Strategies for Anticipating Risk in Heterogeneous System Design

Marisabel Guevara
Duke University
mg@cs.duke.edu

Benjamin Lubin
Boston University
blubin@bu.edu

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

Abstract

Heterogeneous design presents an opportunity to improve energy efficiency but raises a challenge in resource management. Prior design methodologies aim for performance and efficiency, yet a deployed system may miss these targets due to run-time effects, which we denote as risk. We propose design strategies that explicitly aim to mitigate risk. We introduce new processor selection criteria, such as the coefficient of variation in performance, to produce heterogeneous configurations that balance performance risks and efficiency rewards. Out of the tens of strategies we consider, risk-aware approaches account for more than 70% of the strategies that produce systems with the best service quality. Applying these risk-mitigating strategies to heterogeneous datacenter design can produce a system that violates response time targets 50% less often.

1. Introduction

The geometric growth of data stored on the cloud calls for rapid growth in datacenter compute capability. The challenge lies in meeting this demand with datacenters that already dissipate megawatts of power. Datacenters can benefit from unconventional mixes of highly efficient hardware. The power savings afforded by mobile processors outweigh their performance limitations [21, 32, 23]. Mobile technologies that dissipate $\frac{1}{10}$th the power of server components allow datacenters to deploy more servers, increase throughput, and improve service quality within a given power budget [11].

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 [15, 18]. 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 this paper, 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.

We make 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 (§2).
- **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).
- **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 (§5).
- **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 (§6).

Our work lays a foundation for datacenters that are heterogeneous by design. We challenge traditional heterogeneous system design that selects processors from a design space assuming an ideal, yet unlikely, mapping of applications to resources. Instead, we show that our novel risk-aware design strategies produce systems with better service quality than traditional risk-agnostic design. Collectively, our findings make the case for rethinking heterogeneous design strategies to account for run-time risk.

2. 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.

2.1. 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 [15, 18, 7] and datacenter organizations [11]. 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 $\frac{1}{3}\times$ [21, 32]. 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 [23]. 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 [11]. Achieving this improvement in service quality depends on effectively managing heterogeneous resources. In this paper, 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.

2.2. 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 [36] and benchmark redundancy analysis [31]. 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 [14, 7, 18, 40]. 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 [14, 7, 18, 40]. 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.

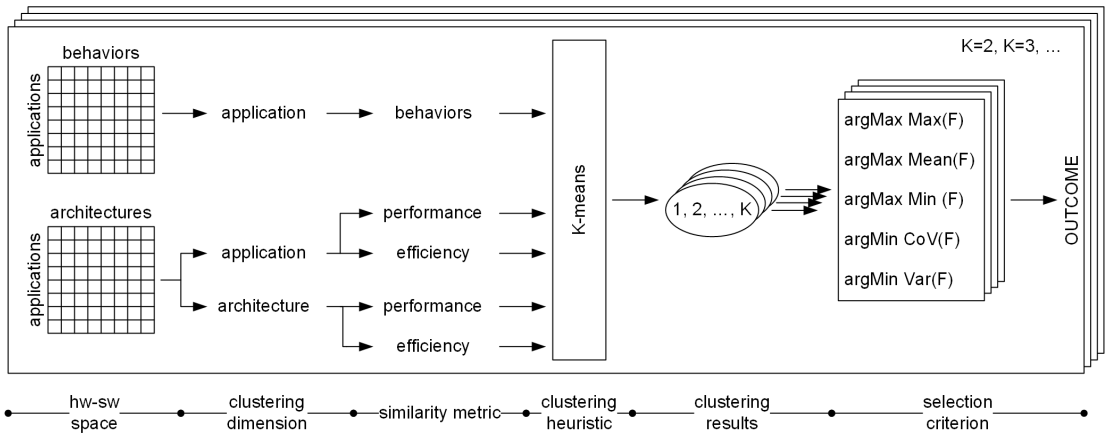


Figure 1: A Strategy Framework shows the numerous approaches to heterogeneous design.

3. 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 1, and detail each stage in which a decision is made (e.g., hardware-software space, clustering dimension, etc.).

3.1. 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. 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.

3.3. 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.

3.4. 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 [15, 18, 7].

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.

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

Table 1: Processor Design Space.

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 applica-

tions. 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 [24].

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.

3.5. 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 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 §5.

[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.

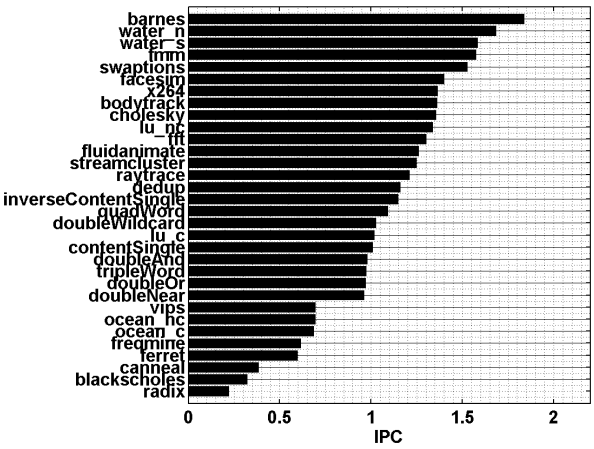


Figure 2: Application Diversity (IPC on Core 42 of Table 1).

4. Experimental Methodology

To evaluate heterogeneous processor design strategies for datacenters, we deploy a comprehensive methodology. Cycle-accurate processor and memory simulations 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 [30], integrated with DRAMSIM2 [34], to simulate the 42 processors listed in Table 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 3–4. We use McPAT/CACTI at 32nm to model power and area [19]. 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 [37]. A query type denotes whether page *content* or *title* is searched. Wildcards and searches for similar words (*near*) are supported. Negative queries 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 2. Diverse search queries vary in computational

	barnes|radix		barnes|quadword		radix|quadword	
Task Profile (Mcycles/task)	2972	1326	2972	32	1326	32
Daily Peak	420	1884	420	76827	942	76827
Weekly Peak (tasks/min)	1261	5652	1261	230482	2826	230482
Value (K$/month)	$5 if $T \leq 200$ms $0 if $T \geq 1000$ms	$5 if $T \leq 200$ms $0 if $T \geq 1000$ms	$5 if $T \leq 200$ms $0 if $T \geq 1000$ms	$5 if $T \leq 20$ms $0 if $T \geq 160$ms	$5 if $T \leq 200$ms $0 if $T \geq 1000$ms	$5 if $T \leq 20$ms $0 if $T \geq 160$ms
Scaling Factors	0.19-1.0	0.79-1.0	0.19-1.0	0.25-1.0	0.79-1.0	0.25-1.0

Table 2: Application Parameters in the Market.

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: Out-of-order.

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 4: In-order.

	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 5: Outcomes from BIPS 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

Table 6: Outcomes from BIPS^3/W strategies.

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 [26]. Such patterns have been used to evaluate other datacenters [5, 11, 22].

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 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 [5, 22, 11], 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$^{\text{th}}$ percentile response time. Value degrades linearly as response time increases up to some cut-off, after which computation has no value (Table 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 [5, 11, 22].

We apply the market to the heterogeneous outcome of a design strategy from Figure 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 [32]. 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[11]. 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 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 [10], VTune [12], or OProfile [29]. 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 [33]. Given sparsely sampled profiles, statistical machine learning can fit models and predict scaling factors [43].

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 §6.1 we evaluate each heterogeneous outcome and its sensitivity to fractional shares.

5. Designing for Manageability

In this section, we compare the strategies that produced the 50 unique outcomes listed in Tables 5–6. 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 produces the outcomes in Tables 5–6, where each entry is a tuple of identifiers that map to the microarchitectures in Table 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
- **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.

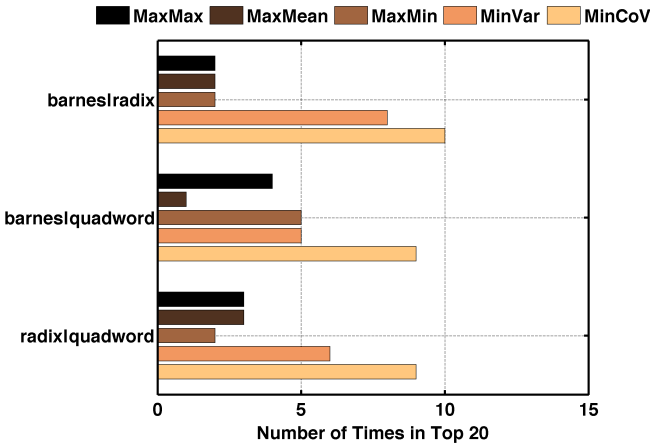


Figure 3: Risk-aware strategies are more likely to produce outcomes with the best service quality.

Mitigating Risk During Design. Prior approaches to heterogeneous design 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 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 cri-

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

teria 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 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 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.

6. 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 (§6.1). 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 (§6.2).

6.1. 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.

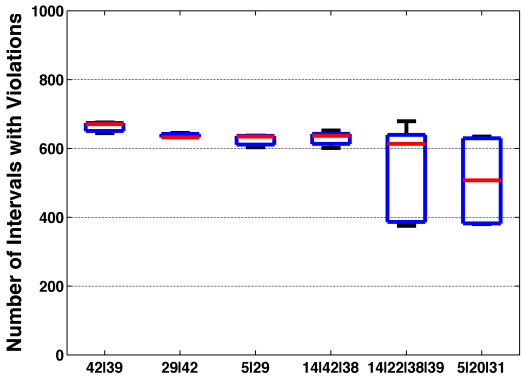


Figure 4: Heterogeneous system `14|22|38|39` exhibits more risk yet reduces response time violations by 50% relative to low-risk system `42|39` (barnes|radix).

In a datacenter, this reward is a reduction in the number of allocation periods that incur response time violations.

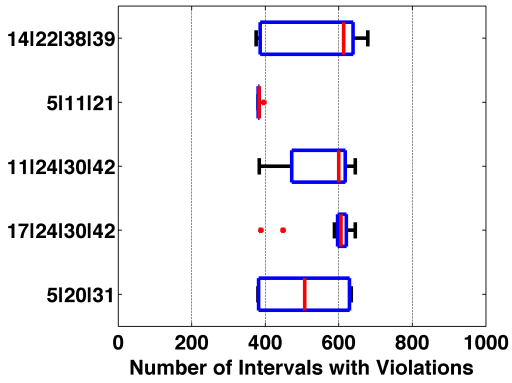
However, it is difficult to provision diverse resources in proportions that match the application mix. A strategy's outcome defines a set of heterogeneous cores, but not their organization in the system. Our evaluation thus far has assumed that the system's power budget is divided amongst heterogeneous core types such that service quality is optimized. Yet identifying each core type's 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 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.

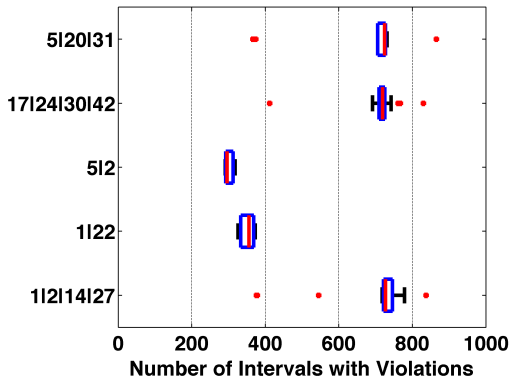
For an application mix, the box shows the effect of different fractional shares. Given K heterogeneous cores, we evaluate all combinations of $\frac{1}{K}$-sized fractions within a fixed power budget (e.g., 20KW). For example, when $K = 2$, core types can be organized into fractions of 1:0, $\frac{1}{2}$:$\frac{1}{2}$, 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 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.

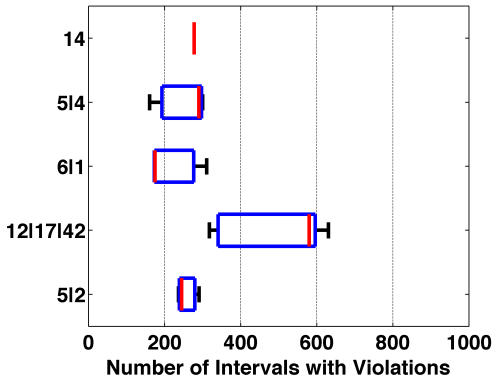
Reward. Despite the increase in system risk, the reward is a significant improvement in service quality. The highest ranked



(a) **barnes|radix**



(b) **barnes|quadword**



(c) **radix|quadword**

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

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 4.

barnes|radix. Most top ranked outcomes in Figure 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 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.

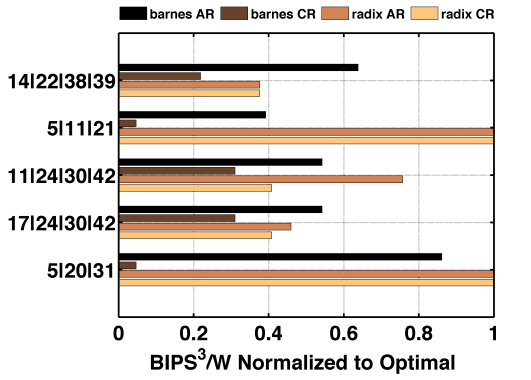
6.2. Quantifying Risks to Efficiency

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.

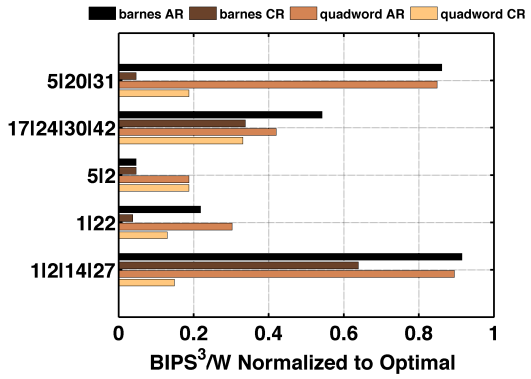
An application may not realize the upper bound on efficiency and instead run on a less efficient core for two reasons. First, an application's most efficient core may not be available in the heterogeneous outcome chosen for the system. In this scenario, efficiency is lost due to application risk. Second, an application's most efficient core may be present but allocated to another application, which is contention risk.

In Figure 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.

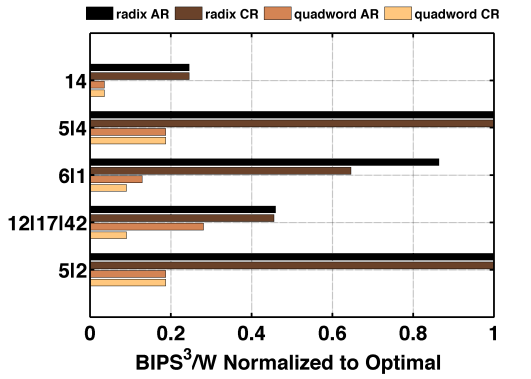
barnes|radix. Figure 6a shows efficiency losses due to application and contention risk. Radix executes at near-optimal efficiency for a few of the top ranked outcomes (i.e., `5|11|21` and `5|20|31`). Reconciled rankings closely align with radix's ranking. If the second- and third-ranked configurations are chosen, efficiency losses are zero.



(a) **barnes|radix**



(b) **barnes|quadword**



(c) **radix|quadword**

Figure 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).

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 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 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.

7. Related Work

Kumar et al. consider existing cores drawn from multiple design generations [14, 16]. Alternatively, Kumar et al. exhaustively simulate and search a space with hundreds of designs to maximize performance subject to power and area constraints [15]. Choudhary et al. evaluate synthesizable core designs in heterogeneous mixes [7]. 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 tradeoffs tractably, and use K-means clustering in their optimization [18]. Strozek and Brooks similarly study clustering strategies for heterogeneous embedded systems [40]. 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 a broad space of strategies. Unlike prior work, we anticipate run-time manageability at design-time.

While much prior work in distributed systems have considered diverse tasks and heterogeneous VMs, the underlying processors are often homogeneous by design. At present, heterogeneity in datacenters is modest and involves multiple processor generations [28, 25] or frequencies [22]. However, studies of datacenter software on diverse hardware motivate greater

heterogeneity due to the potential for efficiency [21, 32, 9].

To anticipate run-time manageability, we deploy a framework that uses a market mechanism to assign cores to tasks [22, 11, 5]. These mechanisms operate at datacenter scale, examining application preferences for hardware and allocating cores to task streams.

A large body of work studies scheduling in heterogeneous CMPs. Much of this work focuses on profiling and thread migration. Scheduling software to heterogeneous hardware might account for memory-level parallelism [8], instruction-level parallelism [2, 13], resource demands [1, 6, 35, 38, 39], thread age [17], load balance [20], or hardware faults [42, 3]. Scheduling is simplified when big or small cores are used for a specific purpose. Big cores can accelerate critical sections in parallel computation [41] while small cores can efficiently support the operating system [27] or managed software [4].

8. Conclusions

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.

Acknowledgements

This work is supported by NSF grants CCF-1149252 (CAREER) and CCF-1337215 (XPS-CLCCA). This work is also supported by STARnet, a Semiconductor Research Corporation program, sponsored by MARCO and DARPA. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of these sponsors.

References

[1] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai, "The impact of performance asymmetry in emerging multicore architectures," in *ISCA*, 2005.
[2] M. Becchi and P. Crowley, "Dynamic thread assignment on heterogeneous multiprocessor architectures," *JILP*, 2008.
[3] F. Bower, D. Sorin, and L. Cox, "The impact of dynamically heterogeneous multicore processors on thread scheduling," *IEEE Micro*, 2008.
[4] T. Cao, S. Blackburn, T. Gao, and K. McKinley, "The yin and yang of power and performance for asymmetric hardware and managed software," in *ISCA*, 2012.
[5] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle, "Managing energy and server resources in hosting centers," in *SOSP*, 2001.
[6] J. Chen and L. John, "Efficient program scheduling for heterogeneous multi-core processors," in *DAC*, 2009.
[7] N. Choudhary, S. Wadhavkar, T. Shah, H. Mayukh, J. Gandhi, B. Dwiel, S. Navada, H. Najaf-abadi, and E. Rotenberg, "FabScalar: Composing synthesizable RTL designs of arbitrary cores within a canonical superscalar template," in *ISCA*, 2011.
[8] K. V. Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, "Scheduling heterogeneous multi-cores through performance impact estimation (PIE)," in *ISCA*, 2012.
[9] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," in *ASPLOS*, 2012.
[10] S. Graham, P. Kessler, and M. Mckusick, "Gprof: A call graph execution profiler," in *CC*, 1982.
[11] M. Guevara, B. Lubin, and B. C. Lee, "Navigating heterogeneous processors with market mechanisms," in *HPCA*, 2013.
[12] Intel, "Vtune," http://software.intel.com/en-us/intel-vtune.
[13] D. Koufaty, D. Reddy, and S. Hahn, "Bias scheduling in heterogeneous multi-core architectures," in *EuroSys*, 2010.
[14] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen, "Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction," in *MICRO*, 2003.
[15] R. Kumar, D. Tullsen, and N. Jouppi, "Core architecture optimization for heterogeneous chip multiprocessors," in *PACT*, 2006.
[16] R. Kumar, D. Tullsen, P. Ranganathan, N. Jouppi, and K. Farkas, "Single-ISA heterogeneous multi-core architectures for multi-threaded workload performance," in *ISCA*, 2004.
[17] N. Lakshminarayana, J. Lee, and H. Kim, "Age based scheduling for asymmetric multiprocessors," in *SC*, 2009.
[18] B. Lee and D. Brooks, "Illustrative design space studies with microarchitectural regression models," in *HPCA*, 2007.
[19] S. Li, J. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, 2009.
[20] T. Li, D. Baumberger, D. Koufaty, and S. Hahn, "Efficient operating system scheduling for performance-asymmetric multi-core architectures," in *SC*, 2007.
[21] 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 *ISCA*, 2008.
[22] B. Lubin, J. Kephart, R. Das, and D. Parkes, "Expressive power-based resource allocation for datacenters," in *IJCAI*, 2009.
[23] K. Malladi, B. Lee, F. Nothaft, C. Kozyrakis, K. Periyathambi, and M. Horowitz, "Towards energy-proportional datacenter memory with mobile DRAM," in *ISCA*, 2012.
[24] H. Markowitz, "Portfolio selection," *The Journal of Finance*, vol. 7, no. 1, pp. 77–91, 1952.
[25] J. Mars, L. Tang, and R. Hundt, "Whare-map: Heterogeneity in "homogeneous" warehouse-scale computers," in *ISCA*, 2013.
[26] D. Meisner, C. Sadler, L. Barroso, W. Weber, and T. Wenisch, "Power management of online data-intensive services," in *ISCA*, 2011.
[27] J. Mogul, J. Mudigonda, N. Binkert, P. Ranganathan, and V. Talwar, "Using asymmetric single-ISA CMPs to save energy on operating systems," *IEEE Computer*, 2008.
[28] R. Nathuji, C. Isci, and E. Gorbatov, "Exploiting platform heterogeneity for power efficient data centers," in *ICAC*, 2007.
[29] Open Source, "OProfile," http://oprofile.sourceforge.net.
[30] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSSx86: A full system simulator for x86 CPUs," in *DAC*, 2011.
[31] A. Phansalkar, A. Joshi, L. Eeckhout, and L. John, "Measuring program similarity: Experiments with SPECCPU benchmark suites," in *ISPASS*, 2005.
[32] V. Reddi, B. Lee, T. Chilimbi, and K. Vaid, "Web search using mobile cores: Quantifying and mitigating the price of efficiency," in *ISCA*, 2010.
[33] 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*, 2010.
[34] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *CAL*, 2011.
[35] D. Shelepov, J. S. Alcaide, S. Jeffery, A. Fedorova, N. Perez, Z. Huang, S. Blagodurov, and V. Kumar, "HASS: A scheduler for heterogeneous multicore systems," in *Operating Systems Review*, 2009.
[36] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *ASPLOS*, 2002.
[37] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz, "Analysis of a very large web search engine query log," in *SIGIR*, 1999.
[38] A. Snavely and D. Tullsen, "Symbiotic job scheduling for simultaneous multithreading processor," in *ASPLOS*, 2000.
[39] S. Srinivasan, L. Zhao, R. Illikkal, and R. Iyer, "Efficient interaction between OS and architecture in heterogeneous platforms," in *Operating System Review*, 2011.
[40] L. Strozek and D. Brooks, "Efficient architectures through application clustering and architectural heterogeneity," in *CASES*, 2006.
[41] M. Suleman, O. Mutlu, M. Qureshi, and Y. Patt, "Accelerating critical section execution with asymmetric multi-core architectures," in *ASPLOS*, 2009.
[42] J. Winter, D. Albonesi, and C. Shoemaker, "Scalable thread scheduling and global power management for heterogeneous many-core architectures," in *PACT*, 2010.
[43] W. Wu and B. Lee, "Inferred models for dynamic and sparse hardware-software spaces," in *MICRO*, 2012.