistent and representative measurements, we warm up the system and set aside data collected immediately after the search engine is started.

After warm-up, we run the full set of queries three times. Across these experiments, we observe an average variation of 7% in query execution time. This reported variation is inflated by large percentage differences for relatively few small queries. Across experiments, a short query’s execution time could vary between 2-4ms, a difference of 50-100%.

4.3.3 Query Diversity and Execution Time

We analyze Solr logs to obtain the execution time for each query. Query execution time varies from milliseconds to seconds as illustrated by the logscale y-axis on Figure 4.5a. The variation in query execution time is surprising and we seek to understand the source of this diversity.

We find that the number of results for a query is a strong indicator of execution time. Figure 4.5a shows the number of hits for each query on the log-scale x-axis. The correlation coefficient between the execution time and number of hits is 0.91, which indicates a strong correlation for the large data set. Intuitively, more computation is required for queries that match a larger part of the index. Evidently, distinguishing between short- and long-running queries is a necessary first step for fair guarantees on service quality.

Table 4.2: Parameters for the four bins of web search query types in the market.

	1-term	**2-term**	**3-term**	**>3-term**
Mcyc/Task	14	79	509	1539
% StdDev	100	200	80	70
% Arrivals	32.5%	32.7%	18.9%	15.9%
Total Tasks/min	1620K			
Daily Tasks/min	13162	13244	7654	6440
Week Tasks/min	52650	52974	30618	25758
Value (K$/month)	\$15 if $T \leqslant$50ms \$0 if $T \geqslant$800ms			

To this end, we separate queries by an indicator that is readily available before query processing: the number of search terms. Figure 4.5b illustrates the distribution of execution times for varying query lengths. Queries with the same number of terms have similar response times. We observe that 1-term queries are likely to take less time than 2-term queries, and so on. The intuition is that adding a term to a query will only add to the number of hits since terms are ORed together.

The relationships between query execution time, the number of hits, and the number of search terms hold even when the index is in shards across multiple nodes (e.g., SolrCloud). Queries execute against all shards in parallel, and the longest running execution determines execution time. In this configuration, the correlation coefficient between execution time and number of hits is 0.90, nearly as high as in the single-server case. Although a query touches only a fraction of the data in distributed search, we find that query execution time is not reduced by the same fraction. Thus, we focus on single-server search, expecting our results to be representative and generalize due to the data parallelism at greater scales.

Our study thus far has shown that number of terms is a good indicator of execution time, and that there is a wide spectrum of execution times for our query set. Given that we can classify query types prior to executing them, we can assign queries to separate agents and queues. Moreover, we can fairly allocate resources to these agents and provide service quality guarantees across query types.

Table 4.3: Architecture parameters for big and small cores.

	Big	**Small**
#Nodes	$0 - 160$	$0 - 225$
#CPU	4	16
Freq	2.0 GHz	1.0 GHz
Issue	6 inst	2 inst
Exe	OOO	IO
L1 I/D	64/64KB	32/32KB
L2	4MB, 8-way	1MB, 8-way

Table 4.4: Performance scaling factors for web search query types on big and small cores.

	IPC_{Big}	IPC_{Small}	**SF**
1-term	1.03	0.44	0.43
2-term	1.01	0.44	0.43
3-term	1.02	0.44	0.45
$>$**3-term**	1.14	0.45	0.40

4.4 Price of Fairness

Fair resource allocation is desirable, yet enforcing fairness likely reduces datacenter throughput and efficiency relative to a welfare-maximizing allocator. Next, we evaluate the price of fairness by examining processor allocations for web search queries of different types. We compare a welfare-maximizing allocation (max) against an envy-free allocation (fair).

4.4.1 Mitigating Envy

Economic mechanisms expose an inherent tension familiar to most system architects. A free market maximizes system throughput but may leave some tasks vulnerable to poor performance. In contrast, a perfectly fair allocation does not efficiently use the resources within the datacenter. To navigate efficiency-fairness trade-offs, we require a spectrum of management mechanisms.

At one end of the spectrum, we implement a market that allocates processors to maximize welfare [39]. At the other end, we constrain the market to enforce fairness by eliminating envy. With these mechanisms, we identify the sources of envy and the price paid for mitigating it. In particular, we examine expressive utility functions for different types of web search queries to identify root causes of envy.

Methodology. We use an in-house simulator to compare the welfare-maximizing (max) and envy-free (fair) market mechanisms. The market allocates processors to web search queries as summarized in Table 4.2 that have diurnal arrival rates shown in Figure 4.10. Four agents contend for resources, each acting on behalf of a particular query type. Each query type demands the same 95$^{\text{th}}$-percentile wait time ($<$800ms) and derives the same value when this target is met.

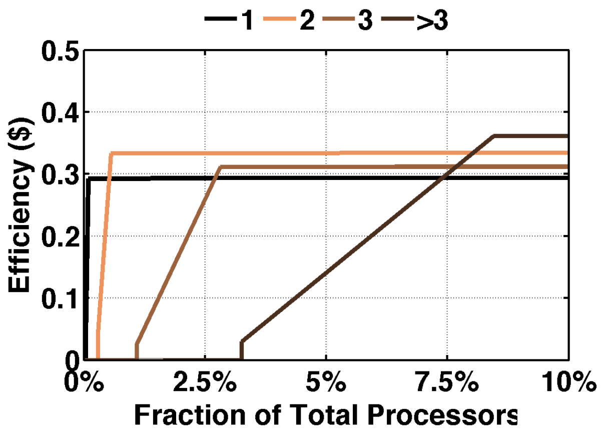
We classify queries into four types based on the number of terms: 1, 2, 3, and $>$3. Each query type is parametrized by its duration in millions of cycles, based on measurements in Figure 4.5b a physical machine. Query arrivals follow sinusoidal patterns and with relative arrival rates based on the characterization of Silverstein et al. [91].

The market allocates processors to agents and their queries. We simulate a 20KW system, which corresponds to approximately two racks of servers equipped with 160 quad-core processors. Each 2.5GHz core dissipates 15.6W, and the uncore components (memory, disk, NIC) dissipate 65W. We assume a power usage effectiveness of 1.6 [6]. The electricity price is \$0.07 per kWh. These parameters determine costs that are used to calculate welfare.

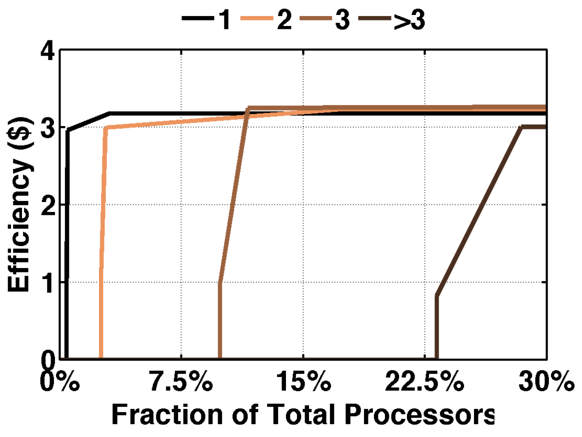
Sources of Envy. To understand the type of envy that max allocations cause, we consider the utility functions of each agent. Utility is the value derived from a particular allocation of processors. As the processor allocation increases, 95th percentile response time falls and value rises. We consider these effects for agents $A_1, A_2, A_3, A_{>3}$ who represent queries with 1, 2, 3 and $>$3 terms.

The effects of a processor allocation vary across query types due to differences in query latency and arrival rates. Arrival rates vary according to diurnal datacenter activity. We examine envy in three activity scenarios: low, average, and high. As shown in Figure 4.6, utility functions assign a dollar value (y-axis) to processor allocations (x-axis).

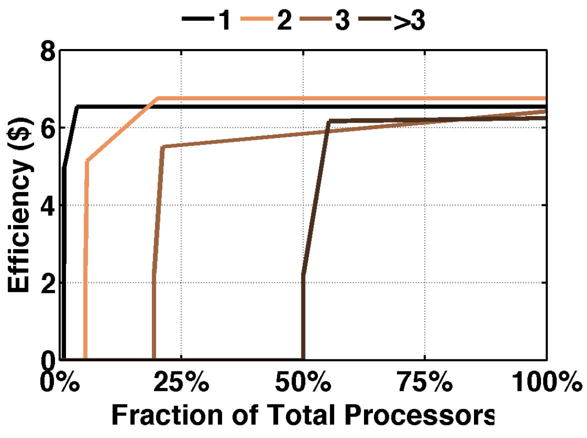
Values on the y-axis originate from the service-level agreement for web search; see Table 4.2. For each activity scenario and query type, there exists a flat region



(a) **Low Load**



(b) **Average Load**



(c) **High Load**

FIGURE 4.6: Utility functions show that 1- and 2-term queries require significantly fewer processors than 3- and >3-term queries.

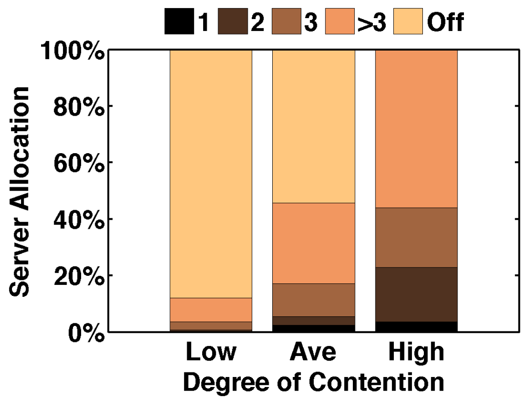
in which additional processors do not increase value. Values in this region vary from \$0.30 at low load, \$3.00 at average load, and almost \$7.00 at peak load. For a particular level of load, however, query types derive similar value from good service.

The key differences in utility manifest on the x-axis and the number of processors demanded by each query type. The x-intercept identifies the minimum number of processors required for an agent. The x-intercept shifts right as query length and complexity increases. Agent $A_{>3}$ demands more processors than agent A_1. Additionally, the x-intercept shifts right as query load increases. Higher query arrival rates require correspondingly high processor allocations to increase service rate and to maintain reasonable queueing times.

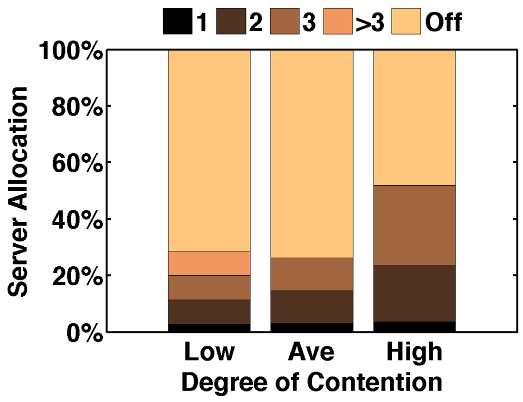
Figure 4.6(a) highlights several of these effects. Short queries, despite comprising more than 60% of the total query stream, require few processors. At low load, agents A_1 and A_2 require less than 1% of the datacenter's processors to maximize their value for computation. With such modest demands for hardware, these agents are unlikely to envy the allocation of others. Agents A_1 and A_2 derive no additional benefit from an allocation greater than 0.1% and 0.6% of the system, respectively. They will not envy the larger allocations of A_3 and $A_{>3}$.

In contrast, envy is likely given high load or an over-subscribed datacenter, scenarios illustrated in Figure 4.6(c). The allocations for agents A_3 and $A_{>3}$ require a large share of the system to meet even their minimum service targets. Indeed, these agents would need more than 100% of the datacenter's processors to derive their maximum value. For example, agent A_3 would continue to benefit from additional processors even if she were given all of the datacenter's processors. In this setting, envy is very likely.

Envying the Allocation of Another. Figure 4.7a compares max and fair processor allocations that are made to each agent and her query type. At high load, consider the allocations made by the max mechanism. Agent A_3 and her 3-



(a) **Max**



(b) **Envy-Free**

FIGURE 4.7: Max allocations more efficiently use processors than the envy-free mechanism, especially at high load.

term queries receive 21.2% of the datacenter's processors. Agent $A_{>3}$ receive 55.9%. Given these allocations, A_3 envies $A_{>3}$. Agent A_3 would derive more value from 55.9% of the datacenter's processors than from the 21.2% she was allocated.

To reason about this envy, consider the users who issue the queries. Users who issue 3-term queries would envy those who issue >3-term queries. The first user would likely experience better service had the 3-term query been classified as a >3-term query. This observation may produce strategic behavior. A user may try to exploit a system under high load by aliasing her request and misleading the manager for greater performance.

The envy-free mechanism explicitly guards against such strategic behavior. Figure 4.7b shows fair processor allocations for the same stream of queries. Unfortunately, under high load, it is not possible to provision resources to agent $A_{>3}$ without inducing envy in agent A_3. As a result, the fair mechanism allocates $A_{>3}$ nothing so that A_3 has no cause for envy.

4.4.2 Price of Fairness

The most efficient allocation of datacenter processors is one that maximizes value across all users. At best, the fair and envy-free mechanism allocates processors just as efficiently as the optimal and welfare-maximizing one. At other times, however, enforcing envy-freeness causes the system to deviate from the most efficient allocations. We quantify the price of fairness by comparing max and fair mechanisms.

Figure 4.11 indicates efficiency from the fair mechanism is noticeably lower than that of the max mechanism when resource contention is high. To be precise about the price of fairness, we adopt the definition from prior work [18]:

$$\text{Price} = \frac{\text{Total utility of agents in optimal allocation}}{\text{Total utility of agents in fair allocation}}$$

We apply this definition to our week-long experiments in processor allocation. For the four query types, the price of fairness is 1.5; the max allocation achieves up to $1.5\times$ the value of the fair allocation. Recent theoretical findings have shown the difficulty of applying tight bounds to the price of envy-freeness. We note that experiments show far better efficiency than what is predicted by the loose upper bound $n - \frac{1}{2}$ [18].

Envy-freeness constraints degrade efficiency most when the system is under heavy load. In this setting, compare max and fair allocations in Figure 4.7b. The key difference between them is whether agent $A_{>3}$ receives any processors at all. These queries need at least 50% of the datacenter's processors to derive any value from the allocation. Unfortunately, both agents A_3 and $A_{>3}$ would benefit from a 50% allocation. This situation leads to envy.

To eliminate envy between A_3 and $A_{>3}$, the market would need to allocate an equal number of processors to both agents. But both agents prefer $\geqslant 50\%$ and the market cannot satisfy both agents without sacrificing all other agents. Given these trade-offs, the market allocates nothing to agent $A_{>3}$ and satisfies the other three agents.

This particular example highlights the price of fairness. Envy-freeness is a highly restrictive form of fairness. In a highly contended system, eliminating envy may force the datacenter manager to degrade throughput and starve agents. Such trade-offs are to be expected from a mechanism that enforces any sort of fairness. Given a formal measure for the price of fairness, we proceed to introduce a knob that allows the market to tune the balance between fairness and efficiency.

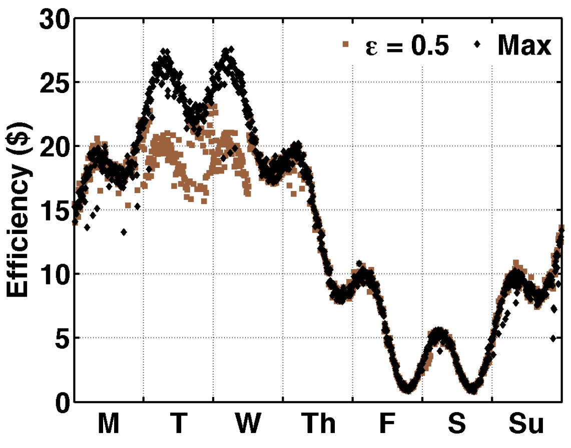
4.4.3 Trading Fairness for Efficiency

Although envy-freeness is a desirable property for an allocation mechanism, its effect on efficiency is costly. When resources are scarce, the price of fairness is particularly unattractive. To solve this problem, we add a parameter to the market mechanism that allows for a tunable amount of envy to exist in the allocations. Constraint Equation (4.5) for envy-freeness in the mixed integer program becomes:

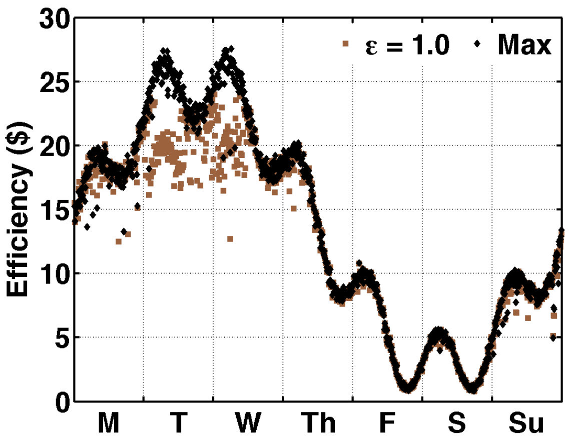
$$\epsilon\text{-EF}: \qquad V_A(X_A) + \epsilon \times V_A(X_A) \geqslant V_A(X_B) \qquad \forall a, b \in \text{Agents}$$

In the above, ϵ is the amount of envy allowed in the system expressed relative to each agent's valuation. When $\epsilon = 0$, the market produces envy-free allocations. When $\epsilon > 0$, optimization constraints are relaxed allowing the market to find more efficient allocations while tolerating some degree of envy.

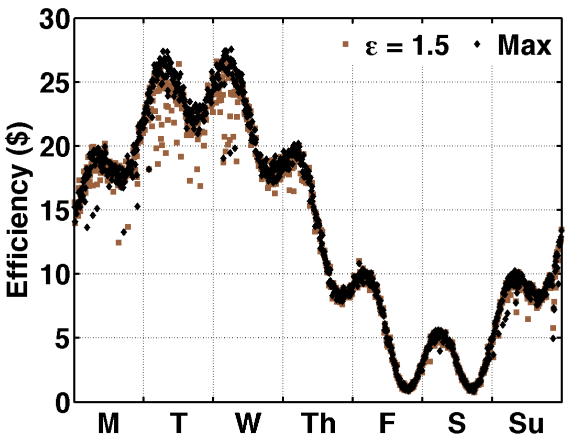
Figure 4.8 shows how efficiency improves as ϵ increases. The market permits agents to envy another agent's allocation and tolerates envy of 0.5×, 1.0×, or even 1.5× of its allocated value $V_A(X_A)$. By progressively increasing ϵ and relaxing constraints on envy, the overall efficiency of ϵ-EF allocations approaches that of the max



(a) $\epsilon = 0.5$-**Envy**

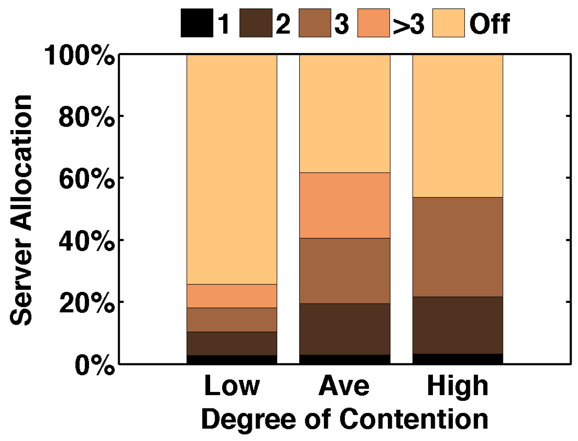


(b) $\epsilon = 1.0$-**Envy**

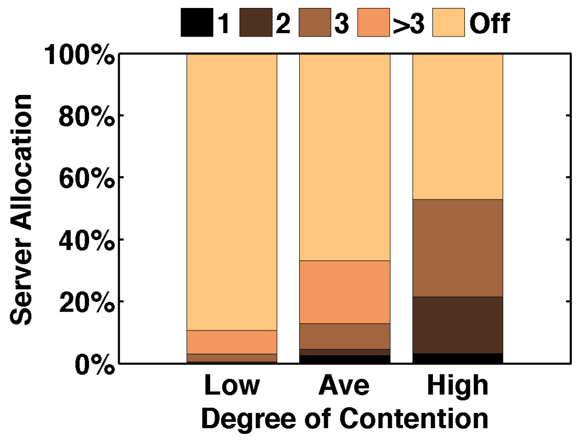


(c) $\epsilon = 1.5$-**Envy**

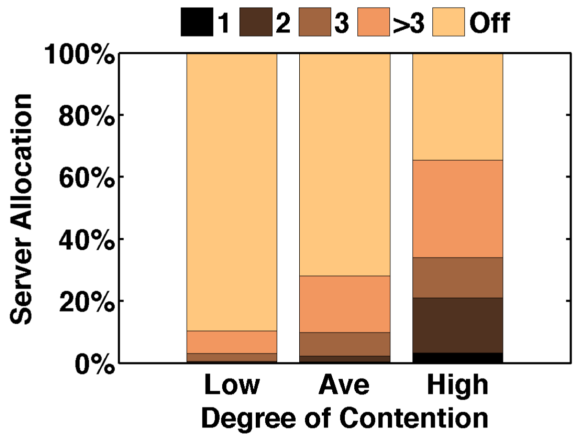
FIGURE 4.8: Increasing the amount of envy in the system via the parameter ϵ improves efficiency.



(a) $\epsilon = 0.5$-**Envy**



(b) $\epsilon = 1.0$-**Envy**



(c) $\epsilon = 1.5$-**Envy**

FIGURE 4.9: As ϵ-envy increases, resource allocations more closely approximate those of the max mechanism.

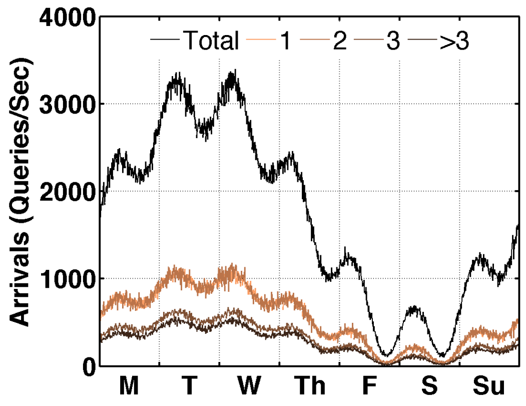


FIGURE 4.10: Query arrival rate.

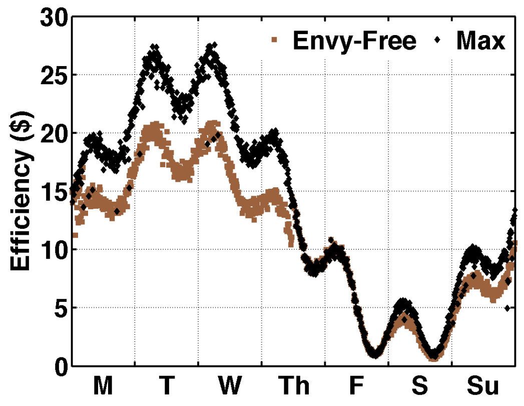


FIGURE 4.11: Max vs EF efficiency.

allocation.

As before, the effects of fairness on throughput are most apparent when the system is under high load. Permitting some envy reduces the price of fairness observed in Figure 4.11. In a deployed system, ϵ can serve as a powerful knob to trade-off efficiency and envy in response to system load.

Improvements in efficiency are a direct result of different allocations for different values of ϵ. Figure 4.9 shows allocations as the allowed amount of envy increases. When $\epsilon = 0.5$, the allocations do not differ significantly from the envy-free ones in Figure 4.7b. Yet even modest allocation changes when $\epsilon = 0.5$ improve efficiency for periods of moderate contention. See Monday and Thursday in Figure 4.8a.

When $\epsilon = 1.0$, allocations under low and average load very closely resemble those from the max mechanism in Figure 4.7a. Lastly, when $\epsilon = 1.5$, the four agents' allocations are close to those of the max mechanism. And as these allocations converge, so do efficiency and welfare.

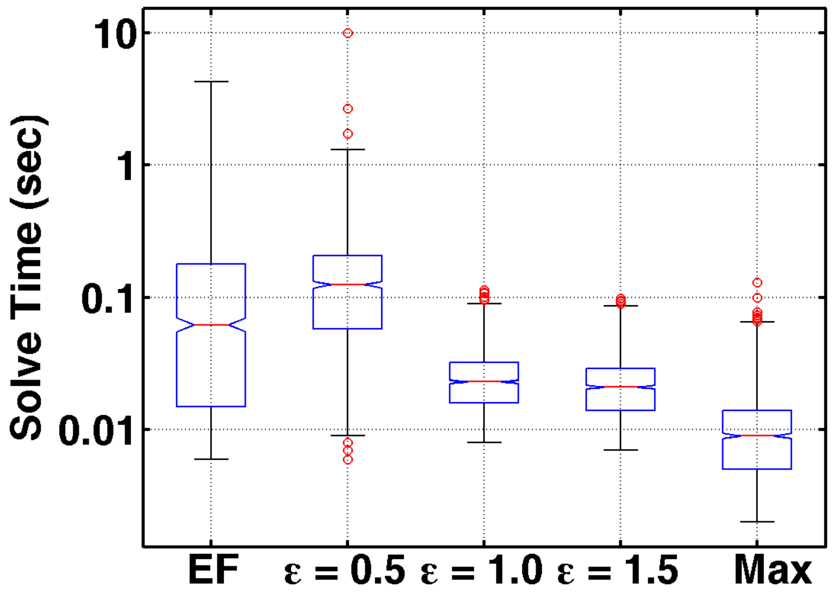


FIGURE 4.12: Market clearing and solve time for varying degrees of envy.

4.5 Fairness and Heterogeneous Processors

Recent industry trends motivate low-power, mobile-class processors in datacenters. Though these processors degrade single query performance, many more of them can be used within the power budget of traditional, server-class processors. Prior mechanisms manage heterogeneous servers to maximize performance and throughput [39, 69]. For the first time, we consider the fair allocation of heterogeneous processors.

In this section, we describe the difficulty of expressing heterogeneity with piecewise-constant Leontief utility functions. Leontief utilities do not apply to latency-sensitive tasks and we cannot leverage recent generalizations of max-min fairness [36]. Instead, we express heterogeneity using piecewise-linear utility functions and implement fairness within a market [39]. We evaluate fairness and envy when allocating big- and small-core processors to web search queries of varying lengths.

4.5.1 Expressing Heterogeneity

Fairness is complicated in settings with multiple resources and resource types. Ghodsi et al. generalize max-min fairness for multiple resources – processors and memory [36]. In this setting, max-min fairness relies on two key assumptions. First, the resources are complementary. Tasks require a minimum allocation of both processors and memory to execute, and more of one resource cannot compensate for less of the other. Second, piecewise-constant utility functions express value for throughput and greater allocations always increases value.

In our setting, fairness is complicated by multiple resource types – big and small processor cores. Processor heterogeneity poses fundamentally different challenges. The processor cores are imperfect substitutes. One core type can be substituted for another as long as the latency penalties are tolerable. Moreover, users derive value when latency targets are met. Piecewise-linear utility functions express value for latency.

For piecewise-linear utilities and heterogeneous processors, Guevara et al. present a market mechanism for resource allocation [39]. We extend this market to mitigate envy and improve fairness. As in the setting with homogeneous processors, we assess envy and quantify the price of fairness when allocating heterogeneous processors.

Fairness and Substitutability. Heterogeneous processors are imperfect substitutes. Tasks may derive the same utility when running on Q_B big processors or Q_S small processors. A stream of memory- or I/O-bound tasks may be indifferent between core types and hence $Q_B = Q_S$. In contrast, a processor-intensive task stream needs many small cores to compensate for individual task slowdowns and hence $Q_B < Q_S$. The key is that, for any homogeneous task stream, the marginal rate of substitution between core types is a given and constant.

Correctly expressing substitutability is important to achieving fairness. Assessing envy requires each agent to compare the value derived from her allocation against the value derived from other agents' allocations. With processor heterogeneity, the market clearing mechanism must differentiate between resource types to properly quantify envy.

Fairness and Optimization Complexity. Processor heterogeneity, if exposed directly to the market clearing solver, would produce an expensive multi-dimensional linear program. For timely allocation decisions, we scale heterogeneous resource allocations into canonical ones. Such scaling reduces the dimensionality of the optimization and reduces solve time.

Specifically, we apply performance scaling factors to represent application preferences for heterogeneous processors. Utility functions express value as a function of cycles from a canonical core (x-coordinate). Canonical cycles are inferred by applying scaling factors $< \kappa_B, \kappa_S >$ to a mix of heterogeneous cycles $< Q_B, Q_S >$. The market clearing mechanism estimates canonical cycles by multiplying $Q = Q_B \times \kappa_B + Q_S \times \kappa_S$ and mapping Q to value (y-coordinate).

Mitigating heterogeneity's effect on solve time is important because enforcing constraints on envy necessarily increases solve time. Fairness impacts solve time by adding constraints on envy and reducing the set of feasible solutions for the linear program. The precise impact on solve time depends on how restrictive the constraints on fairness are.

Figure 4.12 illustrates solve time distributions when clearing a market with heterogeneous processors. At one end of the spectrum, a welfare-maximizing market allocates processors without regard for envy. The market clears repeatedly for one week of activity, and allocations are identified in <0.1 seconds.

In contrast, restricting the extent of envy in the system increases the mean and variance for solve times. When $\epsilon = 1.0 - 1.5$ and more envy is permitted, solve time remains <0.1 seconds. At $\epsilon = 0.5$ or when envy is not permitted, solve times increase by one to two orders of magnitude. Increased allocation overheads are another price of fairness.

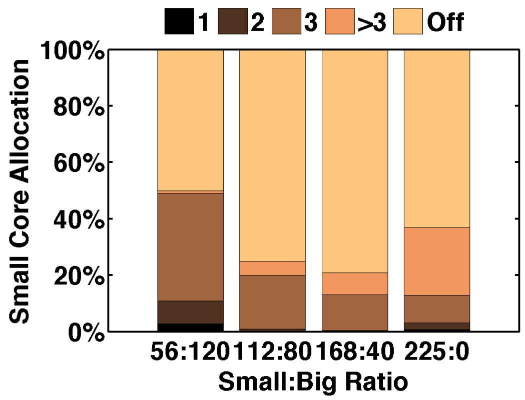
4.5.2 Allocating Big and Small Cores

Let us again consider the problem of allocating processors to search queries. As before, queries are classified upon arrival by the number of search terms. The allocation mechanism partitions resources across query types maximizing welfare while mitigating envy. The datacenter is equipped with two types of processors with either big or small cores. We estimate performance scaling factors in cycle-accurate simulation for each query type.

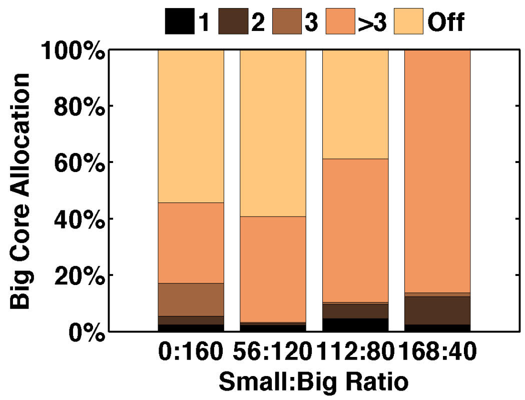
Methodology. We use Marssx86 and DRAMsim2 to simulate Solr with a smaller input set of 50K documents [80, 86]. We configure big and small cores as listed in Table 4.3. Across all query types, the scaling factor for the small core is 0.4 ($\kappa = \frac{IPC_B}{IPC_S}$), shown in Table 4.4. This is similar to prior work that executed the Bing search engine on Intel Atom processors and observed a 3× decrease in throughput [84].

The number of cores per server is determined to equalize die area based on McPAT estimates [61]. Processors that use small cores should integrate more of them on the die to amortize the cost of uncore components across more processing capability. Aside from scaling factors and core power, all parameters are kept the same across processor types.

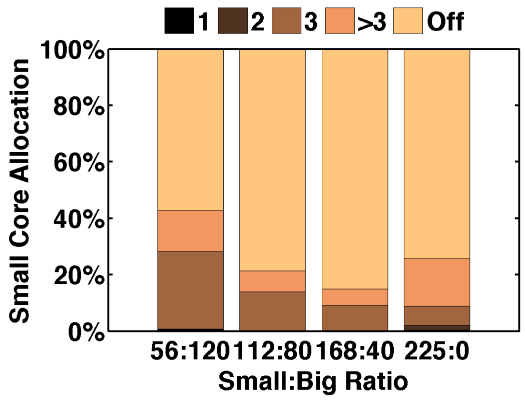
Sources of Envy. Figures 4.13a–4.13b show the allocations of small and big cores with the welfare-maximizing market at average query load. The x-axis enumerates various ratios of small to big cores that fit within a 20KW power budget. Possibilities include a homogeneous system with only big cores **0:160**, only small cores **225:0**, or a heterogeneous mix of the two core types. The y-axis shows the



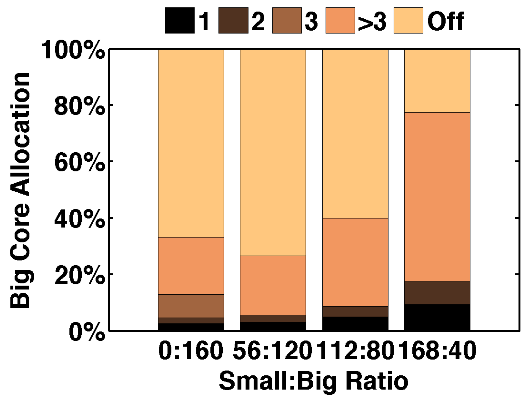
(a) Max Allocation of Small Cores



(b) Max Allocation of Big Cores



(c) $\epsilon = 1.0$, Small Cores



(d) $\epsilon = 1.0$, Big Cores

FIGURE 4.13: Allocations of heterogeneous resources.

percentage of small and big cores allocated to each query type.

Suppose agents A_1, A_2, A_3, $A_{>3}$ represent query streams with 1, 2, 3, and >3 terms. Within each heterogeneous system, the max mechanism allocates the big cores to $A_{>3}$ but this allocation causes envy. Consider system **112:80**. $A_{>3}$'s allocation is 37.5% of the system's big cores and 5.0% of its small cores. The agent derives \$2.86 of utility (Figure 4.6b). Meanwhile, A_3 receives no big cores and 19.1% of the system's small cores for \$3.21 of utility. Agent A_3 envies $A_{>3}$ since her value would increase to \$3.25 if allocations were swapped.

Now let us consider the allocations with the fair, envy-mitigating mechanism shown in Figures 4.13c–4.13d. With $\epsilon = 1.0$, all four query types receive allocations. In the heterogeneous cases, agents A_1 and $A_{>3}$ always receive some fraction of the big cores. The other two agents A_2 and A_3 receive a mix of heterogeneous cores. Although not shown, the efficiencies of the EF and ϵ-EF mechanisms are the same in both homogeneous and heterogeneous settings.

In summary, the fair market mechanisms extend naturally to datacenters with heterogeneous processors. The market produces the desired outcomes. Big processors are allocated to more expensive tasks and small processors are allocated to simpler ones. The market clears to maximize welfare while mitigating envy. And the price of fairness depends on the extent that envy is permissible.

4.6 Related Work

Fairness. In datacenter systems, dominant resource fairness (DRF) allocates resources to throughput-oriented workloads [36]. DRF guarantees attractive game-theoretic properties: sharing incentives, envy-freeness, Pareto efficiency, and strategy-proofness. However, DRF does not apply to latency-sensitive tasks, which are the focus of our work.

In computer architecture, fairness has been defined as equal-slowdown in the presence of resource contention on a shared system [75]. This definition guarantees fairness in outcomes but does not provide guidance for resource allocation. Equal-slowdown mechanisms do not provide sharing incentives; prior work assumes that users have no choice but to share.

In economics, the cake-cutting problem translates naturally into system management. Yet only recently has it been studied for piecewise-linear utility [15, 24]. In these settings, guaranteeing fairness is theoretically unexplored. Fairness necessarily reduces efficiency, yet bounding efficiency losses is an open problem [18]. We present a real-world need for piecewise-linear utility and fairness among search queries.

Markets. Resource allocation using economic mechanisms has typically aimed to maximize performance [31, 46, 96, 101]. There has also been work in applying such mechanisms to account for energy consumption [19]. Another approach to improving energy efficiency of datacenters is to use DVFS [66]. More recently, allocating small cores for their energy efficiency has been shown to improve service within a fixed energy budget [39]. To the best of our knowledge, we are the first to apply market mechanisms to enforce fairness properties across latency-sensitive applications.

Heterogeneity. By profiling requests on a variety of server-class processors, resource managers can make better mappings of applications to systems [27, 69]. To exploit efficient mobile-class processors in heterogeneous systems, new allocation mechanisms are needed [39]. Processor design methodologies can target datacenter applications [65] or produce large-scale, heterogeneous systems [40]. Heterogeneity presents a unique challenge to resource management we take an important first step towards fairly allocating substitutable resources.

4.7 Summary

We use a market mechanism to enforce envy-freeness in shared datacenters. Applying this resource manager to divide resources amongst web search queries reveals that the price of fairness is efficiency, and it is most apparent when system resources are scarce. We present a parametrized version of the EF market that allows a system to trade envy for efficiency. Fair resource management can also be applied to heterogeneous resources and maintain the same strong guarantees in the resulting allocations.

5 Conclusions

The primary contribution of this thesis is a novel approach to architecting heterogeneous systems that coordinates the design and management of the resources. Heterogeneity is desirable due to improvements in energy efficiency. If the heterogeneous resources are well-matched to the application mix, the system is a step closer to being specialized for the payload and can execute it more efficiently than a general-purpose system. This efficiency allows datacenters to save power while servicing the same amount of load, or to increase service rate without increasing power utilization.

This thesis has also contributed a vision and methodology for deploying heterogeneous processors in a datacenter. Though prior work has proposed heterogeneous chip multiprocessors, we assume homogeneous multiprocessors and instead deploy heterogeneity across servers or racks in a datacenter. The benefits of this approach are twofold. First, system architects can determine the fraction of the power budget that should be dedicated to each processor type based on the expected workload mix. With heterogeneous CMPs, the processor architect determines the ratio of small and big cores, and this mix is unlikely to satisfy all cloud services. Second, the management of the heterogeneous resources can be centralized such that once a request arrives at a server, there are no additional resource allocation decisions to be made. In fact, resources can be partitioned at the level of full racks or servers and requests from an application are steered to that part of the datacenter.

We have presented two versions of an allocation mechanism that uses profiling information to manage heterogeneous resources. The market mechanism in Chapter 2 makes allocations that maximize welfare, which is value minus cost. It uses an optimization framework to find the best allocation, where the best allocation is that which provides good quality of service to as many applications as possible, at the lowest operational cost. In Chapter 4 we present an alternative version of the market that enforces envy-free allocations between latency-sensitive applications. In both versions of the market, the resulting allocations provide theoretical guarantees rather than responding to system utilization metrics as approximations for welfare or fairness. The markets are truly a marriage between two fields, computer architecture and computational economics, where techniques from both disciplines contributed to the final product.

The other major contribution in this thesis is in the form of a methodology for designing heterogeneous systems for manageability. In the presence of heterogeneity, resource managers make best-effort allocations in response to application arrival rates and preferences. Instead of designing heterogeneous systems to maximize efficiency, diversity, or peak performance within a fixed area or power budget, we propose design strategies that produce manageable systems. By considering the sources of run-time uncertainty, our design strategies aim to provide resource managers suitable alternatives to a best-case allocation. For example, in the presence of contention an application may not get its preferred core, but if the design includes reasonable alternatives to the preferred core, then the manager is more likely to make reasonably good decisions. The results in 3 show that risk-aware design strategies are likely to produce better systems than strategies from prior work that maximize efficiency under best-case assumptions about resource management. Our work rethinks heterogeneous design and demonstrates the benefit of this novel approach.

5.1 Key Contributions

In summary, the key contributions of this thesis are:

- **Processor Heterogeneity in the Datacenter.** We define a novel design space where heterogeneous processors are deployed in datacenters to increase the compute capability of the system within a fixed power budget.

- **Economic Mechanisms and Heterogeneity.** We allocate processors to applications with a market that navigates performance-efficiency trade-offs of heterogeneity.

- **Anticipating Risk in Heterogeneous Design.** We consider resource management at design-time and ask whether a deployed heterogeneous system is likely to meet design objectives using non-ideal resource allocation.

- **Formalizing Heterogeneous Design Strategies.** We construct a framework of strategies for heterogeneous design, and propose strategies that minimize performance uncertainty.

- **Designing for Manageability.** We explore tens of design strategies and rank the resulting systems based on service quality. Risk-aware design accounts for more than 70% of the top-ranked strategies.

- **Examine Fairness for Latency-Sensitive Tasks.** We demonstrate that prior algorithms for fair allocations do not support expressive piecewise-linear utility functions, which are necessary to describe the performance targets of latency-sensitive tasks.

- **Introduce Fairness to Markets.** We add constraints to a market mechanism so that the resulting resource allocations are fair.
- **Enforcing Fairness for Heterogeneous Tasks.** We find that web search queries have vastly different runtimes and can be classified upon arrival based on the number of search terms.
- **Quantifying the Price of Fairness.** We observe that the efficiency of a welfare-maximizing market is $1.5\times$ that of fair allocations, though we can improve efficiency by parametrizing the amount of envy allowed in the system.
- **Extending Fairness to Heterogeneous Processors.** We show that the microarchitectural differences in heterogeneous processors can be embedded into the market when the fairness constraints are present.

5.2 Future Directions

Many interesting research directions lie at the intersection of systems and computational economics. Small and large-scale systems can benefit from solutions that have a strong theoretical foundation. We can tailor mechanisms from theory based on system and application behaviors. Both disciplines benefit and grow from collaborating wiht each other. Next, we discuss open questions that lie on the same course as this thesis.

5.3 Market Mechanisms

As we continue to build bridges between computer architecture and economic and multi-agent systems, enhancing allocation procedures with greater architectural insight is imperative. This thesis demonstrates that market mechanisms can manage heterogeneous resources when equipped with the relative performance and power of the different architectures. There are other sources of performance and power differences that might be considered by the allocator. For example, differences in the storage technologies of servers have an effect on application performance, and a mechanism for resource management would benefit from this information. Accelerators are another source of performance and power improvements, and appropriately expressing the trade-offs to an allocator is an open question.

There is much research in performance and power modeling in the systems community, and coupling these models with mechanisms from computational economics would increase their relevance. Whereas the market in this thesis relied on data from having profiled applications, using a model in the market that estimates performance and power differences between heterogeneous processors is an interesting alternative. There is also the option of having the market learn whether the profiling information it uses is producing good allocations. The market can compare the value that each application experiences from the allocation to the value that the market expected from each application when it made its allocation decision. If the market is repeatedly over-valuing an allocation decision, then it is possible that the scaling factors that it has for an application to do not adequately represent the sensitivity of that application to processor choice.

Another direction is to have the proxy agents implement bidding strategies in response to the allocations and service quality that an application obtain. If an allocation is repeatedly unsuccessful at obtaining a good allocation, it may have to rethink the parameters of its bid. The dollar value and response time targets are fed to the proxy by each application, yet it is possible that an application is unable to correctly specify these parameters to reflect the type of service it expects. For example, even though an application may have a response time target in mind, this target may not correctly account for queuing or other system delays and hence needs to be reconsidered. The bidding proxy can make recommendations to the application that will allow it to get better service. The proxy might also be strategic and try to increase an application's utility by obtaining the same service quality at a lower price. Based on the status of the system, such as contention and degree of heterogeneity, a proxy agent can define a strategy to bid less than the true valuation of the application.

There is also a broader opportunity to apply market mechanisms to solve problems in systems other than datacenters. In fact, heterogeneous processors are more prevalent in mobile systems-on-chip and chip multiprocessors than they are in datacenters. Tailoring the application and system model in the market to these different architectures would open these platforms to a new approach to resource management. Other parts of the market model probably need to be updated, for example the frequency of allocation decisions and the optimization technique that is used to clear the market. Mobile and desktop platforms also have different application and system settings than a datacenter. Properly reflecting these to the market is necessary to benefit from the mechanism.

5.4 Heterogeneous System Design

Future work also exists along the route of heterogeneous system design. This thesis is the first to consider the importance of resource management during the design phase. Prior work in designing heterogeneous chip multiprocessors aimed to maximize efficiency by assuming a perfect mapping of applications to processors. By breaking this assumption, we have discovered novel appraches to design space exploration that produce systems that perform better under a realistic, imperfect resource allocation setting. We evaluated the risk-aware strategies by applying them to a design space of heterogeneous processors for datacenters. Yet the strategies can also be applied to other heterogeneous systems, such as chip multiprocessors. It is also possible that new strategies are waiting to be discovered that target different application and system models.

Our approach in Chapter 3 ties the design to one allocation mechanism. It would be interesting to know how sensitive the design process is to the specific mechanism. For example, suppose the market mechanism undergoes a small change that typically leads to better allocations. Does this mean that the heterogeneous system we designed with the original market is no longer a good design point? The market may also undergo a large change, or it may altogether be replaced. Ideally, the system we designed would not altogether fail to provide good quality of service under a different resource manager. It is unlikely that the design strategies we propose are poorly matched to other resource allocators, but this remains an open question.

Another interesting direction is to identify key features and parameters in an allocation mechanism that affect design decisions. For example, if a resource allocation mechanism has a policy to always allocate low-power cores first, the designer might consider that the resources will be managed based on their power utilization. An open question is whether there is a may be to relay this information to the design process without requiring an end-to-end evaluation of the different design points, as was done in this work. If key features of an allocator can be extracted and communicated to the designer, it is also likely that small updates to the allocation mechanism will not necessitate updates to heterogeneous design strategies.

Bibliography

[1] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, P. Bahl, and R. Gupta, "Somniloquy : Augmenting network interfaces to reduce PC energy usage," in *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, 2009, pp. 365–380.

[2] Amazon, "Amazon Elastic Compute Cloud," 2011. [Online]. Available: http://aws.amazon.com/ec2/

[3] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan, "FAWN : A fast array of wimpy nodes introduction," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. New York, NY, USA: ACM, 2009, pp. 1–14.

[4] Anonymous, "Space invaders," *The Economist*, 2012.

[5] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai, "The impact of performance asymmetry in emerging multicore architectures," in *Proceedings of the 32nd annual international symposium on Computer Architecture*, ser. ISCA '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 506–517.

[6] L. Barroso, J. Clidaras, and U. Hölzle, *The datacenter as a computer: An introduction to the design of warehouse-scale machines*. Morgan Claypool, 2013.

[7] L. Barroso, K. Gharachorloo, R. McNamara, a. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese, "Piranha: A scalable architecture based on single-chip multiprocessing," *Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201)*, pp. 282–293, 2000.

[8] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, Dec. 2007.

[9] L. A. Barroso and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–108, Jan. 2009.

[10] M. Becchi and P. Crowley, “Dynamic thread assignment on heterogeneous multiprocessor architectures,” in *Proceedings of the 3rd conference on Computing frontiers*, ser. CF ’06. New York, NY, USA: ACM, 2006, pp. 29–40.

[11] D. Bertsekas and R. Gallager, *Data Networks.* Prentice Hall, 1992.

[12] N. Binkert, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, D. a. Wood, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, and T. Krishna, “The gem5 simulator,” *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, p. 1, Aug. 2011.

[13] F. Bower, D. Sorin, and L. Cox, “The impact of dynamically heterogeneous multicore processors on thread scheduling,” *Micro, IEEE*, vol. 28, no. 3, pp. 17–25, 2008.

[14] E. Bragg, M. Guevara, and B. Lee, “Understanding query complexity and its implications for energy-efficient web search,” in *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*, Sept 2013, pp. 401–401.

[15] S. J. Brams, M. Feldman, J. K. Lai, J. Morgenstern, and A. D. Procaccia, “On maxsum fair allocations,” in *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, 2012.

[16] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, “CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, Jan. 2011.

[17] T. Cao, S. M. Blackburn, T. Gao, and K. S. McKinley, “The yin and yang of power and performance for asymmetric hardware and managed software,” in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ser. ISCA ’12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 225–236.

[18] I. Caragiannis, C. Kaklamanis, P. Kanellopoulos, and M. Kyropoulou, “The efficiency of fair division,” *Theory of Computing Systems*, Sep. 2011.

[19] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, “Managing energy and server resources in hosting centers,” *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, p. 103, Dec. 2001.

[20] J. Chen and L. K. John, “Efficient program scheduling for heterogeneous multicore processors,” in *Proceedings of the 46th Annual Design Automation Conference*, ser. DAC ’09. New York, NY, USA: ACM, 2009, pp. 927–930.

[21] Y. Chen, J. Lai, D. Parkes, and A. Procaccia, "Truth, justice and cake cutting," in *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, 2010, pp. 756–761.

[22] ——, "Truth, justice and cake cutting," *Games and Economic Behavior*, vol. 77, no. 1, pp. 284–297, 2013.

[23] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi, and E. Rotenberg, "FabScalar : Composing synthesizable RTL designs of arbitrary cores within a canonical superscalar template," in *Proceeding of the 38th annual international symposium on Computer architecture.* New York, NY, USA: ACM, 2011, pp. 11–22.

[24] Y. Cohler, J. Lai, D. C. Parkes, and A. D. Procaccia, "Optimal envy-free cake cutting," in *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, 2011.

[25] R. Courtland, "The battle between ARM and Intel gets real," *IEEE Spectrum*, 2012.

[26] J. Davis, J. Laudon, and K. Olukotun, "Maximizing CMP throughput with mediocre cores," *14th International Conference on Parallel Architectures and Compilation Techniques (PACT'05)*, pp. 51–62, 2005.

[27] C. Delimitrou and C. Kozyrakis, "Paragon : QoS-Aware scheduling for heterogeneous datacenters," in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2013.

[28] L. Eeckhout, S. Nussbaum, J. Smith, and K. De Bosschere, "Statistical simulation: Adding efficiency to the computer designer's toolbox," *Micro, IEEE*, vol. 23, no. 5, pp. 26–38, 2003.

[29] Facebook, "More effective computing," Tech. Rep., 2011.

[30] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the Clouds: A study of emerging scale-out workloads on modern hardware," in *17th International Conference on Architectural Support for Programming Languages and Operating Systems*, no. Asplos, 2012, pp. 1–11.

[31] D. F. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini, "Economic models for allocating resources in computer systems," in *Market Based Control of Distributed Systems.* World Scientific Press, 1996, pp. 156–183.

[32] A. Gandhi, M. Harchol-balter, and M. A. Kozuch, "The case for sleep states in servers," in *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*. New York, NY, USA: ACM, 2011.

[33] S. Garg, S. Sundaram, and H. D. Patel, "Robust heterogeneous data center design: A principled approach," *SIGMETRICS Perform. Eval. Rev.*, vol. 39, no. 3, pp. 28–30, Dec. 2011.

[34] V. George, S. Jahagirdar, C. Tong, K. Smits, S. Damaraju, S. Siers, V. Naydenov, T. Khondker, S. Sarkar, and P. Singh, "Penryn : 45-nm next generation Intel ® core ™ 2 processor," in *Solid-State Conference*, 2007, pp. 14–17.

[35] G. Gerosa, S. Curtis, M. D'Addeo, B. Kuttanna, F. Merchant, B. Patel, M. Taufique, and H. Samarchi, "A sub-2W low power IA processor for mobile internet devices in 45 nm high-k metal gate CMOS," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 1, pp. 73–82, Jan. 2009.

[36] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness : Fair allocation of multiple resource types," in *Proceedings of the 8th USENIX conference on Networked Systems Design and Implementation*. Berkeley, CA: USENIX Association, 2011.

[37] S. L. Graham, P. B. Kessler, and M. K. Mckusick, "Gprof: A call graph execution profiler," in *Proceedings of the 1982 SIGPLAN symposium on Compiler construction*, ser. SIGPLAN '82. New York, NY, USA: ACM, 1982, pp. 120–126.

[38] B. Guenter, N. Jain, and C. Williams, "Managing cost , performance , and reliability tradeoffs for energy-aware server provisioning," in *Proceedings of the 30th conference on Information communications*, 2011, pp. 1332–1340.

[39] M. Guevara, B. Lubin, and B. Lee, "Navigating heterogeneous processors with market mechanisms," in *Proceedings of the 19th IEEE International Symposium on High-Performance Computer Architecture*, Feb. 2013.

[40] ——, "Strategies for anticipating risk in heterogeneous system design," in *Proceedings of the 20th IEEE International Symposium on High-Performance Computer Architecture*, Feb. 2014.

[41] V. Gupta, M. Harchol-Balter, J. G. Dai, and B. Zwart, "On the inapproximability of M/G/K: Why two moments of job size distribution are not enough," *Queueing Systems*, vol. 64, no. 1, pp. 5–48, Aug. 2009.

[42] Hewlett Packard, "HP moonshot system," http://www.hp.com/go/moonshot.

[43] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, Jul. 2008.

[44] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, “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*, ser. NSDI’11. Berkeley, CA, USA: USENIX Association, 2011, pp. 22–22.

[45] M. Horowitz, “Scaling, power and the future of cmos,” in *Proceedings of the 20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems*, ser. VLSID ’07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 23–.

[46] T. Ibaraki and N. Katoh, *Resource allocation problems: Algorithmic Approaches.* Cambridge, MA, USA: MIT Press, Jan. 1988, vol. 45, no. 1.

[47] Intel, “Vtune,” http://software.intel.com/en-us/intel-vtune.

[48] Intel Corporation, “Intel ® 64 and IA-32 architectures software developer ’ s manual,” Tech. Rep. 326018, 2011.

[49] L. Keys, S. Rivoire, and J. D. Davis, “The search for energy-efficient building blocks for the data center system overview,” in *Workshop on Energy-Efficient Design*, 2010.

[50] P. Kongetira and K. Aingaran, “Niagara: A 32-way multithreaded sparc processor,” *Micro, IEEE*, pp. 21–29, 2005.

[51] J. Koomey, “Growth in data center electricity use 2005 to 2010,” 2011.

[52] D. Koufaty, D. Reddy, and S. Hahn, “Bias scheduling in heterogeneous multi-core architectures,” in *Proceedings of the 5th European conference on Computer systems*, ser. EuroSys ’10. New York, NY, USA: ACM, 2010, pp. 125–138.

[53] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, “Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction,” in *Proceedings of the 36th International Symposium on Microarchitecture*, 2003.

[54] R. Kumar, D. M. Tullsen, and N. P. Jouppi, “Core architecture optimization for heterogeneous chip multiprocessors,” *Proceedings of the 15th international conference on Parallel architectures and compilation techniques - PACT ’06*, p. 23, 2006.

[55] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas, “Single-isa heterogeneous multi-core architectures for multithreaded workload performance,” in *Proceedings of the 31st annual international symposium on Computer architecture*, ser. ISCA ’04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 64–.

[56] J. Lai, "Truthful and fair resource allocation," Ph.D. dissertation, Harvard University, 2013.

[57] N. B. Lakshminarayana, J. Lee, and H. Kim, "Age based scheduling for asymmetric multiprocessors," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, ser. SC '09. New York, NY, USA: ACM, 2009, pp. 25:1–25:12.

[58] B. C. Lee and D. M. Brooks, "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*, ser. ASPLOS XII. New York, NY, USA: ACM, 2006, pp. 185–194.

[59] ——, "Illustrative design space studies with microarchitectural regression models," in *13th International Symposium on High Performance Computer Architecture*, no. 1. IEEE, 2007, pp. 340–351.

[60] G. Lee, B.-G. Chun, and H. Katz, "Heterogeneity-aware resource allocation and scheduling in the cloud," in *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*, ser. HotCloud'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 4–4.

[61] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, no. c. New York, New York, USA: ACM, 2009, pp. 469–480.

[62] S. Li, K. Lim, P. Faraboschi, J. Chang, P. Ranganathan, and N. P. Jouppi, "System-level integrated server architectures for scale-out datacenters," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44 '11. New York, NY, USA: ACM, 2011, pp. 260–271.

[63] T. Li, D. Baumberger, D. A. Koufaty, and S. Hahn, "Efficient operating system scheduling for performance-asymmetric multi-core architectures," in *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, ser. SC '07. New York, NY, USA: ACM, 2007, pp. 53:1–53:11.

[64] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt, "Understanding and designing new server architectures for emerging warehouse-computing environments," in *35th International Symposium on Computer Architecture*, 2008, pp. 315–326.

[65] P. Lotfi-kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, and B. Falsafi, "Scale-out processors," in *Proceedings of the 39th annual international symposium on computer architecture*, vol. 00, no. c, 2012, pp. 500–511.

[66] B. Lubin, J. Kephart, R. Das, and D. Parkes, "Expressive power-based resource allocation for data centers," in *Proc. Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[67] K. T. Malladi, B. C. Lee, F. A. Nothaft, C. Kozyrakis, K. Periyathambi, and M. Horowitz, "Towards energy-proportional datacenter memory with mobile DRAM," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ser. ISCA '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 37–48.

[68] H. Markowitz, "Portfolio selection," *The Journal of Finance*, vol. 7, no. 1, pp. 77–91, 1952.

[69] J. Mars and L. Tang, "Whare-Map : Heterogeneity in âĂIJhomogeneousâĂİ warehouse-scale computers," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, 2013, pp. 619–630.

[70] J. Mars, L. Tang, and R. Hundt, "Heterogeneity in " homogeneous " warehouse-scale computers : A performance opportunity," *IEEE Computer Architecture Letters*, pp. 1–4, 2011.

[71] D. Meisner, B. Gold, and T. Wenisch, "PowerNap: Eliminating server idle power," *ACM SIGPLAN Notices*, vol. 44, no. 3, pp. 205–216, 2009.

[72] D. Meisner, C. Sadler, L. Barroso, W. Weber, and T. Wenisch, "Power management of online data-intensive services," in *Proceeding of the 38th annual international symposium on Computer architecture.* New York, New York, USA: ACM, 2011.

[73] D. Meisner and T. F. Wenisch, "Stochastic queuing simulation for data center workloads," in *Workshop on Exascale Evaluation and Research Techniques*, 2010.

[74] J. Mogul, J. Mudigonda, N. Binkert, P. Ranganathan, and V. Talwar, "Using asymmetric single-isa cmps to save energy on operating systems," *Micro, IEEE*, vol. 28, no. 3, pp. 26–41, 2008.

[75] O. Mutlu and T. Moscibroda, "Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, 2008, pp. 63–74.

[76] R. Nathuji, C. Isci, and E. Gorbatov, “Exploiting platform heterogeneity for power efficient data centers,” *Fourth International Conference on Autonomic Computing (ICAC'07)*, pp. 5–5, Jun. 2007.

[77] Open Source, “OProfile,” http://oprofile.sourceforge.net.

[78] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, S. M. Rumble, E. Stratmann, and R. Stutsman, “The case for RAMClouds: Scalable high-performance storage entirely in DRAM,” *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 4, pp. 92–105, Jan. 2010.

[79] D. C. Parkes, A. D. Procaccia, and N. Shah, “Beyond dominant resource fairness: Extensions, limitations, and indivisibilities,” in *Proceedings of the 13th ACM Conference on Electronic Commerce*, ser. EC '12. New York, NY, USA: ACM, 2012, pp. 808–825.

[80] A. Patel, F. Afram, S. Chen, and K. Ghose, “MARSS: A full system simulator for multicore x86 CPUs,” in *Proceedings of the 48th Design Automation Conference*, ser. DAC '11. New York, NY, USA: ACM, 2011, pp. 1050–1055.

[81] a. Phansalkar, a. Joshi, L. Eeckhout, and L. John, “Measuring program similarity: Experiments with SPEC CPU benchmark suites,” *IEEE International Symposium on Performance Analysis of Systems and Software, 2005. ISPASS 2005.*, pp. 10–20, 2005.

[82] A. Phansalkar, A. Joshi, and L. K. John, “Analysis of redundancy and application balance in the spec cpu2006 benchmark suite,” in *Proceedings of the 34th annual international symposium on Computer architecture.* New York, NY, USA: ACM, 2007.

[83] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, “Cutting the electric bill for internet-scale systems,” *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, pp. 123–134, 2009.

[84] V. Reddi, B. Lee, T. Chilimbi, and K. Vaid, “Web search using mobile cores,” *Proceedings of the 37th annual international symposium on Computer architecture*, no. Table 1, p. 314, 2010.

[85] G. Ren, E. Tune, T. Moseley, Y. Shi, S. Rus, and R. Hundt, “Google-wide profiling: A continuous profiling infrastructure for data centers,” *IEEE Micro*, vol. 30, no. 4, pp. 65–79, Jul. 2010.

[86] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, “Dramsim2: A cycle accurate memory system simulator,” *IEEE Computer Architecture Letters*, vol. 10, no. 1, pp. 16–19, Jan. 2011.

[87] C. Rusu, A. Ferreira, C. Scordino, and A. Watson, “Energy-efficient real-time heterogeneous server clusters,” in *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, ser. RTAS ’06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 418–428.

[88] Seamicro, “Seamicro introduces the SM10000-64HD, setting industry record for energy efficiency and compute density,” 2011.

[89] D. Shelepov, J. C. Saez Alcaide, S. Jeffery, A. Fedorova, N. Perez, Z. F. Huang, S. Blagodurov, and V. Kumar, “HASS: A scheduler for heterogeneous multicore systems,” *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 2, pp. 66–75, Apr. 2009.

[90] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, “Automatically characterizing large scale program behavior,” in *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS X. New York, NY, USA: ACM, 2002, pp. 45–57.

[91] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz, “Analysis of a very large web search engine query log,” *SIGIR Forum*, vol. 33, no. 1, pp. 6–12, Sep. 1999.

[92] A. Snavely and D. M. Tullsen, “Symbiotic jobscheduling for a simultaneous multithreaded processor,” in *Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS IX. New York, NY, USA: ACM, 2000, pp. 234–244.

[93] S. Srinivasan, L. Zhao, R. Illikkal, and R. Iyer, “Efficient interaction between os and architecture in heterogeneous platforms,” *SIGOPS Oper. Syst. Rev.*, vol. 45, no. 1, pp. 62–72, Feb. 2011.

[94] L. Strozek and D. Brooks, “Efficient architectures through application clustering and architectural heterogeneity,” in *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, ser. CASES ’06. New York, NY, USA: ACM, 2006, pp. 190–200.

[95] M. A. Suleman, O. Mutlu, M. K. Qureshi, and Y. N. Patt, “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*, ser. ASPLOS XIV. New York, NY, USA: ACM, 2009, pp. 253–264.

[96] I. E. Sutherland, “A futures market in computer time,” *Commun. ACM*, vol. 11, no. 6, pp. 449–451, Jun. 1968.

[97] U.S. Environmental Protection Agency, “Report to congress on server and data center energy efficiency public law 109-431,” Tech. Rep., 2007.

[98] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (pie)," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ser. ISCA '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 213–224.

[99] H. R. Varian, "Equity, envy, and efficiency," *Journal of Economic Theory*, vol. 9, 1974.

[100] C. Vecchiola, R. N. Calheiros, D. Karunamoorthy, and R. Buyya, "Deadline-driven provisioning of resources for scientific applications in hybrid clouds with aneka," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 58–65, Jan. 2012.

[101] C. Waldspurger, T. Hogg, B. Huberman, J. Kephart, and W. Stornetta, "Spawn: A distributed computational economy," *IEEE Transactions on Software Engineering*, vol. 18, no. 2, pp. 103–117, 1992.

[102] J. Walrand and S. Parekh, *Communication Networks: A concise introduction*. Morgan Claypool, Synthesis Lectures, 2010.

[103] J. A. Winter, D. H. Albonesi, and C. A. Shoemaker, "Scalable thread scheduling and global power management for heterogeneous many-core architectures," in *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, ser. PACT '10. New York, NY, USA: ACM, 2010, pp. 29–40.

[104] W. Wu and B. C. Lee, "Inferred models for dynamic and sparse hardware-software spaces," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 413–424.

[105] S. Xi, M. Guevara, J. Nelson, P. Pensabene, and B. Lee, "Understanding the critical path in power state transition latencies," in *Low Power Electronics and Design (ISLPED), 2013 IEEE International Symposium on*, Sept 2013.

[106] D. H. Yoon, J. Chang, N. Muralimanohar, and P. Ranganathan, "BOOM: Enabling mobile memory based low-power server DIMMs," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, 2012, pp. 25–36.

Biography

Marisabel Guevara was born on August 20, 1986 in Santa Cruz de la Sierra, Bolivia. She received the Chancellor's Scholarship to complete her undergraduate studies at the University of Arkansas. In 2007, she was a recipient of the Google Anita Borg Scholarship. She graduated Magna Cum Laude with Honors in Computer Engineering in 2008. She was awarded a graduate fellowship from the School of Engineering and Applied Sciences at the University of Virginia, where she received a Masters in Computer Engineering in 2010. As a graduate student at Duke University, she received the Dean's Award for Excellence in Mentoring in 2013. Her research lies at the intersection of computer architecture and computational economics. Her work proposed a coordinated approach to the management and design of datacenters equipped with heterogeneous resources [14, 39, 40, 105]. In 2014 she received a PhD in Computer Science from Duke University.