**Optimizing Resource Allocations for Dynamic Interactive Applications**

by

Sarah Lynn Bird

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Krste Asanović, Co-chair
Professor David Patterson, Co-chair
Doctor Burton Smith
Professor David Wessel

Spring 2014

**Optimizing Resource Allocations for Dynamic Interactive Applications**

## Abstract

Optimizing Resource Allocations for Dynamic Interactive Applications

by

Sarah Lynn Bird

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Krste Asanović, Co-chair

Professor David Patterson, Co-chair

Modern computing systems are under intense pressure to provide guaranteed responsiveness to their workloads. Ideally, applications with strict performance requirements should be given just enough resources to meet these requirements consistently, without unnecessarily siphoning resources from other applications. However, executing multiple parallel, real-time applications while satisfying response time requirements is a complex optimization problem and traditionally operating systems have provided little support to provide QoS to applications. As a result, client, cloud, and embedded systems have all resorted to over-provisioning and isolating applications to guarantee responsiveness. Instead, we present PACORA, a resource allocation framework designed to provide responsiveness guarantees to a simultaneous mix of high-throughput parallel, interactive, and real-time applications in an efficient, scalable manner. By measuring application behavior directly and using convex optimization techniques, PACORA is able to understand the resource requirements of applications and perform near-optimal resource allocation—2% from the best allocation in 1.4 ms while only requiring a few hundred bytes of storage per application.

# Contents

# List of Figures

# List of Tables

## Acknowledgments

I would like to first thank my advisors: Krste Asanović, Dave Patterson, and Burton Smith for being amazing mentors. They have all provided immeasurable contributions to my research and career, and it'd be hard to imagine myself or my work without their influence. I'd particularly like to thank Krste for his observant, thoughtful, and practical nature, which was often critical in managing projects, people, and research challenges. I'd like to thank Dave for his passionate and tireless leadership at Berkeley and his incredible ambition for producing world-changing research. He's been an amazing and inspiring example of how turn visions into reality. I'd like to thank Burton for being an incredible role model, mentor, colleague, and co-conspirator since we began working together on PACORA. Burton has encouraged, challenged, and supported all my ideas and has really helped me mature as a researcher. I can't thank him enough for pushing me to tackle more ambitious problems and then helping me with them along the way. I would also like to thank my other committee member, David Wessel, for providing one of the inspiring applications for PACORA and Tessellation and his constant support and feedback as both projects progressed.

Thanks to the ParLab Architecture group, particularly the Cosmic Cube, for being amazing collaborators, friends, and teachers during my time in Berkeley. I would especially like to thank Henry Cook, whose shared interest in resource partitioning and application modeling has led to many fun and fruitful collaborations over the years. Without his shared enthusiasm, PACORA never would have happened. Thanks to Miquel Moreto, who worked with us to improve and evaluate several of our ideas. I'd also like to thank Andrew Waterman for his support, company, and occasional pressure during many late nights working in the ParLab.

Thanks to the Tessellation team, for providing a home for PACORA. Particular thanks to John Kubiatowicz, who has been like an advisor during my time at Berkeley. His wild, crazy, and occasionally overly-ambitious ideas have been a great source of inspiration for my research. I'd also like to thank Gage Eads helping develop many of the support pieces for PACORA in Tessellation, and for spending many hours with me in a tiny room with a large server making things work. Thanks to Juan Colmenares for his endless dedication to making our ideas for Tessellation a reality. Thanks to our technical support staff, Kostadin Ilov and Jon Kuroda, who didn't flinch at any of the strange prototype hardware or software we asked them for to support for our research.

Thanks to Stephen Boyd, for teaching me about convex optimization and providing guidance on our PACORA implementation, and thanks for his infectious enthusiasm for applying convex optimization to real world problems. I'd also like to thank Lin Xiao helping to implement our ADMM algorithm. Thanks to Kevin Peterson for his endless patience while teaching me MATLAB.

Thanks to everyone at Microsoft and Intel who have given advice, encouragement, and constructive criticism as we have refined the ideas in PACORA. Particular thanks to the Windows developers, who despite initial skepticism, have provided invaluable feedback, advice, and support for PACORA. Thanks to everyone at Intel, particularly Gans Srinivasa

and Mark Rowland, for supporting PACORA and providing experimental hardware to help test our ideas.

Thanks to all my colleagues in Parlab, for providing an amazing community during my time at Berkeley. Their brilliant research is a constant source of inspiration, and I'm very grateful to have had the opportunity to interact with them on a daily basis. I'd also like to thank all my friends at Berkeley for helping to make graduate school so much more than just research. Finally, I'd like to thank my family for always understanding and encouraging me to take risks and follow my dreams.

# Chapter 1

# Introduction

As growing on-chip hardware parallelism delivers increasing processing capabilities, users expectations of their personal computing devices grows as well. Today's users expect snappy operation from high-quality multimedia and interactive applications that call for responsive user interfaces and stringent real-time guarantees from the systems that host them. As result, providing responsiveness is a growing need for all types of systems, ranging from webservers and databases running on cloud systems, through interactive multimedia applications on mobile clients, to emerging distributed embedded systems.

Perhaps the most important component of interactive system performance is the behavior of the operating system (OS). Surprisingly, over the last 30 years the operating system kernels on which most systems rely have been built on a minicomputer foundation, and the major advances in performance, human-computer interfaces and graphics, as dramatic as these have been, have left their architecture relatively untouched. The list of standard OS concepts: interrupts, device drivers, priority thread scheduling, demand paging, and the like would be familiar to OS kernel developers of the 1980's. Developers have been able to avoid rethinking the operating system kernel chiefly because the hardware platform it ran on didn't change in any fundamental way [139]. Only the presence of specialized real-time operating systems provide a hint that modern, general-purpose OSes are not up to all of the tasks a developer might demand of them. These traditional OS architectures were designed to maximize utilization with little consideration for individual application quality-of-service (QoS) and thus provide few mechanisms to describe application deadlines or guarantee their responsiveness.

Often the only way to guarantee performance is to remove the possibility of interference from other applications all together, and evidence of this behavior can be found in current systems of all sizes. Cloud computing providers routinely utilize their clusters at only 10% to 50% to keep the system responsive, despite the significant impact on infrastructure capital costs and the additional operational costs of consuming electricity [12, 58, 84]. In some cases, cloud providers run only a single application on a cluster to avoid unexpected interference [84]. Some mobile systems have gone so far as to limit which applications can run in the background [8] in order to preserve responsiveness and battery life, despite the obvious

concerns for the overall user experience. In the embedded space, realtime developers often use completely separate systems for each application to ensure QoS, despite the high cost of this approach.

However, such significant over-provisioning can not continue indefinitely into the future: users will continue to demand increasing performance and functionality for their applications. Simply expanding the number of hardware resources isn't a viable solution: battery life and power bills can not be increased at the rate required to meet this demand [84, 145, 62]. Consequently, the industry has no choice but to improve the efficiency at which these systems operate [84, 11], if they hope to meet customers growing expectations.

## 1.1 Resource Allocation

Now the problem becomes how best to make efficient use of computing resources while satisfying QoS requirements for a dynamically changing and complex mix of simultaneously running applications. Traditionally, this problem was reduced to scheduling threads on a single processor. In most systems, there was only a single processor that ran at top speed almost all the time. Memory was time-multiplexed too, to a much lesser extent, but other resources were deemed so abundant as to require no explicit management at all (*e.g.,* I/O, network bandwidth. With modern hardware diversifying to include a variety of parallel and possibly heterogenous architectures (*e.g.,* multicore, GPUs) and systems running multiple parallel, real-time applications at once, the situation becomes significantly more complex. Concurrently running applications might interfere with each other through shared resources, causing applications to experience unpredictable performance degradations if the system only considers CPU resources. For example, if two compute-bound threads are simultaneously scheduled on two different cores of a multicore processor, there may be no degradation versus running each in isolation. However, if the two threads are memory-bandwidth constrained, simultaneous scheduling could dramatically impair performance. Even more complex behaviors may occur if cores or hardware thread contexts share functional units, caches, or TLBs. Some applications may not scale well enough to utilize a given resource while other applications may fail to meet user-driven deadlines given too few resources. Even if only one application is running, a new responsibility for the OS in the manycore era is to maximize performance and energy-efficiency of that sole app by only allocating the appropriate resources. For example, allocating too many cores may cause the application to slow down, or consume additional power for no additional performance gain.

As a result, the problem begins to look more like a *resource allocation* optimization where the system must figure out how to give just enough of a variety of system resources (*e.g.,* nodes, processor cores, cache slices, memory pages, various kinds of bandwidth) to applications to meet their performance requirements consistently.

Many current systems address this problem by requiring applications to request a specific number of resources [147, 59], and if resources are oversubscribed the system seeks to degrade performance fairly [147, 51, 7, 6, 158, 71]. While simple to implement, this approach has a few

drawbacks: it requires additional application developer effort to understand the resources required and developers often request many times more resources than they need to be safe [84]; it is less robust to resource changes and requires the developer to update the requirements as new hardware is released; and an application does not have the global view of what else has to run in the system, so it can not request resources based on the relative importance of its tasks compared to others in the system.

## 1.2 PACORA

In this thesis, we present PACORA: Performance-Aware Convex Optimization for Resource Allocation, a resource-allocation framework that determines the appropriate resources for applications running in a system without requiring the application developers to understand resource usage. PACORA seeks to dynamically assign resources across multiple applications to guarantee responsiveness without over-provisioning and adapt allocations as the application mix changes. It is a generic framework that we believe is applicable in many resource-allocation scenarios, from cloud providers determining how many resources to give each job to avoid violating Service-Level Agreements (SLAs), through databases allocating resources to queries, to distributed embedded systems allocating bandwidth among devices and sensors.

PACORA specifically focuses on the problem of *how much* of each resource type to assign to each application, and unlike many other resource allocators, PACORA considers *all* resource types when making decisions. Rather than allocating resources to maximize a system's aggregate performance or its hardware utilization as many resource allocation systems do, PACORA mathematically optimizes a single objective function designed to accurately reflect the value of the system to its customer(s) [153]. This point of view is often cast in the literature as the problem of defining and maximizing utility [142]; PACORA minimizes a negative utility - the penalty - instead. PACORA explicitly represents system power as a competing "application", so resources that can do little to reduce the penalty of other applications are automatically powered down to reduce it's penalty and improve efficiency.

PACORA takes an uncommon approach to resource allocation, relying heavily on application-specific functions built through measurement and convex optimization. PACORA's functions explicitly represents application deadlines rather than simply the application's relevant importance, as with priority systems. Knowing the deadlines allows the system to make continuous trade-offs among application responsiveness, system performance, and energy efficiency, and lets the system make optimizations that are difficult in today's systems, such as running just fast enough to make the deadline.

Using runtime measurements, application-specific performance models are built and maintained to help determine the resources required to meet each application's deadlines. PACORA leverages partitioning mechanisms in the system to create isolation between ap-

plications. Reasonable isolation virtualizes the performance of a machine[1], which allows PACORA to build high-fidelity performance models of application runtimes independent of what is happening in the rest of the system.

PACORA uses convex optimization to perform real-time resource allocation inexpensively, dynamically allocating resources to adjust to the changing state of the system. The optimization problems involved are tiny by contemporary standards, and solutions are quite fast. Moreover, the adaptive, closed-loop nature of the allocation process means that a solution need not be optimal to be beneficial; PACORA is incessantly working to reduce total penalty.

We choose to study PACORA implemented in a general-purpose operating system for client systems, because we believe this scenario has some of the most difficult resource allocation challenges: a constantly changing application mix requiring low overhead and fast response times, shared resources that create more interference among the applications, and platforms that are too diverse to allow *a priori* performance prediction. PACORA's models require only a few hundred bytes of additional storage per application in our OS implementation, and with this negligible overhead, PACORA is able to dynamically allocate resources to adjust to the changing state of the system and trade off responsiveness and energy. PACORA makes resource allocation decisions in 350 µs in the worst case and often faster than 50 µs. Static allocation decisions are near optimal—only 2% from the best possible allocation on average. By building application-specific functions online and formulating resource allocation as an optimization problem, PACORA is able to accomplish multi-dimensional resource allocation on a general set of resources, thereby handling heterogeneity and the growing diversity of modern hardware while protecting application developers from needing to understand resources.

## 1.3 Contributions

In this thesis, we have contributed the following:

1. A simple but effective model to represent application performance for resource allocation
2. A function to represent the application deadlines and importance
3. A framework to perform multidimensional resource allocation in realtime

## 1.4 Collaborations

Much of the work in this thesis was part of collaborations with others. Here I attempt to attribute these contributions to the proper individuals.

[1]Virtualized performance means that given a subsection of the machine (*e.g.,* 2 cores and 3 cache slices), the application will behave as if it was on a separate machine of that size

Henry Cook and I shared an interest in hardware partitioning mechanisms and application modeling. The initial RTF study in Chapter 3 was a joint effort to explore these interests. Miquel Moreto joined the collaboration for a study of the effect of cache partitioning on a variety of modern benchmarks [38]. Data collected from this study was reused for the PACORA static allocation analysis in Chapter 4, and many of the results regarding benchmark behavior are shown throughout the thesis.

Burton Smith added the idea of using convexity to Henry's and my initial research and has worked closely with me to develop, revise and extend the PACORA formulation for many years. Stephen Boyd and Lin Xiao provided inspiration and advice regarding the use of convex optimization and worked closely with us to create our ADMM formulation and implementation.

The Tessellation and RAMP teams both provided platforms that were used to collect data shown in this thesis [137, 35]. Gage Eads, in particular, worked closely with me to help develop the dynamic video experiment in Chapter 6.

# Chapter 2

# Related Work

Over the years, there has been much work in scheduling and resource management in batch and high-performance computing systems, operating systems, real-time computing, hardware, and more recently cluster and datacenter management. In this chapter, we present the work most related to PACORA, particularly focusing on performance prediction, modeling, and satisfying QoS requirements such as deadlines or Service-Level Objectives (SLOs).

## 2.1 Batch Scheduling and Cluster Management

Classic resource management systems were designed for batch scheduling [45, 47]. Like PACORA, batch scheduling systems rely on a gang-scheduling model [46] and can allocate multiple resource types. However, they tend to use queues and priorities to schedule jobs while trying to keep all resources busy. Jobs are placed into a queue based on the priority of the job according the system administrator. When resources become available, the job in the highest priority queue is scheduled first. Resource allocations are always the user's responsibility to specify and pay for. Responsiveness can be improved by buying a higher priority or more resources. Few batch systems incorporate deadlines; however, there are a few exceptions such as scheduling for the Tera MTA [4].

Modern cluster resource management has a similar flavor. In systems such as Amazon EC2 [5], Eucalyptus [108], and Condor [118], users must specify their resource requirements. Other systems such as Hadoop [7, 6, 158] and Quincy [71] use a fairness policy to assign resources. In Yarn [147], the Resource Manager also uses fairness to assign resources. However, Application Managers track resource needs of applications and sends them to the Resource Manager, so the application developer does not need to specify them. Mesos [59] uses two-level scheduling to manage resources in a cluster. Mesos decides how many resources to offer to its applications; they decide which resources to accept and how to schedule them. Mesos does not provide a particular resource allocation policy, but is a framework that can support multiple policies. Dominant Resource Fairness [51], a generalization of max-min fairness to multiple resources types, has been implemented in Mesos. PACORA could be implemented

as another resource allocation policy in Mesos.

## 2.2 Co-scheduling Applications

Much recent work has focused on the problem of choosing which applications or VMs to schedule together to minimize interference. Interference, which can significantly slow down applications, typically is the result of applications interacting in caches or other shared resources. Unlike PACORA, which solves the problem of how many resources to give an application, co-scheduling techniques focus on placing applications on particular resources. The majority of these techniques concentrate on quantifying or predicting the interference between co-scheduled applications.

Some of these approaches could be combined with PACORA in different ways. For example, PACORA could determine the total allocation and then one of these approaches could be used to place the applications, or PACORA could be used to partition resources on a single node after one of these techniques selected which applications to co-schedule there. Alternatively, PACORA could potentially replace these techniques by looking for combinations of applications with low penalty to co-schedule together. While not tested specifically, there is no reason to believe that approach wouldn't work in systems with resource partitioning. However, many of the co-scheduling techniques focus on interference from shared resources that are not partitioned. PACORA assumes that its response-time functions (RTFs) are independent of the other applications on the machine, and shared resources could easily violate this assumptions. In systems with significant interference from shared resources, the co-scheduling techniques that quantify or predict interference would work better.

### Disjoint Resource Utilization

One line of work investigated techniques on single nodes to co-schedule applications with disjoint resource requirements to minimize interference, for example, executing a compute-bound and memory-bound application concurrently [136, 27, 125, 160, 166].

Shen *et al.* showed that using hardware measurement information for resource-aware scheduling resulted in a 15-70% reduction in request latency over default Linux for RUBiS, TPC-C, and TPC-H [125]. Zhang *et al.* [160] used a similar technique and found that most applications receiving a 7–8% boost in performance over traditional scheduling and a 58% reduction in unfairness.

Calandrino *et al.* [27] uses working set sizes to make co-scheduling decisions and enhance soft real-time behavior. Merkel and Bellosa [100] try to co-schedule applications with disjoint energy usage. Their technique uses performance counters to predict the energy consumptions of tasks and then tries to schedule to maximum performance within the thermal limits of the system. Merkel and Bellosa [101] later propose *Task Activity Vectors* that describe how much each application uses the various functional units; these vectors are used to balance usage across multiple cores and unbalance usage among hardware threads within each core.

The intended effect is to distribute chip temperature more evenly, but the idea may be more broadly applicable, *e.g.,* for heterogeneous systems.

## Interference Experiments

Another body of work has relied running online experiments with different combinations of applications and then selecting the highest performing combination. Tang *et al.* [140] use an adaptive approach to map threads-to-cores. The approach uses an exploration period where it tries different thread-to-core mappings and then selects the highest performing one. Mars *et al.* [97] present run experiments in advance to characterize the pressure each application generates in the memory subsystem and the sensitivity to memory pressure. They use this information to select applications that will run well together. Zheng *et al.* [162] use a sandboxed environment to run experiments for collocating applications and then use the results to generalize to the larger datacenter.

## Predicting Interference

Another line of work explores past measurements or performance models to predict the expected interference between applications. Cuanta [54] focuses on predicting the slowdown from cache effects by creating a performance lookup table per application, but requires access to physical memory addresses. In [152], West *et al.* use hardware performance counters to estimate cache occupancy. The estimated occupancy is then fed into an analytical model to predict cache misses for co-scheduled applications. Verma *et al.* [148] assumes that the cache occupancy is provided by the applications and then uses heuristics to co-schedule applications to minimize cache interference.

Koh *et al.* [81] predict performance degradation of co-scheduled applications using the resource utilization statics of the applications. For each application they build a resource usage vector which includes cache, processor, disk, and network utilization information. In a technique similar to program similarity analysis [63], they compare the resource vector of a new application with historical information from other applications. The predicted performance degradation is based on a weighted sum of the observed performance degradation of applications with the most similar resource vectors. Stewart *et al.* [131] predict resource usage based on the transaction mix and combine that information with expected queuing delays to co-schedule applications.

Paragon uses profiling data combined with collaborative filtering techniques to determine on which server to place an application based on the server configuration and co-scheduled applications [40]. Dejavu [146] categorizes applications into workload classes, and then uses the workload class to determine an appropriate allocation and co-scheduling for application. Dejavu caches preferred co-schedules and uses an interference index to evaluate new placements.

## 2.3 Model-Based Scheduling and Allocation Frameworks for SLOs and Soft Real-Time Requirements

In both the cloud computing and realtime communities, there is a growing interest in use application-specific performance "models" to try to schedule to meet deadlines or SLOs.

### Autonomic Computing and Utility Functions

Much of this research has been in autonomic computing [103, 141, 143, 82]. Typically, the performance models are utility functions derived from off-line measurements of raw resources utilization. These functions are either interpolations from tables or analytic functions based on queueing theory [103]. The utility functions typically map the number of servers each execution environment receives to its performance relative to its requirements. A central arbiter maximizes total utility. The utility functions are not necessarily concave, so the arbiter must use reinforcement learning [141] or combinatorial search [103] to make allocations. Each application has a manager that schedules the resources given to it by the arbiter. Walsh *et al.*[142] note the importance of basing utility functions on the metrics in which QoS is expressed rather than on the raw quantities of resources. There are other philosophical similarities to PACORA, but since the objective functions are discrete and non-convex their optimization is difficult. A survey of autonomic systems research appears in [66].

Rajkumar *et al.*[117] propose a system Q-RAM that maximizes the weighted sum of utility functions, each of which is a function of the resource allocation to the associated application. Unlike PACORA, there is no distinction between performance and utility, and the utility functions are assumed as input rather than being discovered by the system. The functions are sometimes concave, and in these cases the optimal allocation is easily found by a form of gradient ascent. When the utility functions are not concave, a suboptimal greedy algorithm is proposed.

Chase *et al.* [29] monitor the performance of applications as a function of their resources in cloud environments and use a greedy algorithm to allocate resources to maximize resource utility. Urgaonkar *et al.* [144] create a closed queuing network model and use a Mean Value Analysis (MVA) algorithm to allocate resources for multi-tiered applications. Watson *et al.* [151] also develop queuing-based performance models for enterprise applications, but use a virtualized environment to generate the models.

### Feedback-driven Controllers

Several systems use a feedback-driven reactive approach to resource allocation where a control loop or reinforcement learning adjusts allocations continuously.

Rightscale [120] for Amazon's EC2 [5] monitors the load of applications and automatically creates additional VM instances when the load crosses a certain threshold, using an additive-

increase controller to determine the number of instances to create. Zhu *et al.* [164] use three levels of controllers to meet SLOs in datacenters. Their node controller allocates resources on-chip. The pod controller migrates VM's between nodes, and the pod set controller adjusts the resource allocations for a pod.

AcOS [13] and Metronome [127] feature hardware-thread based maintenance of "heart rate" targets using adaptive reinforcement learning. AcOS also senses thermal conditions and can exploit Dynamic Voltage and Frequency Scaling (DVFS). Bodik *et al.* [23] builds online performance models like PACORA. Initially, their technique begins with an *exploration policy* that avoids nearly all SLO violations while building the model; later, it shifts to allocating with a controller based on the model built with exploration policy. The models are statistical, and bootstrapping is used to estimate performance variance. Major changes in the application model are detected and cause model exploration to resume. The models are not convex or concave in general, and all SLOs must be met with high probability.

Jockey [48] has some similarities to PACORA: it is intended to handle parallel computation, its utility functions are concave, and it adapts dynamically to application behavior. Its performance models are obtained by calibrating either event-based simulation or a version of Amdahl's Law to computations. Jockey does not optimize total utility but simply increases processors until utility flattens for each application, *i.e.,* each deadline is met. A fairly sophisticated control loop prevents oscillatory behavior.

Mars *et al.* [155] build on the offline performance models and co-scheduling algorithm in Bubble-Up [97] to create an adaptive system called Bubble-Flux. After Bubble-Up determines applications placement. Bubble-Flux uses a controller which continuously monitors the QoS of applications and slows down background computation as needed. Q-Clouds [104] creates models online using hardware performance counters to represent the interference in cache, memory, and prefetchers from co-scheduled applications. A controller then adjusts the resource allocations so that applications perform as if they were scheduled alone.

## Specifications and Offline Workload Models

Other systems base decisions on user-provided resource specifications and a real-time scheduling algorithm. In the Redline system [156], compact resource requirement specifications written by hand to guarantee response times. Isolation of resources is strong, as in PACORA. Scheduling is Earliest-Deadline-First. Admission control is lenient but oversubscription situations are remedied by de-admitting some of the non-interactive applications.

Gmach *et al.* [52] use traces of workloads to generate synthetic workloads and predict future resource needs. They use their system to aid capacity planning in datacenters. Soror *et al.* [129] use information about the expected workload of a database to create workload-specific VM configurations. Their framework requires the database management system to represent the workload as a set of SQL statements.

### Resource Management Frameworks

Some frameworks can support multiple scheduling and resource allocation policies. Guo *et al.*[55] present such a framework. They point out that much prior work is insufficient for true QoS; merely partitioning hardware is not enough because there must also be a way to specify performance targets and an admission control policy for jobs. Unlike PACORA, they argue that targets should be expressed in terms of capacity requirements rather than rates or times.

Nesbit *et al.*[106] introduces *Virtual Private Machines* (VPM), a framework for resource allocation and management in multicore systems. A VPM comprises a set of virtual hardware resources, both spatial (physical allocations) and temporal (scheduled time-slices). They break down the framework components into policies and mechanisms which may be implemented in hardware or software. VPM *modeling* maps high-level application objectives via *translation*, which uses models to assign acceptable VPMs to applications while adhering to system-level policies. A *scheduler* decides if the system can accommodate all applications. The VPM approach and terminology are similar to PACORA's at a high level, but no design or implementation of the modeling, translation, or scheduling components is presented [106].

There are several optimization frameworks for datacenters. Kingfisher [124] uses an integer linear program approach to minimize the resource cost for a cloud tenant based anticipated workload changes. They assume a perfect workload predictor. Their optimization uses the workload predictions and then considers the possibilities for scaling up or out and considers the time to transition to new configurations. In [42], Doyle *et al.* create models to predict the response time of web services by using their storage I/O rate, storage response time, memory usage, and CPU latency. They use a hill-climbing approach to assign resources to applications with the greatest marginal improve to a system-level metric. Like PACORA they can provide differentiated QoS. However, since their formulation is non-convex, they cannot guarantee that they are moving towards an optimal allocation.

In Whare-map [96], applications are profiled while they run in the datacenter. Whare-map uses than information to create an opportunity factor that indicts how well a particular application can run on a given node type. It then uses a stochastic hill climbing approach to determine a good mapping of jobs to nodes.

## 2.4 Hardware Resource Partitioning and QoS

PACORA relies on resource partitioning and Quality-of-Service mechanisms when available to enforce its resource allocation decisions. Resource partitioning and QoS research is active for on-chip, cluster, and networking resources [132, 133, 116, 157, 77, 64, 55, 56, 87, 105, 31, 28, 67, 43, 161, 99, 159]. However, the vast majority of research focuses on allocating a single resource type with a fixed policy, typically *fairness*. Some have researched network bandwidth fairness [22, 78, 93], while others [15, 14, 163] have concentrated on processor fairness.

Hardware partitioning research, which has largely focused on caches, provides mechanisms based on policies baked into the hardware, not the flexible allocations PACORA requires [132, 133, 116, 28, 31, 43, 67, 99, 159, 161]. Early work focused on providing adaptive, fair policies that ensure equal performance degradation [77, 157], not guarantees of responsiveness. Other work has focused on maximizing system performance or utilization [132, 133, 116] . Qureshi and Patt [116] create cache utility functions that represent an application's miss-rate as a function of it's cache allocation. They use a greedy allocation technique to partition the cache to minimize the total cache misses across all applications.

More recent proposals have incorporated more sophisticated policy management [56, 55, 64, 73]. Iyer *et al.* [72] suggests a priority-based cache QoS framework, CQoS, for shared cache way-partitioning. The priorities might be specified per core, per application, per memory type, or even per memory reference. However, simultaneous achievement of performance targets as in PACORA is not addressed. Bitirgen *et al.* [20] use artificial neural networks to predict an applications performance as a function of the cache and memory bandwidth allocations and the CPU power states. They then use a heuristic to search for an allocation that has a high weighted speedup.

## 2.5 Summary

Resource management and scheduling has been a common area of research for many communities over the years including high-performance computing systems, operating systems, real-time computing, computer architecture, and distributed systems. However, much of the past work has focused on maximizing system utilization or providing fairness to applications. PACORA instead focuses allocating on resources to guarantee differentiated QoS to applications to they can make their deadlines. Recently, resource allocation and scheduling systems, particularly for cloud and web services, have begun to work on performance prediction, modeling, and satisfying deadlines or SLOs like PACORA. Few of these approaches perform multidimensional resource allocation as in PACORA, but instead focus on co-scheduling applications. The most similar line of work typically relies on controllers to adjust resource allocations dynamically when deadlines are being missed. PACORA is the only framework that determines multidimensional resources allocations based convex models and then finds the globally optimal resource allocation using optimization.

# Chapter 3

# PACORA Framework

In this chapter, we describe the architecture of the PACORA framework. We present the mathematical formulation and prove its convexity. We also describe the two application specific-functions in detail and present experiments that helped guide the selection of the response-time function.

## 3.1 PACORA Architecture

PACORA is a framework designed to determine the proper amount of each resource type to give each application.

For our purposes, an application is an entity to which the system allocates resources: these can be a complete application (*e.g.,* a video player), a component of an application (*e.g.,* a music synthesizer), a background OS process (*e.g.,* indexing), a job in warehouse-scale computing, or a distributed application in a distributed embedded system.

Resources are anything that the system can "partition" using hardware or software. Resources can typically be thought of as one of three types: compute, communication, and capacity[1]. In our operating system experiments, we use cores (compute), network bandwidth (communication), and cache ways and memory pages (capacity). Other operating scenarios would have resources that perform similar functions at a different scale. For warehouse-scale computing, resources are more likely to be different types of nodes, network bandwidth, and storage. For distributed embedded systems, resources would include compute devices, link bandwidths, and memories.

### Resource Allocation as Optimization

PACORA formulates resource allocation as an optimization problem designed to determine the ideal resource allocation across all active applications by trying to minimize the total

[1]PACORA does not treat any resource types differently so classification is not strictly necessary. It is only described here to demonstrate the range of resources that could be controlled by PACORA.

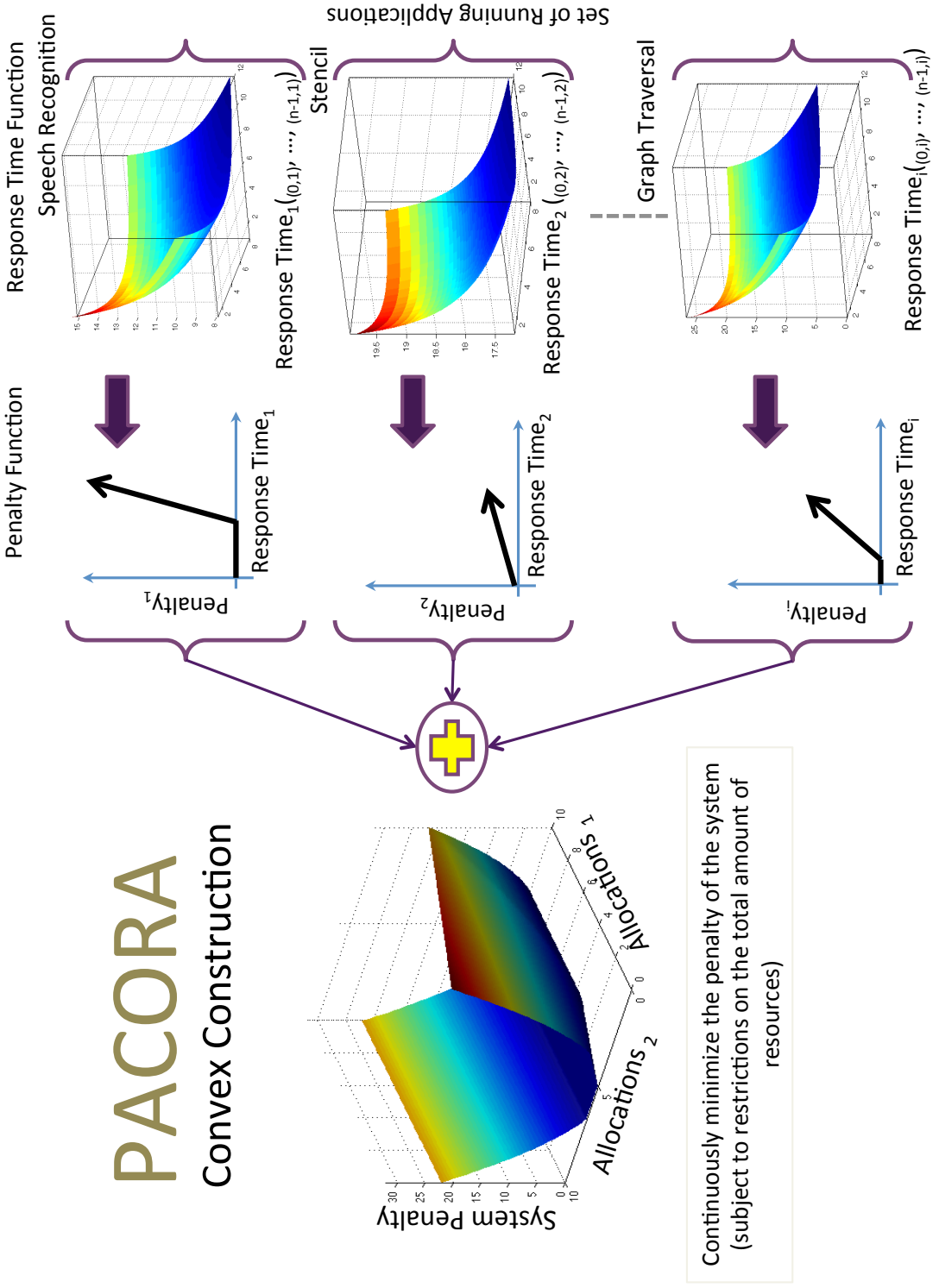


Figure 3.1: Visual representation of PACORA's optimization formulation. The runtime functions represented are the speech recognition, stencil kernel, and graph traversal applications from the evaluation Chapter 4.

penalty of the system. This approach is analogous to minimizing dissatisfaction with the user experience due to missed deadlines in a client system and minimizing the contract penalties paid for violated Service-Level Agreements (SLAs) in a cloud system. Figure 3.1 presents the formulation visually.

The optimization selects the allocations for all resources and resource types at once. This approach enables the system to make tradeoffs between resource types. For example, the system could choose to allocate more memory bandwidth in lieu of on-chip cache, or one large core instead of several small cores. Given that all of the resources allocated to an application contribute to the response time, independently allocating each resource type would make it difficult to provide predictable response times for applications without over-provisioning.

PACORA employs two types of application-specific functions in its optimization: a response-time function (RTF) and a penalty function. The response-time function represents the performance of the application with different resources and is built with runtime measurements. The penalty function represents the user-level goals for the application (*i.e.,* the deadline and how important it is to meet) and is set by the system, developer, or administrator.

A succinct mathematical characterization of this resource optimization scheme is the following:

$$\text{Minimize} \quad \sum_{p \in P} \pi_p(\tau_p(a_{p,1} \ldots a_{p,n})) \tag{3.1}$$

$$\text{Subject to} \quad \sum_{p \in P} a_{p,r} \leq A_r, r = 1, \ldots n \tag{3.2}$$

$$\text{and} \quad a_{p,r} \geq 0 \tag{3.3}$$

Here $\pi_p$ is the penalty function for application $p$, $\tau_p$ is its response time function, $a_{p,r}$ is the allocation of resource $r$ to application $p$, and $A_r$ is the total amount of resource $r$ available.

PACORA is designed to be convex by construction to take advantage of efficient convex optimization methods for solving the optimization problem [24].

## Assumptions

In Section 3.1, we have presented the mathematical framework behind PACORA. However, in order to deploy PACORA in a real system, we also need to make three assumptions about the design of the system. Here we describe these assumptions in detail.

### 1) Hierarchical Scheduling

PACORA is designed for systems where resource allocation is separate from scheduling. This split enables the use of application-specific scheduling policies, which have the potential to be easier to design and more efficient than general-purpose schedulers that have to do everything. The resource allocation system is then able to focus on the problem of *how much* of each resource type to assign to each application.

In client machines, PACORA makes coarse-grain resource-allocation decisions (*e.g.,* cores and memory pages) at the OS level, while the micro-management of these resources is left to user-level runtimes, such as Intel Threaded Building Blocks [36] or Lithe [111], and to user-level memory managers. However, a user-level runtime is not strictly necessary: in Linux, for example, we have used PACORA to set thread affinity or size resource containers.

If the machine is operating in a cloud computing environment, PACORA could be used in a hypervisor to allocate resources among guest OSes. For warehouse-scale computers, PACORA could be used to allocate resources (*e.g.,* nodes and storage) to jobs, while scheduling is left to other entities such as the MapReduce framework[39] or the node OS.

PACORA could be used in a system designed to consolidate realtime systems. Resources can be allocated to various realtime user-level schedulers such as Earliest-Deadline-First or Rate-Monotonic schedulers, and PACORA will guarantee quality-of-service to the schedulers, eliminating the need in the case of many applications for a realtime OS designed around a single real-time scheduler.

**2) Allocation Enforcement**

PACORA relies on resource allocation mechanisms to assign resources and enforce allocations. For PACORA to be able to use a resource, the system must be able to allocate the resource (*e.g.,* a core) or a fraction of it (*e.g.,* a percentage of network bandwidth) to an application and enforce this allocation. Enforcement can be in hardware or software. For example, cache partitioning could be implemented in hardware easily by changing the replacement algorithm to limit in which ways an application can write data (as is done in our Sandy Bridge prototype used in the experiments in Chapter 4) or the operating system could use page coloring emulate cache partitioning.

We have found that hardware mechanisms are readily available in most systems for some resources (*e.g.,* cores and memory pages) and others can easily be managed in with software (*e.g.,* network bandwidth). During the course of this work we have also observed QoS mechanisms being added to commercial systems (*e.g.,* cache partitioning) [38]. As more QoS mechanisms become available on future systems, other resources could be easily added to PACORA.

**3) Performance Isolation and Shared Resources**

PACORA assumes some amount of performance isolation between applications. In order for the RTFs to accurately reflect the expected response times of the applications, it is important that the response time does not change much as a function of the other applications currently running on the machine. However, the performance isolation need not be perfect: all of our evaluation was run on current x86 hardware with some shared resources, and PACORA was still effective. Chapter 7 discusses handling shared resources in more detail.

## 3.2 Convex Optimization

If the penalty functions, response-time functions, and resource constraints were arbitrary, little could be done to optimize the total penalty beyond searching at random for the best allocation. However, we designed PACORA's optimization, RTFs, and penalty functions to be convex by construction, which enables us to use convex optimization [24] methods when optimizing. By framing our resource allocation problem as a convex optimization problem, we get two significant benefits: for each problem an optimal solution exists without multiple local extrema, and fast optimization methods with practical incremental solutions become feasible. In this section, we prove the convexity of PACORA's optimization formulation. The RTF and penalty function convexity are discussed in Sections 3.3 and 3.4 respectively. PACORA also formulates RTF *creation* as a convex optimization problem, as explained in Section 5.4.

### Resource Allocation Optimization Convexity

A constrained optimization problem is *convex* if both the objective function to be minimized and the constraint functions that define its feasible solutions are convex functions. A function $f$ is convex if its domain is a convex set and $f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y)$ for all $x$ and $y$ in its domain and for $\theta$ between 0 and 1. A set is convex if for any two points $x$ and $y$ in the set, the point $\theta x + (1-\theta)y$ is also in the set for all $\theta$ between 0 and 1. If $f$ is differentiable, it is convex if its domain is an open convex set and $f(y) \geq f(x) + \nabla f^T \cdot (y - x)$ where $\nabla f$ is the gradient of $f$. Put another way, $f$ is convex if its first-order Taylor approximations are always global underestimates of its true value.

A convex optimization problem is one that can be expressed in this form:

$$\begin{aligned} \text{Minimize} \quad & f_0(x_1, \ldots x_m) \\ \text{Subject to} \quad & f_i(x_1, \ldots f_m) \leq 0, i = 1, \ldots k \\ \text{where} \quad & \forall i \quad f_i : \Re^m \rightarrow \Re \text{ is convex.} \end{aligned}$$

PACORA's resource allocation problem can be transformed into a convex optimization problem in the $m = |P| \cdot n$ variables $a_{p,r}$ as long as the penalty functions $\pi_p$ are convex non-decreasing and the response-time functions $\tau_p$ are convex. We designed our functions to meet these constraints, and proofs of their convexity are shown below.

The resource constraints are affine and therefore convex; they can be rewritten as

$$\sum_{p \in P} (a_{p,r} - A_r) \leq 0 - a_{p,r} \leq 0 \tag{3.4}$$

$$-a_{p,r} \leq 0 \tag{3.5}$$

The convex formulation makes the optimization scale linearly in the number of resource types and the number of applications. For client operating systems with around 100 applications running and 10 resource dimensions, the total number of variables in the optimization

problem is 1000—a very small convex optimization problem that can be solved in microseconds on current systems. Cloud systems could have many more than 100 applications running, but the problem size scales linearly, and the potential benefits of a good allocation should scale rapidly with the size of the system.

## 3.3 Response-Time Function Design

In this section, we discuss the design considerations and requirements for PACORA's RTF, evaluate potential RTFs with different complexities, and describe the chosen design in more detail. Chapter 4 evaluates the performance of the chosen RTF. Chapter 7 discusses alternative and enhanced RTF models.

### Purpose

In order for a resource allocation framework to make informed decisions about application performance, there must be a way for it to understand the performance impact of an application's resource allocation in the system. One can imagine several high-level approaches to provide this information to the framework. One option would be for the the system to try a variety of allocations and select the best one. However, there are a few disadvantages to this method: first, the system may need to try many points to find an efficient resource allocation for multidimensional allocation problems; second, the result for a single application may not compose well with multiple applications; and third, it doesn't give the system much understanding of the value of individual resources making resource tradeoffs difficult.

Another option would be something similar to hill climbing, where the system incrementally adds or removes resources and measures the change in performance. However, there are several challenges for an incremental approach as well. Since the system relies on measuring the incremental gradients, it could get stuck in local minima or remain on a performance plateau for a particular resource without discovering the threshold that gives significant performance improvement (*e.g.,* the point where the application fits in cache). It could also be difficult to explore more than one resource dimension at a time. Additionally it could take quite a long time to reach an efficient resource allocation, particularly for a system with multiple applications running, and could violate the application's quality-of-service while exploring resource allocations.

While obviously these techniques can be improved upon, we felt the fundamental problems of composablity and potentially high overhead to find an efficient multidimensional allocation would be very difficult to overcome. For PACORA, we instead chose to take a modeling approach to represent an application's performance given its resource assignments. We explicitly create RTFs from measured values that capture information about the performance impact of a particular resource to an application on the current hardware at a particular time. We chose to use models because they can be easily used in an optimization that considers multiple resources and applications at the same time.

## Design Considerations

When considering what was necessary for a performance model to be used in a real system, we came up with the following requirements to guarantee that the model would be low cost to produce and use and work with real applications:

- Low cost to produce and maintain;
- Low storage overhead;
- Works with any number of resource dimensions;
- Tolerant of noisy measurements;
- Convex by construction; and
- Easily computed gradients.

One approach to creating explicit resource-performance models would have been to model response times by recording past values and interpolating among them. Tthis idea has serious shortcomings for resource allocation problems, however:

- The multidimensional response time tables would be large and thus more expensive to measure and store;
- Interpolation in many dimensions is computationally expensive thereby increasing the overhead of the resource allocation optimization;
- The measurements will be noisy and require smoothing;
- Convexity in the resources may be violated and as a result significantly increasing the cost of the resource allocation optimization by eliminating the opportunity to use efficient convex optimization techniques; and
- Gradient estimation will be slow and difficult.

Instead of interpolating, PACORA maintains a parameterized analytic response time model with the partial derivatives evaluated from the model *a priori*. Application responsiveness is highly nonlinear for an increasing variety of applications like streaming media or gaming, thus requiring many data points to represent the response times without a model. Using models, each application can be described in a small number of parameters. Models can be built from just a few data points and can naturally smooth out noisy data. The gradients, needed by PACORA to solve the optimization problem efficiently, are easy to calculate.

However, to realize the potential advantages of modeling, we first needed to demonstrate that simple models could adequately represent the response time of an application for resource allocation purposes. To determine if simple models would work, and to select an

appropriate model for PACORA's RTF functions, we used three steps. First, we performed a simple study using microbenchmarks in a real system to determine the complexity required for the model (Section 3.3). Once we had determined the general form of the model from the experiments, we then designed a model that seemed logical using our domain knowledge of computer hardware, applications, and performance (Section 3.3). Finally, we performed experiments using real benchmarks and kernels on a real system to validate that our model fit the measured values. Chapter 4 presents these final experiments and results.

## Model Format Evaluation

To test the potential of different model formats, we first performed a study comparing the accuracy of RTFs created from linear models, quadratic models, and genetically-programed response surfaces for eight synthetic benchmarks and five phases of a real speech recognition kernel. The RTFs studied use three resources dimensions: cores, off-chip memory bandwidth, and cache banks. In this section, we describe these experiments and their results.

### Applications

We created a set of synthetic microbenchmarks specifically designed to evaluate our modeling techniques by representing the space of possible resource behaviors in three dimensions.

| Name | Processor | Cache | Offchip BW | Description |
|---|---|---|---|---|
| p–c–b– | oblivious | oblivious | oblivious | Pointer chases through a long list (single threaded) |
| p–c–b+ | oblivious | oblivious | benefits | Streaming copy with no reuse (single threaded) |
| p–c+b– | oblivious | benefits | oblivious | Copies data repeatedly from a large block (single threaded) |
| p–c+b+ | oblivious | benefits | benefits | Copies data from large blocks, with reuse (single threaded) |
| p+c–b– | benefits | oblivious | oblivious | Pointer chases through long lists (multithreaded) |
| p+c–b+ | benefits | oblivious | benefits | Streaming copies with no reuse (multithreaded) |
| p+c+b– | benefits | benefits | oblivious | Copies data repeatedly from large blocks (multithreaded) |
| p+c+b+ | benefits | benefits | benefits | Copies data from large blocks, with reuse (multithreaded) |

Table 3.1: Synthetic microbenchmark descriptions. Each benchmark captures a different combination of responses to resource allocations. "Benefits" means that application performance improves as more of that resource is allocated to it (though sometimes only up to a point). "Oblivious" means that the application performance barely improves or does not improve at all as more of that resource is allocated to it.

| Phase | Name | Description | Behavior |
|---|---|---|---|
| 1 | Cluster | Compute probability, step 1 | Accumulate, up to 6 MB data read, 800KB written |
| 2 | Gaussian | Compute probability, step 2 | Calculate, up to 800KB read, 40KB written |
| 3 | Update | Non-epsilon arc transitions | 40KB read, small blocks, dependent on graph connectivity |
| 4 | Pruning | Pruning states | Small blocks, dependent on graph connectivity |
| 5 | Epsilon | Epsilon arc transitions | Small blocks, dependent on graph connectivity |

Table 3.2: Description of phase behavior in large-vocabulary continuous-speech-recognition (LVSCR) application.

Table 3.1 describes these benchmarks. In general, each benchmark represents a generic category of behavior that we might expect to see in phases of real applications. We classified the benchmarks based on whether they benefit from additional processor, cache or bandwidth resources, or whether they derive no benefit from running on a large allocation of a given resource. We also limited the size of the benchmarks along these resource dimensions so that they encounter performance plateaus on our simulated machine. For example, a benchmark's performance might benefit from additional cores up to four cores but not from more than four cores. Parallel benchmarks were parallelized with `pthreads` [90]. The benchmarks each run an average of 1.9 billion cycles per execution.

We also evaluated a real multithreaded application with multiple phases of behavior, specifically a Hidden-Markov-Model (HMM)-based inference algorithm that is part of a large-vocabulary continuous-speech-recognition (LVCSR) application [32, 65]. LVCSR applications analyze a set of audio waveforms and attempt to distinguish and interpret the human utterances contained within them. The recognition network we used here models a vocabulary of over 60,000 words and consists of millions of states and arcs. The inference process is divided into a series of five phases, and the algorithm iterates through the sequence of phases repeatedly with one iteration for each input frame. This application case study demonstrates the varying ability of our models to capture real application behavior. Table 3.2 lists the characteristics of each phase of the application. Each phase runs for an average of 24 billion cycles per execution.

### Resources

To properly test the potential RTF functions, we implement hardware partitioning mechanisms for three of most important shared on–chip resources: cores (compute), interconnect bandwidth to DRAM (communication), and L2 cache capacity (capacity). We specifically chose one of each of the resource types described in Section 3.1.

**Core Pinning** To partition cores, our implementation uses the thread affinity feature built into the Linux 2.6 kernel. We restrict the threads belonging to an application to run on the cores assigned to that application. We assume a homogeneous collection of cores, and that only last level caches are shared among applications, which means that there is nothing to differentiate a core from any other when they are allocated.

**Globally Synchronized Frames** To partition off-chip bandwidth, we use the Globally Synchronized Frames (GSF) approach presented in Lee et al. [87]. We chose this approach because it does not require complex hardware modifications, provides strict QoS guarantees for minimum bandwidth and the maximum delay of the network, and provides proportional sharing of excess bandwidth. GSF controls the number of packets that a core can inject into the network per frame, and each core is guaranteed to get the number of packets allocated to it each frame. GSF enables cores to inject packets into future frames if their current allocation is already exceeded. This feature allows excess bandwidth to be shared among

cores proportional to their packet allocation. To simplify prediction by making performance more deterministic, our current implementation does not make any future frames available during training, meaning that applications get exactly their allocation each frame.

**Cache Partitioning** We implemented bank-based cache partitioning as opposed to way-based partitioning to preserve cache associativity in our machine. We assumed banks are sized at 1 MB, and we assume no additional overhead to lookup the correct bank. Our experiments do not reallocate banks during execution, so we also do not add a reallocation overhead.

### Modeling Techniques

We use multivariate regression techniques to create explicit statistical models for predicting the performance of an application given a resource allocation of a particular size. We create one regression model per application phase.

Linear least-squares regression techniques produce simple models that can be expressed concisely and are therefore more portable. Linear regression techniques can outperform nonlinear ones when training sets are small, the data has a low signal to noise ratio, or sparse sampling is used[57]. These criteria apply in our case. These models are attractive due to their simplicity, but their restricted expressiveness may reduce their accuracy of the underlying system.

Linear models may be realized in varying forms (i.e. it is the combination of terms that is linear, rather than the degree of each term). The simplest models are linear additive models, which take the form:

$$y(x) = a_0 + \sum_{i=1}^{N} a_i x_i \tag{3.6}$$

Multivariate linear additive models contain one term for each variable (i.e. an allocation, $x_i$) and an intercept term ($a_0$). The regression tunes the coefficient associated with each term ($a_i$) to fit the sample data as accurately as possible. Note that the linear additive model has no way to represent any possible interaction between the variables, implying that all variables are independent—which is expressly not true in our scheduling scenario. For example, a smaller cache size will result in increased cache misses and an increased demand for memory bandwidth, meaning that the effect of a change in bandwidth allocation is not independent from a change in cache size in terms of its effect on performance. However, their interactions may be small enough in practice to ignore in the model.

More complex multivariate linear regression models often include terms for variable interaction and polynomial terms of degree two or more. Such models are commonly termed *response surface models,* and have the general form:

$$y(x) = a_0 + \sum_{i=1}^{N} a_i x_i + \sum_{i=1}^{N} \sum_{j=i}^{N} a_{ij} x_i x_j + ... \tag{3.7}$$

These polynomial models capture more complex dependencies between the input variables. However, we as modelers must still explicitly express the nature of the relationship between input and output in the form we give the polynomial equation. We choose to test a quadratic model, in addition to the basic linear model, in order to explore the potential benefit of including interaction terms.

Selecting the best possible equation form for the data automatically requires the use of nonlinear regression techniques such as local regression, cubic splines, neural networks, or genetic programming [3, 23, 150]. The disadvantage of these techniques is that the models are difficult to use for many optimization methods and can be quite expensive to build. However, we include a genetic programming approach to explore how accurately we can model applications despite that fact that they are likely to be too costly and slow to use in a real system.

Genetic programming is a technique, based on evolutionary biology, used to optimize a population of computer programs according to their ability to perform a computational task. In our case, the 'program' is an analytic equation whose evaluation embodies a response surface model, and the 'task' is to match sample data points obtained from full–scale simulations [3]. The output, termed a *genetically-programmed response surface* (GPRS), is a set of nonlinear models that create explicit equations describing the relationship between design variables and performance, and we incorporate them into our framework as an example of a nonlinear modeling alternative. A GPRS is generated automatically, meaning that the modeler does not have to specify the form of the response surface equation in advance. Instead, genetic programming [83] is used to create an equation and tune the coefficients. For more information on GPRS creation, see [3] or [37].

We also explored a statistical machine learning technique, Kernel Canonical Correlation Analysis (KCCA) [10], to predict the response time in the style of [50]. However, the results are not presented here as we found that we generally had too few sample points for the models to function.

### Experimental Testbed

We use Virtutech Simics [149] to simulate a multicore system with a two-level on-chip memory hierarchy to collect the data used to create our performance models and to test the effectiveness of our resource scheduling framework. Simics is a full system simulator capable of running a commodity OS and completing simulations consisting of billions of cycles. We modify the Simics cache and memory timing modules to reflect the capabilities of our hardware partitioning mechanisms. Our target machine has 10 cores, private 64KB L1D and 32KB L1 I caches for each core and a shared L2 (16 MB, 16-way set associative). All caches have 128 B lines. All banks in the L2 cache have uniform access time of 7 cycles. Our target machine runs Fedora Core 5 Linux (kernel 2.6.15). We constrain the simulated system to a maximum allocation of 10 cores, 16 MB of L2 cache, and 4 cache lines/cycle of off-chip bandwidth, and a minimum allocation of 1 core, 16 KB of L2 cache, and 5 cache lines/thousand cycles of off-chip bandwidth.

| Name | Linear | Quadratic | GPRS |
|---|---|---|---|
| p–c–b– | 0.06% (0.04) | 0.04% (0.04) | **0.02%** (0.02) |
| p–c–b+ | 234.07% (287.44) | 139.21% (167.10) | **0.23%** (0.40) |
| p–c+b– | 12.67% (5.30) | 8.26% (5.30) | **0.02%** (0.02) |
| p–c+b+ | 12.04% (5.09) | 7.83% (4.69) | **0.06%** (0.06) |
| p+c–b– | 0.07% (0.05) | **0.05%** (0.06) | **0.05%** (0.03) |
| p+c–b+ | 271.05% (377.53) | 164.23% (226.23) | **0.51%** (1.08) |
| p+c+b– | 13.08% (6.91) | 8.79% (7.32) | **0.06%** (0.07) |
| p+c+b+ | 12.07% (4.86) | 8.05% (5.49) | **0.08%** (0.06) |

Table 3.3: Means (standard deviations) of percentage error in runtime cycles for each of the predictive models for each of the synthetic microbenchmarks. Lowest are in bold.

### Evaluation of Model Accuracy

To test the accuracy of the different model types, we chose a sample of 55 points from the space of 19200 possible allocations (or 0.3%) using an Audze-Eglais Uniform Latin Hypercube design of experiments [16], and trained the models in MATLAB [98] using this sample set. Audze-Eglais selects sample points that are as evenly distributed as possible through the space of possible allocations. We used the benchmark runtime in cycles to represent the response time for the benchmarks. We then evaluated the accuracy of the model relative to measured performance on a training set which contains all points disjoint from the sample set.

Table 3.3 reports the mean and standard deviation of percentage error for the synthetic benchmarks of each of the models in predicting runtime cycles of each allocation versus the actual measured performance of that allocation. We can see that two of the benchmarks, p+c–b– and p–c–b–, seem to be very easy to predict and all three model types have less than 0.1% mean error. Four of the remaining benchmarks, p–c+b–, p–c+b+, p+c+b–, and p+c+b+, are more difficult to predict for the linear and the quadratic model. The linear model has an error around 12% with a significant standard deviation, 5%, for each benchmark. The quadratic model performs a bit better and has an average error around 8% for each of these benchmarks but still has a very large standard deviation. The GPRS model, however, performs extremely well on these benchmarks and again has an average error of less than 0.1%. The final two benchmarks, p–c–b+ and p+c–b+, which are streaming benchmarks, prove quite challenging for the linear and quadratic model resulting in an average error of 250% and 150% respectively. The GPRS model, however, has very little trouble with these benchmarks and produces an error around 0.5%.

Figure 3.2 visually represents what is happening in each of the these cases. The figure plots the predictions from each model versus measured data for all the benchmarks. The x-axis is resource configurations ordered by runtime and the y-axis is the runtime. Looking at the plots we can see that all models perform well on the benchmarks that have a very predictable structure and no performance plateau. However, benchmarks which encounter a

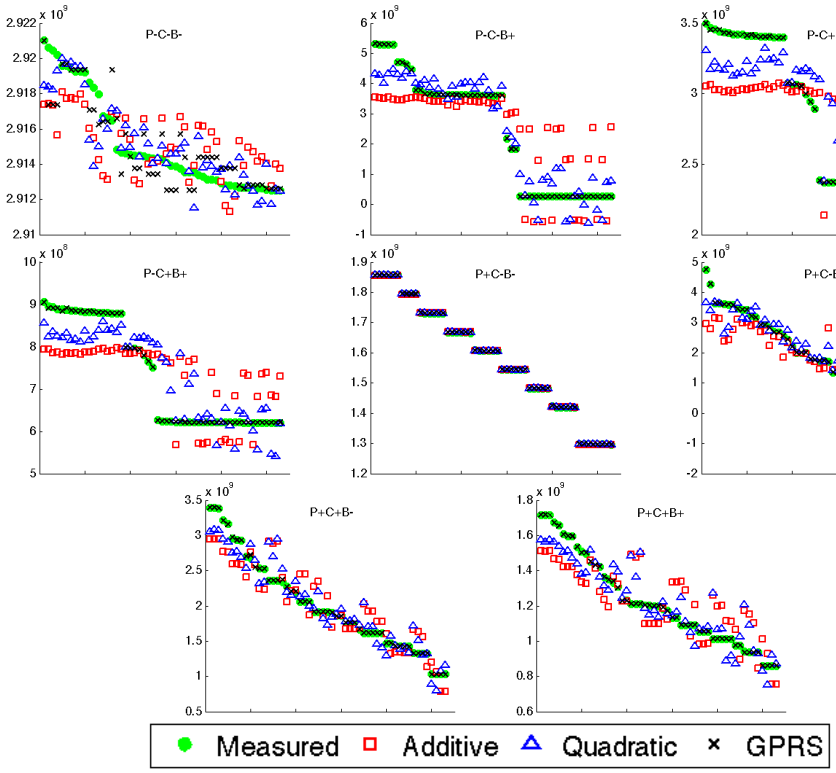


Figure 3.2: Comparison of model accuracy for the eight microbenchmarks when predicting runtime in cycles. Each point represents a prediction for a machine configuration, and points are ordered along the x-axis based on decreasing measured run time. Y-axis plots predicted or measured runtime in cycles; note the differing ranges. In most cases, the nonlinear GPRS–based model is so accurate that it precisely captures all sample points.

cliff and then saturate (such as the working set fitting in cache) prove difficult for the linear and quadratic models and the larger the cliff, the larger the error.

Clearly, from the synthetic benchmark results, the GPRS models are extremely accurate, and the other models have the potential to have problems. However, the GPRS–based models used here took up to six hours each to build and are also be very expensive to use in an optimization. The linear and quadratic models can be trained extremely rapidly and are very efficient to use in the optimization problem, but this efficiency clearly comes with a cost in terms of accuracy. However, there a few things to consider when evaluating these results. First, these are synthetic benchmarks and may not be representative of real applications. Second, the true metric of whether or not a model is accurate "enough" depends on the quality of the resource allocations decisions produced from using it.

Looking at the results for our real speech application (Table 3.4), we can see that the phases of the LVSRC application were actually easier to predict for the linear and quadratic model than most of the synthetic benchmarks, and the GPRS actually performs worse than

| Name | Linear | Quadratic | GPRS |
|---|---|---|---|
| cluster | **4.27%** (3.66) | 6.90% (5.67) | 15.52% (10.36) |
| gaussian | **1.83%** (0.72) | 4.16% (2.49) | 2.33% (3.19) |
| update | **4.98%** (3.04) | 7.94% (7.12) | 5.89% (5.09) |
| pruning | **2.27%** (1.08) | 10.70% (10.29) | 3.07% (2.91) |
| epsilon | 3.88% (4.01) | 4.66% (4.27) | **2.69%** (1.09) |

Table 3.4: Means (standard deviations) of percentage error in runtime cycles for each of the predictive models for each of the phases of the LVSRC application. Lowest are in bold.

on the synthetic benchmarks. These results make the linear and quadratic models seem like potentially reasonable choices. Cook et al. [38] found in their study of 44 real benchmarks that they did not actually exhibit performance cliffs like the ones the models struggled on in our synthetic benchmarks. These results further improve our confidence in the potential of the linear and quadratic models for modeling real applications.

However, our simulator is too slow to test the quality of resource allocations the models produce, so we perform an additional study using an FPGA system to test the allocation quality. These results are presented in Chapter 4. For this discussion, it is worth mentioning that we found that the linear and quadratic models outperform the GPRS models because they work well with the optimizer and that the quadratic model often produced near optimal allocations. As a result, we chose to move forward with a quadratic model. The following section describes how we designed PACORA's RTF using a quadratic model. We then perform accuracy tests with a wide range of benchmarks on the new model. Chapter 4 presents these results.

## Response-Time Functions

In this section, we discuss the final design for PACORA's RTFs.

While the initial studies only looked at total runtime as the metric of performance for an application, we decided that the RTF actually should represent the expected *response time* of an application as a function of the resources allocated to the application. Response time is an application-specific measure of the performance representing the time to run the critical function of the application. For example, the response time of an application might be:

- The time from a mouse click to its result;
- The time to produce a frame;
- The time from a service request to its response;
- The time from job launch to job completion; or
- The time to execute a specified amount of work.

As explained in Section 3.2, PACORA needs to model response times with functions that are convex by construction in order to take advantage of the efficient solution methods

available in convex optimization. All applications have a function of the same form, but the application-specific weights are set using the performance history of the application. Equation 3.8 below shows the RTF we selected for PACORA, and Figure 3.3 shows two example RTFs we have created from applications we studied.

$$\tau(w, a) = \tau_0 + \sum_{i \in n, j \in n} \frac{w_{i,j}}{\sqrt{a_i * a_j}} \tag{3.8}$$

Here $\tau$ is the response time, $i$ and $j$ are resource types, $n$ is the total number of resource types, $a_i$ and $a_j$ are the allocations of resource types $i$ and $j$, and $w_{i,j}$ is the application-specific weight for the term representing resources $i$ and $j$.

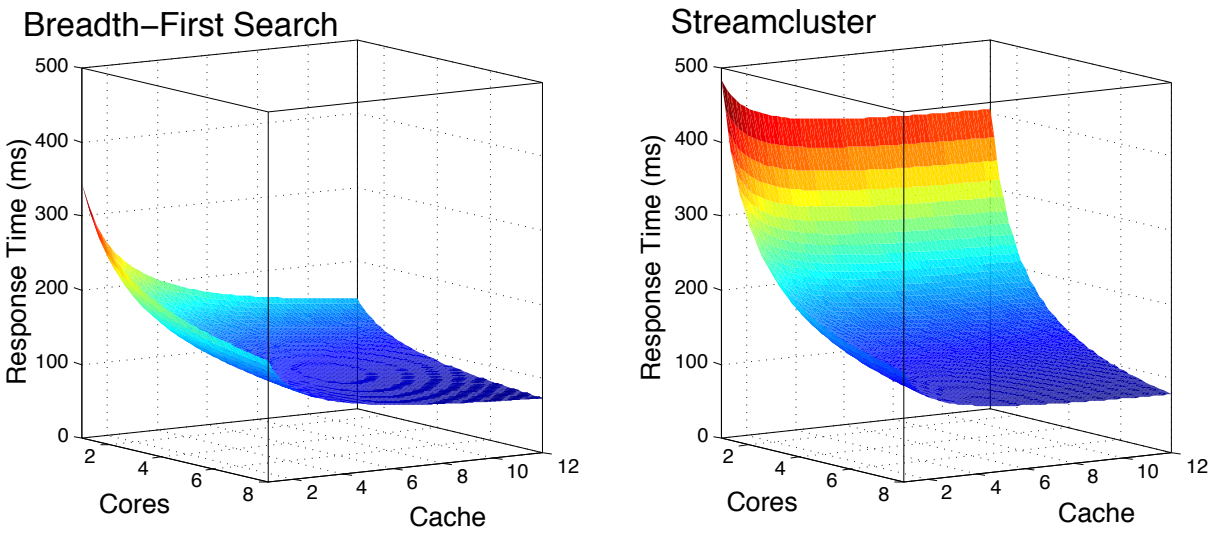


Figure 3.3: Response-Time Functions for a breadth-first search algorithm and `streamcluster` from the PARSEC benchmark suite [17]. We show two resource dimensions: cores and cache ways. Chapter 4 presents the experiments where these models were generated.

In this equation, the response time is modeled as a weighted sum of component terms, roughly one per resource, where a term $w_i/a_i$ is the amount of work $w_i \geq 0$ divided by $a_i$, the allocation of the $i$th resource [128]. We felt that this naturally represented approximately how resources behave. For example, one term might model instructions executed divided by total processor MIPS, so the application-specific $w$ PACORA is learning is the number of instructions. As we increase the allocation of MIPS, then we'll see the contribution to total runtime from this term decrease since the additional processing power reduces the time to execute the instructions. Other terms follow the same pattern but for different resources such as model network accesses divided by bandwidth, and so forth.

The examples described above all contain only a single resource type. However, our intuition was that there may be relationships between resource types and asynchrony and latency tolerance may make response time components overlap partly or fully. For example, one can easily imagine that the amount of cache an application has affects its required memory bandwidth. Thus, we added additional terms to represent the interactions between resources. To our surprise, in most of our experiments, we have found that the interaction terms are nearly always negligible and can be eliminated to save space and computation. This omission allows the dimensionality of the function, and thus the storage space required, to increase roughly linearly with the number of resource types. However, it is possible that some systems may still require them.

It is obviously important to guarantee the positivity of the resource allocations. This guarantee can be enforced as the allocations are selected during penalty optimization, or the response time model can be made to return $\infty$ if any allocation is less than or equal to zero. This latter idea preserves the convexity of the model and extends its domain to all of $\Re^n$ and consequently we used this approach in our implementation.

Our chosen model design satisfies the design requirements listed above. The model is low cost to produce: we can use convex optimization to produce it as described in Chapter 5. The models (without the interaction terms) scale linearly with the number of resource dimensions and only require a small number of history values to produce a good model so they are compact to store. They can capture information about all of the resource types and are tolerant of noise (See Chapters 4 and 7 for variability results and discussion). Such models are automatically convex in the allocations because $1/a$ is convex for positive $a$ and because a positively-weighted sum of convex functions is convex. Lastly, the gradient $\nabla\tau$, which is needed by the penalty optimization algorithm, is simple to compute since $\tau$ is analytic, generic, and symbolically differentiable. However, we leave it to Chapter 4 to demonstrate the efficacy of our model in a real system.

## RTF Convexity

We now show that response time functions $\tau$ are convex in the resources $a_i$ given any of the possibilities we have considered.

A function is defined to be *log-convex* if its logarithm is convex. A log-convex function is itself convex because exponentiation preserves convexity, and the product of log-convex functions is convex because the log of the product is the sum of the logs, each of which is convex by hypothesis. Now $1/a$ is log-convex for $a > 0$ because $-\log a$ is convex on that domain. In a similar way, $\log(1/\sqrt{a_i \cdot a_j}) = -(\log a_i + \log a_j)/2$ and $\log a^{-1/d} = -(\log a)/d$ are convex, implying $\log(1/\sqrt{a_i \cdot a_j})$ and $\log a^{-1/d}$ are also. Finally, $\log(1/\log a)$ is convex because its second derivative is positive for $a > 1$:

$$\frac{d^2}{da^2}\log(1/\log a) \;=\; \frac{d^2}{da^2}(-\log\log a)$$

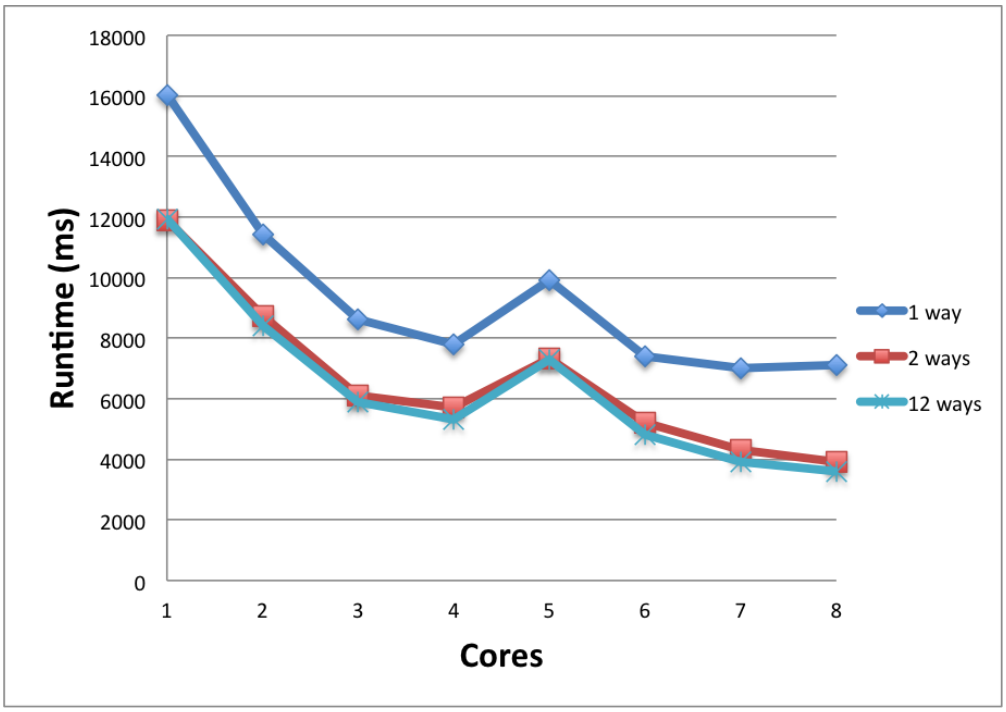


Figure 3.4: Measured runtimes for the `dedup` benchmark in PARSEC varying cores from 1-8 and allocating 1, 2, and 12 cache ways. Ways 3-11 are not shown, but look nearly identical to 2 and 12. Chapter 4 presents the experiments where this data was generated.

$$= \frac{d}{da}\left(\frac{-1}{a \log a}\right)$$
$$= \frac{1 + \log a}{(a \log a)^2}.$$

### Non-Convexity

Forcing RTFs to be convex assumes that the actual response times are close to convex. We believe this to be a plausible requirement as applications usually follow the "Law of Diminishing Returns" for resource allocations, and in our implementation and evaluation, we found our convexity assumption to be reasonably true. In cases where the assumption was not completely valid, PACORA was still able to produce near optimal allocations (See Chapter 4). The reason that non-convex response time versus resource behavior did not result in bad resource allocations was that for the most part the non-convex behavior we measured was usually particular resource allocations producing much worse results than their surrounding allocations and these points were ignored as outliers in the model and rarely selected by the optimization. For example, we have seen non-convex performance in applications when dealing with hyperthreads or memory pages. For two of our applications, five hyperthreads

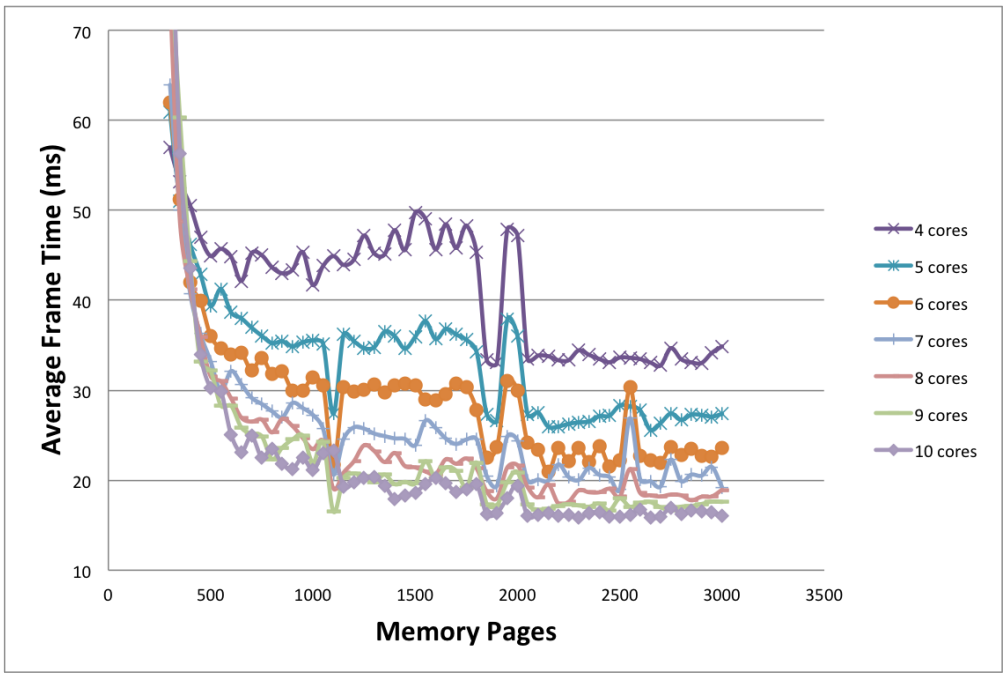


Figure 3.5: Average frame time for an n-bodies application running on Windows 7 while varying the memory pages and cores.

resulted in significantly worse performance than either four or six. Figure 3.4 show this behavior with PARSEC's `dedup` benchmark. When studying some other applications, we found that particular numbers of memory pages, (*e.g.,* 2K), resulted in much better performance than the adjacent page allocations as shown in Figure 3.5. Chapter 7 discusses these outliers and additional challenges to response time modeling along with additional techniques that could be employed to handle them.

Another potential kind of convexity violation might not be so easily ignored is where "plateaus" can sometimes occur as in Figure 3.6. Such plateaus can be caused by adaptations within the application such as adjusting the algorithm or output quality (For example, a video player may choose to increase resolution having received an increase in network bandwidth and thus the system may not measure an improvement in frame rate) or certain resources that only provide performance improvements in increments rather than smoothly. In these applications, the response time is really the *minimum* of several convex functions depending on allocation, and the point-wise minimum that the application implements fails to preserve convexity. The effect of the plateaus will be a non-convex penalty as Figure 3.7 shows and multiple extrema in the optimization problem will be a likely result.

There are several ways to avoid this problem. One is based on the observation that such response time functions will at least be *quasiconvex*. Another idea is to use additional constraints to explore convex sub-domains of $\tau$. (These approaches are described in more detail in Chapter 7.) Either approach adds significant computational cost, and we found

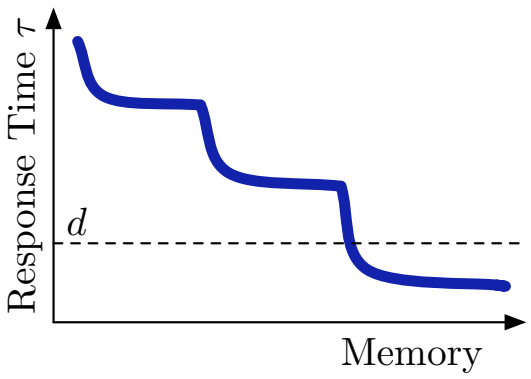


Figure 3.6: Response time function with some resource "plateaus".

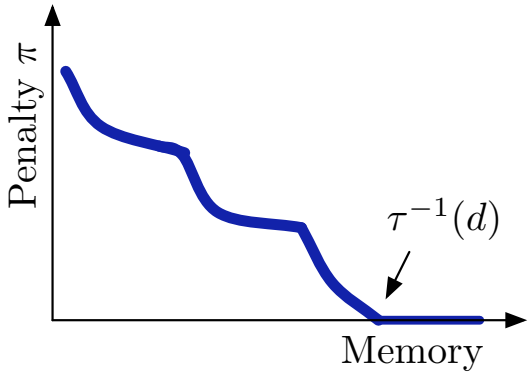


Figure 3.7: Net effect of the resource plateaus on the application penalty.

that our simple convex models still resulted in high-quality resource allocations. Thus we chose not to implement either.

## 3.4 Penalty Functions

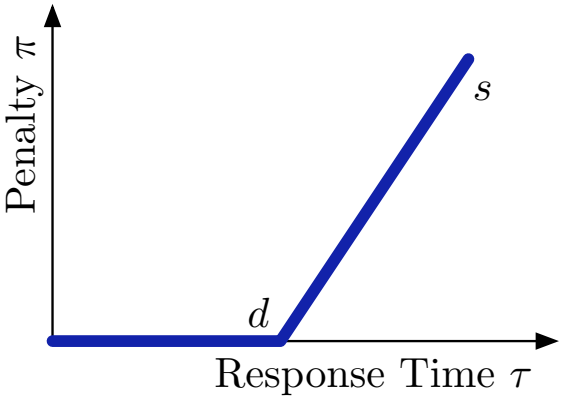


Figure 3.8: A penalty function with a response time constraint.

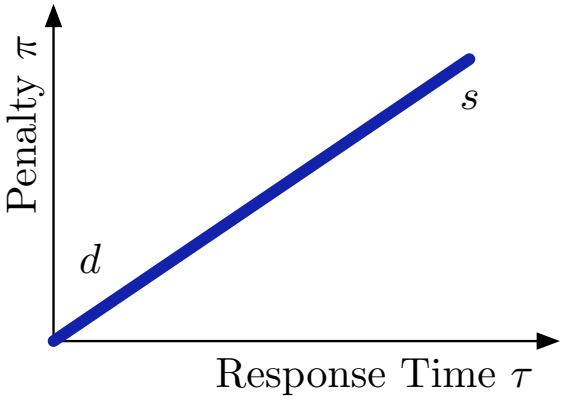


Figure 3.9: A penalty function with no response time constraint.

In addition to understanding how an application's performance responds to resources (represented with the application RTF), in resource allocation it is also necessary to know the relative importance of the applications: one application may use a resource type more efficiently, but another, less efficient, application may be more important to the user. To embody user-level preferences about the application, we added a second application-specific function called the *penalty function* in PACORA.

Although similar to priorities, penalty functions are *functions* of the response time rather than simple *scalar values*, so they can explicitly represent deadlines. Knowing deadlines lets PACORA make optimizations that are difficult in today's systems, such as running just fast enough to make the deadline. Like priorities, the penalty functions are typically set by the system on behalf of the user. However, one could imagine in future systems potentially learning them through user interactions.

PACORA's penalty functions $\pi$ are non-decreasing piecewise-linear functions of the response time $\tau$ of the form $\pi(\tau) = \max(0, (\tau - d)s)$ where $d$ represents the deadline of the application and $s$ (slope) defines the rate the penalty increases as response time exceeds $d$. For applications without response-time constraints the deadline can be set to 0. Two representative graphs of this type appear in Figures 3.8 and 3.9.

An application penalty function can be represented using only $d$ and $s$, which makes them extremely lightweight to store, and the storage size per application is constant regardless of the number of resource types.

### Penalty Function Convexity

In this section, we discuss the convexity of PACORA's penalty functions. A few facts about convex functions will be useful in what follows. First, a *concave* function is one whose negative is convex. Maximization of a concave function is equivalent to minimization of its convex negative. An affine function, one whose graph is a straight line in two dimensions or a hyperplane in n dimensions, is both convex and concave. A non-negative weighted sum or point-wise maximum (minimum) of convex (concave) functions is convex (concave), as is either kind of function composed with an affine function. The composition of a convex non-decreasing (concave non-increasing) scalar function with a convex function remains convex (concave).

Each penalty function $\pi$ is the point-wise maximum of two affine functions and is therefore convex. Moreover, since each penalty function is scalar and nondecreasing, its composition with a convex response time function will also be convex.

## 3.5 Managing Power and Energy

The optimization in Equations 3.1 to 3.3 does not include a cost for allocating resources, and thus all the resulting allocations would divide *all* the resources among the applications. While that may have been reasonable in former computing paradigms (*e.g.,* desktop computers), in current systems it is essential to operate efficiently in order to extend battery life or reduce power consumption. As a result, for PACORA to be practical in today's system, it is also necessary to consider the power required to run the resource in the allocation decision.

To represent the cost of operating a resource, we create an artificial application called *application 0*. Application 0 is designated the idle application and receives allocations of all resources that are left idle, *i.e.,* not allocated to other applications. If the system has the

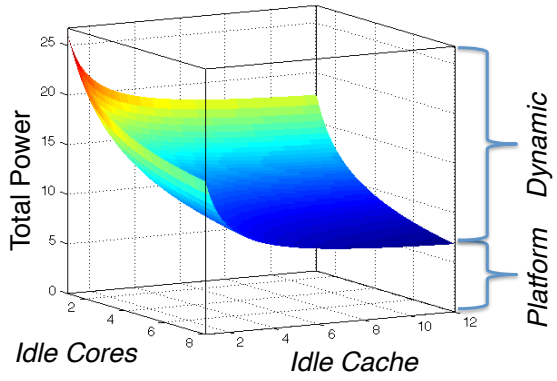


Figure 3.10: Example application 0 RTF.

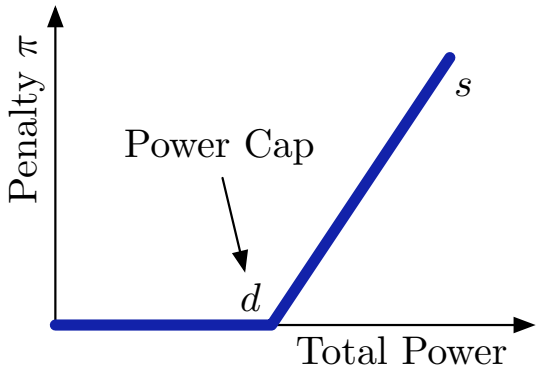


Figure 3.11: Example application 0 penalty function using the deadline as a power cap.

appropriate power management mechanisms, these idle resources can be powered off or put in a low power model to save power.

Additionally, application 0's resources allocations act as *slack* variables in our optimization problem, turning the resource bounds into equalities:

$$\sum_{p \in P} a_{p,r} - A_r = 0, r = 1, \ldots n. \tag{3.9}$$

The "response time" for application 0, $\tau_0$, is artificially defined to be the total system power consumption. Application 0's RTF represents how the system power improves when particular resources are left idle (*i.e.,* allocated to application 0), which is similar to other RTFs since they represent how the response time of an application improves when allocated particular resource types. Figure 3.10 shows an example application 0 RTF.

The penalty function $\pi_0$ establishes a system tradeoff between power and performance that will determine which resources are allocated to applications to improve performance and which are left idle. The penalty function $\pi_0$ can be used to keep total system power below the parameter $d_0$ to the extent that the penalties of other applications cannot overcome its penalty slope $s_0$. Both $s_0$ and $d_0$ can be adjusted to reflect the current battery charge in mobile devices. For example, as the battery depletes, $d_0$ could be decreased or $s_0$ increased to force other applications to slow or cease execution.

The power response function is affine and monotone non-increasing in its arguments $a_{0,r}$, which satisfies our convexity requirements for RTFs, thus making it safe for us perform this application 0 trick in our optimization. Additionally, creating slack variables turns the resources constraint inequalities into equalities, which makes the optimization easier to solve.

We chose to use the application 0 abstraction to represent power and energy over the more traditional approach of directly adding an allocation cost to the optimization because we found it to be more expressive of real life scenarios. Using the RTF machinery, we are

able to represent the power of the resources running as a function rather than simply a value, which enables us to express things like the fact that using more of a resource increases the power consumption per resource, thanks to thermal interactions. The penalty functions deadline allows us to represent scenarios like "I need my battery to last until I plug it in when I get home tonight." In this case, the power needs to be capped so that the battery does not drain too fast, but there is little advantage to saving more power below the cap. As shown in Figure 3.11, with PACORA's deadline and slope arguments this scenario can easily be captured; as long as the power consumption is less than the deadline then there is no penalty to the system, but greater than the deadline, the slope is quite steep.

## 3.6 Summary

In this chapter, we present the mathematical formulation of PACORA and prove it's convexity. We show some initial modeling experiments that we used to guide PACORA's design.

# Chapter 4

# RTF Exploration and Feasibility Study

In this chapter, we present two early studies we performed to test the potential of a model-based framework for resource allocation. Building on the results for the experiments presented in Chapter 3, the first study in this chapter further evaluates several different model formats using our MATLAB [98] framework. However, rather than collecting data using a simulator we instead use RAMP Gold [9, 138, 137], a multiprocessor emulator. The emulator performance enables us to evaluate the quality of the resource allocation decisions produced by the framework using real benchmarks in addition to the synthetic microbenchmarks. The second study uses the MATLAB framework to evaluate PACORA using data collected from several benchmark suites on a current hardware platform with a modern operating system.

## 4.1 RTF Exploration and System Potential using an FPGA-based System Simulator

In Chapter 3, we looked at the accuracy of modeling eight synthetic microbenchmarks using linear, quadratic, KCAA and GPRS models. We found that the more complex models were indeed more accurate, but more expensive to build. However, since the ultimate measure of performance for our resource allocation system is the quality of the allocation decisions and not the accuracy of the models, we felt it was important to evaluate the resource allocations produced by each model type before selecting one. The following experiments are intended to study quality of resource allocation decisions using real benchmarks in addition to the synthetic microbenchmarks.

### Platform

For our experiments, we choose to use an FPGA-based multiprocessor emulator RAMP Gold [9, 138, 137] because it allows us to conduct resource allocation experiments at a realistic

| Attribute | Setting |
|---|---|
| CPUs | 64 single-issue in-order cores @ 1 GHz |
| L1 Instruction Cache | Private, 32 KB, 4-way set-associative, 128-byte lines |
| L1 Data Cache | Private, 32 KB, 4-way set-associative, 128-byte lines |
| L2 Unified Cache | Shared, 8 MB, 16-way set-associative, 128-byte lines, inclusive, 4 banks, 10 ns latency |
| Off-Chip DRAM | 2 GB, 4×3.2 GB/sec channels, 70 ns latency |

Table 4.1: Target machine parameters simulated by RAMP Gold.

scale—something that was not possible with the simulator used in Chapter 3. Using RAMP Gold, we are able to emulate an operating system running real benchmarks to completion on a 64-core machine.

RAMP Gold is a cycle-accurate level-7 FAME simulator [137]. We run RAMP Gold on five Xilinx XUP FPGA boards. Each board is programmed to simulate one instance of our target architecture, and we use the multiple boards to provide higher throughput of independent emulation runs. We selected a 64-core machine to increase the space of possible allocations in order to stress our framework. Table 4.1 lists the target machine parameters. We implemented the hardware performance measurement system described in [19] to collect data.

### Operating System

We use an in-house prototype operating system, The Research Operating System (ROS) [79, 92, 33], in our experiments. ROS is a simple OS designed to assign resources to applications[1]. We selected ROS because its basic two-level scheduling design matched well with the system assumptions made by our resource allocation framework and because it was easy to modify to support additional partitioning mechanisms. We ported ROS to boot on RAMP Gold and modified its functionality to support our scheduling framework (including paging management and threading libraries).

### Partitioning Mechanisms

In our experiments, we wanted to explore each of the three resource types (*i.e.,* computation, capacity, and bandwidth) mentioned in Chapter 3. As a result we chose to determine allocations for the cores and their private caches, the shared last-level cache, and shared memory bandwidth. For each resource, we provide a mechanism to prevent applications from exceeding their allocated share. The OS assigns cores and their associated private resources to a specific application. For the shared last-level cache, we modify the OS page-coloring algorithm so that applications are never given a page from a different application's color allocation.

[1]ROS is the starting design and implementation for both Tessellation OS [35] and Akaros [1].

| Name | Type | Parallelism | Working Set | Bandwidth Demand |
|---|---|---|---|---|
| Blackscholes | financial PDE solver | coarse data parallel | 2.0 MB | minimal |
| Bodytrack | vision | medium data parallel | 8.0 MB | grows with cores |
| Fluidanimate | animation | fine data parallel | 64.0 MB | grows with cores |
| Streamcluster | data mining | medium data parallel | 16.0 MB | high |
| Swaptions | financial simulation | coarse data parallel | 0.5 MB | grows with cores |
| x264 | media encoder | pipeline | 16.0 MB | grows with cores |
| Tiny | synthetic | one thread does all work | 1 KB | minimal |
| Greedy | synthetic | data parallel | 16.0 MB | high |

Table 4.2: Benchmark description. PARSEC benchmarks use `simlarge` input set sizes, except for x264 and fluidanimate, which use `simmedium` due to limited physical memory capacity. PARSEC characterizations are from [17].

To partition off-chip memory bandwidth, we use Globally-Synchronized Frames (GSF)[87]. GSF provides strict Quality-of-Service guarantees for minimum bandwidth and the maximum delay of a point-to-point network—in our case the memory network—by controlling the number of packets that each core can inject per frame. We use a modified version of the original GSF design, which tracks allocations per application instead of per core, does not reclaim frames early, and does not allow applications use any excess bandwidth. These changes make GSF more suited to our study since we want to strictly bound the maximum bandwidth per application. To implementing GSF, we modified the target machine's memory controller in RAMP Gold to synchronize the frames and track application packet injections.

## Description of Workloads

In our experiments, we use the PARSEC 2.0 benchmark suite [17], as well as two of the synthetic microbenchmarks from Chapter 3. We selected the PARSEC suite because the applications are highly scaleable and thus well suited to varying the core allocations in our experiments. Due to library and OS dependencies, we were only able to able to port six of the PARSEC benchmarks to the RAMP Gold/ROS platform, so we use those six in our experiments. We use the `simlarge` input set sizes, except for `x264` and `fluidanimate`, which use `simmedium` due to the limited physical memory capacity on the Xilinix FPGAs. Table 4.2 summarizes the benchmarks.

## Resource Allocation Experiments

To test the quality of decisions produced by each model type, we first ran each of the benchmarks alone on the machine several times, each time varying the number of cores, and cache and bandwidth allocations to create a training sample set. We use a design of experiments (DoE) technique known as the Audze-Eglais Uniform Latin Hypercube [16] to

select the points included in the sample set using 20% of the possible allocations. Audze-Eglais selects sample points which are as evenly distributed as possible through the space of possible allocations.

We use the training samples to create linear additive models, quadratic response surface models, and non-linear models based on Kernel Canonical Correlation Analysis (KCCA) [10] and Genetically-Programed Resource Surface (GPRS). We use MATLAB's [98] multivariate regression techniques to create the quadratic and linear models. The KCCA and GPRS models are created using custom C code.

Using the performance models from the applications running alone, our MATLAB framework makes resource allocations for four pairs of benchmarks. We use MATLAB's implementation of the medium-scale active-set algorithm, which is a sequential quadratic programming based solver to maximize an object function that represents the quality of the resource allocation.

For these experiments, we evaluate two objective functions. Our first objective function is minimizing the maximum cycles, *i.e.,* makespan. We selected makespan both because it is a classic scheduling criterion [135] and because it makes sense for mobile systems were the goal is to complete everything as quickly as possible and then go back into a low power sleep. Second, we chose a simple proxy for total energy consumed, the total number of cycles run (the sum of the cycles on each core) + 10× the total number of off-chip accesses since energy efficiency is becoming increasingly important for all systems. Our chosen optimizer depends on the convexity of the function to guarantee optimality and only some of our objective functions are convex. As a result, the optimization algorithm may choose local minima allocations even with perfect models. However, we felt optimizers to handle non-convex functions optimally would be prohibitively expensive for resource allocation in operating systems, so we chose not to explore more complex optimizers.

To test the quality of decisions produced by our MATLAB framework, we simulated all possible schedules of allocations for the four pairs of benchmarks running simultaneously, for a combined total of 68.5 trillion target core-cycles [2].

We compare the quality of our allocations with a few baselines: the optimal allocation, naively giving each application half of the machine, or time-multiplexing each application across the entire machine. The time-multiplexing scheme runs the first application to completion and then runs the second application to completion. We believe this is an overly optimistic representation of time-multiplexing; more fined-grained time-multiplexing could lead to longer runtimes due to cache interference effects and other context swap overheads. Figure 4.1 shows the quality of our resource allocation decisions for the two objective functions as compared with the baselines. We show the results for `blackscholes` vs. `streamcluster` as a representative example. We do not show results for the GPRS as we found that they were extremely non-convex and as a result often produced very poor results when paired with our optimizer.

[2] A Core-cycle is 1 clock cycle of execution on 1 core; simulating a 64-core CMP for 1,000,000 cycles would be 64,000,000 core-cycles.

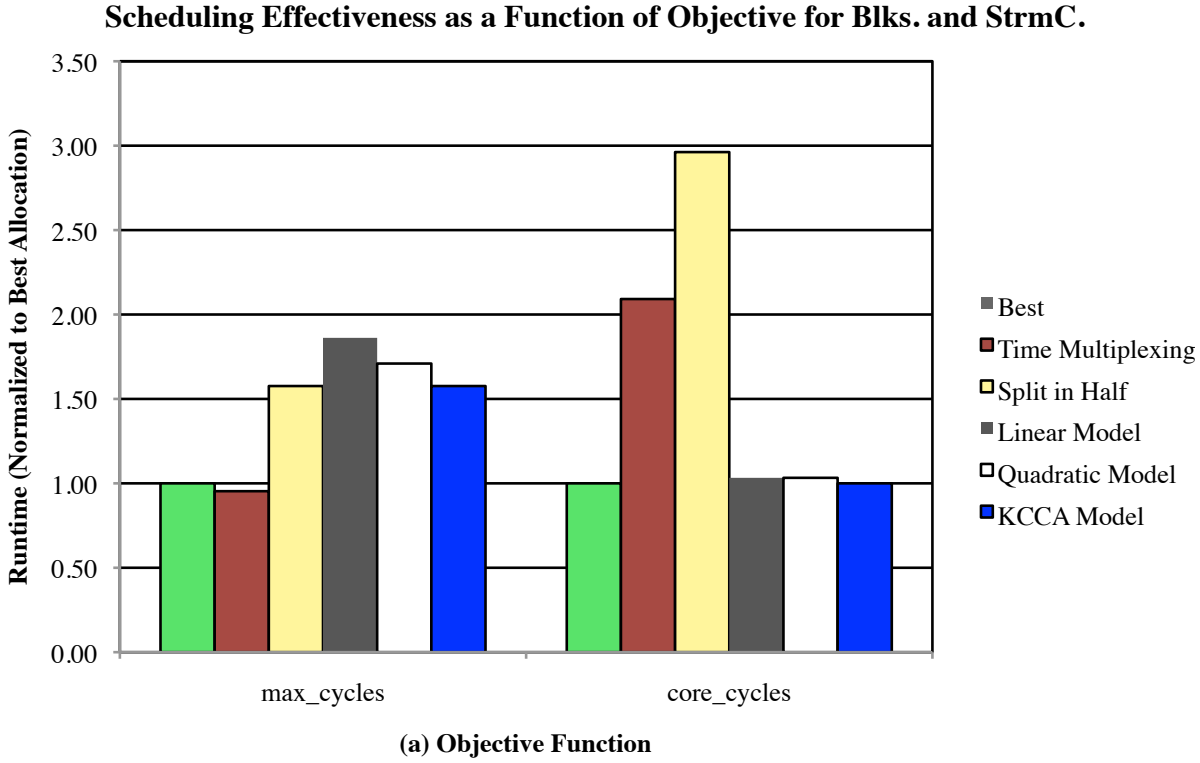


Figure 4.1: The performance of model-based allocation decisions for two objective functions (makespan/max_cycles and energy/core_cycles) compared with baselines. The results are normalized to the optimal resource allocation. The results shown are for `blackscholes` vs. `streamcluster`.

We found that the performance of the resource allocation framework depends a lot on the objective function. Our energy function is convex and thus the optimizer is able to find allocations near the true optimal allocation, whereas makespan is not convex and thus the selected allocations are often not even better than just dividing the resource in half. Additionally, since the PARSEC benchmarks scale so well, time-multiplexing the applications is near optimal for makespan. However, for applications that do not scale efficiently to 64-cores, time-multiplexing is unlikely to be the optimal choice.

In Figure 4.2, we separate out the results for the energy objective by benchmark pair. We do not show the GPRS or KCAA results because the optimizer struggled with their non-convexity and as a result often produced poor allocations. The simple linear and quadratic models perform much better with only a small set of sample points and have significantly lower overhead, which makes them more likely candidates for an actual implementation.

In these results, the performance of the linear and quadratic models are nearly identical. Both models perform significantly better than our naive baselines. They beat naively dividing

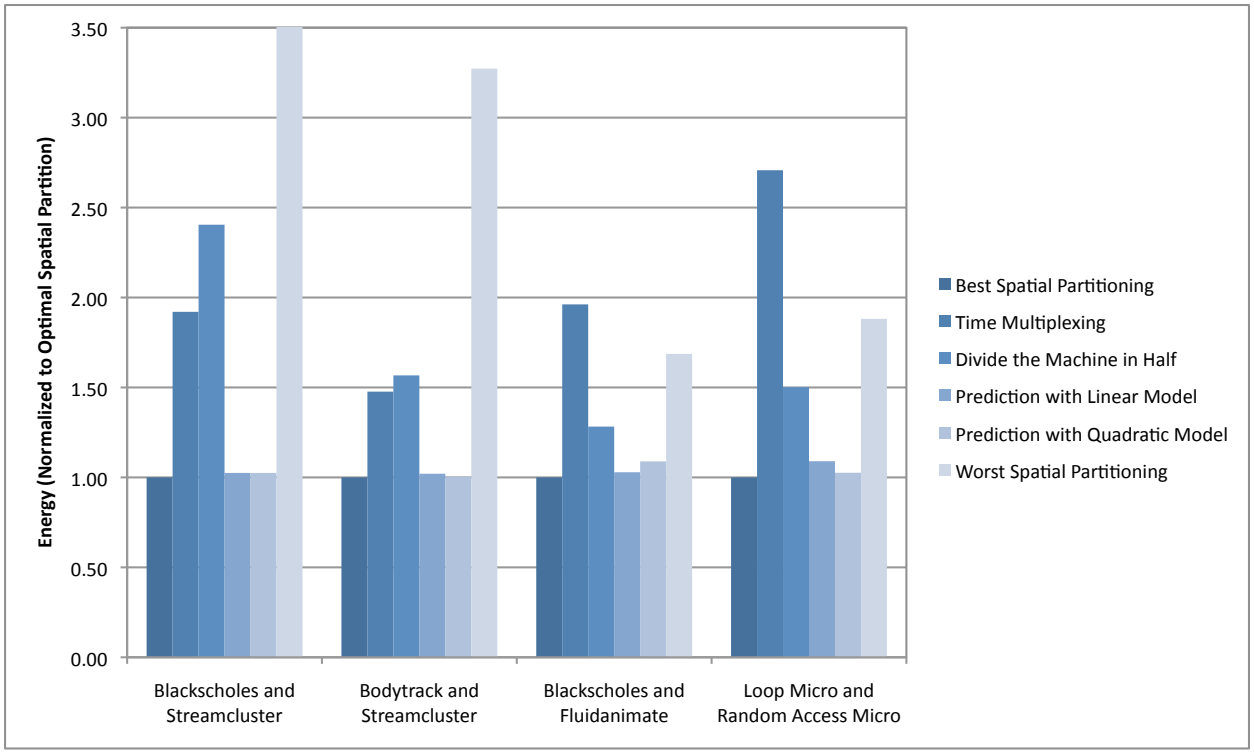


Figure 4.2: Comparison of the effectiveness of different scheduling techniques normalized to our quadratic model-based approach. The metric (sum of cycles on all cores + 10× sum of off-chip accesses) is a proxy for energy, so lower numbers are better.

the machine by 65% and time multiplexing by 100% on average. Furthermore, the chosen allocations are within a few percent of optimal every time. We also include the worst-case results to show that the penalty for poor decision making can be quite large, with an energy cost 3.25× greater than our allocation on average. As we scale up the problem size to include more resources and more applications, we only expect this gap to widen.

As a result of this study, we felt there was real potential available for model-based resource allocation. However, clearly the convexity of both the models and objective functions matter tremendously for consistently producing good resource allocations without using a heavy-weight optimizer. These results helped push us towards the convex-by-construction formulation for PACORA presented in Chapter 3. While the linear and quadratic models have nearly identical performance on the benchmarks tested, we chose to go with a quadratic model for PACORA because we felt it would be more robust to noise and outliers.

Figure 4.3 shows an additional study we performed to help guide our future experiments. Here we compared the allocation results for the makespan objective based on data from the PARSEC large and small benchmark sizes and the synthetic microbenchmarks. With the smaller benchmark sizes, we found that the relative difference in performance of the various

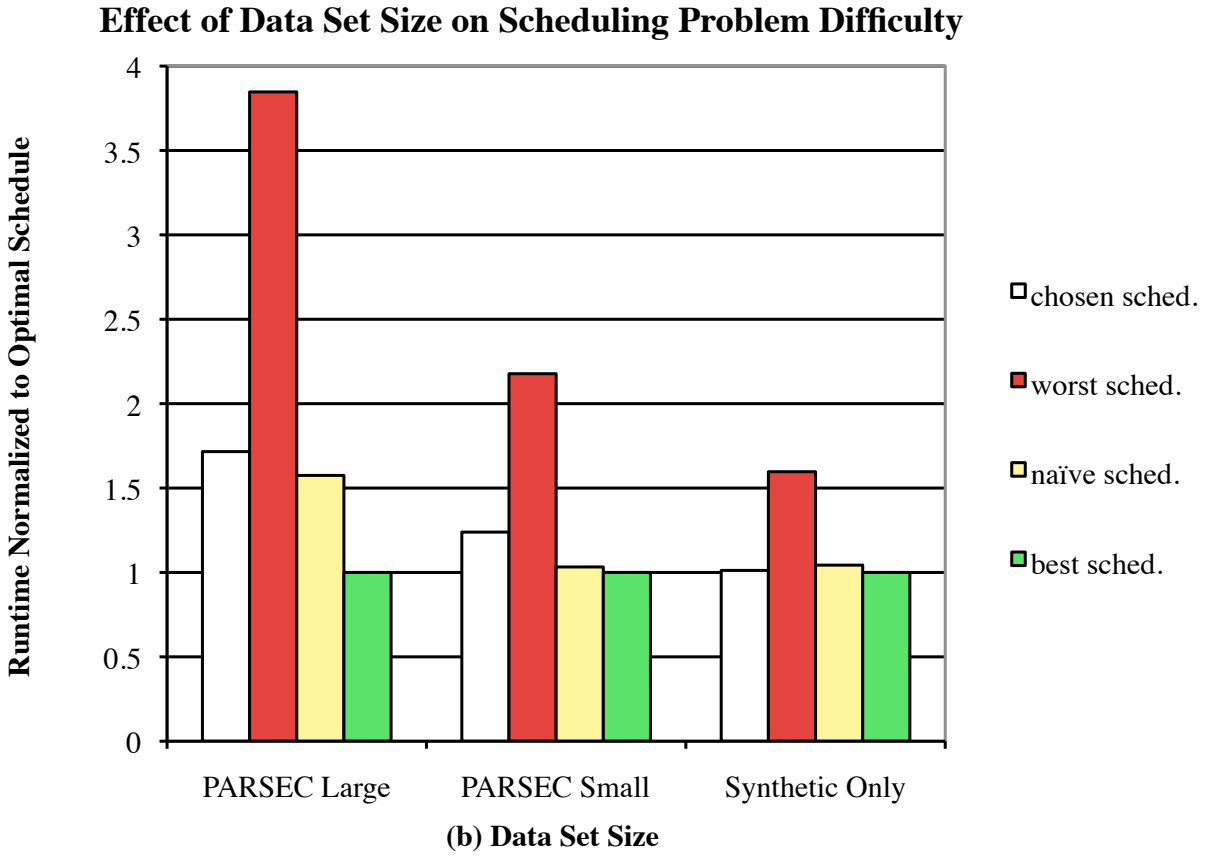


Figure 4.3: The effect of benchmark size on the difficulty of the resource allocation problem. The average chosen resource allocation from all pairs of benchmarks, worst case allocation and naive baseline case are normalized to the optimal allocation for each dataset. The objective is makespan.

allocation approaches becomes too small to reach any conclusions. In fact the difference between the worse and optimal allocations becomes so small it would be hard to justify smarter resource allocation techniques because the cost of a bad decision is very low. As a result, in future studies we only test PACORA on a real hardware running full applications and large-sized benchmarks to make sure our results are realistic.

## 4.2 PACORA Feasibility in a Real System

After formulating PACORA, we created a MATLAB implementation to test the effectiveness of PACORA's model-based convex optimization for allocating resources. We used it to experiment with the accuracy of different types of models and test the quality of the resource

allocation decisions. Data is collected online by running application benchmarks on a recent x86 processor running Linux-2.6.36. The measured data is processed using Python and then fed into MATLAB [98] to build the RTFs. MATLAB uses the RTFs to make resource allocation decisions. We compare performance of the chosen resource allocations with the actual measured performance of all possible resource allocations to test quality of the resource allocation decisions. We use CVX [68] in MATLAB to perform the convex optimization for building RTFs and making resource allocation decisions. We chose this static approach because it let us test many applications—44 in total—and many resource allocations rapidly.

## Platform

To collect data, we use a prototype version of Intel's Sandy Bridge x86 processor that is similar to the commercially available client chip, but with additional hardware support for way-based LLC partitioning. The Sandy Bridge client chip has four quad-issue out-of-order superscalar cores, each of which supports two hyperthreads using simultaneous multithreading [69]. Each core has private 32 KB instruction and data caches, as well as a 256 KB private non-inclusive L2 cache. The LLC is a 12-way set-associative 6 MB inclusive L3 cache, shared among all cores using a ring-based interconnect. All three cache levels are write-back.

The cache-partitioning mechanism is way-based and works by modifying the cache-replacement algorithm. To allocate cache ways, we assign a subset of the 12 ways to a set of hyperthreads, thereby allowing only those hyperthreads to replace data in those ways. Although all hyperthreads can hit on data stored in any way, a hyperthread can only replace data in its assigned ways. Data is not flushed when the way allocation changes.

We use a customized BIOS that enables the cache partitioning mechanism, and run unmodified Linux-2.6.36 for all of our experiments. To allocate cores, we use the Linux `taskset` command to pin applications to sets of hyperthreads. The standard Linux scheduler performs the scheduling for applications within these containers of hyperthreads. For our experiments we consider each hyperthread to be an independent core. To minimize inter-application interference, we first assign both hyperthreads available in one core before moving on to the next core. For example, a four-core allocation from PACORA represents four hyperthreads on two real cores on the machine.

## Performance and Energy Measurement

To measure application performance, we use the `libpfm` library [44, 114], built on top of the `perf_events` infrastructure in Linux, to access available performance counters [70].

To measure on-chip energy, we use the energy counters available on Sandy Bridge to measure the consumption of the entire socket and also the total combined energy of cores, their private caches, and the LLC. The counters measure power at a $1/2^{16}$ second granularity. We access these counters using the Running Average Power Limit (RAPL) interfaces [70].

## Description of Workloads

Our workload contains a range of applications from three different popular benchmark suites: SPEC CPU 2006 [130], DaCapo [21], and PARSEC [17]. We selected this set of applications to represent a wide variety of possible resource behaviors in order to properly stress PACORA's RTFs. We include some additional in-house research applications to broaden the scope of the study, and some microbenchmarks to exercise certain features.

The **SPEC CPU2006** benchmark suite [130] is a CPU-intensive, single-threaded benchmark suite, designed to stress a system's processor, memory subsystem, and compiler. Using the similarity analysis performed by Phansalkar *et al.* [115], we subset the suite, selecting 4 integer benchmarks ( `astar`, `libquantum`, `mcf`, `omnetpp`) and 4 floating-point benchmarks (`cactusADM, calculix, lbm, povray`). Based on the characterization study by Jaleel [74], we also pick 4 extra floating-point benchmarks that stress the LLC: `GemsFDTD`, `leslie3d`, `soplex` and `sphinx3`. When multiple input sets are available, we pick the single *ref* input indicated by [115].

We include the **DaCapo** Java benchmark suite as a representative of managed-language workloads. We use the latest 2009 release, which consists of a set of open-source, real-world applications with non-trivial memory loads, and includes both client and server-side applications.

The **PARSEC** benchmark suite is intended to be representative of parallel real-world applications [17]. PARSEC programs use various parallelization approaches, including data- and task-parallelization. We use native input sets and the `pthreads` version for all benchmarks, with the exception of `freqmine`, which is only available in OpenMP.

We add four **additional parallel applications** to help ensure we cover the space of interest: `Browser_animation` is a multithreaded kernel representing a browser layout animation; `G500_csr` code is a breadth-first search algorithm; `Paradecoder` is a parallel speech-recognition application that takes audio waveforms of human speech and infers the most likely word sequence intended by the speaker; `Stencilprobe` simulates heat transfer in a fluid using a parallel stencil kernel over a regular grid [75].

We also add two **microbenchmarks** that stress the memory system and cause increased interference between applications: `stream_uncached` is a memory and on-chip bandwidth hog that continuously brings data from memory without caching it, while `ccbench` explores arrays of different sizes to determine the structure of the cache hierarchy.

## RTF Experiments

Using a performance characterization of the applications, we select a subset of the benchmarks that are representative of different possible responses to resource allocations in order to reduce our study to a feasible size. Similar to [115], we use machine learning to select representative benchmarks. We use a hierarchical clustering algorithm [115] provided by the Python library `scipy-cluster` with the *single-linkage* method. The feature vector contains parameters to represent core scaling, cache scaling, prefetcher sensitivity and bandwidth

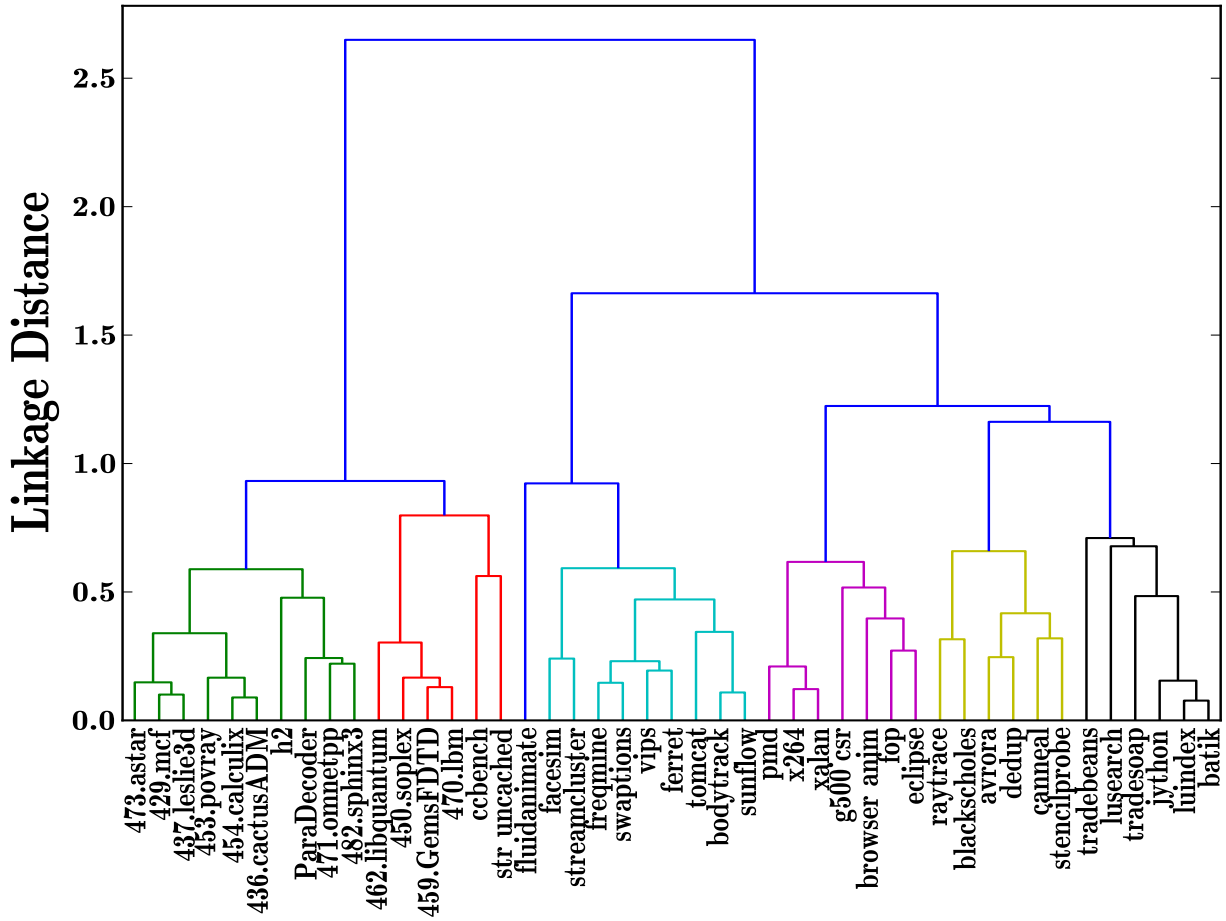


Figure 4.4: Dendrogram representing the results of clustering 44 PARSEC, DaCapo and SPEC benchmarks based on core scaling, cache scaling, prefetcher sensitivity and bandwidth sensitivity.

sensitivity. The clustering algorithm uses Euclidean distance between vectors to determine clusters.

The clustering, which is shown in Figure 4.4, results in six clusters representing the following (applications at the cluster center are listed in parenthesis):

- no scalability, high cache utility, (`429.mcf`)
- no scalability, low cache utility, (`459.gemsFDTD`)
- high scalability, low cache utility, (`ferret`)
- limited scalability, high cache utility, (`fop`)
- limited scalability, low cache utility, (`dedup`)
- limited scalability, low bandwidth sensitivity, (`batik`)

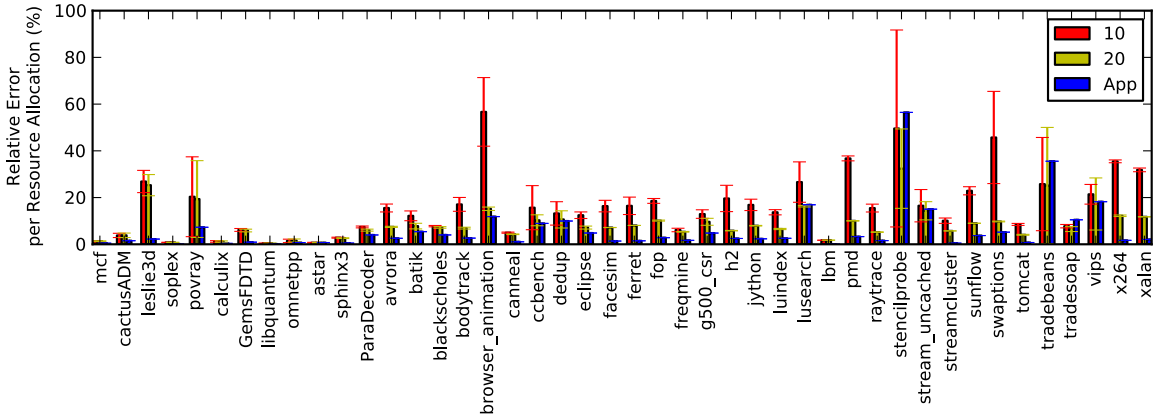


Figure 4.5: 1-norm of relative error from RTF predicted response time compared to actual response time. The actual response time is the median over three trials. 10 and 20 represent RTFs built with 10 and 20 training points respectively. App represents the variability (average standard deviation) in performance of the application between the three trials.

To test the effectiveness of our RTFs in capturing real application behavior, we measure each of our 44 benchmarks running alone on the machine for all possible resource allocations of cache ways and cores. Cores can be allocated from 1–8 and cache ways from 1–12 resulting in 96 possible allocations for each application. We use a genetic algorithm design of experiments [16] to select 10 and 20 of the collected allocations to build the RTFs. We also experimented with building RTFs with more data points but found that they provided little improvement over 20. We then use the model to predict the performance of every resource allocation and compare it with the actual measured performance (median value of three trials) of that resource allocation. We built three different models from three trials and tested each of them against median measured value.

Figure 4.5 shows the 1-norm of the relative error of the predicted response times per resource allocation for an RTF built with 10 training points and one built with 20. The average error per point is 16% for an RTF built with 10 training points and 9% for an RTF built with 20 training points. We also calculated the percentage variability (average standard deviation) for each resource allocation in the application between the 3 trials (shown as "App" in Figure 4.5). The average variability is 9%, so we can see that PACORA's RTFs are not much more inaccurate than the natural variation in response time in the application. It is not for possible an RTF to be more accurate than the application variability, and we can also see that applications with higher variability result in RTFs with larger relative errors, (*e.g.,* `stencilprobe`, `tradebeans`). Chapter 7 discusses application variability in more detail.

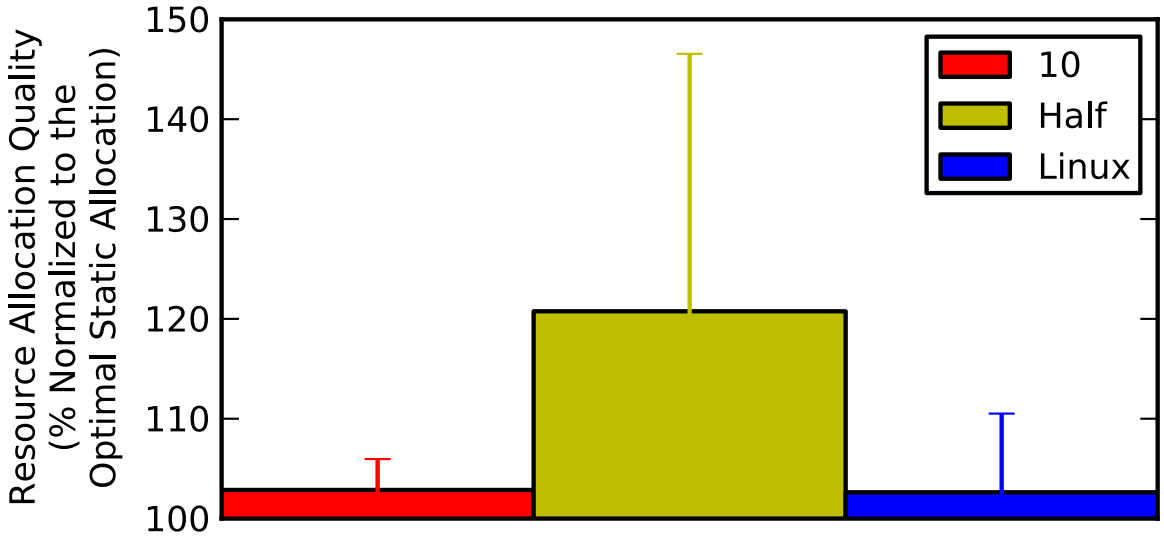


Figure 4.6: Resource allocation decisions for each pair of the cluster representative applications compared equally dividing the machine and a shared resources Linux baseline. Quality is measured is allocation performance divided by performance of the best possible allocation.

## Resource Allocation Experiments

Using the RTFs built for the applications, we let PACORA make static resource allocations for all possible pairs of the cluster representative applications. We then ran an exhaustive study of all possible resource allocations for each pair on our Sandy Bridge-Linux platform, measure the performance, and compare it with the best performing, *i.e.,* optimal, resource allocation. We also compare this result to equally dividing the resources between the two applications and to sharing all of the resources using the standard Linux scheduler. We only experiment with pairs of applications in order to make the exhaustive study computationally feasible; Chapter 6 presents results using more applications.

Figure 4.6 shows these results for our 10 point RTFs. As we might expect, simple naive heuristics do not perform well, and dividing the machine in half is around 20% slower than either PACORA or shared resources with standard Linux. PACORA's resource allocations are 2% from the optimal static allocation on average. Using shared resources with the standard Linux scheduler performs similarly but with a higher standard deviation. These results show that PACORA is able to provide performance comparable to Linux scheduling on shared resources with more predictable performance on average. While it may seem a bit counterintuitive to propose using a more complex system to get the same performance, this is actually a good result. Resource partitioning provides desirable benefits for applications

such as increased predictability and reducing interference[3]; however, it is often viewed as having a high cost. The belief is that sharing resources can result in higher utilization, as the applications can dynamically take advantage of available resources. In these results, we're seeing that PACORA can provide increased predictability with very little overhead. Increased predictability should allow increased utilization of machines compared to the current practice of using resource overprovisioning to guarantee QoS to sensitive applications. Additionally, as the shown in Chapter 6, PACORA's resource allocation decisions do not need to be static, but can be made dynamically to adjust to the changing needs of the applications, which should provide additional performance improvements.

### Effect of Model Accuracy on Decision Quality

There are two main sources of challenges for PACORA's design: performance non-convexity and performance variability. The main concern with performance non-convexity and variability is their effects on the accuracy of the response-time functions. However, an important result we have found while evaluating PACORA is that model accuracy has less impact on the quality of resource-allocation decisions than we anticipated. When experimenting with possible models for the RTFs, we found that while some models were always a little too inaccurate and did degrade the performance of the resource-allocation decisions, often better models provided insignificant improvement in resource-allocation decisions. Figure 4.7 shows the effect of model accuracy on the quality of the resource-allocation decisions made using the RTF model in Equation 3.8. Although there is a slight correlation between model accuracy and decision quality, many decisions with inaccurate models still result in near optimal allocations. This effect enables PACORA's model-based design to be feasible in a noisy system with real applications.

## 4.3 Summary

In this chapter, we presented two sets of experiments to evaluate model-based frameworks. Our first experiments used an FPGA-based emulator to evaluate the effect of different model types on the resource allocations produced. Our second set of experiments use current hardware and a modern OS to evaluate our PACORA framework in terms of model accuracy and allocation quality for 44 benchmarks. In both cases, we found that our allocation system was able to produce near optimal allocations and beat the baselines given a convex formulation.

[3]In [38], using the same partitioning mechanisms and applications, Cook et al. were able to reduce the worse cast inference from 36% to 7%.

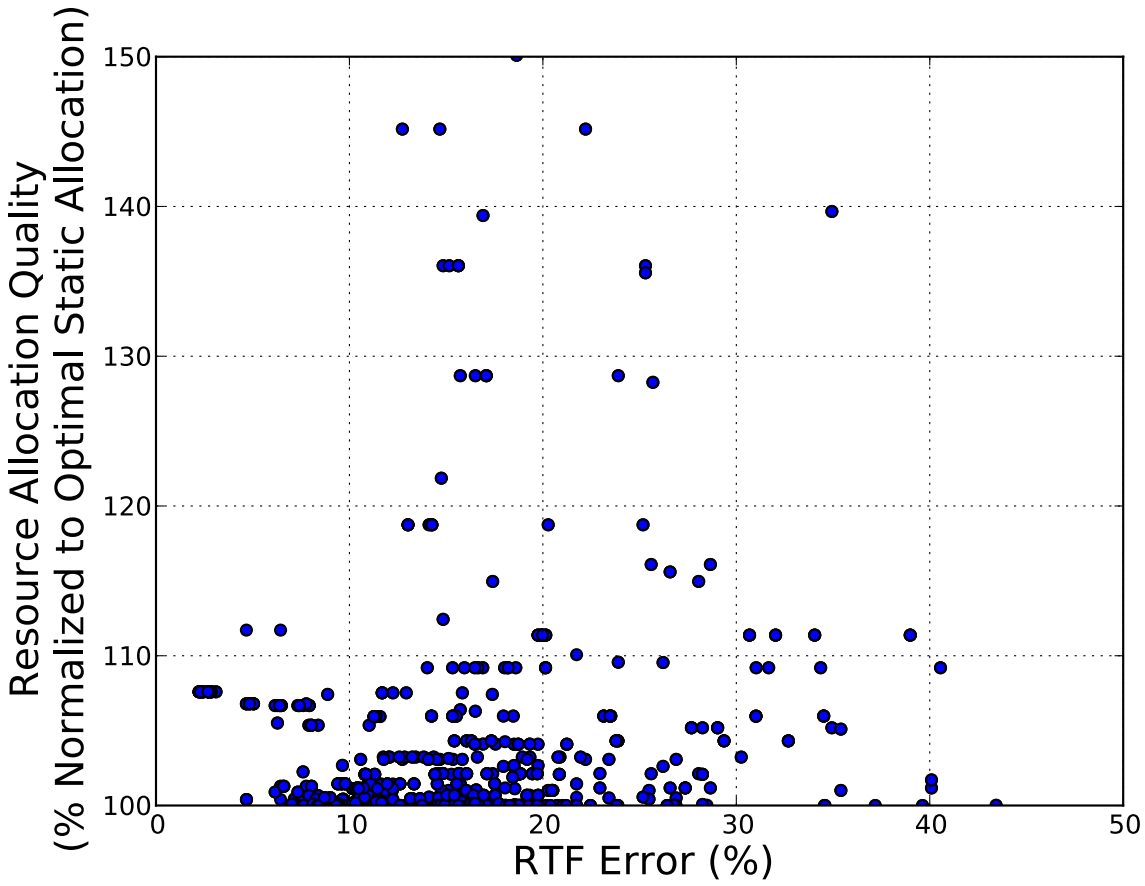


Figure 4.7: Effect of Model Accuracy on Decision Quality. The x axis represents the combined relative error of all RTFs used in the decision.

# Chapter 5

# PACORA Implementation in a Manycore OS

In this chapter, we present our implementation of PACORA in the Tessellation OS, a manycore research operating system [33, 92, 35, 34, 76]. We give an overview of Tessellation and why we chose it for our implementation. We then discuss the details for building response-time functions (RTFs) online in the operating system. Finally, we present our implementation of the resource allocator using an Alternating Direction Method of Multipliers (ADMM) optimization method.

## 5.1 Motivation

We believe PACORA is applicable to many resource-allocation scenarios from cloud computing to distributed embedded systems. For our initial prototype, we chose to study PACORA implemented in a general-purpose operating system for client systems, because we believe this scenario has some of the most difficult resource allocation challenges: a constantly changing application mix requiring low overhead and fast response times, shared resources that create more interference among the applications, and platforms that are too diverse to allow *a priori* performance prediction.

To evaluate PACORA's ability to make real-time decisions in a real operating system, we implemented it in an in-house research operating system, Tessellation. We chose to implement PACORA in Tessellation rather than a more conventional operating system such as Linux for three reasons:

1. Tessellation separates resource allocation from scheduling, so is closer to the OS architecture assumed by PACORA.
2. Tessellation allows resource revocation, enabling PACORA to dynamically reallocate resources.
3. Tessellation implements additional resource partitioning mechanisms letting PACORA manage more resource types.

Further, Investigating new resources management schemas by modifying a full-fledged production OS such as Linux is complex and requires more implementation effort than developing for Tessellation's resource-centric OS[1]. We use the Tessellation port to test our implementations of the algorithms, measure the overhead and reaction times, and illustrate PACORA's ability to work in a real system.

## 5.2 Tessellation Overview

This section briefly describes the key components of Tessellation OS [92, 33, 35, 34, 76]. The Tessellation kernel is a thin, hypervisor-like layer that provides support for dynamic resource management. It implements resource containers called cells along with interfaces for user-level scheduling, resource adaptation, and cell composition. Tessellation currently runs on x86 hardware platforms (*e.g.,* with Intel's Sandy Bridge processors).

### Cells

In Tessellation, resources are distributed to QoS domains called *cells*, which are explicitly parallel, light-weight, performance-isolated containers with guaranteed, user-level access to resources. The software running within each cell has full user-level control of the cell's resources.

As depicted in Figure 5.1, applications in Tessellation are created by composing cells via *channels*, which provide fast, user-level asynchronous message-passing between cells. Applications can then be split into performance-incompatible and mutually distrusting cells with controlled communication—thereby making them secure and easier to schedule efficiently.

Tessellation OS implements cells on x86 platforms by partitioning resources using *space-time partitioning* [122, 94], a multiplexing technique that divides the hardware into a sequence of simultaneously-resident spatial partitions. Cores and other resources are *gang-scheduled* [110, 49], so cells provide to their hosted applications an environment that is very similar to a dedicated machine.

### Resources and Services

Partitionable resources include cores, memory pages, and guaranteed fractional services from other cells (*e.g.,* a throughput reservation of 150 Mbps from the network service). They may also include cache slices, portions of memory bandwidth, and fractions of the energy budget, when hardware support is available [2, 88, 112, 123].

[1]For example, the Earliest Deadline First (EDF) scheduler in Tessellation is only 800 lines of user-space code, contained in four files. By contrast, support for EDF in Linux requires kernel modifications and substantially more code: the best-known EDF kernel patch for Linux, SCHED_DEADLINE, has over 3500 modified lines in over 50 files.

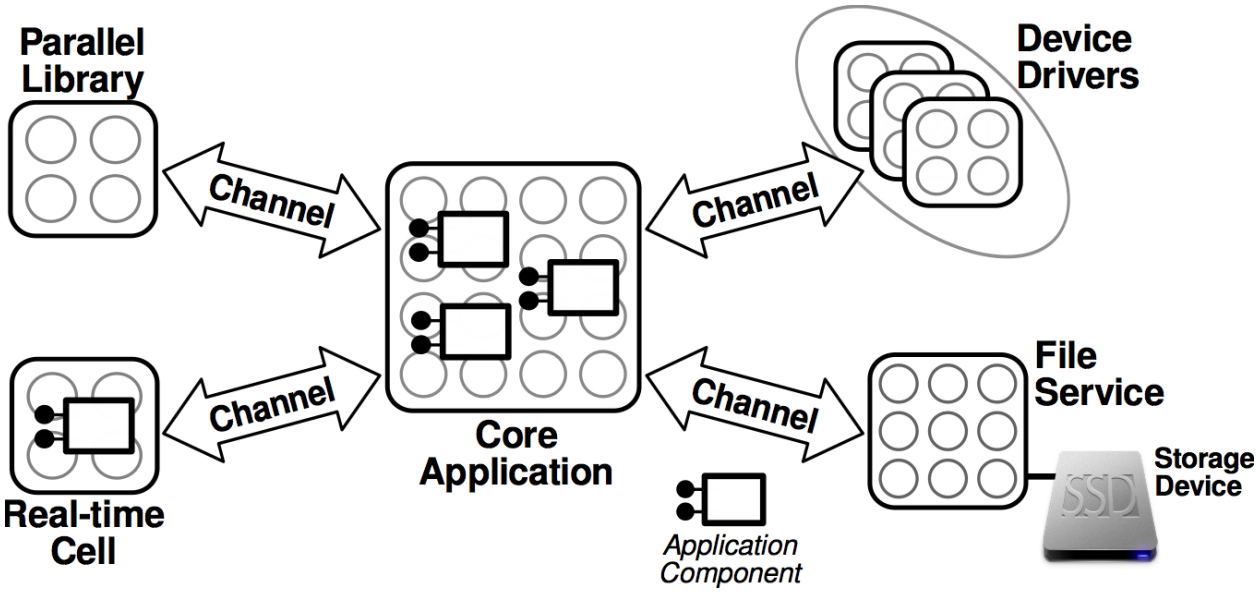


Figure 5.1: Applications in Tessellation are created as sets of interacting components hosted in different cells that communicate over channels. Standard OS services (*e.g.,* the file service) are also hosted in cells and accessed via channels.

Tessellation also creates *service cells* to encapsulate user-level device drivers and control devices. Each service can thus arbitrate access to its enclosed devices to offer service guarantees to other cells. Tessellation treats the services offered by the service cells as additional resources to be allocated to applications.

Tessellation currently has two such service cells implemented: the *Network Service*, which provides access to network adapters and guarantees that the data flows are provisioned with the agreed levels of throughput; and the *GUI Service*, which provides a windowing system with response-time guarantees for visual applications.

## Resource Management and Scheduling

Tessellation uses *Two-level scheduling* [89, 109] to separate global decisions about allocation of resources *to* cells (*first level*) from application-specific usage of resources *within* cells (*second level*). Resource allocation occurs at a coarse time scale to allow time for cell scheduling decisions to become effective.

### Scheduling

Scheduling within cells functions purely at the user-level, as close to the *bare metal* as possible, improving efficiency and eliminating unpredictable OS interference. Tessellation provides a framework for preemptive scheduling, called *Pulse*, enables customization and support for

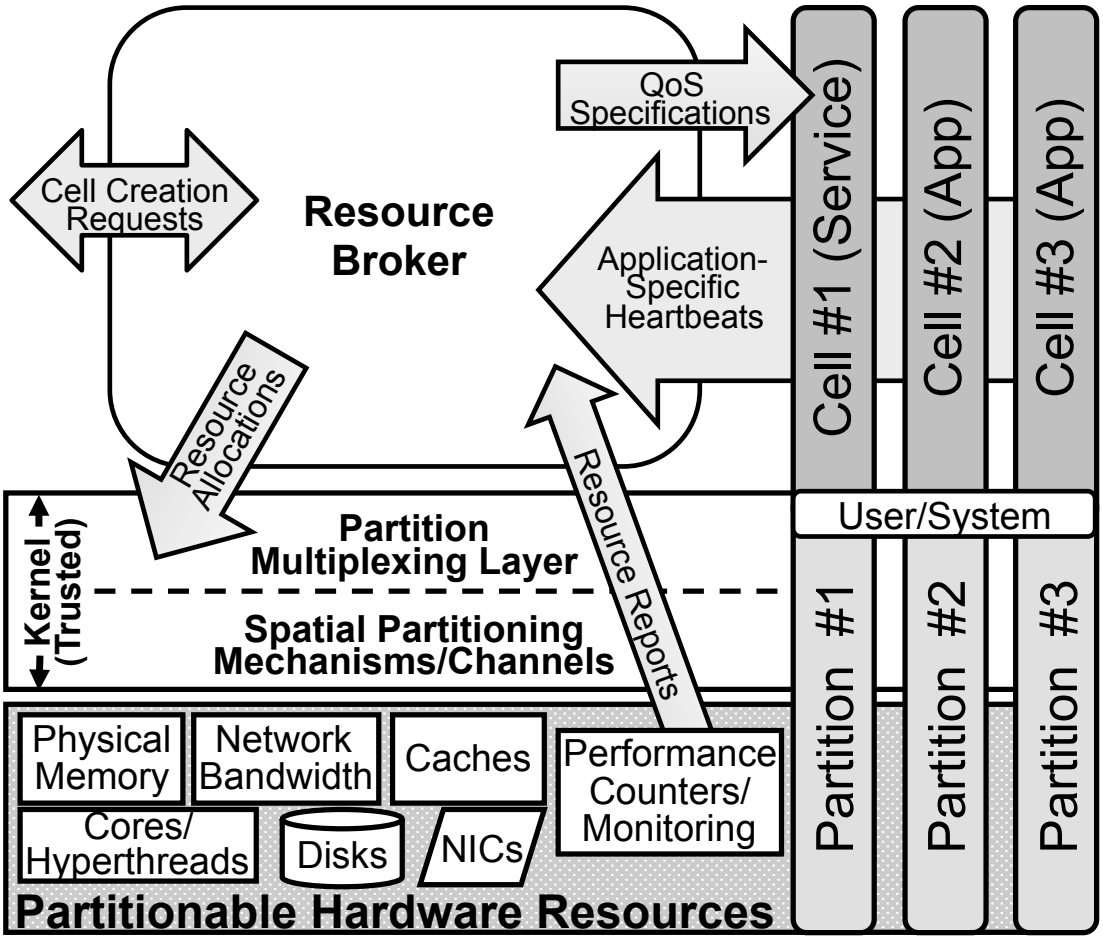


Figure 5.2: The Tessellation kernel implements *cells* through *spatial-partitioning*. The *Resource Broker* redistributes resources after consulting application-specific *heartbeats* and system-wide *resource reports*.

a wide variety of application-specific runtimes and schedulers without kernel-level modifications. The user-level runtime within each cell can be tuned for a specific application or application domain with a custom scheduling algorithm. Using Pulse, Tessellation provides pre-canned implementations for TBB [119] and a number of scheduling algorithms, including Global Round-Robin (GRR), Earliest Deadline First (EDF), and Speed Balancing [61].

Pulse provides support for revoking resource from schedulers. If a core is removed, Pulse's auxiliary scheduler runs the cell's outstanding scheduler contexts in a globally *cooperative*, Round-Robin manner; *i.e.,* a scheduler context runs until it either completes and transitions into an application context, or yields into Pulse, allowing other contexts to run. Additionally, the Pulse API provides callbacks to notify schedulers when the number of available cores changes, enabling resource-aware scheduling.

#### Adaptive Resource Allocation

Global resource allocation in Tessellation is performed by the *Resource Broker*, as Figure 5.2 shows. The Broker assigns resources to cells and communicates its allocation decisions to the kernel and services for enforcement. It reallocates resources, for example, when a cell starts or finishes or when a cell significantly changes performance. The Broker can periodically adjust allocations; the reallocation frequency provides a tradeoff between adaptability (to changes in state) and stability (of user-level scheduling).

Rather than implementing a single policy, the Broker is a resource-allocation framework that supports rapid development and testing of new allocation policies. We've implemented PACORA as a resource-allocation policy inside the Resource Broker.

## 5.3 PACORA in Tessellation

In this section, we provide details of PACORA's implementation in Tessellation's Resource Broker. Figure 5.3 shows the design. The Resource Broker runs in its own cell and communicates with applications and services through channels. PACORA leverages the existing Resource Broker interfaces to communicate with the cells, services, and kernel. The RTF Creation and Dynamic Penalty Optimization modules contain PACORA's model creation and resource allocation functions. Sections 5.4 and 5.5 describe these modules respectively.

### Cell Creation

When a cell is started, it opens its own channel with the Resource Broker and sends a cell creation message to register. The registration message contains the deadline and slope for PACORA's penalty function and optionally, a starting RTF model. The message format is shown below. PACORA uses the `message_type` field to determine how to unpack each of the message formats.

```
typedef struct perf_function {

    char message_type;
    uint64_t runtime_target;
    float penalty_slope;
    float model_constants[MODEL_SIZE];

} perf_func_t;
```

Ideally, penalty functions would be inferred by the system or provided by a more trusted source than the applications themselves. The simplest approach to implement this functionality in current operating systems would be to use an application's priority as the penalty slope and its interaction class [107] for the deadline. However, for our prototype the di-

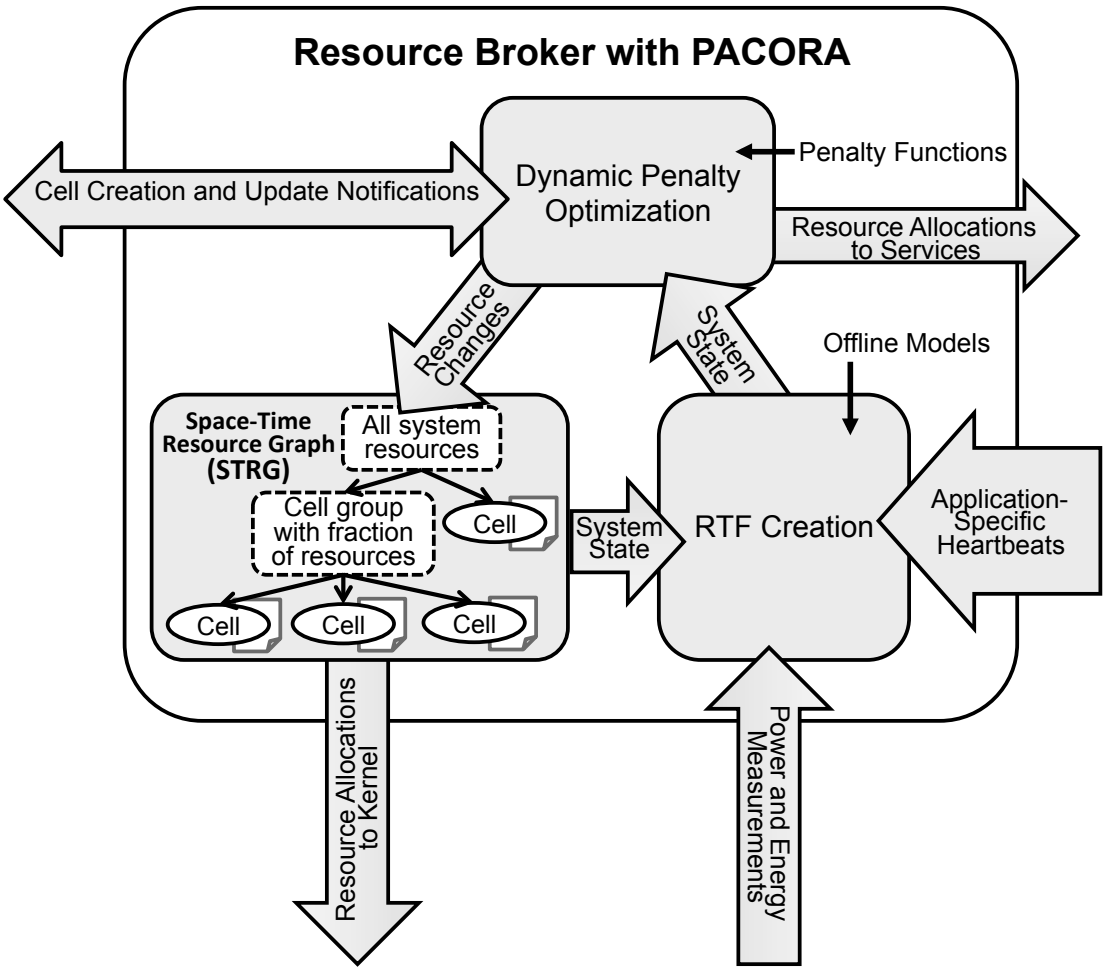


Figure 5.3: Overview of PACORA implementation in Tessellation. PACORA leverages the existing Resource Broker interfaces to communicate with the cells, services, and kernel. The RTF Creation and Dynamic Penalty Optimization modules contain PACORA's model creation and resource-allocation functions.

rect approach was straightforward to implement, and we believe does not detract from the validity of the resource-allocation experiments[2].

The Resource Broker also provides an interface for cells to update their penalty function or RTF while they are running, which we currently use to change RTF functions when an application changes phase or to update the penalty function of application 0 when the computer changes operating mode (*i.e.,* from battery to power source). The message formats for updating RTFs or Penalty Functions are shown below.

[2]For cloud systems, this approach is, in fact, common practice: applications typically provide their resource requirements to the system.

```
typedef struct penalty_update{

    char message_type;
    float penalty_slope;

} penalty_update_t;

typedef struct deadline_update{

    char message_type;
    uint64_t runtime_target;

} deadline_update_t;

typedef struct model_update{

    char message_type;
    float model_constants[MODEL_SIZE];

} model_update_t;
```

### Performance and Power Measurement

Applications report their own measured response times to PACORA by periodically sending performance report messages, called *heartbeats* [60]. Messages may contain the value for a single heartbeat or heartbeats may be batched together. The batch size is configurable, but is bounded by a maximum size, `MAX_VALUES_IN_PERF_REPORT`, set by the system. The code below shows the heartbeat message format.

```
typedef struct perf_report {

    char message_type;
    uint64_t data_values[MAX_VALUES_IN_PERF_REPORT];
    int32_t num_values;

} perf_report_t;
```

PACORA uses this information to build RTFs offline or online. Section 5.4 describes this process. As with the cell creation interface, it would be better for the system to directly measure application heartbeats rather than needing to trust the application's measurements. However, measuring application-specific heartbeats in a general-purpose way is a challenging problem, and we chose not to address it in this work. We instead focus exploring the value

of resource allocation using application-specific measurements first. Chapter 8 discusses heartbeat measurement further.

Tessellation provides a system call (show below) for PACORA to directly measure the system energy using the energy counters available on current x86 systems. PACORA uses this information to build application 0's RTF.

```
int sys_read_energy_counter(int32_t counter_id);
```

## Resource Allocation

PACORA periodically optimizes the system penalty and produces resource allocations. Section 5.5 describes the details of this process . Allocation decisions are communicated to the kernel and services for enforcement. Updates are sent to via the kernel the `sys_update_cells system` call, which adjusts the Space-Time Resource Graph. The function prototype for the system call is shown below.

```
int sys_update_cells(cell_spec_t* updated_cell_specs,
                     int32_t num_of_updated_cell_specs,
                      start_cell_params_t* new_cell_params,
                      int32_t* new_cel\tau_ids,
                      int32_t num_of_new_cells);
```

The Resource Broker has a channel with each service to communicate allocations. To update an allocation, PACORA sends a QoS Specification message to the service. The function prototype to send the message is shown below. The data field is service specific.

```
int change_allocation(int service_id, int cell_id, void *data,
                        size_t len, channel_gate_t* service_ch);
```

The Resource Broker design provides an adjustable reallocation frequency; the reallocation frequency provides a tradeoff between adaptability (to changes in state) and stability (of user-level scheduling).

If PACORA is using offline models then it only makes sense to reallocate resources when a cell starts or finishes or when a cell updates its penalty function or RTF, since those are only points at which the inputs to the optimization change. The exception to this is if optimization is terminated early for latency reasons, then each successive reallocation would move the allocations closer to optimal[3].

Online modeling reallocation could, in theory, be performed more frequently since the RTF functions can change as a cell runs. However, the models will only change significantly as a result of an application phase change or application input change and so, in practice, it is similar to the offline modeling case[4].

[3]We found early termination to be unnecessary since the complete optimization runs so quickly. Chapter 6 shows these results.

[4]In the cloud, RTFs for applications such as web services may also change as a result of the incoming request load and thus require more frequent reallocation.

In our experiments (See Chapter 6), we run PACORA continuously so that we can observe more resource-allocation decisions. However, the allocations rarely change outside of the cases described above (*i.e.,* cell start/stop or phase/penalty change), so in practice it performs the same (in terms of resulting allocations) as if it were run periodically but with a higher overhead since it continuously occupies a hardware thread—instead only for each optimization.

### Application Requirements

Our Tessellation implementation of PACORA requires a few minor modifications to applications[5]. First, during the application initialization phase where the cell registers with Tessellation, a cell creation call must be added to open a channel with the Resource Broker and send the application's penalty function. Second, the application must be modified to measure its response time and send these results to PACORA using the heartbeat interface. For our applications, this simply required adding two timer calls and one message send. Finally, if the system is using offline modeling and the application has multiple phases with different RTFs, then an update RTF message must be sent when the application changes phase.

These modifications are mostly a product of our prototype implementation decisions more than PACORA's fundamental design, and we hope that more advanced future implementations would eliminate the need to modify applications. Chapter 8 discusses heartbeats further.

## 5.4 RTF Creation

There are many ways to collect the response-time data for applications. The user-level runtime scheduler is one possible source, or the operating system could measure progress using performance counters. In our implementation, applications report their own measured values; however, this solution was chosen simply as a way to test the validity of the concept. In a production operating system, it may not be a good idea because applications could lie about their performance. In a single-operator datacenter environment, this might be less of a concern.

There are also many different possible moments to create response-time functions. RTFs could be created in advance and distributed with the application. This approach could make lots of sense for app stores since most of them cater to just a few platforms. RTFs could also be crowd-sourced and built in the cloud, which has the advantage of making it easy to collect a diverse set of training points. However, all of these approaches lack adaptability. As a result, we have chosen to implement two solutions that collect data directly from the user's machine. The first approach is to adapt to the system by collecting all of the training points at application install time and building the model then. The most highly adaptive

[5]In addition to the modifications already required for an application to run on Tessellation

approach collects data continuously as the application runs, uses the data to modify the model training set, and rebuilds the model. A hybrid approach may be the most effective: applications can begin with a generic or crowdsourced model and personalize it over time. The remainder of this section describes our model creation application in detail.

## Install Time Data Collection

To create RTF models either at install time or online, we use a convex least-squares approach described below. At install time, we use a genetic algorithm, Audze-Eglasis Design of Experiments [16], to select the resource allocation vectors to use for training. The application is run with each resource vector for a configurable number of heartbeats to record the response time. We average the response times collected for an allocation and use that result as the response time for the model[6]. These vectors and their response times are fed into the convex least-squares algorithm. Offline models are built entirely from this install time data. Online modeling uses the response times measured as the application runs in the models, but it could also start with a model built at install time.

## Least-Squares Minimization

After enough measurements, the model parameters $w$ of an application's RTF $\tau$ (Equation 3.8 in Chapter 3) can be discovered by solving an over-determined linear system $t = Dw$, where $t$ is a column vector of actual response times measured for the application and $D$ is a matrix whose $i$th row $D_{i,*}$ contains the corresponding resource vector. Estimating $w$ is relatively straightforward: we've implemented a least-squares solution using *QR factorization* [53] of $D$ to determine the $w$ that minimizes the $r$esidual error of $\|Dw - t\|_2^2 = \|Rw - Q^T t\|_2^2$. The solution proceeds as follows:

$$\begin{aligned} t &= Dw - \varepsilon \\ &= QRw - \varepsilon \\ Q^T t &= Rw - Q^T \varepsilon \end{aligned}$$

The individual elementary orthogonal transformations, *e.g.,* Givens rotations, that triangularize $R$ by progressively zeroing out $D$'s sub-diagonal elements are simultaneously applied to $t$. The elements of the resulting vector $Q^T t$ that correspond to zero rows in $R$ comprise $-Q^T \varepsilon$. Since $Rw$ exactly equals the upper part of $Q^T t$, the upper part of $Q^T \varepsilon$ is zero. The residual error for the $t_i$ can be found by premultiplying $Q^T \varepsilon$ by $Q$.

This formulation assumes a model norm $p = 1$. If a different model norm $p$ is desired, such as $p = 2$, we could first square each measurement in $t$ and each reciprocal bandwidth term in $D$ and then follow the foregoing procedure. The elements of the result $w$ will be squares as well, and the 2-norm of the difference in the squared quantities will be minimized[7].

[6]The Tessellation OS and our applications both have very little variability so average works fine for our purposes; however, Chapter 7 discusses why average may not be the right choice in other situations.

[7]This is not the same as minimizing the 4-norm; what is being minimized is $1/2\|\mathrm{diag}(Dww^T D^T - tt^T)\|_2^2$.

## Incremental Least-Squares

As resource allocation continues, more measurements will become available to augment $t$ and $D$. Moreover, older data may poorly represent the current behavior of the application. One option to adapt the RTF models to this incoming data would be to periodically rebuild the model once a sufficient amount of new data has accumulated. However, if the model is rebuilt too frequently, it can be quite expensive. If it is rebuilt rarely, then the models, and consequently the resource allocations, will be slow to respond to changes in applications. As an alternative, we've implemented an incremental approach described below to replace old data and efficiently update RTFs with each new data value.

To perform incremental least squares, we need a factorization $\tilde{Q}\tilde{R}$ of a new matrix $\tilde{D}$ derived from $D$ by dropping a row and adding a row. Corresponding elements of $t$ are dropped and added to form $\tilde{t}$.

The matrices $\tilde{Q}$ and $\tilde{R}$ can be generated by applying Givens rotations as described in Section 12 of [53] to *downdate* or *update* the factorization much more cheaply than recomputing it *ab initio*. The method requires retention and maintenance of $Q^T$ but not of $D$. Every update in PACORA is preceded by a downdate that makes room for it. Downdated rows are *not* always the oldest (bottom) ones, but an update always adds a new top row. For several reasons, the number of rows $m$ in $R$ will be at least twice the number of columns $n$. Rows selected for downdating will always be in the lower $m - n$ rows of $R$, guaranteeing that the most recent $n$ updates are always part of the model.

To guarantee convexity of the RTF, the solution $w$ to $t \approx QRw$ must have no negative components. Intuitively, when a resource is associated with more than a single $w_j$ or when the measured response time increases with allocation then negative $w_j$ may occur. *Non-negative Least-Squares* problems (NNLS) are common linear algebra, and there are several well-known techniques [30]. However since PACORA's online model maintenance calls for incremental downdates and updates to rows of $Q^T$, $Q^Tt$ and $R$, the NNLS problem is handled with a scheme based on the *active-set* method [86] that also downdates and updates the *columns* of $R$ incrementally, roughly in the spirit of Algorithm 3 in [95]. However, PACORA's algorithm cannot ignore downdated columns of $R$ because subsequent *row* updates and downdates must have due effect on these columns to allow their later reintroduction via column updates as necessary. This problem is solved by leaving the downdated columns in place, skipping over them in maintaining and using the QR factorization.

The memory used in maintaining a model with $n$ weights is modest, $24n^2 + 21n + \mathrm{O}(1)$ bytes. For $n = 8$ this is under 2 KB, fitting nicely in L1 cache. Our NNLS implementation takes 4 µs per update-downdate pair in Tessellation. The sections below describe our row and column update/downdate, rank preservation, and outlier minimization algorithms in more detail.

### Row Update and Downdate

A row downdate[8] operation applies a sequence of Givens rotations to the rows of $Q^T$. The rotations are calculated to set every $Q^T_{i,dd}$, $i \neq dd$ to zero. In the end, only the diagonal element $Q^T_{dd,dd}$ of column $dd$ will be nonzero. Since $Q^T$ remains orthogonal, the non-diagonal elements of row $dd$ will also have been zeroed automatically and the diagonal element will have absolute value 1. These same rotations are concurrently applied to the elements of $Q^T t$ and to the rows of $R$ $(= Q^T D)$ to reflect the effect that these transformations have on $Q^T$.

It is crucial to select pairs of rows and an order of rotations that preserves the upper triangular structure of $R$ while zeroing all but the diagonal entry of the chosen column $dd$ of $Q^T$. Since $dd$ is always below the diagonal of $R$ it initially will contain only zeros. It is therefore sufficient to rotate every non-$dd$ row with row $dd$, proceeding from bottom to top. The first $m - n - 1$ rotations will keep row $R_{dd,*}$ entirely zero, and the remaining $n$ rotations will introduce nonzeros in $R_{dd,*}$ from right to left. The effect on $R$ will be to replace zero elements by nonzero elements only within row $dd$. At this point, except for a possible difference in overall sign, $R_{dd,*} = D_{dd,*}$.

Now the rows from 0 down through $dd$ of the modified matrices $Q^T t$ and $R$ and both the rows and columns of the modified $Q^T$ are circularly shifted by one position, moving row $dd$ to the top (and column $dd$ of $Q^T$ to the left edge). The following is the result:

$$\begin{bmatrix} \pm 1 & 0 \\ 0 & \tilde{Q}^T \end{bmatrix} \begin{bmatrix} t_{dd} \\ \tilde{t} \end{bmatrix} = \begin{bmatrix} \pm D_{dd,*} \\ \tilde{R} \end{bmatrix} w - \begin{bmatrix} \pm 1 & 0 \\ 0 & \tilde{Q}^T \end{bmatrix} \begin{bmatrix} \varepsilon_{dd} \\ \tilde{\varepsilon} \end{bmatrix}$$

The top row has thus been decoupled from the rest of the factorization and may either be deleted or updated with new data.

The update application more or less reverses these steps, adding a new top row to $R$ and $t$ and a row and column to $Q^T$. Then $R$ is made upper triangular once more by a sequence of Givens rotations that zero its sub-diagonal elements (formerly the diagonal elements of $\tilde{R}$) one at a time. These rotations are applied not just to $R$ but also to $Q^T t$ and of course to $Q^T$ itself.

### Rank Preservation

If care is not taken in downdating $R$, its rows may become so linearly dependent, perhaps from repetitive resource allocations, that determining a unique $w$ is impossible. The rank of $R$ depends on both the resource optimization trajectory and the choices made in the row downdate-update algorithm. PACORA exploits the latter idea and simply avoids downdating any row that will make $R$ rank-deficient.

Deciding in advance whether downdating a row of $R$ will reduce its rank is equivalent to predicting whether one of the Givens rotations, when applied to $R$, will zero or nearly zero a diagonal entry of $R$. This property is particularly easy to determine because $dd$, the

[8]Here we use downdate to meaning removing a row.

row to be downdated, is initially all zeros in $R$, *i.e.* in the lower part of the matrix. In this situation, a diagonal entry of $R$, $R_{i,i}$ say, will be compromised if and only if the cosine of the Givens rotation that involves rows $dd$ and $i$ is nearly zero. The result will be an interchange of the zero in $R_{dd,i}$ with the nonzero diagonal element $R_{i,i}$. $R_{dd,i}$ is zero before the rotation because $R$ was originally upper triangular and prior rotations only involved row subscripts greater than $i$.

PACORA keeps track of the sequence of values in $Q^T_{dd,dd}$ without actually changing $Q^T$ so that if the downdate at location $dd$ is eventually aborted there is nothing to undo. It is also possible to remember the sines and cosines of the sequence of rotations, so they don't have to be recomputed if success ensues. A rank-preserving row to downdate will always be available as long as $R$ is sufficiently "tall". Having at least twice as many rows as columns is enough since the number of available rows to downdate matches or exceeds the maximum possible rank of $R$.

### Column Update and Downdate

The active-set NNLS method is based on the idea that since the only constraints are variable positivity, then for all components either the variable or its gradient will be zero at a solution point; see [24], page 142. The active set, denoted by $\mathbf{Z}$, comprises the column subscripts $j$ for which the variable $w_j$ is zero and the gradient $v_j$ is positive. If a column $j$ not currently in $\mathbf{Z}$ happens to acquire a negative $w_j$ after a back-solve, $w_j$ is zeroed, $j$ is moved into $\mathbf{Z}$ and column $j$ is downdated in $R$, thereby making the gradient positive. Conversely, if a column already in $\mathbf{Z}$ happens to acquire a negative gradient $v_j$ it is removed from $\mathbf{Z}$ and updated in $R$, allowing it to further reduce the value of the objective function.

After initial acquisition of data and $QR$ factorization, each step of PACORA's NNLS algorithm combines incremental row and column downdates and updates as follows:

**Algorithm 5.4.1:** INCREMENTALNNLS$(t_0, d_0)$

**local** $R, Q^T, Q^Tt, w, v, idx, d, u, done$
$R, Q^T, Q^Tt \leftarrow$ DNDTROW$(R, Q^T, Q^Tt, idx)$
$R, Q^T, Q^Tt \leftarrow$ UPDTROW$(t_0, d_0, R, Q^T, Q^Tt, idx)$
$w \leftarrow$ BACKSOLVE$(R, Q^Tt, idx)$
$v \leftarrow$ GRADIENT$(R, Q^Tt, idx)$
**repeat**
  $done \leftarrow$ **true**
  $d \leftarrow \arg\min(w)$
  **if** $w_d < 0$
    **then**
    $done \leftarrow$ **false**
    $R, Q^T, Q^Tt, idx \leftarrow$ DNDTCOL$(R, Q^T, Q^Tt, idx, d)$
    $w \leftarrow$ BACKSOLVE$(R, Q^Tt, idx)$
    $v \leftarrow$ GRADIENT$(R, Q^Tt, idx)$
  $u \leftarrow \arg\min(v)$
  **if** $v_u < 0$
    **then**
    $done \leftarrow$ **false**
    $R, Q^T, Q^Tt, idx \leftarrow$ UPDTCOL$(R, Q^T, Q^Tt, idx, u)$
    $w \leftarrow$ BACKSOLVE$(R, Q^Tt, idx)$
    $v \leftarrow$ GRADIENT$(R, Q^Tt, idx)$
**until** $done$
**return** $(w, v)$

The set **Z** and its complement **P** are implemented as an index $idx$ containing a vector of the column subscripts comprising **P** in increasing order followed by the column subscripts of **Z** in increasing order; $idx$ also contains an offset defining the beginning of **Z** in the vector. For example, if columns 1, 3, and 4 are in **Z** and columns 0, 2, and 5 are in **P** then the resulting vector is [0 2 5 1 3 4] and the offset is 3. Since the offset is just the size of the set **P** it is naturally called $p$.

Regardless of status, columns are left in place in $R$ The columns of $R$ belonging to **P** are denoted by $R^p$ and those in **Z** by $R^z$. The updating or downdating of a column only involves modifying the index $idx$ to redefine **P** and **Z** and then applying Givens rotations to the rows of $R$ to restore $R^p$ to upper triangular form.

When a column indexed by $d$ in $R^p$ is downdated because $w_d < 0$, that column is moved from **P** to **Z** in $idx$. To restore $R^p$ to upper triangular form, Givens rotations are applied to $R$ at rows $R_{d,*}$ and $R_{k,*}$ where $d < k < p$. The row subscripts $k$ are used in decreasing order from $p-1$ down to $d+1$, and each rotation zeros the subdiagonal element in $R^p$ of the column indexed by $k$. As usual, these rotations are also applied to $Q^T$ and $Q^Tt$. The

result in $R^z$ is a "spike" of nonzeros in the column that was moved; it can eventually extend to the bottom of $R$ as *row* updates occur.

Column movements from $\mathbf{Z}$ to $\mathbf{P}$ are based on the gradient $v$ of the objective function, namely

$$\begin{aligned} v &= 1/2\nabla\|Dw - t\|_2^2 \\ &= D^T(Dw - t) \\ &= R^T Q^T(QRw - t) \\ &= R^T(Rw - Q^T t) \\ &= R^T(-Q^T\varepsilon). \end{aligned}$$

If for some column in $\mathbf{Z}$ the inner product of the corresponding spiked row in $R^T$ and $-Q^T\varepsilon$ is negative, the column subscript must be moved to P. Updating $R^p$ reverses the downdating steps by zeroing the spike via a sequence of Givens rotations on $R$ between adjacent pairs of rows, starting at the bottom and ending at $m, m+1$ where $m$ is the position of the new column in $idx$. These rotations conveniently extend the columns to the right of $m$ in $R^p$ by one, thus restoring $R^p$ to upper triangular form. Once again, the rotations are also applied to $Q^T$ and $Q^T t$.

A new gradient computation and new back-solve for $w$ are clearly necessary after either downdates or updates to columns of $R$.

#### Outliers and Phase Changes

Some response time measurements may be "noisy" or even erroneous. A weakness of least-squares modeling is the high importance it gives to outlying values. On the other hand, when an application changes phase it is important to adapt quickly, and what looks like an outlier when it first appears may be a harbinger of change. What is needed is a way to discard either old or outlying data with a judicious balance between age and anomaly.

The downdating algorithm accomplishes this balance by weighting the errors in $\varepsilon = Q(Q^T t - Rw)$ between the predicted response times $\tau$ and the measured ones $t$ by a factor that increases exponentially with the age $g(i)$ of the error $\varepsilon_i$. Age can be modeled coarsely by the number of time quanta of some size since the measurement; PACORA simply lets $g(i) = i$. The weighting factor for the $i$th row is then $\eta^{g(i)}$ where $\eta$ is a constant somewhat greater than 1. The candidate row to downdate is the row with the largest weighted error, *i.e.*, $dd = \arg\max_i |\varepsilon_i| \cdot \eta^{g(i)}$ and that does not reduce the rank of $R$.

## 5.5 Dynamic Penalty Optimization

We chose to implement PACORA's resource allocation optimization using an Alternating Direction Method of Multipliers (ADMM) algorithm [25]. We selected ADMM for two reasons.

First, it works well with PACORA's RTF and penalty functions[9]. Second, it provides a natural way to distribute the algorithm. Using ADMM global optimization problem only needs to know the resource quantities and costs. The RTF and penally functions can be stored locally, and their gradients computation can be performed locally. While this feature may be less important in current client operating systems, it is very natural for the cloud. Running in a distributed fashion, the algorithm looks like a resource market expressed as an exchange problem. The applications send their resource requirements each round. The global leader processes the requirements and returns the resource costs. The applications adjust their resource requirements based on the new cost. The cycle continues until there is an agreed upon resource cost and thus a resource allocation. The advantage of this formulation is that very little information needs to be communicated. In addition to the performance benefits, the division can also be seen as potentially benefiting security or privacy since application's RTF models need never leave the local machine.

The remainder of this section describes our ADMM formulation in more detail.

## ADMM Overview

We follow the notation in §7.3 of Boyd et al. [25]. We have $n$ resources and $N$ applications. We let $a_i \in \mathbf{R}^n_+$ denote the vector of resources that application $i$ consumes. The (vector of) total resource consumption is then $a_1 + \cdots + a_N$; for future use we let $z$ denote the average resource usage per application, *i.e.,* , the total divided by $N$. Application $i$ has a cost (penalty in PACORA) function $\pi_i : \mathbf{R}^n \to \mathbf{R} \cup \{\infty\}$, where $\pi_i$ is convex. We let $\pi_i$ take on the value $+\infty$ to encode constraints on the resource allocation.

The total cost function is

$$\pi_1(a_1) + \cdots + \pi_N(a_N) + g(Nz),$$

where $g : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$ is the cost of consuming a total amount of resources (including any limits on total available resources). Note that the first $N$ terms are the costs associated with the applications (*i.e.,* their penalty contribution to the system), and the last term is the cost of providing the total resources (*i.e.,* penalty from application 0 for using the system power/energy). The problem is to choose the allocations $a_i$ to minimize the total cost (in terms of penalty), which is a convex optimization problem [24, 25].

We will solve this problem using the *sharing ADMM algorithm* from [25]:

$$\begin{aligned} a_i^{k+1} &:= \operatorname*{argmin}_{a_i} \left(\pi_i(a_i) + (\rho/2)\|a_i - a_i^k + \overline{a}^k - z^k + u^k\|_2^2\right) \\ z^{k+1} &:= \operatorname*{argmin}_{z} \left(g(Nz) + (N\rho/2)\|z - u^k - \overline{a}^{k+1}\|_2^2\right) \\ u^{k+1} &:= u^k + \overline{a}^{k+1} - z^{k+1}. \end{aligned}$$

[9]Since PACORA's penalty functions contain a discontinuity in the gradient, other approaches such as gradient decent don't behave appropriately.

Here $\rho > 0$ is an algorithm parameter, $k$ is the iteration number, and $\overline{a}^k$ is the average of the consumption vectors $a_1^k, \ldots, a_N^k$. We interpret $a_i^k$ as the (proposed) resource consumption of application $i$, $z^k$ as the (proposed) average resource consumption, and $u^k$ as a dual variable, all at iteration $k$. This algorithm converges to an optimal allocation, and $(1/\rho)u^k$ converges to the optimal dual variables (prices) for the resources.

The $x$-update can be carried out in parallel, for $i = 1, \ldots, N$. Each application, in each iteration, must minimize a function of the form

$$\pi_i(a_i) + (\rho/2)\|a_i - v\|_2^2,$$

*i.e.,* , each application evaluates a proximal operator; see [113].

The $z$-update step requires gathering $a_i^{k+1}$ to form the averages, and then solving a problem with $n$ variables. This step is also a proximal evaluation.

After the $u$-update, the new value of $\overline{a}^{k+1} - z^{k+1} + u^{k+1}$ is scattered to the subsystems.

## Applications

We disregard the interaction terms in our RTFs for this implementation as we found them rarely useful, and therefore did not seem worth the cost of the increased computation. Thus, our RTF model of application $i$ is

$$\tau_i(a_i) = \tau_{0,i} + \sum_{j=1}^{n} (w_i)_j/(a_i)_j$$

for $a_i > 0$, and $+\infty$ otherwise, where $\tau_{0,i} \in \mathbf{R}_+$ and $w_i \in \mathbf{R}_{++}^n$ are given response time model parameters. Recall the application cost is given by PACORA's penalty function

$$\pi_i(a_i) = s_i(\tau_i(a_i) - d_i)_+,$$

where $s > 0$ is a parameter and $d_i$ is the deadline. Note that $\tau_i(a_i) - d_i$ is the excess response time.

In each iteration of the sharing algorithm, we need to evaluate the proximal operator of $\pi_i$. To simply notation, we drop the subscript $i$ and consider one application. We need to minimize

$$s\left(\tau_0 + \sum_{j=1}^{n} w_j/a_j - d\right)_+ + (\rho/2)\sum_{j=1}^{n}(a_j - v_j)^2$$

over $a_j \geq 0$. Note that we can combine $s$ and $\rho$ (say, by dividing by $s$) and we can combine $\tau_0$ and $d$. So we now assume that $s = 1$ and $\tau_0 = 0$, with the understanding that $s$ and $\tau_0$ have been incorporated into $\rho$ and $d$.

We work out several cases. First suppose that the excess response time is negative. The first term above is zero and we must have $a = v$. So $a = v$ is the solution when $v > 0$ and

$$\sum_{j=1}^{n} w_j/v_j - d \leq 0.$$

Now consider the case when the excess response time is positive. In this case we simply minimize

$$\sum_{j=1}^{n} w_j/a_j + (\rho/2)\sum_{j=1}^{n}(a_j - v_j)^2,$$

which can be done for each $a_j$ separately. Each $a_j$ must satisfy

$$w_j/a_j^2 = \rho(a_j - v_j).$$

This equation can be solved extremely quickly, using a bisection method, Newton's method, or many others to find the unique (positive) values $a_j^\star$ that satisfy the equation. We then check if the resulting values of $a_j$ give nonnegative excess response time, *i.e.,* , if

$$\sum_{j=1}^{n} w_j/a_j^\star - d \geq 0.$$

If so, we are done: $a^\star$ is the value of the proximal operator.

Finally, we consider the special case (which occurs often) when the optimal values have zero excess response time, *i.e.,* ,

$$\sum_{j=1}^{n} w_j/a_j - d = 0.$$

The optimality condition in this case is that exists a $\theta \in [0, 1]$ for which

$$\theta w_j/a_j^2 = \rho(a_j - v_j)$$

(along with the condition $\sum_{j=1}^{n} w_j/a_j - d = 0$). We solve this equation by bisection on $\theta$. For each value of $\theta$, we use the same method as above to find $a_j$. We then check if $\sum_{j=1}^{n} w_j/a_j - d = 0$ is positive or negative. If it is positive, we increase $\theta$; otherwise we decrease it.

## Total Resource Cost

We take the resource cost to have the form

$$g(z) = \sum_{i=1}^{n} g_i(z_i).$$

Here $g_i(z_i)$ is the cost of providing resource $i$ at level $z_i$. A simple model is

$$g_i(z_i) = \begin{cases} c_i z_i & z_i \leq Z_i \\ +\infty & z_i > Z_i \text{ or } z_i < 0, \end{cases}$$

where $Z_i > 0$ is the maximum available, and $c_i > 0$ is the price for resource $i$. Since $g$ is separable, we can minimize over each resource separately; these are scalar problems.

We need to minimize

$$g_i(Nz_i) + (N\rho/2)(z_i - v_i)^2$$

over the (scalar) $z_i$. (Here $v_i = u_i^k + \overline{a}_i^{k+1}$.) The solution is simple:

$$z_i = \max\{0, \min\{v_i - c_i/\rho, Z_i/N\}\}.$$

Note that $N$ drops out, but we need to scale the bound accordingly. Also note that when the average $z_i = Z_i/N$, the total amount of resource is $Nz_i = Z_i$, meaning that resource $i$ is at its maximum possible level.

**Total Resource Cost with Free Zone**

We model the resource cost in terms of energy consumed with a function of the form

$$g(z) = \lambda \left( \sum_{i=1}^{n} c_i z_i - b \right)_+ ,$$

where $c_i > 0$ represents the amount of energy consumed by unit amount of resource $i$, the constant $b > 0$ is a threshold below which power consumption is free, and $\lambda > 0$ is the price charged for excess energy used (or relative weight used to tradeoff between response time and energy). We also impose lower and upper bounds on $z$, *i.e.,* , $0 \leq z_i \leq Z_i$ for $i = 1, \ldots, n$.

To evaluate the proximal operator, we need to minimize

$$g(Nz) + \frac{N\rho}{2} \sum_{i=1}^{n} (z_i - v_i)^2,$$

where $z$ is the averge resource vector, and $v_i = u_i^k + \overline{x}_i^{k+1}$. This is equivalent to

$$\begin{array}{ll} \text{minimize} & \lambda \left(\sum_{i=1}^n c_i z_i - b/N\right)_+ + (\rho/2) \sum_{i=1}^n (z_i - v_i)^2 \\ \text{subject to} & 0 \leq z_i \leq Z_i/N, \quad i = 1, \ldots, n. \end{array}$$

The solution can be obtained in a way similar to that used for evaluating the proximal operator of the response-time penalty functions.

We work out several cases. First suppose that the excess energy is negative. The first term in the objective function is zero. So the solution is

$$z_i = \max\{0, \min\{v_i, Z_i/N\}\}, \quad i = 1, \ldots, n$$

provided that

$$\sum_{i=1}^{n} c_i z_i - b/N \leq 0.$$

Next we consider the case when the excess energy is positive. In this case, we simply minimize

$$\lambda \left( \sum_{i=1}^{n} c_i z_i - b/N \right) + (\rho/2) \sum_{i=1}^{n} (z_i - v_i)^2,$$

which can be solved for each $z_i$ separately and the solutions are

$$z_i = \max\{0, \min\{v_i - \lambda c_i/\rho, Z_i/N\}\}, \quad i = 1, \ldots, n.$$

We then need to check

$$\sum_{i=1}^{n} c_i z_i - b/N \geq 0.$$

If so, we are done.

Finally we consider the case when the optimal allocations have zero excess energy, *i.e.,* ,

$$\sum_{i=1}^{n} c_i z_i - b/N = 0.$$

The solution takes the form

$$z_i = \max\{0, \min\{v_i - \theta c_i/\rho, Z_i/N\}\}, \quad i = 1, \ldots, n.$$

where $0 \leq \theta \leq \lambda$. We can do a bisection on $\theta$ to make the solution satisfy $\sum_{i=1}^{n} c_i z_i - b/N = 0$. Basically, if the excess energy is positive, then we increase $\theta$; otherwise we decrease it.

**Stopping criteria**

Here we describe a stopping criterion that is similar to the one in §3.3 of [25].

For our resource-allocation problem, the primal residuals at iteration $k$ are

$$r_i^k = a_i^k - z_i^k, \quad i = 1, \ldots, N,$$

and the dual residuals are

$$s_i^k = \rho(z_i^{k-1} - z_i^k), \quad i = 1, \ldots, N.$$

Here $z_i$ for $i = 1, \ldots, N$ are the variables that were eliminated to simplify the $z$-update in the sharing problem (see §7.3 of [25]). The variable $z$ in the simplified update is actually their average $\overline{z} = (1/N)\sum_{i=1}^{N} z_i$. Based on the derivation in [25, §7.3],

$$z_i^k = a_i^k - \overline{a}^k + \overline{z}^k, \quad i = 1, \ldots, N.$$

Therefore we have

$$r_i^k = a_i^k - z_i^k = \overline{a}^k - \overline{z}^k, \quad i = 1, \ldots, N,$$

*i.e.,* , the primal residuals for the applications are the same as the average primal residue. The dual residuals become

$$\begin{aligned} s_i^k &= \rho(z_i^{k-1} - z_i^k) \\ &= \rho((a_i^{k-1} - a_i^k) + (\overline{a}^k - \overline{z}^k) - (\overline{a}^{k-1} - \overline{z}^{k-1})) \\ &= \rho((a_i^{k-1} - a_i^k) + r^k - r^{k-1}), \quad i = 1, \ldots, N. \end{aligned}$$

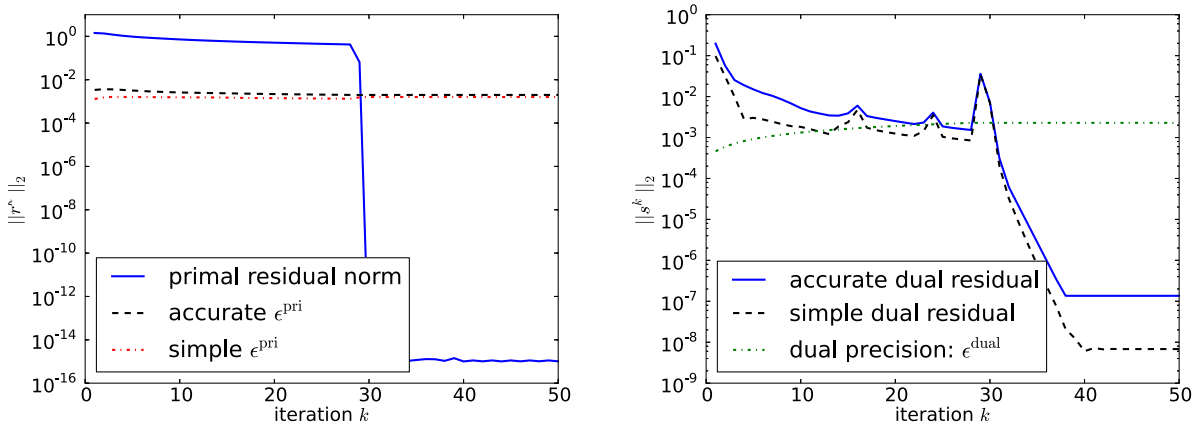


Figure 5.4: Progress of reducing primal and dual residual norms in ADMM. This is the case of expensive energy, notice that the simple dual residual often works as well as the accurate dual residual. Since the resource allocations are not reaching their bounds, so the simple dual residual $\|s^k\|_2 = \rho\|z^k - z^{k+1}\|_2$ converges to zero only asymptotically, and can serve as a stopping criterion.

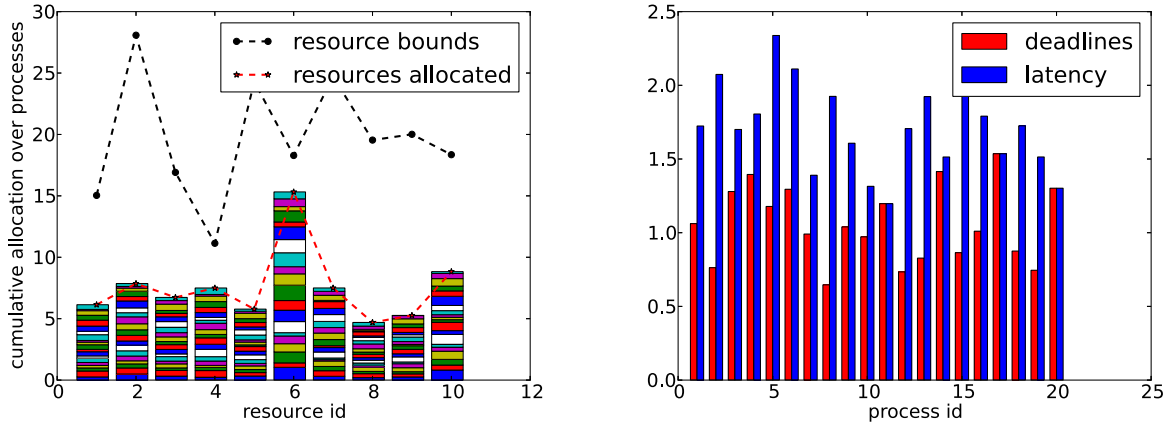


Figure 5.5: Visualization of optimal resource allocation (left) and resulting response time for each application (right). There are $n = 10$ resources and $N = 20$ applications. This is the case of expensive energy, so the total resources allocated are mostly well below their bounds, but the application response times are mostly exceeding the deadlines, which is the desirable result for this case where using resources has a higher penalty than missing deadlines.

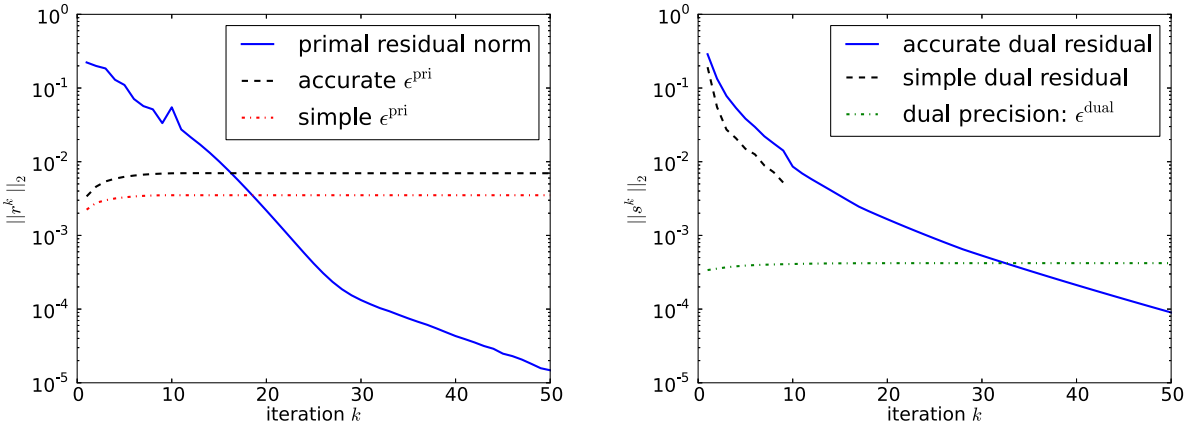


Figure 5.6: Progress of reducing primal and dual residual norms in ADMM. This experiment is the case of cheap energy, notice that the simple dual residual becomes exactly zero (discontinued in the plot) after 10 iterations, Since the energy is cheap, the resource allocations reach their bounds easily, so the simple dual residual $\|s^k\|_2 = \rho\|z^k - z^{k+1}\|_2 = 0$ become zero quickly, therefore can *not* serve as a stopping criterion.

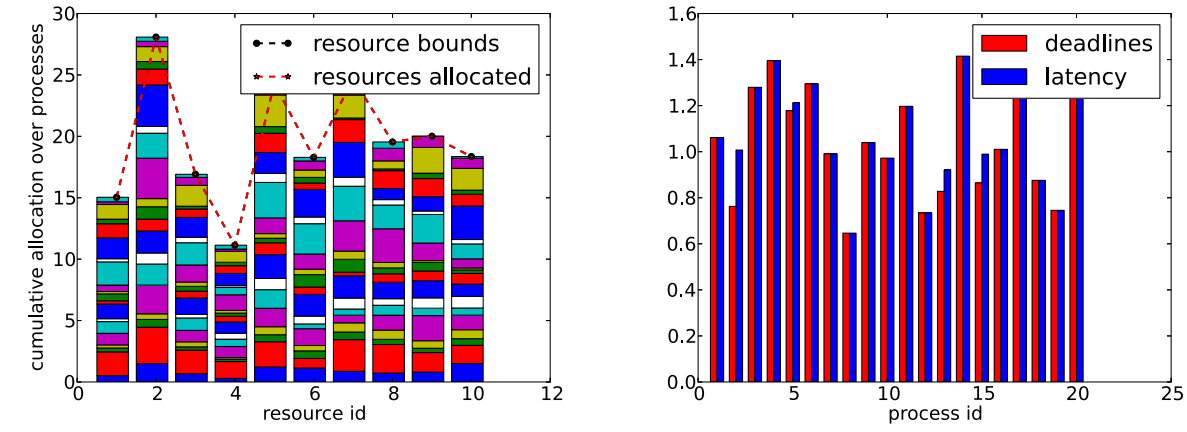


Figure 5.7: Visualization of optimal resource allocation (left) and resulting response time for each application (right). There are $n = 10$ resources and $N = 20$ applications. This experiment is the case of cheap energy, so the total resources allocated all reach their bounds, and most of the application response times are within their deadlines.

So the dual residuals are different for different applications.

The following termination criterion is similar to the one proposed in [25, §3.3]:

$$\begin{aligned} \|r^k\|_2 = \|\overline{x}^k - \overline{z}^k\|_2 &\leq \epsilon^{\text{pri}}, \\ \max\{\|s_1^k\|_2, \ldots, \|s_N^k\|_2\} &\leq \epsilon^{\text{dual}}, \end{aligned}$$

with

$$\begin{aligned} \epsilon^{\text{pri}} &= \sqrt{n}\epsilon^{\text{abs}} + \epsilon^{\text{rel}} \max\{\|x_1\|_2, \ldots, \|x_N\|_2, \|z_1\|_2, \ldots, \|z_N\|_2\}, \\ \epsilon^{\text{dual}} &= \sqrt{n}\epsilon^{\text{abs}} + \epsilon^{\text{rel}}\rho\|u^k\|_2. \end{aligned}$$

Since the computation involved in the above stopping criterion are rather heavy, and we also experimented with simplified conditions. In particular, we tried the following conditions which only uses the average vectors:

$$\begin{aligned} \|r^k\|_2 = \|\overline{x}^k - \overline{z}^k\|_2 &\leq \sqrt{n}\epsilon^{\text{abs}} + \epsilon^{\text{rel}}\|\overline{z}^k\|_2, \\ \|s^k\|_2 = \rho(\overline{z}^{k-1} - \overline{z}^k\|_2) &\leq \sqrt{n}\epsilon^{\text{abs}} + \epsilon^{\text{rel}}\rho\|u^k\|_2. \end{aligned}$$

Basically we simplified the calculation of $\epsilon^{\text{pri}}$ and the dual residual, while leaving the calculation of primal residual and $\epsilon^{\text{dual}}$ unchanged.

We perform two simple experiments with different resource costs to evaluate potential stopping criteria. Figures 5.4 and 5.5 represent the case where resources are very expensive, so there may be a lower penalty for applications to miss deadlines than for using more resources. Figures 5.6 and 5.7 represent the case where resources are very cheap and so applications should not miss their deadlines.

Figure 5.4 shows both the accurate calculations and their simplified counterparts. We see that the simple primal $\epsilon^{\text{pri}}$ is slightly smaller than the more accurate calculation, making primal residual a little harder to satisfy the termination condition. On the other hand, the simple dual residual calculation is smaller than the accurate dual residual calculation, making the dual residual easier to satisfy the termination condition. Figure 5.5 illustrates the optimal resource allocation and resulting response time for each of the applications.

It looks that the simplified stopping criterion is effective and sufficient. However, when we vary the parameters in the resource allocation problem, the simple conditions may breakdown. Figure 5.6 and 5.7 plot the same quantities but in the case of cheap energy (meaning that price $\lambda$ in the total resource cost function is small). In this case, the total resources allocated all reach their bounds, and most of the application response time are within their deadlines. Notice that the simplified dual residual becomes exactly zero (discontinued in the right plot of Figure 5.6) after 10 iterations. The reason is that since the energy is cheap, the resource allocations reach their bounds easily, so the simple dual residual $\|s^k\|_2 = \rho\|z^k - z^{k+1}\|_2 = 0$ become zero quickly, therefore can *not* serve as a stopping criterion.

In our implementation, we use the simplified calculation of $\epsilon^{\text{pri}}$, but do not simplify the calculation of the dual residual.

## 5.6 Summary

In this chapter, we presented details of PACORA's implementation in Tessellation, a many-core research OS. We gave an overview of Tessellation and our rational for selecting it. We provided the interfaces for applications and Tessellation to communicate with PACORA. We then presented the mathematical details for our online RTF creation using incremental non-negative least squares. Our incremental NNLS algorithm can add a new value to an application's model in just 4 µs by removing an appropriate row from the matrix and replacing it with the new value. We then presented our penalty optimization using sharing ADMM. ADMM naturally distributes the optimization by alternating between solving the primal and the dual problems. We performed a few simple experiments in simulation to test potential stopping criteria and found that we could simplify some of the calculation, but not all. Our NNLS and ADMM algorithms are self-contained and not Tessellation-specific and thus can be reused in any PACORA implementation. The next chapter evaluates their implementation in Tessellation.

# Chapter 6

# Evaluation in a Manycore OS

In this chapter, we evaluate our PACORA implementation in Tessellation OS using a video conference as a motivating scenario.

## 6.1 Dynamic Resource Allocation in a Manycore OS

In evaluating PACORA's ability to allocate resources dynamically to many applications, we selected a motivating scenario that we felt could easily occur on current laptops, if not yet mobile devices. We constructed a video conference scenario similar to chatting with a group of friends on Google Hangout or meeting with coworkers on Microsoft's Lync. In our video conference, every person in the meeting has a separate performance-guaranteed video stream. Typically, the videos are a small size, but the current speaker has a larger, high resolution video. Simultaneously, participants may be collaborating through web browsers, or watching shared video clips and web searching, while their systems run compute-intensive background tasks such as updates, virus scans, or file indexing.

Although it may be relatively straightforward to provide responsiveness guarantees for individual applications such as video streams in current systems, it is a real challenge to do so without reserving excessive resources, which will compromise system utilization, power consumption, or responsiveness of other applications. The goal was to show that PACORA can allocate for a mix of throughput and realtime applications effectively without significant overprovisioning.

## 6.2 Experimental Setup

In this section, we describe our platform, data collection system, and workloads for our dynamic resource allocation experiments.

## Platform

Our dynamic experiments are all run on an Intel Nehalem-EP system with two 2.66-GHz Xeon X5550 quad-core processors and hyperthreading enabled, resulting in 16 total hardware threads. This system contains a 1-Gbps Intel Pro/1000 Ethernet network adapter, which we use to receive incoming video streams and data. Tessellation allocates resources directly to applications. In addition to allocating cores and cache ways as in the experiments in Chapter 4, Tessellation can also allocate fractions of network bandwidth. This platform has the advantage of more cores, which allows us to simultaneously run more applications; however, it lacks cache-partitioning hardware, so we are only able show PACORA allocating cores and network bandwidth. In this system, PACORA has eight cores available to allocate, and Tessellation uses the remaining cores to run OS services. We artificially limit the available network bandwidth to 1500 kbits/s to make the resources more constrained.

The applications employ a second-level scheduler to schedule work onto the resources. Our experiments use a preemptive scheduling framework called PULSE (Preemptive User-Level SchEduling) [35], with two different scheduling strategies: applications with responsiveness requirements use an earliest-deadline-first (EDF) scheduler and throughput-oriented applications use a round-robin (GRR) scheduler.

## Performance and Energy Measurement

Applications report their own measured response times to PACORA through the heartbeat interface presented in Chapter 5, and PACORA uses this information to build response-time functions (RTFs) offline or online. Our offline modeling uses CVX [68] in MATLAB [98] to build the models. Online we use our Non-Negative Least Squares (NNLS) implementation described in Chapter 5. The online models are identical to our offline models for the same inputs, so the method used has no effect on resource-allocation decisions.

We also use the same heartbeat information to show if the application is making its deadlines for the experiments. Tessellation enables PACORA to directly measure the system energy. However, energy counters are not available on our Nehalem-EP system and thus we extend the power model from the Sandy Bridge system to function as our application 0 RTF.

## Description of Workloads

Our video conference scenario has three types of applications: a video application, a network bandwidth hog, and a file indexer.

Our streaming video application is a multi-threaded, CPU- and network-intensive workload intended to simulate multiparty video-chat applications like Google Hangout or Microsoft Lync. The application has a separate, performance-guaranteed incoming video stream for each participant and adjusts video sizes based on which person is speaking. The speaker's video is larger and has higher resolution than the other video streams, and as the speaker

Figure 6.1: Screenshot of our video-chat scenario with all small videos (right) and one large video (right).

changes, the requirements for the video streams change. Figure 6.1 shows screenshots of our video application running.

Our application can have up to nine incoming video streams each handled by a thread in our video cell. In the cell, each video stream has an EDF-scheduled thread with 33 ms deadlines. Tessellation provides separate network bandwidth allocation to each video thread, and the videos share their core allocations using the EDF scheduler.

In our experiments, videos are resized using a keyboard command. Small videos require roughly 90 kbit/s of network bandwidth while large videos require 275 kbit/s of network bandwidth. We use Big Buck Bunny [18] for all videos. Each video stream is encoded offline in the H.264 format using libx264, transported across the network through a TCP connection from a Linux Xeon E5-based server, and decoded and displayed by the Tessellation client. The client receives, decodes, and displays each frame using libffmpeg and libx264.

Our network bandwidth hog application is designed to represent an application such as Google Drive or Dropbox uploading files to the cloud in the background during the video conference. The bandwidth hog is a simple, single-threaded application that transmits data at the fastest possible rate. The hog contends with the video player for bandwidth by constantly sending UDP messages to the Linux server.

Finally, we use psearchy [91], a parallel text indexer, from MOSBENCH [26] to represent compute-intensive tasks such as virus scans or file indexing, which could be executing in the background. Psearchy was designed to index and query Web pages, but instead we index the Linux 2.4.0 source code. It runs on top of a pthread-compatible runtime system implemented in Pulse.

## 6.3 Resource Allocation Experiments

Now we demonstrate how PACORA can be used to efficiently allocate resources for different parts of the video conference scenario. This section demonstrates using PACORA as the overall resource allocation system, dividing resources between the incoming video streams, a file indexer, and outgoing network data. We assign a moderate penalty (10.0) for missing the deadline to small videos, and a significant penalty (50.0) for the large videos. We assign a small penalty for the network hog (5.0) and a very small penalty to the file indexer (0.1), with no deadlines for either.

Our first experiment uses offline modeling and assumes that system is running on wall power and thus sets application 0's penalty slope to 0. Figure 6.2 shows the results. All network allocations were initially set to 120 kbits/s, and as shown in plots (a) and (b). The first adaptation event occurs at t= 2 s, when PACORA changes the allocations from their initial settings to application-specific allocations. PACORA changes all of the video threads to 96 kbits/s, just above the required network bandwidth for small videos. PACORA removes all bandwidth from the file indexer since it does not use network bandwidth and gives the remaining bandwidth in the system to the network hog. Plot (d) shows that PACORA removes cores from the video cell and gives them to the file indexer.

Additional resizing events occur at 25, 35, 52, 58 and 65 seconds, when videos 1, 2, 3, and 4 change size. As shown in plots (a) and (b) , PACORA reduces the network hog's allocation in order to give sufficient bandwidth to the large video. However, when all the videos are small the bandwidth is returned to the network hog. We can see in plot (f) that PACORA allocates 99.6% of the total bandwidth on average throughout the experiment with a standard deviation of 0.5%.

Plot (d) shows that larger videos do not need enough additional processing power to require an increase in cores, so the core allocations do not change after the initial adaptation. Plot (c) shows that the videos do not drop below the required frame rate except when resizing. These glitches while resizing are an artifact of the application implementation.

Plot (e) shows the time for PACORA to run it's resource-allocation algorithm. The average runtime is 285 µs, but optimizations where the allocations change significantly can take as long as 1.4 ms.

We believe these results show the potential of PACORA. Running in a real operating system, PACORA is able to dynamically allocate resources to a mix of realtime and high-throughput applications without missing any deadlines. PACORA takes less than 1.4 ms to calculate new allocations, and allocates 99% of the resources.

Figure 6.3 shows results for the same experiment except now that we have changed application 0's penalty slope to 10 to represent that the system is running on battery power and saving energy is now important. We can see that the allocations differ slightly for those in Figure 6.2—most likely as a result of some inaccuracies in our power model that was built for a different machine and our certain stopping criteria. All allocations stay at their initial allocations until video 2 becomes large at 19 seconds. We can see that PACORA takes a few steps to increase the network allocation for video 2 above threshold. When video 2 becomes

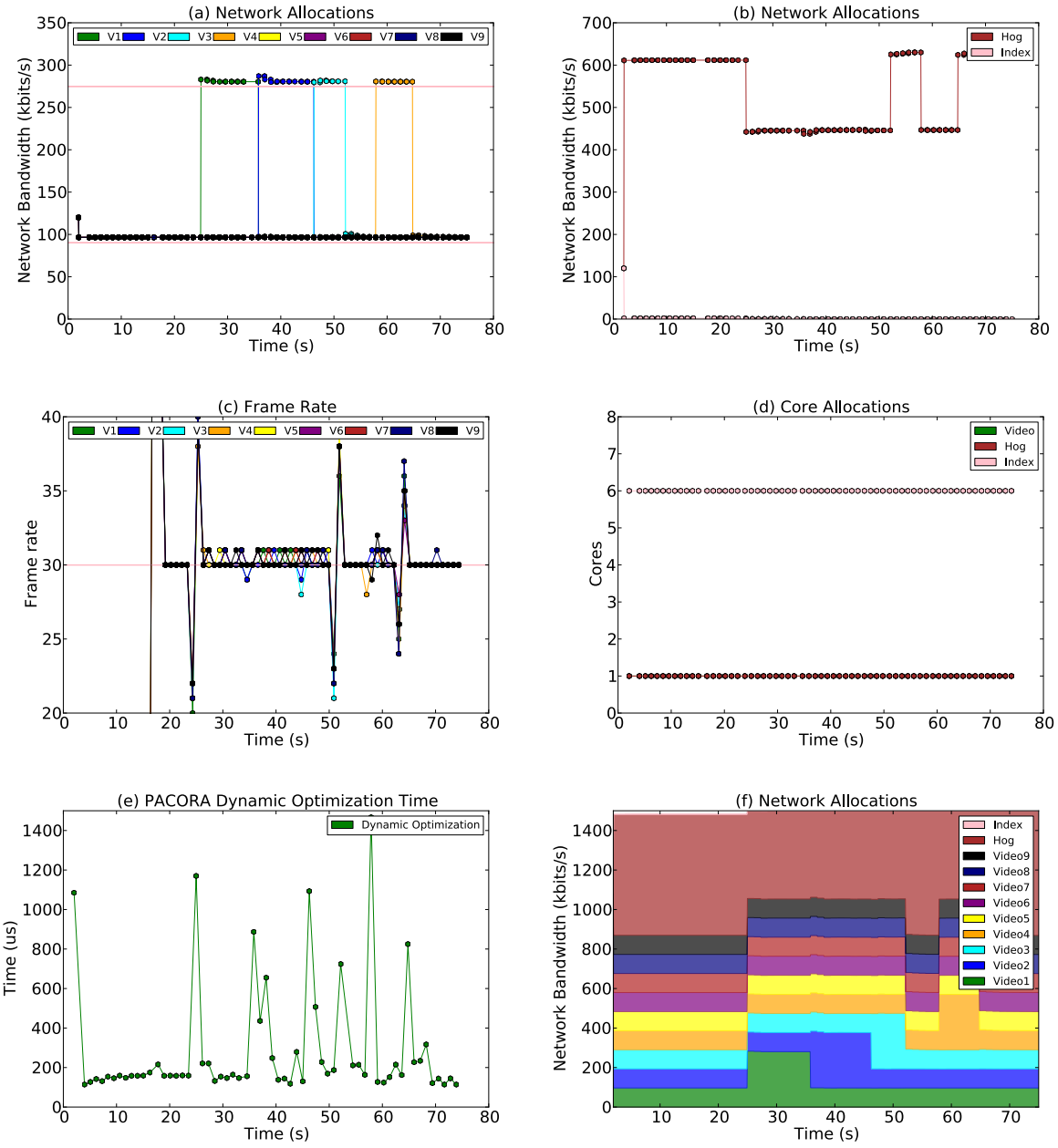


Figure 6.2: Allocation results for video conference with 9 videos, a bandwidth hog, and a file indexer with wall power and offline modeling. Periodically, one of the videos becomes large causing the allocations to change. Plot (a) shows the network bandwidth allocations for the nine video threads. The two red lines represent the required network bandwidth for a large and small video. Plot (b) shows the network bandwidth allocations for the bandwidth hog and the file indexer. Plot (c) shows the measured frame rate for the video threads. The red line represents the desired frame rate of 30 frames per second. Plot (d) shows the core allocations for the video cell, bandwidth hog, and file indexer. Plot (e) shows the time to run PACORA's resource allocation algorithm. Plot (f) shows the network allocations in plots (a) and (b) stacked.

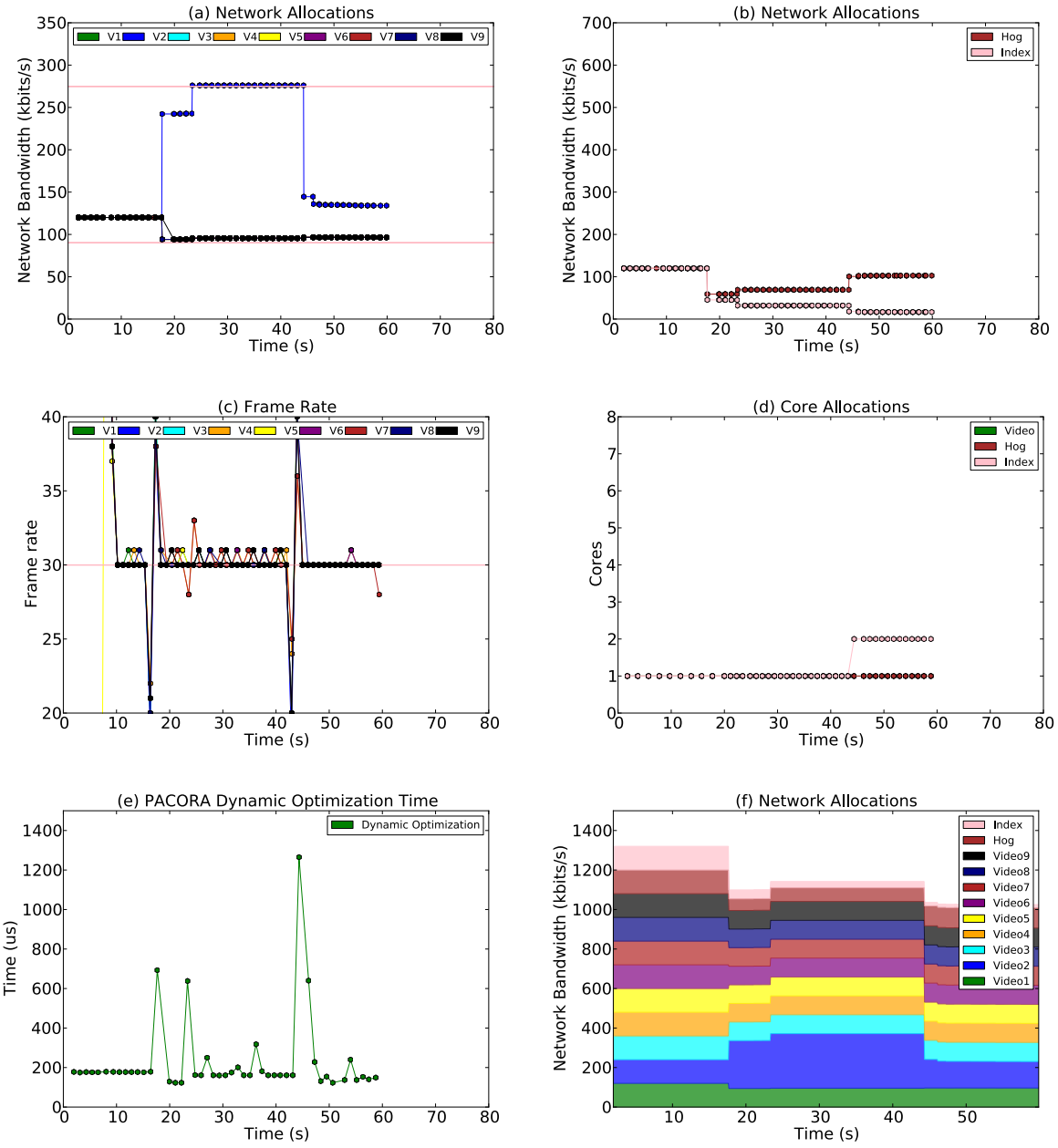


Figure 6.3: Allocation results for video conference with 9 videos, a bandwidth hog, and a file indexer with battery power and offline modeling. Periodically, one of the videos becomes large causing the allocations to change. Plot (a) shows the network bandwidth allocations for the nine video threads. The two red lines represent the required network bandwidth for a large and small video. Plot (b) shows the network bandwidth allocations for the bandwidth hog and the file indexer. Plot (c) shows the measured frame rate for the video threads. The red line represents the desired frame rate of 30 frames per second. Plot (d) shows the core allocations for the video cell, bandwidth hog, and file indexer. Plot (e) shows the time to run PACORA's resource allocation algorithm. Plot (f) shows the network allocations in plots (a) and (b) stacked.

small again at 42 seconds, the video allocation says slightly above the lowest possible, and index is given an extra core. However, despite these minor quirks, which we believe can be fixed with a better power model and more experimentation with stopping criteria, the results in plot (f) look promising. PACORA now only allocates 76.9% of the total available network allocation, leaving the rest idle.

## 6.4 Summary

In this chapter, we demonstrated PACORA's ability to allocate resources in a manycore OS for a video conference scenario. We believe the results are a good proof-of-concept for PACORA; however, the next step for future work will be to experiment with a larger range of applications.

# Chapter 7

# Discussion

In this chapter, we discuss some of the potential challenges that could arise when deploying PACORA in a real system with real applications and present techniques we have considered to address these challenges. PACORA's challenges can be broadly categorized into two types: performance non-convexity and performance variability. The main concern with performance non-convexity and variability is their effects on the accuracy of the response-time functions (RTFs). In the following sections, we describe the different sources of non-convexity and variability and how to cope with them to reduce their effect on model accuracy. However, as shown in the experiments in Chapter 4, we have found that model accuracy has less impact on the quality of resource allocation decisions than we anticipated. As a result, we feel that many of the challenges discussed in this chapter may arise more often in theory than in practice.

## 7.1 Performance Non-Convexity

Since the RTF models are convex, non-convex application performance can be a challenge for PACORA. Generally, non-convex behavior can occur in two ways: outliers and quasiconvex response times. In this section, we describe these problems in more detail, present examples we have observed in our studies, and discuss potential ways mitigate their effects.

### Outliers

Outliers are particular resource allocations whose response times are significantly different (typically much worse) than the general surface of the response time of the application. These are often a result of interference between different interacting systems in modern hardware and operating systems. For example, (as presented in Chapter 3) we have seen outliers in applications when dealing with hyperthreads (Figure 3.4 and `stencilprobe` in Figure 7.1), which are likely the result of prefetching or other data management failures due to the mismatched execution rates of threads. (Figure 3.5). Many applications also have

outliers for extremely small allocations of particular resources (*i.e.,* one cache way, as shown in `blackscholes` Figure 7.1). While outliers will likely always be a reality in real systems, as responsiveness, predictability, and efficiency increase in importance we expect to see an increased number of chip designs that provide more performance convexity and reduce the total number of outliers.

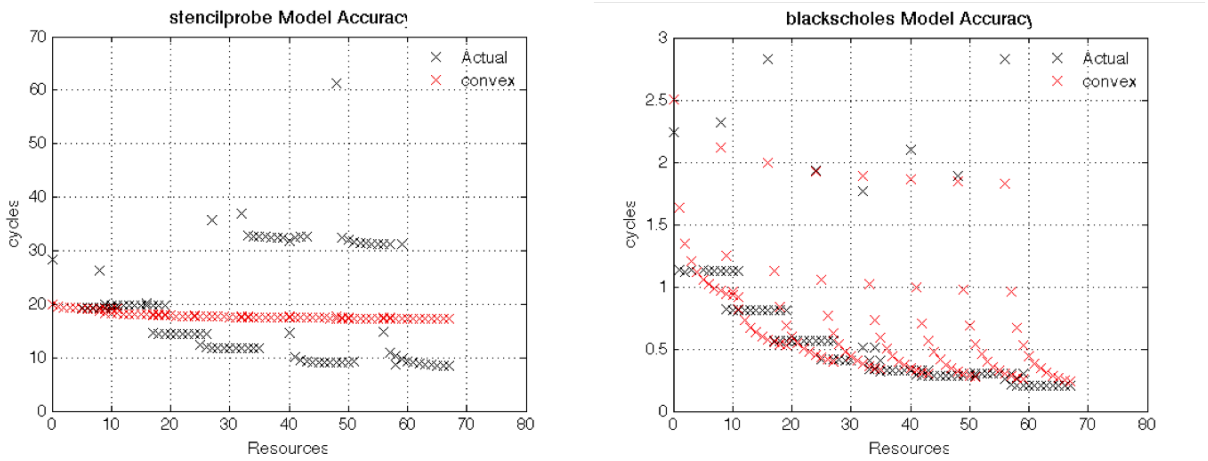


Figure 7.1: Actual measured response times (black X) and the predicted response times (red X) for the `stencilprobe` and `blackscholes` benchmarks. Each point represents a prediction for particular allocation, and points are ordered along the x-axis by increasing resource amounts (clusters count up 1 core, 2 cores, etc. and within a cluster cache ways increase 1-12). Y-axis plots predicted or measured runtime in cycles.

Two potential problems arise from outliers. First, they can distort the accuracy of the model for other resource allocations. Our model construction optimization (described in Chapter 5) tries to minimize total error and since outliers can often be very far away from the other points they tend to pull the model towards them. Figure 7.1, which was produced from the experiments in Chapter 4, demonstrates this effect. The typical result of outliers is that PACORA will have an overly pessimistic view of response times and likely over allocate. To alleviate this problem, outliers should be thrown out during the model creation phase. Chapter 5 describes our implementation's scheme for identifying and removing outliers.

The second issue occurs when PACORA's resource-allocation optimization unknowingly selects an outlier allocation to give the application. In this case, the actual performance will be significantly worse than expected—possibly increasing total penalty and violating application SLOs. To prevent this, we propose an approach where PACORA keeps track of the points with extreme error in the model and uses heuristics to adjust the resource allocations coming out of the optimization to avoid such points. Allocations could be reduced or increased slightly to move off the outlier point, and the RTF model could be used as an oracle to determine the efficacy of either approach. Increasing allocations would require either retaining some slack resources in the system or removing resources from another application.

We did not implement any of these heuristics in our current evaluation as outlier points were so rarely selected, but we do plan to experiment with them in future work.

## Quasiconvex Response Time Functions

The other potential form of non-convex behavior is where the basic shape of the response-time function is not actually convex, as opposed to just a few outlier points that violate convexity. Since applications usually follow the "Law of Diminishing Returns" for resource allocations, the only realistic example of this behavior is performance "plateaus" (Figure 3.6). Such plateaus can be caused by adaptations within the application, such as adjusting the algorithm or output quality, or certain resources that only provide performance improvements in increments rather than smoothly. For example, a video player may choose to increase resolution having received an increase in network bandwidth and thus the system may not measure an improvement in frame rate.

In these applications, the response time is really the *minimum* of several convex functions depending on allocation, and the point-wise minimum that the application implements fails to preserve convexity. The effect of the plateaus will be a non-convex penalty as shown in Figure 3.7 and multiple extrema in the optimization problem will be a likely result.

There are a few potential ways to avoid this problem. One is based on the observation that such response-time functions will at least be *quasiconvex*. A function $f$ is quasiconvex if all of its *sublevel sets* $S_\ell = \{x | f(x) \leq \ell\}$ are convex sets. Alternatively, $f$ is quasiconvex if its domain is convex and

$$f(\theta x + (1-\theta)y) \leq \max(f(x), f(y)), 0 \leq \theta \leq 1$$

Quasiconvex optimization can be performed by selecting a threshold $\ell$ and replacing the objective function with a convex constraint function whose sublevel set $S_\ell$ is the same as that of $f$. Next, the algorithm determines whether there is a feasible solution for that particular threshold $\ell$. Using repeated application of this technique while performing a binary search on $\ell$ should reduce the level of feasibility until the solution is approximated reasonably well.

An alternative approach is to use additional constraints to explore convex sub-domains of $\tau$. For example, the affine constraint $a_{p,r} - \mu \leq 0$ excludes application $p$ from any assignment of resource $r$ exceeding $\mu$. Similarly, $\mu - a_{p,r} \leq 0$ excludes the opposite possibility. A binary (or even linear) search of such sub-domains could be used to find the optimal value.

In practice, we did not observe plateaus[1] because modern hardware is fairly effective at gracefully degrading performance as a function of resources[2] and most applications do not frequently adapt to their resources. Since both approaches add significant computational cost, we chose not to use either in our PACORA implementation.

[1] except in our synthetic microbenchmarks running on simulated hardware (Figure 3.2)

[2] Similar results were found in the experiments in [38]

## 7.2 Variability

Variability is when an application does not consistently have the same performance for a given allocation. Large variability can make it difficult to create an accurate model since a single predicted response time value for an allocation may not convey how likely an application is to achieve that response time. For example, Figures 7.2a and 7.2b show the recorded frame rate for an n-bodies application running on Windows 7 over 100 frames with different memory page and core allocations. Figure 7.2a has an allocation of 5 cores and 2500 memory pages while Figure 7.2b has significantly more cores at 15 but only 550 memory pages. The two figures have very similar average frame rates of 36.6 and 37.0 frames/second respectively. However, if the application quality-of-service requirement was 30 frames/second, despite having the higher average frame rate, the allocation in Figure 7.2b's would miss 15% of the deadlines while the allocation in Figure 7.2a misses no deadlines.

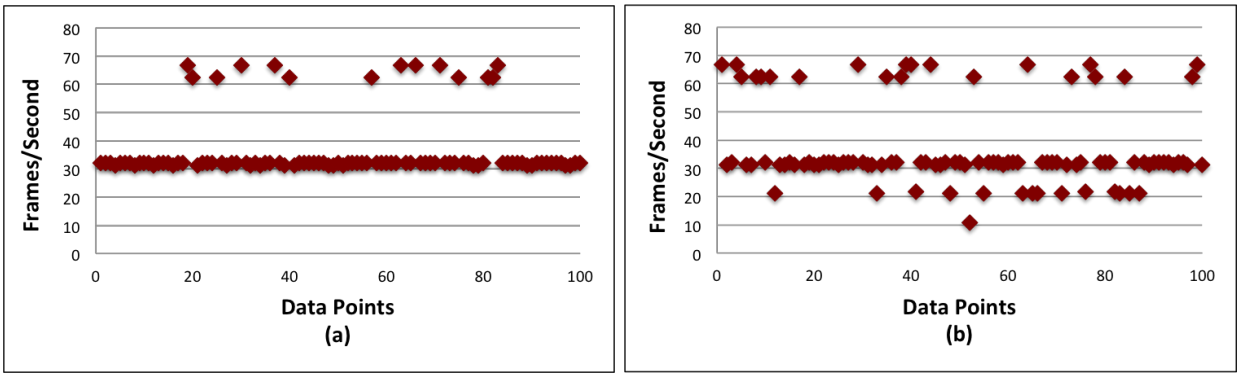


Figure 7.2: Actual measured frame rate for an n-bodies application when allocated (a) 5 cores and 2500 memory pages and (b) 15 cores and 550 memory pages. Each point represents frames/second achieved by the application.

### Techniques to Address Variability

Variability in application performance typically has three potential sources: phase changes, performance changes due to differing inputs, and variable resource performance due to external causes or interference from sharing. Depending on the source of the variability and magnitude of the variability, we have developed three primary ways to address it. The first approach is to rapidly adapt the models online, the second is to build more than one model per application, and the third is to use stochastic models. In this section, we first describe each of the techniques and then discuss how they can be applied to the different sources of variability. We imagine the techniques can be used independently or in combination to address the challenge of variability.

### Online Modeling

PACORA's online modeling, which was presented in Chapter 5, can be used to rapidly adapt models to the current state of the machine and application. Online modeling has the advantage that it can react to performance situations that PACORA has not seen before whereas offline modeling requires all of the potentially variability to be observed in advance during the training phase. The downside of online modeling is that it requires several samples before the new results begin to affect the resource allocations.

### Multiple Models

An orthogonal technique is to build multiple models for the application and change the model as appropriate. For example, in the experiments in Chapter 6 the video application has two distinct operating modes: large video and small video. PACORA could build one model for each operating mode and then change the model in use as the operating mode changes. This approach has the advantage over pure online modeling, which tries to rapidly adapt the existing model to changes in applications, that it may not need many samples to adapt to a change in operating mode. However, it does require additional overhead to maintain and store multiple models. Furthermore, it requires identifying different operating modes in an application in order to know when to build new models and when to change models. With the help of the application identifying different modes could be feasible; however, without the application input, identifying changes such as phases is still an active area of research [41].

### Stochastic Models

The most natural and potentially simplest way to address the problem demonstrated in Figures 7.2a and 7.2b to use stochastic models in PACORA. In our offline modeling experiments, we used the average value measured to build the model, and our online modeling approach uses the most recent value (unless it is believed to be an outlier). Alternatively, we could choose to select values for modeling building that provide the higher degree of confidence required to meet the QoS requirements of the applications.

The lowest cost approach to this would be to maintain the mean and standard deviation for the application and use a Chebyshev bound to adjust the response times accordingly. Random fluctuations will be reflected in the runtime measurements t and the residual error $\epsilon = t - \sigma$. The sample mean and standard deviation of the error can bound the probability that the actual runtime will exceed the prediction $\tau$. Using this information, a Chebyshev bound on the error of the form

$$Pr\left\{\left|\frac{(\epsilon - \overline{\epsilon})}{\sigma_\epsilon}\right| > k\right\} \leq \frac{1}{k^2} \tag{7.1}$$

can guarantee the probability that a particular runtime requirement is met. For example, if an application needs a probability 0.99, and has given an error sample mean of 3 microseconds (meaning $\tau$ underestimates t by that much) and a sample standard deviation

of 2 microseconds, then for a predicted runtime of $\tau$ of 27 microseconds we need to solve $\tau + 3 <= x - k\sigma\epsilon$ for $x$ to determine which $x$ to use as the model input to guarantee that $t = \tau + \epsilon$ exceeds $x$ microseconds only 1% of the time. In this example, $x$ should be 50 microseconds. The downside of Chebyshev is that it can be a very loose bound and thus significantly over-estimate resources.

For a tighter bound, we could instead use quantile regression [80] and select the value at the appropriate quantile as input to the model. For example, we could use the 99th percentile value for a particular allocation. However, quantile regression requires significantly more samples than the Chebyshev approach, particularly for high quantiles.

Stochastic models would be straightforward to use in our current PACORA implementation since they only require preprocessing the data that is sent to the model creation optimization. However, stochastic models really require online modeling because it would be extremely difficult to capture all the variability in advance. Depending on the metric chosen and the variability of the application, they may require significantly more samples and would likely necessitate a lower reallocation frequencies in order to collect the appropriate number of samples at any given allocation. Additionally, they will result in larger resource allocations, but this can be viewed not a cost specific to stochastic models, but the general cost of guaranteeing performance in a noisy environment.

## Sources of Variability

We have identified three primary sources of variability in applications: phases, input dependence, and variable resources. In this section, we discuss how we see the techniques described above being applied to address variability from each of these sources.

### Phases

Application phases can be handled with online modeling or multiple models; both of which were demonstrated in the experiments in Chapter 6. One concern with using the multiple models approach more generally with phases is that phase detection is an active area of research[3]. However, if the application can be modified to signal phase changes—as was the case in our video application—then this becomes less of a concern.

Another possible approach is to build a model that represents the resource requirements of the most demanding phase. The system can be designed to make use of the idle resources when available or power management mechanisms can put them in low-power mode to reduce their energy overhead.

### Input Dependence

Some applications may significantly change performance as a function of their inputs. In the case of our video application, we ignored its input dependence without significant effect.

[3] [41] provides an overview of techniques.

However, for other applications the effect may be more pronounced. If the input dependency is coarse-grained, for example, it only changes at the start of the application, a solution might be to keep multiple models for the application and select one based on the current input. This approach assumes that it is possible to identify the input and cluster it with other inputs that produce similar performance effects. Online modeling is also a reasonable solution for coarse-grain input dependencies.

For fine-grain input dependencies, such as the performance changing as a function of the frame that needs to be rendered, then ignoring the variability but adding a little slack is a reasonable solution. Alternatively, fine-grained input dependencies are a natural fit for stochastic models.

### Resource Variability

Resource variability arising from non-deterministic and shared resources was the most common form of variability observed in our studies. As Figure 4.5 shows, the average variability per allocation per application is 9%, and applications like `tradebeans` that use the network connection or other non-deterministic resources have an even higher variability. Stochastic models are most likely the right way to deal with non-deterministic resources and may be particularly necessary for representing disk-based storage in warehouse-scale computing.

Shared resources can also be handled with stochastic models. However, since the models can be built online while other applications are running, the interference from a loaded machine is already captured in the model. In our evaluation in Chapter 4, we built the models in isolation but found that PACORA was still able to make near-optimal resource allocations for a loaded machine despite the shared resources. These results are supported by the data in Figure 7.3. In the experiments, bandwidth is the primary shared resource between applications. In Figure 7.3 we have run each application with a bandwidth hog, `stream_uncached`, and presented the slowdown over the application executing on the machine alone with the same resource allocation. `stream_uncached` is really a worst case test for bandwidth sensitivity and still most of the applications experience little or no slow down. While there will likely always be some shared resources, we expect results like this to be the norm in the future. There appears to be an trend towards minimizing interference in emerging chip designs[4], as efficiency and predictability begin to trump utilization as primary concerns. Alternately, shared resources could be turned into PACORA-controlled resources by adding hardware or software QoS mechanisms to them.

The final source of resource variability comes from resources that can dynamically vary their performance. Dynamic frequency scaling in processors is the most common example of this in modern systems. The best way to handle frequency scaling in PACORA is still open research; however, we have imagined a few possible alternatives. The simplest approach would be to ignore it or assume that online modeling is sufficient. Alternatively, we could view higher frequencies as an efficiency optimization (like Turbo Mode) and build models

[4]This is particularly relevant for cloud providers who are often hosting VMs from different customers on the same machines.

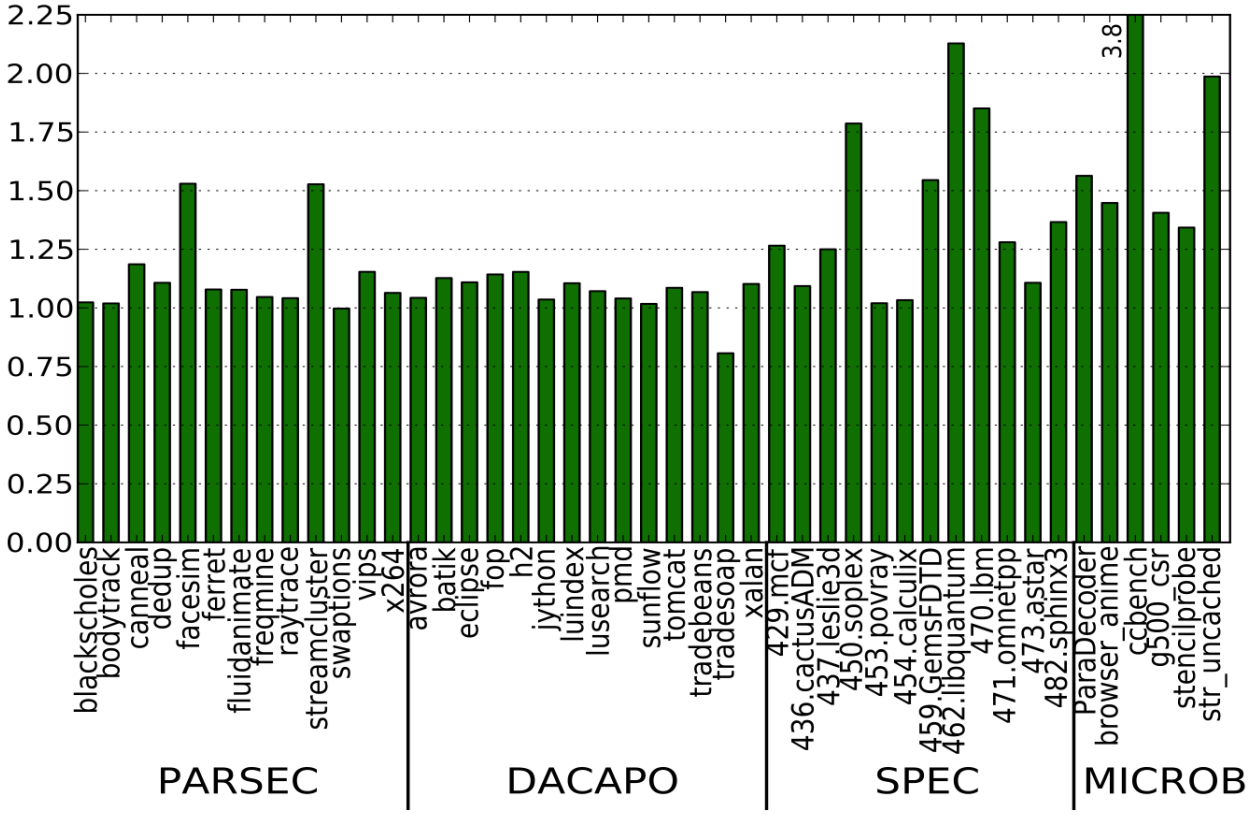


Figure 7.3: Application performance when run with a bandwidth hog, `stream_uncached`, normalized to running on the machine alone with the same resource allocation.

based on lower frequencies. If the cores happen to run faster than expected, the system can reclaim or power down idle resources. Stochastic models would also be a reasonable approach to represent variability due to frequency changes. Opportunities in this area are discussed further in Chapter 8.

Another set of techniques involves explicitly embracing frequency changes in the models rather than simply viewing them as noise. We could use the multiple models approach and have a different model for each frequency range. This assumes that all the cores of an application are running at the same frequency, which is a reasonable assumption for current single-node hardware, but may not be for future hardware and clusters. We could also consider treating frequency as a resource dimension; again if all the cores of a single type are running at the same frequency. Another approach would be to consider cores running at different frequencies as different types of resources (like a heterogenous system) with different energy costs and use PACORA to select the lowest frequencies possible.

## 7.3 Summary

In this chapter, we discussed some of the potential challenges for PACORA in a real system, namely non-convex behavior and performance variability. We described the different sources of these challenges and presented various technique to potentially reduce their effect on model accuracy. Given the limited value of extremely accurate models, PACORA's design is likely good enough for many scenarios that appear in practice today. However, more experimentation is needed to test this hypothesis.

# Chapter 8

# Conclusion and Future Work

In this chapter, we give our closing thoughts on the PACORA research presented in this thesis and discuss future work and other possible extensions to the research.

## 8.1 Concluding Thoughts

In this thesis, we have presented PACORA, which is a framework designed to determine the proper amount of each resource type to give each application. PACORA takes a different approach to resource allocation than traditional systems, relying heavily on application-specific functions built through measurement and convex optimization. By building application-specific functions online and formulating resource allocation as an optimization problem, PACORA is able to accomplish multi-dimensional resource allocation on a general set of resources, thereby handling heterogeneity and the growing diversity of modern hardware while protecting application developers from needing to understand resources. Using convex optimization lets PACORA perform real-time resource allocation inexpensively, enabling PACORA to dynamically allocate resources to adjust to the changing state of the the system.

We constructed the PACORA prototype in Tessellation from the ground up, providing a "clean-room" view of the embodied ideas. We feel our initial implementation of PACORA in the Tessellation OS shows real promise as a proof-of-concept. PACORA is able to make decisions in microseconds and only requires a few hundred bytes of additional storage per application, which makes PACORA's overhead negligible for most systems. For the small allocation decisions studied in Chapter 4, allocations were near optimal—only 2% from the best possible allocation on average. When making larger allocations decisions in Chapter 6, we found that PACORA was able to allocate resources very efficiently to provide QoS to the applications with deadlines, while still providing significant throughput for background applications. We feel that PACORA can be applied in many settings in addition to client operating systems such as Tessellation: it could adaptively and continuously right-size the multiple virtual machines in a corporate server, deliver real-time responsiveness in an embedded system, or adjust resources to meet SLAs in an implementation of a cloud service.

However, that's not to say that PACORA is completely production ready. As discussed in Chapter 7, in practice, variability could potentially provide a challenge to using PACORA. Of course, the primary concern with variability is it's potential to affect model accuracy, and in our studies, we have found that the impact of model accuracy on the quality of resource decisions is not significant. Models with errors above 20% still produce near optimal allocations, and so it's quite possible that variability will be less impactful than originally expected. Particularly since PACORA can always over-provision slightly to make guarantees and still be more effective than most of the current resource allocation practices. Therefore, we believe PACORA or other modeling-based approaches to be feasible in real systems with noisy applications.

Additionally, we simply need more experiments with additional applications and resources to show that PACORA can cover the wide variety of situations that it may encounter in a production system. While we tried to work with a representative variety of benchmarks and applications in our studies, we were primarily limited to applications that could be ported to Tessellation OS; that is applications that rely on few libraries and have source code available. We do not expect most applications to provide any additional challenges to PACORA; however, it will be necessary to explore the corner cases to be confident enough to deploy it. In the following section, we discuss the directions we are taking PACORA to test it further and increase the number of situations it can handle.

## 8.2 Future Work

While there are many possible directions to explore for future PACORA research, we found the following few to be the most promising based on immediate needs of potential users and have begun to work on these ideas, which we describe below.

### PACORA in the Cloud

While a client OS was an interesting system for a proof-of-concept because it required extremely low overheads and fast reallocation, PACORA is more likely to be deployed in cloud systems. Unlike client OSs, which do not separate resource allocation from scheduling, many cloud systems are already used to dividing resources among competing applications that perform their own scheduling. As a result, all of the resource-control mechanisms required by PACORA are already available. We've also found that the measurement mechanisms are often already there as well because they are used to communicate SLA-relevant metrics. However, a major factor which makes PACORA more appealing in the cloud is not the availability of mechanisms, but an appropriate cost structure. As more players have entered the market, the cloud is starting to look like a commodity where it's difficult for companies to differentiate their products with more than price. Therefore, providers are looking for solutions to make their cloud offerings less expensive; one option is to use something like PACORA to increase resource utilization without violating SLAs.

We're exploring several different levels for deploying PACORA in the cloud. The first is close to the original client OS implementation; the idea is to use PACORA to consolidate virtual machines on a node. PACORA could be used as an oracle to find VM combinations that have low total penalties or it could divide the hardware resources on the machine among VMs. Alternatively, PACORA could calculate the number of underutilized resources on a machine that could be claimed for background computation without violating SLAs of the applications. We could also imagine performing the same types of decisions at the rack level rather than for a single machine. Finally, rather than requiring applications to specify resource requirements, PACORA could be used to determine the correct number of nodes, storage, and bandwidth to give an application. This application of PACORA also affords some potentially interesting opportunities to explore alternative business models for the cloud. For example, a provider could simply sell a performance guarantee to a customer and then use whatever resources necessary to meet it or the provider could just bill a customer for the resources they actually used rather than the resources they requested.

We believe PACORA has significant potential in the cloud because it is a natural fit in the current system architectures and economic ecosystem and because the problem dimensions are much larger (*i.e.,* more resources and more applications), which makes the problems more difficult. As a result, the heuristics used in practice are often further from the true optimal than in client systems, which affords an opportunity to PACORA to help bridge that potentially significant gap.

## Heterogeneity

We are also exploring the potential of PACORA to help determine the right computational resources from a heterogeneous set for collection of tasks. In theory, heterogeneity is naturally handled by PACORA. Each core type can be viewed as a different resource system, so fat cores may be resource 1, thin cores may be resource 2, and GPUs could be resource 3. The application developer would not need to specify which core types the application uses best; PACORA would try allocating the cores and the resulting performance would be captured in the response-time function (RTFs). So, for example, if an application did not use a GPU then this would be discovered empirically and the RTF would show no performance improvement along the GPU dimension. We believe this would be particularly powerful when combined with a system such as Dandelion [121] that can automatically compile and schedule applications on different core types.

## PACORA with Dependencies

We are also working to reformulate PACORA to handle dependencies between applications. One of the assumptions baked into PACORA's formulation is the idea that all applications are independent, so allocating resources to or removing resources from one application does not impact the performance of other applications. While this is a reasonable assumption in many scenarios, we have found compelling cases where it would be nice to express the

dependencies in the optimization. One such example is services that support applications. In the video experiment in Chapter 6, we first sized the network service appropriately to have enough capacity to accommodate all the applications and then left PACORA to allocate the service capacity and remaining resources between the applications. While this is certainly a reasonable approach, particularly when the number of services is small, it would be better to represent the dependency in the optimization formulation. PACORA would then be able to trade-off giving resources to applications or the services that support them depending on their deadlines and relative importance.

This idea can be extended further to a general hierarchical resource allocation formulation. A hierarchical formulation would let PACORA allocate resources for pipelined or graph computations such as those created by Dandelion [121] and Naiad [102].

## 8.3 Other Possible PACORA Extensions and Improvements

In this section, we describe possible extensions to the PACORA work that we believe would be interesting and could provide meaningful additions to the PACORA system. Many of these are still open research problems in other domains whose solutions would benefit more than just PACORA.

### Measurement

Our PACORA implementation requires modifying applications to measure their heartbeats. However, we believe this process could be automated. One option would be for the compiler to automatically insert hints around the critical section. Alternatively, we could take advantage of the second-level scheduler's close relationship with the application to see if there is a way for the scheduler to infer the response time. A less invasive approach could be for the system to try to infer performance information by measuring the application at its resource-container boundaries and/or using performance counters.

### Application Quality

Exploring the relationship between resource allocation and applications that can adjust quality seems extremely relevant. PACORA only considers the response time of an application as affecting the user experience (system penalty), but if applications can adjust their quality than that is another component that impacts the user experience. For example, consider a video application that naturally adjusts its resolution to guarantee that it always makes its frame rate. The current implementation of PACORA would view this application as being resource insensitive and give it the smallest amount of resources; however, the user watching the low-resolution video may not consider that as successfully minimizing penalty. Experimenting with the interface for quality information between PACORA and applications to an-

swer questions like "should quality be an explicit parameter in the PACORA optimization?" or "is there a negotiation process between PACORA and applications regarding quality adjustments?" could be very interesting. We think this area becomes particularly intriguing when considered in conjugation with some of the recent work on approximate computing that automatically adjusts application quality using techniques like loop perforation [126, 165].

## Prediction

We believe there significant potential to use predictions from machine learning algorithms as inputs into the resource-allocation optimization. For example, we could use reinforcement learning or explore-exploit techniques to infer the penalty or deadlines for applications. These values could even be personalized and adjusted dynamically. For example, after a page is loaded we may potentially have some time until the user clicks again. If we could predict when the click will occur, we could deprioritize the interactive application to complete more background work without hurting the user experience.

Recent work on predicting application load order and timing, such as SuperFetch in Windows [134] or Falcon on mobile devices [154], could be used to provide hints to the resource-allocation system on whether a proposed reallocation will be long lived enough to justify the cost, if resources could be powered down when nothing is likely to happen, or if resources should be preallocated for a high-priority application with little slack expect to start soon.

## PACORA as an Oracle

A possible extension of PACORA that we have discussed with hardware designers and cloud providers alike is the idea of using PACORA as an oracle to help the system make other types of decisions. For example, PACORA's RTF and penalty functions could be used provide power management hardware hints about runtime slack available for a given application. This information could then be used to decide when to slow down or power down particular resources to save energy or redirect thermal capacity to other computations. PACORA resource allocation optimization runs so quickly it could be used to select which applications to run together. The idea would be to query it with combinations of applications to determine the penalty of running these together. Exploring different subsets of applications with PACORA would enable the system to determine which applications are best paired. This mode could be particularly useful for cloud providers when deciding whether it is safe to co-locate VMs on a machine.

## Alternative Resource Representations in RTFs

Another interesting area to explore is alternative resource representations outside of those in the initial PACORA exploration study. One option we've considered is to use *bandwidth*

*amplification factors* from H. T. Kung [85]'s work to turn all resources into bandwidths and represent the relationship between resources. For example, response time due to memory accesses might be approximated by a combination of memory bandwidth allocation $b_{r1}$ and cache allocation $m_{r2}$. Here we denote an allocation of a bandwidth resource by $b_r$ and of a memory resource by $m_r$.

In this case, memory resources, such as the cache, permit exploitation of temporal locality and thereby *amplify* associated bandwidths. For example, additional main memory may reduce the need for storage or network bandwidth, and of course, increased cache capacity may reduce the need for memory bandwidth.

Kung developed developed tight asymptotic bounds on the bandwidth amplification factor $\alpha(m)$ resulting from a quantity of memory $m$ acting as cache for a variety of computations. He shows that

$$\begin{array}{lll} \alpha(m) & = \Theta(\sqrt{m}) & \text{for dense linear algebra solvers} \\ & = \Theta(m^{1/d}) & \text{for d-dimensional PDE solvers} \\ & = \Theta(\log m) & \text{for comparison sorting and FFTs} \\ & = \Theta(1) & \text{when temporal locality is absent} \end{array}$$

A model could represent relationship between the two allocations with the geometric mean of in the denominator, *viz.* $w_{r1,r2}/\sqrt{b_{r1} \cdot m_{r2}}$, without compromising convexity. Each bandwidth amplification factor could then be described by one of the functions above and included also in the denominator of the appropriate component in the response time function model. For example, the storage response time component for the model of an out-of-core sort application might be the quantity of storage accesses divided by the product of the storage bandwidth allocation and $\log m$, the amplification function associated with sorting given a memory allocation of $m$. Amplification functions for each application might be learned from response time measurements by observing the effect of varying the associated memory resource while keeping the bandwidth allocation constant. Alternatively, redundant components, similar except for amplification function, could be included in the model to let the model fitting process decide among them.

## 8.4 Summary

In this chapter, we described our concluding thoughts on PACORA and presented future work for the project. We believe this thesis is a reasonable step towards proving the feasibility using of model-base resource allocation for real systems; however, more experimentation is needed to determine how to handle additional resources, applications and noisy systems in practice.

# Bibliography


[1] *Akaros*. http://akaros.cs.berkeley.edu.

[2] Benny Akesson, Kees Goossens, and Markus Ringhofer. "Predator: a predictable SDRAM memory controller". In: *Proc. of CODES+ISSS*. Salzburg, Austria, 2007, pp. 251–256.

[3] Luis Alvarez. "Design Optimization based on Genetic Programming". PhD thesis. University of Bradford, 2000.

[4] Gail Alverson et al. "Scheduling on the Tera MTA". In: *In Job Scheduling Strategies for Parallel Processing*. Springer-Verlag, 1995, pp. 19–44.

[5] Amazon.com. *EC2*. http://aws.amazon.com/ec2.

[6] Apache.org. *Hadoop Capacity Scheduler*. http://hadoop.apache.org/common/docs/r0.20.2/capacity_scheduler.html.

[7] Apache.org. *Hadoop Fair Scheduler*. http://hadoop.apache.org/common/docs/r0.20.2/fair_scheduler.html.

[8] Apple Inc. *iOS App Programming Guide*. http://developer.apple.com/library/ios/DOCUMENTATION/iPhone/Conceptual/iPhoneOSProgrammingGuide/iPhoneAppProgrammingG
pdf.

[9] Krste Asanović et al. "RAMP Gold: An FPGA-based Architecture Simulator for Multiprocessors". In: *Proc. of the 4th Workshop on Architectural Research Prototyping (WARP-2009)*. 2009.

[10] Francis R. Bach and Michael I. Jordan. "Kernel Independent Component Analysis". In: *J. Mach. Learn. Res.* 3 (2003), pp. 1–48. ISSN: 1532-4435.

[11] Luiz Andre Barroso. "Warehouse-Scale Computing: Entering the Teenage Decade". In: *Proceedings of the 38th Annual International Symposium on Computer Architecture*. ISCA '11. San Jose, California, USA: ACM, 2011, pp. –. ISBN: 978-1-4503-0472-6. URL: http://dl.acm.org/citation.cfm?id=2000064.2019527.

[12] Luiz André Barroso and Urs Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2009.

[13] Davide B. Bartolini et al. "AcOS: an Autonomic Management Layer Enhancing Commodity Operating Systems". In: *In DAC Workshop on Computing in Heterogeneous, Autonomous, 'N' Goal-oriented Environments (CHANGE), co-located with the Annual Design Automation Conference (DAC)*. San Francisco, California, 2012.

[14] Sanjoy K. Baruah, Johannes E. Gehrke, and C. Greg Plaxton. "Fast Scheduling of Periodic Tasks on Multiple Resources". In: *In Proceedings of the 9th International Parallel Processing Symposium*. 1995, pp. 280–288.

[15] Sanjoy K. Baruah et al. "Proportionate progress: A notion of fairness in resource allocation". In: *Algorithmica* 15 (1996), pp. 600–625.

[16] Stuart J. Bates, Jonathan Sienz, and Dean S. Langley. "Formulation of the Audze–Eglais uniform Latin hypercube design of experiments". In: *Adv. Eng. Softw.* 34.8 (2003), pp. 493–506. ISSN: 0965-9978.

[17] Christian Bienia et al. *The PARSEC Benchmark Suite: Characterization and The PARSEC Benchmark Suite: Characterization and Architectural Implications*. Tech. rep. TR-811-08. Princeton University, 2008.

[18] *Big Buck Bunny*. `http://www.bigbuckbunny.org/`.

[19] Sarah Bird. "Software Knows Best: A Case for Hardware Transparency and Measurability". MA thesis. EECS Department, University of California, Berkeley, 2010.

[20] Ramazan Bitirgen, Engin Ipek, and Jose F. Martinez. "Coordinated Management of Multiple Interacting Resources in Chip Multiprocessors: A Machine Learning Approach". In: *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO 41. Washington, DC, USA: IEEE Computer Society, 2008, pp. 318–329. ISBN: 978-1-4244-2836-6. DOI: `10.1109/MICRO.2008.4771801`. URL: `http://dx.doi.org/10.1109/MICRO.2008.4771801`.

[21] Stephen M. Blackburn et al. "The DaCapo Benchmarks: Java Benchmarking Development and Analysis". In: *OOPSLA*. 2006, pp. 169–190.

[22] Josep M. Blanquer and Banu Özden. "Fair queuing for aggregated multiple links". In: *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. SIGCOMM '01. San Diego, California, USA: ACM, 2001, pp. 189–197. ISBN: 1-58113-411-8. DOI: `10.1145/383059.383074`. URL: `http://doi.acm.org/10.1145/383059.383074`.

[23] Peter Bodik et al. "Automatic Exploration of Datacenter Performance Regimes". In: *Proc. ACDC*. 2009.

[24] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge, England: Cambridge University Press, 2004.

[25] Stephen Boyd et al. "Distributed optimization and statistical learning via the alternating direction method of multipliers". In: *Foundations and Trends in Machine Learning* 3.1 (2011), pp. 1–122.

[26] Silas Boyd-Wickizer et al. *MOSBENCH*. `http://pdos.csail.mit.edu/mosbench/`. 2012.

[27] John M. Calandrino and James H. Anderson. "On the Design and Implementation of a Cache-Aware Multicore Real-Time Scheduler". In: *ECRTS*. 2009, pp. 194–204.

[28] Jichuan Chang and Gurindar S. Sohi. "Cooperative cache partitioning for chip multiprocessors". In: *Proc. ICS '07*. Seattle, Washington: ACM, 2007, pp. 242–252. ISBN: 978-1-59593-768-1.

[29] Jeffrey S. Chase et al. "Managing Energy and Server Resources in Hosting Centers". In: *SIGOPS Oper. Syst. Rev.* 35.5 (Oct. 2001), pp. 103–116. ISSN: 0163-5980. DOI: `10.1145/502059.502045`. URL: `http://doi.acm.org/10.1145/502059.502045`.

[30] Donghui Chen and Robert J. Plemmons. *Nonnegativity Constraints in Numerical Analysis*. Lecture presented at the symposium to celebrate the 60th birthday of numerical analysis, Leuven, Belgium. 2007.

[31] Sangyeun Cho and Lei Jin. "Managing Distributed, Shared L2 Caches through OS-Level Page Allocation". In: *Proc. MICRO 39*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 455–468. ISBN: 0-7695-2732-9.

[32] Jike Chong et al. "Data- parallel large vocabulary continuous speech recognition on graphics processors". In: *Intl. Workshop on Emerging Applications and Manycore Architectures*. 2008.

[33] Juan Colmenares et al. "Resource Management in the Tessellation Manycore OS". In: *HotPar10*. Berkeley, CA, 2010.

[34] Juan A. Colmenares et al. "A Multicore Operating System with QoS Guarantees for Network Audio Applications". In: *Journal of the Audio Engineering Society* 61.4 (2013), pp. 174–184.

[35] Juan A. Colmenares et al. "Tessellation: refactoring the OS around explicit resource containers with continuous adaptation". In: *Proceedings of the 50th Annual Design Automation Conference*. DAC'13. Austin, Texas, USA, 2013, 76:1–76:10.

[36] Gilberto Contreras and Margaret Martonosi. "Characterizing and improving the performance of Intel Threading Building Blocks". In: *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*. Sept. Pp. 57–66.

[37] Henry Cook and Kevin Skadron. "Predictive Design Space Exploration Using Genetically Programmed Response Surfaces". In: *45th ACM/IEEE Conference on Design Automation (DAC)*. 2008.

[38] Henry Cook et al. "A Hardware Evaluation of Cache Partitioning to Improve Utilization and Energy-efficiency While Preserving Responsiveness". In: *Proceedings of the 40th Annual International Symposium on Computer Architecture*. ISCA '13. Tel-Aviv, Israel: ACM, 2013, pp. 308–319. ISBN: 978-1-4503-2079-5. DOI: `10.1145/2485922.2485949`. URL: `http://doi.acm.org/10.1145/2485922.2485949`.

[39] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Commun. ACM* 51.1 (Jan. 2008), pp. 107–113. ISSN: 0001-0782. DOI: 10.1145/1327452.1327492. URL: http://doi.acm.org/10.1145/1327452.1327492.

[40] Christina Delimitrou and Christos Kozyrakis. "Paragon: QoS-aware Scheduling for Heterogeneous Datacenters". In: *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '13. Houston, Texas, USA: ACM, 2013, pp. 77–88. ISBN: 978-1-4503-1870-9. DOI: 10.1145/2451116.2451125. URL: http://doi.acm.org/10.1145/2451116.2451125.

[41] Ashutosh S. Dhodapkar and James E. Smith. "Comparing Program Phase Detection Techniques". In: *Proc. MICRO 36*. Washington, DC, USA: IEEE Computer Society, 2003, p. 217. ISBN: 0-7695-2043-X.

[42] Ronald P. Doyle et al. "Model-based Resource Provisioning in a Web Service Utility". In: *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*. USITS'03. Seattle, WA: USENIX Association, 2003, pp. 5–5. URL: http://dl.acm.org/citation.cfm?id=1251460.1251465.

[43] Haakon Dybdahl and Per Stenstrom. "An Adaptive Shared/Private NUCA Cache Partitioning Scheme for Chip Multiprocessors". In: *Proc. HPCA '07*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 2–12. ISBN: 1-4244-0804-0.

[44] Stephane Eranian. "Perfmon2: a flexible performance monitoring interface for Linux". In: *Ottawa Linux Symposium*. 2006, 269288.

[45] Dror G. Feitelson. *Job Scheduling in Multiprogrammed Parallel Systems*. 1997.

[46] Dror G. Feitelson and Larry Rudolph. "Gang Scheduling Performance Benefits for Fine-Grain Synchronization". In: *J. Parallel Distrib. Comput.* 16.4 (1992), pp. 306–318.

[47] Dror G. Feitelson, Larry Rudolph, and Uwe Schwiegelshohn. "Parallel Job Scheduling - A Status Report". In: *JSSPP*. 2004, pp. 1–16.

[48] Andrew D. Ferguson et al. "Jockey: guaranteed job latency in data parallel clusters". In: *Proceedings of the 7th ACM european conference on Computer Systems*. EuroSys '12. Bern, Switzerland: ACM, 2012, pp. 99–112. ISBN: 978-1-4503-1223-3. DOI: 10.1145/2168836.2168847. URL: http://doi.acm.org/10.1145/2168836.2168847.

[49] Liana Liyow Fong et al. *Gang scheduling for resource allocation in a cluster computing environment*. Patent US 6345287. 1997.

[50] Archana Ganapathi et al. "A Case for Machine Learning to Optimize Multicore Performance". In: *HotPar09*. Berkeley, CA, 2009. URL: http://www.usenix.org/event/hotpar09/tech/.

[51] Ali Ghodsi et al. "Dominant resource fairness: fair allocation of multiple resource types". In: *Proceedings of the 8th USENIX conference on Networked systems design and implementation*. NSDI'11. Boston, MA: USENIX Association, 2011, pp. 24–24. URL: http://dl.acm.org/citation.cfm?id=1972457.1972490.

[52] Daniel Gmach et al. "Workload Analysis and Demand Prediction of Enterprise Data Center Applications". In: *Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization*. IISWC '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 171–180. ISBN: 978-1-4244-1561-8. DOI: 10.1109/IISWC.2007.4362193. URL: http://dx.doi.org/10.1109/IISWC.2007.4362193.

[53] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. third. Baltimore, Maryland: Johns Hopkins University Press, 1996.

[54] Sriram Govindan et al. "Cuanta: Quantifying Effects of Shared On-chip Resource Interference for Consolidated Virtual Machines". In: *Proceedings of the 2Nd ACM Symposium on Cloud Computing*. SOCC '11. Cascais, Portugal: ACM, 2011, 22:1–22:14. ISBN: 978-1-4503-0976-9. DOI: 10.1145/2038916.2038938. URL: http://doi.acm.org/10.1145/2038916.2038938.

[55] Fei Guo et al. "A Framework for Providing Quality of Service in Chip Multi-Processors". In: *Proc. MICRO '07*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 343–355. ISBN: 0-7695-3047-8.

[56] Fei Guo et al. "From chaos to QoS: case studies in CMP resource management". In: *SIGARCH Comput. Archit. News* 35.1 (2007), pp. 21–30. ISSN: 0163-5964.

[57] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning, Second Edition*. Springer New York, 2005.

[58] John L. Hennessy and David A. Patterson. *Computer Architecture - A Quantitative Approach (5. ed.)* Morgan Kaufmann, 2012. ISBN: 978-0-12-383872-8.

[59] Benjamin Hindman et al. "Mesos: a platform for fine-grained resource sharing in the data center". In: *Proceedings of the 8th USENIX conference on Networked systems design and implementation*. NSDI'11. Boston, MA: USENIX Association, 2011, pp. 22–22. URL: http://dl.acm.org/citation.cfm?id=1972457.1972488.

[60] Henry Hoffmann et al. *SEEC: a general and extensible framework for self-aware computing*. Tech. rep. MIT-CSAIL-TR-2011-046. Massachusetts Institute of Technology, 2011. URL: http://hdl.handle.net/1721.1/67020.

[61] Steven Hofmeyr et al. "Juggle: addressing extrinsic load imbalances in SPMD applications on multicore computers". In: *Cluster Computing* 16.2 (2013), pp. 299–319.

[62] Mark. Horowitz et al. "Scaling, power, and the future of CMOS". In: *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*. 2005, 7 pp.–15. DOI: 10.1109/IEDM.2005.1609253.

[63] Kenneth Hoste et al. "Performance Prediction Based on Inherent Program Similarity". In: *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*. PACT '06. Seattle, Washington, USA: ACM, 2006, pp. 114–122. ISBN: 1-59593-264-X. DOI: 10.1145/1152154.1152174. URL: http://doi.acm.org/10.1145/1152154.1152174.

[64] Lisa R. Hsu et al. "Communist, utilitarian, and capitalist cache policies on CMPs: caches as a shared resource". In: *Proc. PACT '06*. Seattle, Washington, USA: ACM, 2006, pp. 13–22. ISBN: 1-59593-264-X.

[65] Xuedong Huang, Alex Acero, and Hsaio-Wuen Hon. *Spoken Language Processing: A Guide to Theory, Algorithm and System Development*. Prentice Hall, 2001.

[66] Markus C. Huebscher and Julie A. McCann. "A survey of autonomic computing—degrees, models, and applications". In: *ACM Comput. Surv.* 40.3 (2008), pp. 1–28. ISSN: 0360-0300.

[67] Jaehyuk Huh et al. "A NUCA substrate for flexible CMP cache sharing". In: *Proc. ICS '05*. Cambridge, Massachusetts: ACM, 2005, pp. 31–40. ISBN: 1-59593-167-8.

[68] CVX Research Inc. *CVX: Matlab Software for Disciplined Convex Programming, version 2.0 beta*. http://cvxr.com/cvx. Sept. 2012.

[69] Intel Corp. *Intel 64 and IA-32 Architectures Optimization Reference Manual*. 2011.

[70] Intel Corp. *Intel 64 and IA-32 Architectures Software Developer's Manual*. March 2012.

[71] Michael Isard et al. "Quincy: fair scheduling for distributed computing clusters". In: *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. SOSP '09. Big Sky, Montana, USA: ACM, 2009, pp. 261–276. ISBN: 978-1-60558-752-3. DOI: 10.1145/1629575.1629601. URL: http://doi.acm.org/10.1145/1629575.1629601.

[72] Ravi Iyer. "CQoS: a framework for enabling QoS in shared caches of CMP platforms". In: *Proc. ICS '04*. St. Malo, France: ACM, 2004, pp. 257–266. ISBN: 1-58113-839-3.

[73] Ravi Iyer et al. "QoS policies and architecture for cache/memory in CMP platforms". In: *Proc. SIGMETRICS '07*. San Diego, California, USA: ACM, 2007, pp. 25–36. ISBN: 978-1-59593-639-4.

[74] Aamer Jaleel. *Memory Characterization of Workloads Using Instrumentation-Driven Simulation – A Pin-based Memory Characterization of the SPEC CPU2000 and SPEC CPU2006 Benchmark Suites*. Tech. rep. VSSAD, Intel Corporation, 2007.

[75] Shoaib Kamil. *Stencil Probe*. http://www.cs.berkeley.edu/~skamil/projects/stencilprobe/. 2012.

[76] Albert Kim et al. "A Soft Real-Time Parallel GUI Service in Tessellation Many-Core OS". In: *Proceedings of the ISCA 27th International Conference on Computers and Their Applications*. CATA'12. Las Vegas, Nevada, 2012.

[77] Changkyu Kim, Doug Burger, and Stephen W. Keckler. "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches". In: *Proc. ASPLOS-X*. San Jose, California: ACM, 2002, pp. 211–222. ISBN: 1-58113-574-2.

[78] Jon Kleinberg, Yuval Rabani, and Eva Tardos. "Fairness in routing and load balancing". In: *J. Comput. Syst. Sci.* 1999, pp. 568–578.

[79] Kevin Klues et al. "Processes and Resource Management in a Scalable Many-core OS". In: *HotPar10*. Berkeley, CA, 2010.

[80] Roger Koenker. *Quantile regression*. 38. Cambridge university press, 2005.

[81] Younggyun Koh et al. "An analysis of performance interference effects in virtual environments". In: *In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS*. 2007.

[82] Samuel Kounev, Ramon Nou, and Jordi Torres. "Autonomic QoS-aware resource management in grid computing using online performance models". In: *Proc. ValueTools '07*. Nantes, France: ICST, 2007, pp. 1–10. ISBN: 978-963-9799-00-4.

[83] John R. Koza. *Genetic Programming: On the programming of computers by means of natural selection*. MIT Press, 1992.

[84] Christos Kozyrakis. "Resource efficient computing for warehouse-scale datacenters". In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*. 2013, pp. 1351–1356. DOI: 10.7873/DATE.2013.278.

[85] H. T. Kung. "Memory Requirements for Balanced Computer Architectures". In: *International Symposium on Computer Architecture*. 1986, pp. 49–54.

[86] Charles L. Lawson and Richard J. Hanson. *Solving Least Squares Problems*. Englewood Cliffs, NJ: Prentice Hall, 1974.

[87] Jae W. Lee, Man Cheuk Ng, and Krste Asanovic. "Globally-Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks". In: *Proc. ISCA '08*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 89–100. ISBN: 978-0-7695-3174-8.

[88] Jae W. Lee, Man Cheuk Ng, and Krste Asanovic. "Globally-Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks". In: *SIGARCH Comput. Archit. News* 36.3 (2008), pp. 89–100.

[89] Bernhard Leiner et al. "A Comparison of Partitioning Operating Systems for Integrated Systems". In: *Proc. of SAFECOMP*. Nuremberg, Germany, 2007, pp. 342–355.

[90] Bil Lewis and Daniel J. Berg. *Multithreaded Programming with Pthreads*. Prentice Hall, 1998.

[91] Jinyang Li et al. "On the Feasibility of Peer-to-Peer Web Indexing and Search". In: *Peer-to-Peer Systems II*. Ed. by M.Frans Kaashoek and Ion Stoica. Vol. 2735. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, pp. 207–215. ISBN: 978-3-540-40724-9. DOI: 10.1007/978-3-540-45172-3_19. URL: http://dx.doi.org/10.1007/978-3-540-45172-3_19.

[92] Rose Liu et al. "Tessellation: Space-Time Partitioning in a Manycore Client OS". In: *HotPar09*. Berkeley, CA, 2009. URL: http://www.usenix.org/event/hotpar09/tech/.

[93] Yonghe Liu and E. Knightly. "Opportunistic fair scheduling over multiple wireless channels". In: *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*. Vol. 2. 2003, 1106–1115 vol.2. DOI: 10.1109/INFCOM.2003.1208947.

[94] Lei Luo and Ming-Yuan Zhu. "Partitioning based operating system: a formal model". In: *ACM SIGOPS Oper. Syst. Rev.* 37.3 (2003).

[95] Yuancheng Luo and Ramani Duraiswami. "Efficient Parallel Nonnegative Least Squares on Multicore Architectures". In: *SIAM Journal on Scientific Computing* 33.5 (2011), pp. 2848–2863.

[96] Jason Mars and Lingjia Tang. "Whare-map: Heterogeneity in "Homogeneous" Warehouse-scale Computers". In: *Proceedings of the 40th Annual International Symposium on Computer Architecture*. ISCA '13. Tel-Aviv, Israel: ACM, 2013, pp. 619–630. ISBN: 978-1-4503-2079-5. DOI: 10.1145/2485922.2485975. URL: http://doi.acm.org/10.1145/2485922.2485975.

[97] Jason Mars et al. "Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations". In: *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-44. Porto Alegre, Brazil: ACM, 2011, pp. 248–259. ISBN: 978-1-4503-1053-6. DOI: 10.1145/2155620.2155650. URL: http://doi.acm.org/10.1145/2155620.2155650.

[98] Mathworks. *Matlab 2009b*. http://www.mathworks.com/. 2009.

[99] Javier Merino et al. "SP-NUCA: a cost effective dynamic non-uniform cache architecture". In: *SIGARCH Comput. Archit. News* 36.2 (2008), pp. 64–71. ISSN: 0163-5964.

[100] Andreas Merkel and Frank Bellosa. "Balancing Power Consumption in Multiprocessor Systems". In: *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*. EuroSys '06. Leuven, Belgium: ACM, 2006, pp. 403–414. ISBN: 1-59593-322-0. DOI: 10.1145/1217935.1217974. URL: http://doi.acm.org/10.1145/1217935.1217974.

[101] Andreas Merkel and Frank Bellosa. "Task activity vectors: a new metric for temperature-aware scheduling". In: *Proc. Eurosys '08*. Glasgow, Scotland UK: ACM, 2008, pp. 1–12. ISBN: 978-1-60558-013-5. DOI: http://doi.acm.org/10.1145/1352592.1352594. URL: http://portal.acm.org/ft_gateway.cfm?id=1352594.

[102] Derek G. Murray et al. "Naiad: A Timely Dataflow System". In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. SOSP '13. Farminton, Pennsylvania: ACM, 2013, pp. 439–455. ISBN: 978-1-4503-2388-8. DOI: 10.1145/2517349.2522738. URL: http://doi.acm.org/10.1145/2517349.2522738.

[103] Mohamed N. Bennani and Daniel A. Menasce. "Resource Allocation for Autonomic Data Centers using Analytic Performance Models". In: *ICAC '05*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 229–240. ISBN: 0-7965-2276-9.

[104] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. "Q-clouds: Managing Performance Interference Effects for QoS-aware Clouds". In: *Proceedings of the 5th European Conference on Computer Systems*. EuroSys '10. Paris, France: ACM, 2010, pp. 237–250. ISBN: 978-1-60558-577-2. DOI: 10.1145/1755913.1755938. URL: http://doi.acm.org/10.1145/1755913.1755938.

[105] Kyle J. Nesbit, James Laudon, and James E. Smith. "Virtual private caches". In: *Proc. ISCA '07*. San Diego, California, USA: ACM, 2007, pp. 57–68. ISBN: 978-1-59593-706-3.

[106] Kyle J. Nesbit et al. "Multicore Resource Management". In: *IEEE Micro* 28.3 (2008), pp. 6–16. ISSN: 0272-1732.

[107] Microsoft Developer Network. *Interaction Class*. 2014. URL: http://msdn.microsoft.com/en-us/library/system.windows.interactivity.interaction(v=expression.40).aspx (visited on 02/23/2014).

[108] Daniel Nurmi et al. "The Eucalyptus Open-Source Cloud-Computing System". In: *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. CCGRID '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 124–131. ISBN: 978-0-7695-3622-4. DOI: 10.1109/CCGRID.2009.93. URL: http://dx.doi.org/10.1109/CCGRID.2009.93.

[109] Roman Obermaisser and Bernhard Leiner. "Temporal and Spatial Partitioning of a Time-Triggered Operating System Based on Real-Time Linux". In: *Proc. of ISORC*. Orlando, Florida, USA, 2008.

[110] John K. Ousterhout. "Scheduling techniques for concurrent systems". In: *Proc. of ICDCS*. Miami/Ft. Lauderdale, FL, USA, 1982.

[111] Heidi Pan, Benjamin Hindman, and Krste Asanović. "Lithe: Enabling Efficient Composition of Parallel Libraries". In: *HotPar09*. Berkeley, CA, 2009. URL: http://www.usenix.org/event/hotpar09/tech/.

[112] Marco Paolieri et al. "Hardware support for WCET analysis of hard real-time multicore systems". In: *SIGARCH Comput. Archit. News* 37.3 (2009), pp. 57–68.

[113] Neal Parikh and Stephen Boyd. "Proximal Algorithms". In: *Foundations and Trends in Optimization* 1.3 (2014), pp. 123–231.

[114] *Perfmon2 webpage*. perfmon2.sourceforge.net/.

[115] Aashish Phansalkar, Ajay Joshi, and Lizy Kurian John. "Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite". In: *ISCA*. 2007, pp. 412–423.

[116] Moinuddin K. Qureshi and Yale N. Patt. "Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches". In: *Proc. MICRO 39*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 423–432. ISBN: 0-7695-2732-9.

[117] Ragunathan Rajkumar et al. "A resource allocation model for QoS management". In: *Proc. RTSS '97*. Washington, DC, USA: IEEE Computer Society, 1997, p. 298. ISBN: 0-8186-8268-X.

[118] Rajesh Raman, Miron Livny, and Marv Solomon. "Matchmaking: An extensible framework for distributed resource management". In: *Cluster Computing* 2.2 (Apr. 1999), pp. 129–138. ISSN: 1386-7857. DOI: 10.1023/A:1019022624119. URL: http://dx.doi.org/10.1023/A:1019022624119.

[119] James Reiders. *Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism*. O'Reilly, 2007.

[120] Rightscale, Inc. *Amazon EC2: Rightscale*. http://aws.amazon.com/solution-providers/isv/rightscale.

[121] Christopher J. Rossbach et al. "Dandelion: A Compiler and Runtime for Heterogeneous Systems". In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. SOSP '13. Farminton, Pennsylvania: ACM, 2013, pp. 49–68. ISBN: 978-1-4503-2388-8. DOI: 10.1145/2517349.2522715. URL: http://doi.acm.org/10.1145/2517349.2522715.

[122] John Rushby. *Partitioning for avionics architectures: requirements, mechanisms, and assurance*. Tech. rep. CR-1999-209347. NASA Langley Research Center, 1999.

[123] Daniel Sanchez and Christos Kozyrakis. "Vantage: scalable and efficient fine-grain cache partitioning". In: *SIGARCH Comput. Archit. News* 39.3 (2011), pp. 57–68.

[124] Upendra Sharma et al. "A Cost-Aware Elasticity Provisioning System for the Cloud". In: *Proceedings of the 2011 31st International Conference on Distributed Computing Systems*. ICDCS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 559–570. ISBN: 978-0-7695-4364-2. DOI: 10.1109/ICDCS.2011.59. URL: http://dx.doi.org/10.1109/ICDCS.2011.59.

[125] Kai Shen et al. "Hardware counter driven on-the-fly request signatures". In: *SIGOPS Oper. Syst. Rev.* 42.2 (2008), pp. 189–200. ISSN: 0163-5980. DOI: http://doi.acm.org/10.1145/1353535.1346306.

[126] Stelios Sidiroglou-Douskos et al. "Managing Performance vs. Accuracy Trade-offs with Loop Perforation". In: *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*. ESEC/FSE '11. Szeged, Hungary: ACM, 2011, pp. 124–134. ISBN: 978-1-4503-0443-6. DOI: 10.1145/2025113.2025133. URL: http://doi.acm.org/10.1145/2025113.2025133.

[127] Filippo Sironi et al. "Metronome: operating system level performance management via self-adaptive computing". In: *Proceedings of the 49th Annual Design Automation Conference*. DAC '12. San Francisco, California: ACM, 2012, pp. 856–865. ISBN: 978-1-4503-1199-1. DOI: 10.1145/2228360.2228514. URL: http://doi.acm.org/10.1145/2228360.2228514.

[128] Allan Snavely et al. "A framework for performance modeling and prediction". In: *SC*. 2002, pp. 1–17.

[129] Ahmed A. Soror et al. "Automatic Virtual Machine Configuration for Database Workloads". In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD '08. Vancouver, Canada: ACM, 2008, pp. 953–966. ISBN: 978-1-60558-102-6. DOI: 10.1145/1376616.1376711. URL: http://doi.acm.org/10.1145/1376616.1376711.

[130] Standard Performance Evaluation Corporation. *SPEC CPU 2006 benchmark suite*. http://www.spec.org.

[131] Christopher Stewart, Terence Kelly, and Alex Zhang. "Exploiting Nonstationarity for Performance Prediction". In: *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. EuroSys '07. Lisbon, Portugal: ACM, 2007, pp. 31–44. ISBN: 978-1-59593-636-3. DOI: 10.1145/1272996.1273002. URL: http://doi.acm.org/10.1145/1272996.1273002.

[132] G. Edward Suh, Srinivas Devadas, and Larry Rudolph. "A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning". In: *Proc. HPCA '02*. Washington, DC, USA: IEEE Computer Society, 2002, p. 117.

[133] G. Edward Suh, Larry Rudolph, and Srinivas Devadas. "Dynamic Partitioning of Shared Cache Memory". In: *Journal of Supercomputing* 28.1 (2004), pp. 7–26. ISSN: 0920-8542.

[134] *SuperFetch*. http://en.wikipedia.org/wiki/Windows_Vista_I/O_technologies\#SuperFetch.

[135] Eric Taillard. "Benchmarks for basic scheduling problems". In: *European Journal of Operational Research* 64.2 (1993), pp. 278–285. URL: http://ideas.repec.org/a/eee/ejores/v64y1993i2p278-285.html.

[136] David Tam, Reza Azimi, and Michael Stumm. "Thread clustering: sharing-aware scheduling on SMP-CMP-SMT multiprocessors". In: *EuroSys '07: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. Lisbon, Portugal: ACM, 2007, pp. 47–58. ISBN: 978-1-59593-636-3. DOI: http://doi.acm.org/10.1145/1272996.1273004.

[137] Zhangxi Tan et al. "A Case for FAME: FPGA Architecture Model Execution". In: *Proc. of the 37th ACM/IEEE Int'l Symposium on Computer Architecture (ISCA 2010)*. Saint-Malo, France, 2010.

[138] Zhangxi Tan et al. "RAMP Gold: An FPGA-based Architecture Simulator for Multiprocessors". In: *Proc. of the 47th Design Automation Conference (DAC 2010)*. Anaheim, CA, USA, 2010.

[139] Andrew S Tanenbaum and Albert S Woodhull. *Operating Systems Design and Implementation (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2005. ISBN: 0131429388.

[140] Lingjia Tang et al. "The Impact of Memory Subsystem Resource Sharing on Datacenter Applications". In: *Proceedings of the 38th Annual International Symposium on Computer Architecture*. ISCA '11. San Jose, California, USA: ACM, 2011, pp. 283–294. ISBN: 978-1-4503-0472-6. DOI: 10.1145/2000064.2000099. URL: http://doi.acm.org/10.1145/2000064.2000099.

[141] Gerald Tesauro, William E. Walsh, and Jeffrey O. Kephart. "Utility-Function-Driven Resource Allocation in Autonomic Systems". In: *Proc. ICAC '05*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 342–343. ISBN: 0-7965-2276-9.

[142] Gerald Tesauro and Jeffrey O. Kephart. "Utility Functions in Autonomic Systems". In: *Proc. ICAC '04*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 70–77. ISBN: 0-7695-2114-2.

[143] Gerald Tesauro et al. "On the use of hybrid reinforcement learning for autonomic resource allocation". In: *Cluster Computing* 10.3 (2007), pp. 287–299. ISSN: 1386-7857.

[144] Bhuvan Urgaonkar, Prashant Shenoy, and Timothy Roscoe. "Resource Overbooking and Application Profiling in Shared Hosting Platforms". In: *SIGOPS Oper. Syst. Rev.* 36.SI (Dec. 2002), pp. 239–254. ISSN: 0163-5980. DOI: 10.1145/844128.844151. URL: http://doi.acm.org/10.1145/844128.844151.

[145] Kushagra Vaid. *Datacenter Power Efficiency: Separating Fact from Fiction*. Invited talk at the 2010 Workshop on Power Aware Computing and Systems. 2010.

[146] Nedeljko Vasic et al. "DejaVu: accelerating resource allocation in virtualized environments". In: *ASPLOS*. 2012, pp. 423–436.

[147] Vinod Kumar Vavilapalli et al. "Apache Hadoop YARN: Yet Another Resource Negotiator". In: *Proceedings of the 4th Annual Symposium on Cloud Computing*. SOCC '13. Santa Clara, California: ACM, 2013, 5:1–5:16. ISBN: 978-1-4503-2428-1. DOI: 10.1145/2523616.2523633. URL: http://doi.acm.org/10.1145/2523616.2523633.

[148] Akshat Verma, Puneet Ahuja, and Anindya Neogi. "Power-aware Dynamic Placement of HPC Applications". In: *Proceedings of the 22Nd Annual International Conference on Supercomputing*. ICS '08. Island of Kos, Greece: ACM, 2008, pp. 175–184. ISBN: 978-1-60558-158-3. DOI: 10.1145/1375527.1375555. URL: http://doi.acm.org/10.1145/1375527.1375555.

[149] Virtutech. *Simics ISA Simulator*. www.simics.net. 2008.

[150] Larry Wasserman. *All of Nonparametric Statistics (Springer Texts in Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[151] Brian J. Watson et al. "Probabilistic Performance Modeling of Virtualized Resource Allocation". In: *Proceedings of the 7th International Conference on Autonomic Computing*. ICAC '10. Washington, DC, USA: ACM, 2010, pp. 99–108. ISBN: 978-1-4503-0074-2. DOI: 10.1145/1809049.1809067. URL: http://doi.acm.org/10.1145/1809049.1809067.

[152] Richard West et al. "Online Cache Modeling for Commodity Multicore Processors". In: *SIGOPS Oper. Syst. Rev.* 44.4 (Dec. 2010), pp. 19–29. ISSN: 0163-5980. DOI: 10.1145/1899928.1899931. URL: http://doi.acm.org/10.1145/1899928.1899931.

[153] Jonathan A. Winter, David H. Albonesi, and Christine 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*. PACT '10. Vienna, Austria: ACM, 2010, pp. 29–40. ISBN: 978-1-4503-0178-7. DOI: 10.1145/1854273.1854283. URL: http://doi.acm.org/10.1145/1854273.1854283.

[154] Tingxin Yan et al. "Fast App Launching for Mobile Devices Using Predictive User Context". In: *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*. MobiSys '12. Low Wood Bay, Lake District, UK: ACM, 2012, pp. 113–126. ISBN: 978-1-4503-1301-8. DOI: 10.1145/2307636.2307648. URL: http://doi.acm.org/10.1145/2307636.2307648.

[155] Hailong Yang et al. "Bubble-flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers". In: *Proceedings of the 40th Annual International Symposium on Computer Architecture*. ISCA '13. Tel-Aviv, Israel: ACM, 2013, pp. 607–618. ISBN: 978-1-4503-2079-5. DOI: 10.1145/2485922.2485974. URL: http://doi.acm.org/10.1145/2485922.2485974.

[156] Ting Yang et al. "Redline: first class support for interactivity in commodity operating systems". In: *Proceedings of the 8th USENIX conference on Operating systems design and implementation*. OSDI'08. San Diego, California: USENIX Association, 2008, pp. 73–86. URL: http://dl.acm.org/citation.cfm?id=1855741.1855747.

[157] Thomas Y. Yeh and Glenn Reinman. "Fast and fair: data-stream quality of service". In: *Proc. CASES '05*. San Francisco, California, USA: ACM, 2005, pp. 237–248. ISBN: 1-59593-149-X.

[158] Matei Zaharia et al. "Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling". In: *Proceedings of the 5th European conference on Computer systems*. EuroSys '10. Paris, France: ACM, 2010, pp. 265–278. ISBN: 978-1-60558-577-2. DOI: 10.1145/1755913.1755940. URL: http://doi.acm.org/10.1145/1755913.1755940.

[159] Michael Zhang and Krste Asanovic. "Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors". In: *Proc. ISCA '05*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 336–345. ISBN: 0-7695-2270-X.

[160] Xiao Zhang et al. "Processor hardware counter statistics as a first-class system resource". In: *HOTOS'07: Proceedings of the 11th USENIX workshop on Hot topics in operating systems*. San Diego, CA: USENIX Association, 2007, pp. 1–6.

[161] Li Zhao et al. "Towards hybrid last level caches for chip-multiprocessors". In: *SIGARCH Comput. Archit. News* 36.2 (2008), pp. 56–63. ISSN: 0163-5964.

[162] Wei Zheng et al. "JustRunIt: Experiment-based Management of Virtualized Data Centers". In: *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*. USENIX'09. San Diego, California: USENIX Association, 2009, pp. 18–18. URL: http://dl.acm.org/citation.cfm?id=1855807.1855825.

[163] Dakai Zhu, Daniel Mossé, and Rami Melhem. "Multiple-Resource Periodic Scheduling Problem: how much fairness is necessary?" In: *Proceedings of the 24th IEEE International Real-Time Systems Symposium*. RTSS '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 142–. ISBN: 0-7695-2044-8. URL: http://dl.acm.org/citation.cfm?id=956418.956616.

[164] Xiaoyun Zhu et al. "1000 Islands: An Integrated Approach to Resource Management for Virtualized Data Centers". In: *Cluster Computing* 12.1 (Mar. 2009), pp. 45–57. ISSN: 1386-7857. DOI: 10.1007/s10586-008-0067-6. URL: http://dx.doi.org/10.1007/s10586-008-0067-6.

[165] Zeyuan Allen Zhu et al. "Randomized Accuracy-aware Program Transformations for Efficient Approximate Computations". In: *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '12. Philadelphia, PA, USA: ACM, 2012, pp. 441–454. ISBN: 978-1-4503-1083-3. DOI: 10.1145/2103656.2103710. URL: http://doi.acm.org/10.1145/2103656.2103710.

[166] Sergey Zhuravlev, Sergey Blagodurov, and Alexandra Fedorova. "Addressing Shared Resource Contention in Multicore Processors via Scheduling". In: *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*. ASPLOS XV. Pittsburgh, Pennsylvania, USA: ACM, 2010, pp. 129–142. ISBN: 978-1-60558-839-1. DOI: 10.1145/1736020.1736036. URL: http://doi.acm.org/10.1145/1736020.1736036.