# ORADIA: Optimizing Resource Allocations for Dynamic Interactive Applications

Paper 68; Authors Redacted

## Abstract

*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 ORADIA, 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, ORADIA is able to understand the resource requirements of applications and perform near-optimal resource allocation—2% from the best allocation in 350 µs while only requiring a few hundred bytes of storage per application.*

## 1. Introduction

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. Additionally, systems are required to meet these response-time demands while improving energy efficiency in order to reduce system power consumption and improve battery life.

Ideally, applications should be given just enough system resources (*e.g.,* nodes, processor cores, cache slices, memory pages, various kinds of bandwidth) to meet their performance requirements consistently. However, allocating resources among multiple parallel, real-time applications while satisfying all of their *Quality-of-Service* (QoS) requirements is a complex optimization problem, particularly as modern hardware diversifies to include a variety of parallel architectures (*e.g.,* multicore, GPUs).

Historically, general-purpose OSes often attempted to represent performance requirements with a single value (usually called a *priority*) associated with a thread of computation and adjusted within the OS by a variety of *ad-hoc* mechanisms. OSes have also been rather unsystematic about resource allocation and hence have not provided useful mechanisms to implement stronger performance guarantees among multiple competing applications, making it difficult to reason about the expected response time of an application.

Priority-based approaches have no mechanism to describe deadlines or the resources required to meet them, and so must run the highest priority applications as fast as possible on all the resources requested. Consequently, interactive and real-time applications are often run needlessly fast with significant over-provisioning—wasting energy and reducing throughput by preventing other applications from using the resources. 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 [7, 35]. In some cases, cloud providers run only a single application on a cluster to avoid unexpected interference. Some mobile systems have gone so far as to limit which applications can run in the background [6] 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.

In this paper, we present ORADIA, a resource-allocation framework designed to provide responsiveness guarantees for a simultaneous mix of throughput, interactive, and real-time applications in an efficient, scalable manner. It allows both application deadlines and the relative importance of meeting these deadlines to be specified. Unlike traditional systems, ORADIA considers *all* resource types when making decisions. Using runtime measurements, application-specific performance models are built and maintained to help determine the resources required to meet each application's performance goals. The quantity of each resource to give each application is determined using convex optimization, allowing the system to make continuous trade-offs among application responsiveness, system performance, and energy efficiency.

We believe ORADIA is applicable to 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. In this paper we choose to study ORADIA 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.

ORADIA makes resource allocation decisions in 350 µs in the worst case and often faster than 50 µs in our many-

core OS implementation. Static allocation decisions are near optimal—only 2% from the best possible allocation on average. ORADIA is able to dynamically allocate resources to adjust to the changing state of the system and trade off responsiveness and energy. By automatically generating performance models of applications, ORADIA alleviates the developer from having to understand hardware resources and requires only a few hundred bytes of additional storage per application. 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. This effect enables modeling to be feasible in real systems with noisy applications.

In this paper we first present the architecture of ORADIA in Section 2. In Section 3, we evaluate its effectiveness at static resource allocation decisions, and in Section 4, we assess how well it dynamically allocates resources in a manycore OS. Section 5 describes our dynamic penalty optimization algorithm. Section 6 discusses response time functions and their creation in detail. Section 7 describes some challenges and possible solutions for ORADIA. Section 8 discusses related work, and Section 9 concludes.

## 2. ORADIA Architecture

ORADIA is a framework designed to determine the proper amount of each resource type to give each application. For example, consider a video conference scenario where each participant requires a separate, performance-guaranteed video stream. New participants may join the conference and others may leave, increasing or decreasing the number of streams running at any given time. 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, the real challenge is to do so without reserving excessive resources, which will compromise system utilization or power consumption. The purpose of ORADIA is to dynamically assign resources across multiple applications to guarantee responsiveness without over-provisioning and to adapt allocations as the application mix changes.

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.

ORADIA is designed for systems where resource allocation is separated 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 work for everything. This approach leaves the system to focus on the problem of *how much* of each resource type to assign to each application. In client machines, ORADIA is used to make coarse-grain resource-allocation decisions (*e.g.,* cores and memory pages) at the OS level, while the micro-management of these resources to run application tasks is left to user-level runtimes such as Intel Threaded Building Blocks [21] or Lithe [59], and to user-level memory managers. In the cloud environment, ORADIA allocates resources (*e.g.,* nodes and storage) to jobs, and scheduling is left to other entities such as the MapReduce framework [23] or the node OS. ORADIA can also be used in a hypervisor to allocate resources among guest OSes.

ORADIA formulates resource allocation as an optimization problem built from two types of application-specific functions: a response-time function 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). ORADIA uses convex optimization [16] to determine the ideal resource allocation across all active applications. The following subsections briefly introduce the primary components of ORADIA and the optimization formulation.

### 2.1. Resources

In our client system, resources are anything that the system can "partition" in hardware or software: specifically we use cores, network bandwidth, cache ways, and memory pages in our system. However, other resources could be easily added assuming they have QoS enforcement mechanisms. The other scenarios would have resources that perform similar functions (compute, network, capacity), but 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.

### 2.2. Response-Time Functions

Response-time functions (RTF) represent the expected *response time* of an application as a function of the resources allocated to the application. The response time is an application-specific measure of the performance 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;
- The time to execute a specified amount of work.

The RTFs are built to be convex functions. All applications have a function of the same form but the application-specific weights are set using the performance history of the application. RTFs are designed to capture information such as how well an application scales with a particular resource. As a result, RTFs naturally support heterogeneity. Each CPU or GPU type is simply viewed as a different resource type by the

system, and thus the RTFs will represent how effectively an application uses a particular type of core. Figure 1 shows two example RTFs we have created from applications we studied.

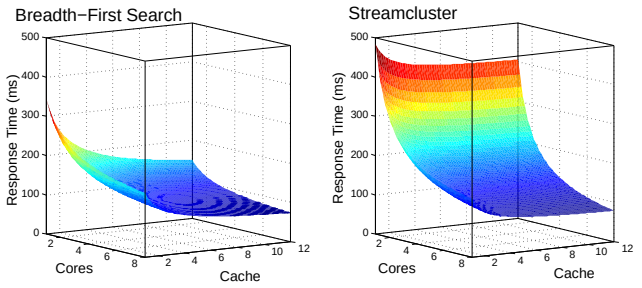


**Figure 1: Response Time Functions for a breadth-first search algorithm and streamcluster from PARSEC. We show two resource dimensions: cores and cache slices.**

Equation 1 below shows the RTF we use in ORADIA.

$$\tau(w,a) = \tau_0 + \sum_{i\in n, j\in n} \frac{w_{i,j}}{\sqrt{a_i * a_j}} \quad (1)$$

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 $j$. We choose this specific function because it is convex in the resources and in initial application studies we found it models response time behavior accurately enough to allow the optimization to make good decisions. In our experience the cross term weights (those with $i \neq j$) are almost always negligible and can be omitted. The dimensionality of the function increases roughly linearly with the number of resource types. The RTFs will be described further in Section 6. Alternative models and the initial model evaluations are described in [3].

ORADIA 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 completely perfect: all of our evaluation was run on current x86 hardware with some shared resources, and ORADIA is still effective. Section 7 discusses handling shared resources in more detail.

### 2.3. Penalty Functions

Penalty functions are designed to represent the user-level goals of the application. They are similar to priorities but are functions of the response time rather than simply values, so they can explicitly represent deadlines. Knowing the deadlines lets the system 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 set by the system on behalf of the user.

ORADIA'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 increases. For applications without response-time constraints the deadline can be set to 0. Two representative graphs of this type appear in Figures 2 and 3.

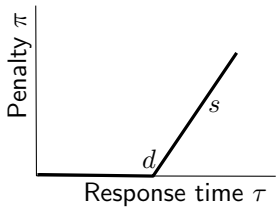


**Figure 2: A penalty function with a response time constraint.**

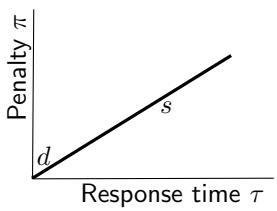


**Figure 3: A penalty function with no response time constraint.**

### 2.4. Resource Allocation as Optimization

ORADIA formulates resource allocation as an optimization problem designed to minimize the total penalty of the system. This approach is analogous to minimizing user dissatisfaction with the user experience due to missed deadlines in a client system, and minimizing the contract penalties paid for violated SLAs in a cloud system.

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, it would difficult to provide predictable response times for applications by considering the allocation of only one resource type at a time.

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

$$\begin{aligned} \text{Minimize} \quad & \sum_{p\in P} \pi_p(\tau_p(a_{p,1}\ldots a_{p,n})) \\ \text{Subject to} \quad & \sum_{p\in P} a_{p,r} \leq A_r, r = 1,\ldots n \\ \text{and} \quad & a_{p,r} \geq 0 \end{aligned}$$

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. Optimization details are described in Section 5.

### 2.5. 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, by framing our resource allocation problem as a *convex optimization problem* [16], two benefits accrue: an optimal solution will exist without multiple local extrema, and fast, incremental solutions will become feasible.

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 $\theta$ 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.

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

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

ORADIA'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 can be found in [3]. Note that the resource constraints are affine and therefore convex; they can be rewritten as $\sum_{p \in P}(a_{p,r} - A_r) \leq 0$ and $-a_{p,r} \leq 0$.

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 problem which is 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.

ORADIA also formulates RTF *creation* as a convex optimization problem, as explained in Section 6.

### 2.6. Power and Energy

In ORADIA, we create an artificial application to represent the interest in reducing system power and energy. Application 0 is designated the idle application and receives allocations of all resources that are left idle, *i.e.,* not allocated to other applications. The idle resources can be powered off or put to sleep if possible to save power.

The "response time" for application 0, $\tau_0$, is artificially defined to be the total system power consumption. 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 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.

## 3. Static Resource Allocation Experiments

We use two different implementations of the ORADIA framework for evaluation. Our static framework, described in this Section, collects data online using an x86 processor running Linux and uses that data to build models and make static resource allocation decisions. Our dynamic framework implemented in a research-prototype manycore operating system, MOS, is presented in Section 4.

Our static framework was designed to test the effectiveness of ORADIA'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 to MATLAB [53] 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 [22] 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.

### 3.1. 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 4 quad-issue out-of-order superscalar cores, each of which supports 2 hyperthreads using simultaneous multithreading [40]. The LLC is a 12-way set-associative 6 MB inclusive L3 cache, shared among all cores using a ring-based interconnect. 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.

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 4-core allocation from ORADIA represents 4 hyperthreads on 2 cores on the machine.

### 3.2. Performance and Energy Measurement

To measure application performance, we use the `libpfm` library [26, 60], built on top of the `perf_events` infrastructure in Linux, to access available performance counters [41].

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. We access these counters using the Running Average Power Limit (RAPL) interfaces [41].

### 3.3. Description of Workloads

Our workload contains a range of applications from three different popular benchmark suites: SPEC CPU 2006 [67], DaCapo [13], and PARSEC [12]. We selected this set of applications to represent a wide variety of possible resource behaviors in order to properly stress ORADIA's RTFs. We include some additional applications to broaden the scope of the study, and some microbenchmarks to exercise certain system features.

The **SPEC CPU2006** benchmark suite [67] 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.* [61], 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 [45], 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 [61].

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 [12]. 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 [46].

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.

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 [61], we use machine learning to select representative benchmarks. We use a hierarchical clustering algorithm [61] 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 sensitivity. The clustering algorithm uses Euclidean distance between vectors to determine clusters.

The clustering results in 6 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`)

### 3.4. RTF Experiments

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 [11] 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 [3]. We then use the model to predict the performance of every resource allocation and compare it with the actual measured performance (median value of 3 trials) of that resource allocation. We built 3 different models from 3 trials and tested each of them against median measured value.

Figure 4 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). The average variability is 9%, so we can see that ORADIA'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 rel-

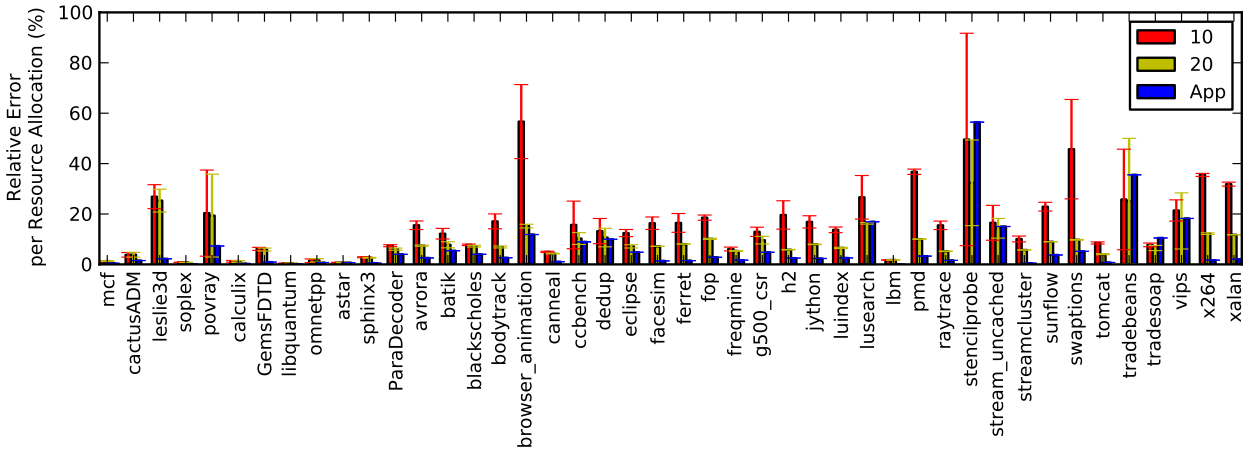


**Figure 4: 1-norm of relative error from RTF predicted response time compared to actual response time. The actual response time is the median over 3 trials. 10 and 20 represent RTFs built with 10 and 20 training points respectively. App represents the variability (average standard deviation) in performance of the application between the 3 trials.**

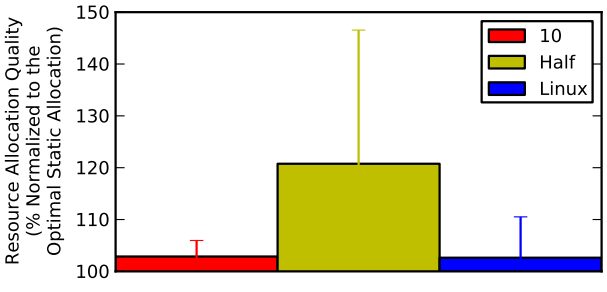


**Figure 5: Resource allocation decisions for each pair of the cluster representative applications compared equally dividing the machine and a shared resources Linux baseline. Quality is measured is allocation performance divided by performance of the best possible allocation.**

ative errors, (*e.g.,* `stencilprobe`, `tradebeans`). Section 7 discusses application variability in more detail.

### 3.5. Resource Allocation Experiments

Using the RTFs built for the applications, we let ORADIA make static resource allocations for all possible pairs of the cluster representative applications. We then run 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.

Figure 5 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 ORADIA or standard Linux. ORADIA'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. The shared resources comparison is interesting: while most of the time sharing resources can result in higher utilization, as the applications can dynamically take advantage of available resources, in some cases the interference between applications was so harmful to performance that on average optimal static partitioning performs slightly better. As a result ORADIA is able to provide performance comparable to Linux scheduling on shared resources with more predictable performance on average (lower worst cases). Additionally, as the shown in the next Section, ORADIA's resource allocation decisions do not need to be static, but can be made dynamically to adjust to the changing needs of the applications.

## 4. Dynamic Resource Allocation in a Manycore OS

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

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

This dynamic framework is used to test our implementations of the algorithms, measure the overhead and reaction times, and illustrate ORADIA's ability to work in a real system.

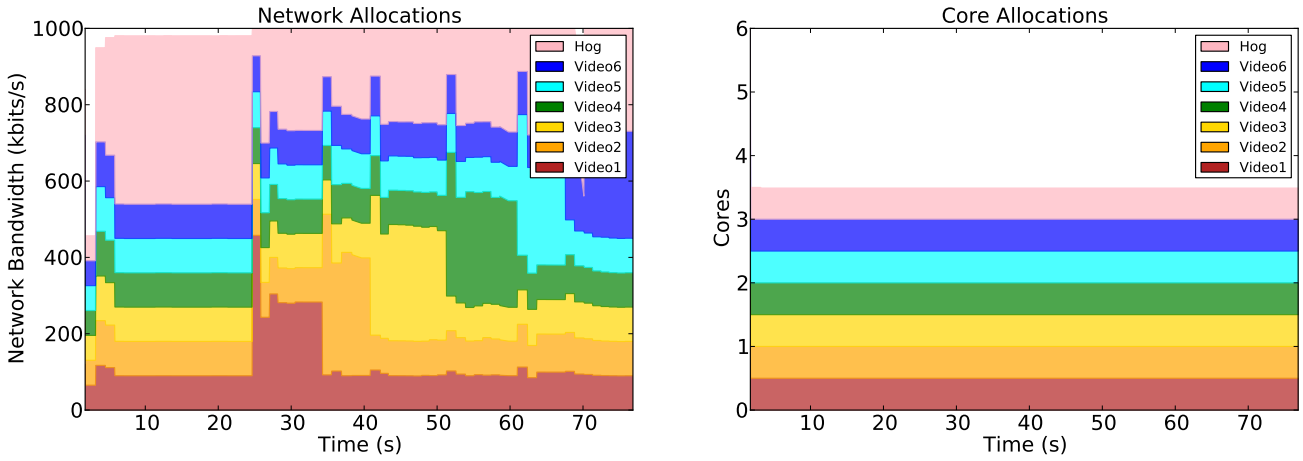


**Figure 6: Wall Power Mode: Resource Allocations of video conferencing through time as the videos change resolution to adjust for which person is speaking. Hog represents a background task uploading files.**

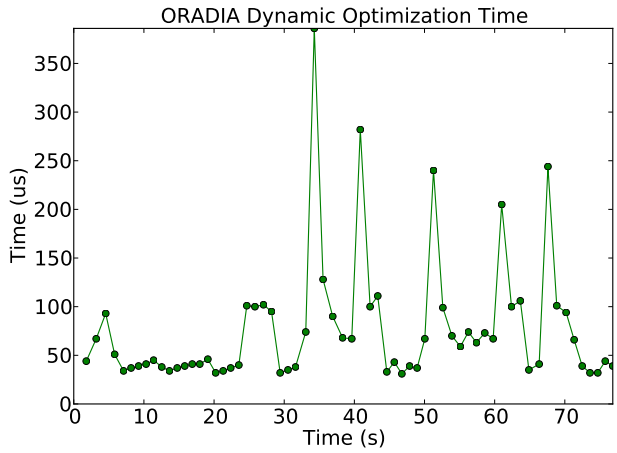


**Figure 7: Performance of our penalty optimization algorithm**

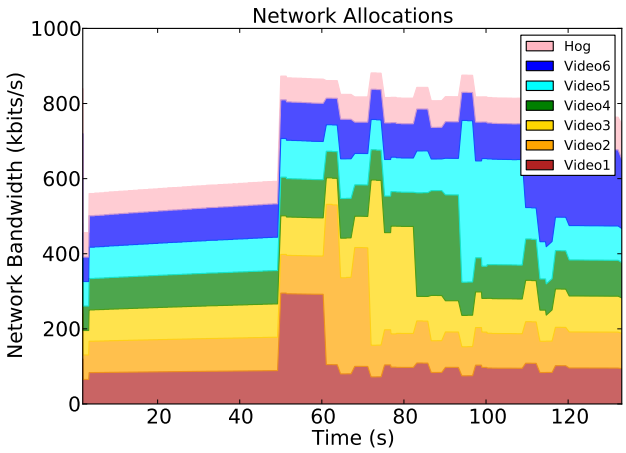


**Figure 8: Battery Power Mode: Resource Allocations of video conferencing through time as the videos change resolution to adjust for which person is speaking. Hog represents a background task uploading files.**

### 4.1. 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 with 16 hardware threads. This system also contains a 1-Gbps Intel Pro/1000 Ethernet network adapter. MOS allocates resources directly to applications, and the applications employ a second-level scheduler to schedule work onto the resources. All of our experiments use a preemptive scheduling framework called PULSE (Preemptive User-Level SchEduling), 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.

In addition to allocating cores and cache ways as with our static framework, MOS can also allocate fractions of network bandwidth. In our experiments, we show ORADIA allocating cores and network bandwidth because the experimental platform, which has the advantage of more cores, does not have cache partitioning. We have also successfully experimented with additional resources such as memory pages and CPU utilization, but the results are not presented here for brevity.

### 4.2. Performance and Energy Measurement

Applications report their own measured response times to ORADIA through a message-passing interface built into MOS, and ORADIA uses this information to build models offline or online. The online models are identical to our offline models for the same inputs, so the method used has no effect on resource allocation decisions. However, online models are able to adapt changes in applications. Modeling is discussed

further in Section 6.

MOS enables ORADIA 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.

### 4.3. Description of Workloads

Our workload is designed to represent a video conference with participants from many remote locations. There is a separate, performance-guaranteed incoming video stream for each participant. The application 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 changes, the requirements for the video streams change. In the background, network-intensive tasks such as uploading and downloading files are executed.

Our streaming video application is a multi-threaded, CPU- and network-intensive workload intended to simulate video chat applications like Skype and Facetime. Each video stream is encoded offline in the H.264 format using libx264, transported across the network through a TCP connection from a Linux E5-based server, and decoded and displayed by the MOS client. The client receives, decodes, and displays each frame using libffmpeg and libx264. Each video stream has a corresponding EDF-scheduled thread with 33 ms deadlines using our second-level EDF scheduler. We also use a network bandwidth hog application designed to represent an application such as Google Drive or Dropbox uploading files to the cloud. The hog contends with the video player for bandwidth by constantly sending UDP messages to the Linux server.

### 4.4. Resource Allocation Experiments

We constrain the available resources in the system for ORADIA to 1 Mbit/s network bandwidth and 6 cores to simulate a resource-constrained system as this is a more challenging resource allocation problem.

In our experiment the video conference contains 6 incoming video streams. At first no one is speaking and thus all the videos are small. The person speaking rotates across the incoming videos 1–6, with the videos resizing around every 10 seconds. We set the deadline to be 33 ms to represent a frame rate of 30 frames/sec. As this rate, small videos require 80 to 87 kbits/s depending on the particular frames. Large videos require 250 to 270 kbits/s. 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 (1.0) with no deadline.

Figure 6 shows the resource allocations for our video conference scenario when the computer is using "wall power". We mean wall power to imply that saving power is less critical and thus we assign a low penalty slope to application 0. ORADIA gives 90 kbits/s to small videos, 280 kbit/s to large videos, and the remaining bandwidth is given to the network hog. Video resizes can be seen as the "steps" in the graph: for example, Video1 becomes large at 24 seconds resulting in its allocation increasing from 90 kbits/s to 280 kbits/s. Video2 becomes large at 33 seconds and Video1 returns to its original size.

Since there is a single set of extreme points in a convex problem after a sufficient number of iterations, our dynamic algorithm will produce the same allocations as our static framework. The runtime is controllable based on the exit conditions of the optimization algorithm, however earlier exits may result in intermediate reallocations. We can see intermediate resource allocations (the "columns" in the graph) as ORADIA transitions for one set of allocations to another. If run much longer (50 ms), ORADIA can completely eliminate the intermediate allocations, which may be appropriate in environments where resource reallocation is expensive and allocations do not need to be adapted frequently. It is also possible to provide guaranteed optimization times of around 20 µs, but at the cost of many more intermediate reallocations. These results can be found in [3].We choose this particular point in the space because we believed it strikes the right balance between reactivity and reallocation amounts for a client operating system. Figure 7 shows the runtime of the optimization algorithm as it makes resource allocation decisions for the video scenario. The algorithm runs in 50 µs on average, with more significant reallocations taking from 250-350 µs. Significant changes result in 1-2 intermediate reallocations. In MOS, network bandwidth is easily reallocated and thus intermediate reallocations do not have much overhead. Core allocations are also easily adjusted between fractional core amounts. Careful tuning of the algorithm parameters could potentially yield additional performance improvements.

Core allocations are also shown in Figure 6. Neither application in our scenario can take advantage of additional compute power thus each optimization results in the same core allocations for the applications regardless of video size.

Figure 8 shows same scenario except the computer is now using "battery power" and thus we have increased the importance of application 0 (penalty 2.0). The video applications are still allocated the necessary amount of resources for each video size. However, the allocation of the network hog is significantly reduced to less than 70 kbits/s, and the remaining resources are left idle.

## 5. Dynamic Penalty Optimization

ORADIA's penalty optimization algorithm dynamically decides resource allocations. The algorithm can be run periodically, when applications start or stop, when an application changes phase or when the system changes operating scenarios. One of the advantages of convex optimization is that it enables fast, incremental solutions. As shown in our experiments, the algorithm can terminate earlier to decrease overhead and still be moving towards an optimal solution as it runs.

Convex optimization is simplest when it is unconstrained, so we reformulated ORADIA's construction to be unconstrained. Extending the response time model functions to all of $\Re^n$

moves the requirement that allocations must be positive into the objective function, and introducing application 0 for slack resources turns the affine inequalities into equalities:

$$\begin{aligned} \text{Minimize} \quad & \sum_{p \in P} \pi_p(\tau_p(a_{p,1} \ldots a_{p,n})) \\ \text{Subject to} \quad & \sum_{p \in P} a_{p,r} = A_r, r = 1, \ldots n \end{aligned}$$

The only remaining constraints are those on the $a_{p,r}$. These can be removed by letting the $a_{p,r}$ be unbounded above for $p \neq 0$ and changing the domain of $\tau_0$ to be the whole resource allocation matrix. The definition of $\tau_0$ might take the form

$$\begin{aligned} \tau_0 &= \sum_r \Delta_r \sum_{p \neq 0} a_{p,r} \\ &= \sum_r \Delta_r (A_r - a_{0,r}) \end{aligned}$$

where $\Delta_r$ is the (constant) power dissipation of one unit of resource $r$. However, if any of the allocations $a_{0,r}$ turns out to be negative then $\tau_0$ should instead return the value $+\infty$.

The penalty optimization algorithm used in ORADIA is gradient descent via backtracking line search along the negative gradient direction [16]. This algorithm rejects and refines any step that yields insufficient relative improvement in the objective function, so infinite values from infeasible allocations will automatically be avoided by the search. The negative gradient $-\nabla\pi$ of the overall objective function $\pi$ with respect to the resource allocations $a$ is computed analytically from the response time models and penalty functions. When a component of this overall gradient is negative, it means the penalty will be reduced by increasing the associated allocation if possible. The gradient search at the boundaries of the feasible region must ignore components that lead in infeasible directions; these can be detected by noting whether for some $p$ and $r$, $a_{p,r} = 0$ with $(-\nabla\pi)_{p,r} > 0$. In such cases, the associated step component is set to zero.

We added an additional optimization to move along boundaries more rapidly in the scenario when a completely allocated resource had a large gradient. We scale all the allocations of that resource type down to satisfy resource constraint while leaving the allocations of other resources untouched.

The rate of convergence of gradient descent depends on how well the sub-level sets of the objective function are conditioned (basically, how "spherical" they are). Conditioning will improve if resource allocation units are scaled to make their relative effects similar. For example, when compared with processor allocation units, memory allocation units of 4 MB are probably a better choice than 4 KB. In addition, penalty function slopes should not differ by more than perhaps two orders of magnitude. If these measures prove insufficient, stronger preconditioners can be used. Our implementation conditions all resource allocations to be in the range of 0-100.

## 6. Response Time Functions

In this section, we describe the design of ORADIA's RTFs in more detail. RTFs describe an application's performance given its resource assignments. These functions capture information about the performance impact of a particular resource to an application on the current hardware at a particular time. Without this information, it would be difficult for any resource allocation system to make informed decisions short of blindly trying a variety of allocations and picking the best one.

**Modeling versus Interpolating** While it might have been possible to model response times by recording past values and interpolating among them, this idea has serious shortcomings:

- The multidimensional response time tables would be large;
- Interpolation in many dimensions is computationally expensive;
- The measurements will be noisy and require smoothing;
- Convexity in the resources may be violated;
- Gradient estimation will be slow and difficult.

Instead of interpolating, ORADIA 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. Their gradients, needed by ORADIA to solve the optimization problem efficiently, are easy to calculate.

ORADIA models response times with functions that are convex by construction. The specific function chosen for ORADIA is shown in Equation 1 above. 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 [66]. For example, one term might model instructions executed divided by total processor MIPS; another might model network accesses divided by bandwidth, and so forth.

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. The models are also linear in the weights.

**Non-Convexity** Forcing RTFs to be convex assumes that the actual response times are close to convex. We find this to be a plausible requirement as applications usually follow the "Law of Diminishing Returns" for resource allocations.

However, there are examples of response time versus resource behavior that violate convexity. For example, we have seen non-convex performance in applications when dealing with hyperthreads or memory pages. For two of our applications, 5 hyperthreads resulted in significantly worse performance than either 4 or 6. 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. Outliers and additional challenges to response time modeling are discussed in Section 7.

Another kind of convexity violation can occur in memory allocation, where "plateaus" can sometimes occur as in Figure 9. Such plateaus are typically caused by adaptations within the application (*e.g.,* adjusting the algorithm or output quality). 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 10 and multiple extrema in the optimization problem will be a likely result.

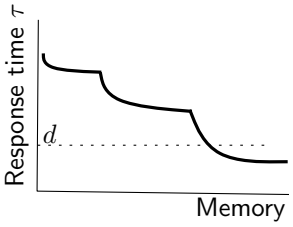


**Figure 9: Response time function with some resource "plateaus".**

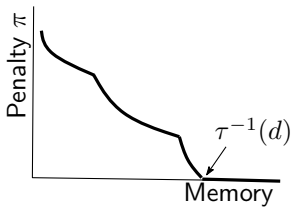


**Figure 10: Net effect of the resource plateaus on the application penalty.**

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 $\tau$. Either approach adds significant computational cost, and we found that our simple convex models still resulted in high-quality resource allocations. Thus we chose not to implement any of these approaches. Alternative approaches to handling non-convex behavior are described in [3].

**Data Collection and Model Creation Time** 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 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 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 model and improve it over time. The remainder of this Section describes our model creation process in detail.

### 6.1. Response Time Model Creation

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 [11], to select the resource vectors to use for training. These vectors and their response times are fed into the convex least-squares algorithm. Online model building uses data from the application's response time history.

**Least-Squares Minimization** After enough measurements, discovery of the model parameters $w$ that define the function $\tau$ can be based on a solution to the 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: a least-squares solution accomplished via *QR factorization* [32] of $D$ will determine the $w$ that minimizes the residual error of $\|Dw-t\|_2^2 = \|Rw-Q^T t\|_2^2$. 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 $t$.

### 6.2. On-line Response Time Modeling

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. ORADIA uses an incremental approach described below to replace old data and efficiently update RTFs.

**Incremental Least-Squares** What is needed is a factorization $\tilde{Q}\tilde{R}$ of a new matrix $\tilde{D}$ derived from $D$ by dropping a row, perhaps from the bottom, and adding a row, perhaps at the top. 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 [32] 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 ORADIA 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 [19]. However since ORADIA's online model maintenance calls for incremental downdates and updates to rows of $Q^T$, $Q^T t$ and $R$, the NNLS problem is handled with a scheme based on the *active-set* method [50] that also downdates and updates the *columns* of $R$ incrementally, roughly in the spirit of Algorithm 3 in [52]. However, ORADIA'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, $24n^2 + 21n + O(1)$ 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 MOS.

**Model 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. ORADIA exploits the latter idea and simply avoids downdating any row that will make $R$ rank-deficient.

**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 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; ORADIA 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 $R$.

**Power Response Modeling** Recall that we manage power and battery energy with an artificial application named application 0 which receives all resources not allocated to other applications. Application 0's "response time" function is similar to the other applications' RTFs. The function inputs are resource allocations just as with the other applications. However, the function output is system power rather than response time. To create the RTF, system power can be measured directly from on-chip energy counters in systems where they are available or from a power meter. These models can be built in advance, during a training phase or online while the system runs, just as with the application RTFs. Alternatively, the model could be part of the operating system's platform-specific information.

Although system power may not be perfectly convex in reality, forcing it to be convex is reasonable because idling a resource should not increase power. As a result, application 0 still fufills its purpose of keeping applications from using additional resources that have poor performance/power ratios.

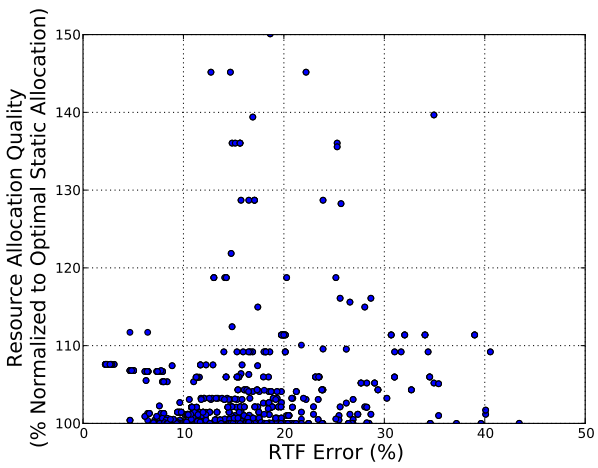


**Figure 11: Effect of Model Accuracy on Decision Quality. The x axis represents the combined relative error of all RTFs used in the decision.**

## 7. Discussion

There are two main sources of challenges for ORADIA's design: performance non-convexity and performance variability. In the following section we describe possible techniques for coping with these challenges.

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 ORADIA 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, once a model crossed a certain threshold of accuracy then better models provided insignificant improvement in resource allocation decisions [3]. Figure 11 shows the effect of model accuracy on the quality of the resource allocation decisions made using the RTF model in Equation 1. 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 ORADIA's model-based design to be feasible

in a noisy system with real applications.

### 7.1. Outliers and Performance Non-Convexity

Outliers and performance non-convexity can be handled with a combination of two techniques. Outliers should be thrown out during the model creation phase as described in Section 6 to prevent those points from distorting the accuracy of the other allocations. The second part of the solution is to keep track of these points with extreme error in the model and use heuristics to adjust the resource allocations to avoid such points. We did not implement this second idea in our current evaluation, but it is a subject of future study. In our study, we experienced outliers for a few applications with a particular number of hyperthreads and for many applications when given only one cache way. However, as responsiveness, predictability, and efficiency increase in importance for systems, we expect to see an increased number of chip designs that provide more performance convexity.

### 7.2. Variability

We observe three main sources of variability in application performance: phase changes, performance changes due to differing inputs, and variable resource performance due to external causes or interference from sharing.

**Phases** As described in Section 6, phases can be handled with online creation of the RTFs. However, another option is to build different models, one per phase, and swap models when a change occurs. We used this method for the video application in Section 4. This approach has the advantage that it can more rapidly adapt to phases; however it requires identifying phases and additional space to store the extra models. Phase detection is an active area of research, and [24] provides an overview of techniques. Another possible approach is to build a model that represents the resource requirements of the most demanding phase. The system can be designed make use of the idle resources when available or power management mechanisms can put them in low-power mode.

**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. However for other applications the effect may be more pronounced. As with phases, a solution might be to keep multiple models for the application and select one based on the current input. A different solution is to make the RTFs *stochastic models* representing the distribution of response times of the application. Stochastic models for ORADIA are discussed in more detail in [3].

**Resource Variability** In our study we experience resource variability both in the form of non-deterministic resources such as the network connection in `tradebeans` or dynamic frequency changes in the CPU and shared resources such as memory bandwidth. 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 of static allocation, we built the models in isolation but found that ORADIA was still able to make near-optimal resource allocations for a loaded machine despite the shared resources.

While there will likely always be some shared resources, we have found a trend towards minimizing interference in emerging chip designs, as efficiency and predictability begin to trump utilization as primary concerns.

## 8. Related Work

### 8.1. Batch Scheduling and Cluster Management

Classic resource management systems were designed for batch scheduling [27, 29]. Like ORADIA, batch scheduling systems can allocate multiple resource types and rely on a gang-scheduling model [28]. However, they tend to use queues and priorities to schedule jobs while trying to keep all resources busy. Resource allocations are always the user's responsibility to specify and pay for. Responsiveness can be improved by buying a higher priority. Few batch systems incorporate deadlines, but one such is [1].

Modern cluster resource management has a similar flavor. In systems such as Amazon EC2 [2], Eucalyptus [58], and Condor [64], users must specify their resource requirements. Other systems such as Hadoop [4,5,75] and Quincy [42] use a fairness policy to assign resources. Mesos [36] 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 [31], a generalization of max-min fairness to multiple resources types, has been implemented in Mesos.

### 8.2. Model-Based Scheduling and Allocation Frameworks for SLAs and Soft Real-Time Requirements

Research is growing in the cloud computing and realtime communities, which use application-specific performance "models" to try to schedule to meet deadlines or SLAs.

Much of this research has been in autonomic computing [49, 56, 70, 71]. 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. A central arbiter maximizes total utility. The utility functions are not necessarily concave, so the arbiter must use reinforcement learning or combinatorial search to make allocations. Walsh *et al.* [72] 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 ORADIA, but since the objective functions are discrete and non-convex their optimization is difficult. A survey of autonomic systems research appears in [38].

Rajkumar *et al.* [63] 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 ORADIA, 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.

Several systems use a feedback-driven reactive approach to resource allocation where a control loop or reinforcement learning adjusts allocations continuously. AcOS [8] and Metronome [65] feature hardware-thread based maintenance of "heart rate" targets using adaptive reinforcement learning. Bodik *et al.* [15] builds online performance models like ORADIA. Initially, it uses an *exploration policy* that avoids nearly all SLA violations while it builds the model; later, it shifts to a controller based on the model it has built. 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.

Jockey [30] has some similarities to ORADIA: 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.

Other systems base decisions on user-provided resource specifications and a real-time scheduling algorithm. In the Redline system [73], compact resource requirement specifications written by hand to guarantee response times are scheduled Earliest-Deadline-First.

Much research focuses on measuring resource usage to make smart co-scheduling decisions. For example, Calandrino *et al.* [17] uses working set sizes to make co-scheduling decisions and enhance soft real-time behavior. Merkel and Bellosa [55] 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.

Some frameworks can support multiple scheduling and resource allocation policies. Guo *et al.* [34] 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 ORADIA, they argue that targets should be expressed in terms of capacity requirements rather than rates or times.

Nesbit *et al.* [57] 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). 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 ORADIA's at a high level, but no design or implementation of the modeling, translation, or scheduling components is presented [57].

### 8.3. Resource Partitioning and QoS

ORADIA 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. 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 [14, 48, 51], while others [9, 10, 78] have concentrated on CPU fairness.

Hardware partitioning research, which has largely focused on caches, provides mechanisms based on policies baked into the hardware, not the flexible allocations ORADIA requires [18, 20, 25, 39, 54, 62, 68, 69, 76, 77]. Early work focused on providing adaptive, fair policies that ensure equal performance degradation [47, 74], not guarantees of responsiveness. More recent proposals have incorporated more sophisticated policy management [33, 34, 37, 44]. Iyer [43] suggests a priority-based cache QoS framework, CQoS, for shared cache way-partitioning. However, simultaneous achievement of performance targets as in ORADIA is not addressed.

## 9. Conclusion

ORADIA takes a different approach to resource allocation than traditional systems, relying heavily on application-specific functions built through measurement and convex optimization. Using convex optimization lets ORADIA perform real-time resource allocation inexpensively, enabling ORADIA to dynamically allocate resources to adjust to the changing state of the the system. ORADIA makes resource allocation decisions in less than 350 µs in the worst case and often faster than 50 µs in our manycore OS implementation. By building application-specific functions online and formulating resource allocation as an optimization problem, ORADIA 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. Allocation decisions are near optimal—only 2% from the best possible allocation on average, and ORADIA only requires a few hundred bytes of additional storage per application.

## References


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

[2] Amazon.com, "Ec2," http://aws.amazon.com/ec2.

[3] Anonymized, "Oradia: Optimizing resource allocations for dynamic interactive applications," Anonymized, Tech. Rep., 2013.

[4] Apache.org, "Hadoop capacity scheduler," http://hadoop.apache.org/common/docs/r0.20.2/capacity_scheduler.html.

[5] Apache.org, "Hadoop fair scheduler," http://hadoop.apache.org/common/docs/r0.20.2/fair_scheduler.html.

[6] Apple Inc., "iOS App Programming Guide," http://developer.apple.com/library/ios/DOCUMENTATION/iPhone/Conceptual/iPhoneOSProgrammingGuide/iPhoneAppProgrammingGuide.pdf.

[7] L. A. Barroso and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, ser. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2009.

[8] D. 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)*, 2012.

[9] S. K. Baruah *et al.*, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, pp. 600–625, 1996.

[10] S. K. Baruah, J. E. Gehrke, and C. G. Plaxton, "Fast scheduling of periodic tasks on multiple resources," in *In Proceedings of the 9th International Parallel Processing Symposium*, 1995, pp. 280–288.

[11] S. J. Bates, J. Sienz, and D. S. Langley, "Formulation of the audze–eglais uniform latin hypercube design of experiments," *Adv. Eng. Softw.*, vol. 34, no. 8, pp. 493–506, 2003.

[12] C. Bienia *et al.*, "The parsec benchmark suite: Characterization and the parsec benchmark suite: Characterization and architectural implications," Princeton University, Tech. Rep. TR-811-08, January 2008.

[13] S. M. Blackburn *et al.*, "The DaCapo benchmarks: Java benchmarking development and analysis," in *OOPSLA*, 2006, pp. 169–190.

[14] J. M. Blanquer and B. Özden, "Fair queuing for aggregated multiple links," in *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '01. New York, NY, USA: ACM, 2001, pp. 189–197. Available: http://doi.acm.org/10.1145/383059.383074

[15] P. Bodik *et al.*, "Automatic exploration of datacenter performance regimes," in *Proc. ACDC*, 2009.

[16] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, England: Cambridge University Press, 2004.

[17] J. M. Calandrino and J. H. Anderson, "On the design and implementation of a cache-aware multicore real-time scheduler," in *ECRTS*, 2009, pp. 194–204.

[18] J. Chang and G. S. Sohi, "Cooperative cache partitioning for chip multiprocessors," in *Proc. ICS '07*. New York, NY, USA: ACM, 2007, pp. 242–252.

[19] D. Chen and R. J. Plemmons, "Nonnegativity constraints in numerical analysis," Lecture presented at the symposium to celebrate the 60th birthday of numerical analysis, Leuven, Belgium, October 2007.

[20] S. Cho and L. 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.

[21] G. Contreras and M. 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.

[22] I. CVX Research, "CVX: Matlab software for disciplined convex programming, version 2.0 beta," http://cvxr.com/cvx, Sep. 2012.

[23] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. Available: http://doi.acm.org/10.1145/1327452.1327492

[24] A. S. Dhodapkar and J. E. Smith, "Comparing program phase detection techniques," in *Proc. MICRO 36*. Washington, DC, USA: IEEE Computer Society, 2003, p. 217.

[25] H. Dybdahl and P. 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.

[26] S. Eranian, "Perfmon2: a flexible performance monitoring interface for linux," in *Ottawa Linux Symposium*, 2006, p. 269–288.

[27] D. G. Feitelson, "Job scheduling in multiprogrammed parallel systems," 1997.

[28] D. G. Feitelson and L. Rudolph, "Gang scheduling performance benefits for fine-grain synchronization," *J. Parallel Distrib. Comput.*, vol. 16, no. 4, pp. 306–318, 1992.

[29] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, "Parallel job scheduling - a status report," in *JSSPP*, 2004, pp. 1–16.

[30] A. D. Ferguson *et al.*, "Jockey: guaranteed job latency in data parallel clusters," in *Proceedings of the 7th ACM european conference on Computer Systems*, ser. EuroSys '12. New York, NY, USA: ACM, 2012, pp. 99–112. Available: http://doi.acm.org/10.1145/2168836.2168847

[31] A. 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*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 24–24. Available: http://dl.acm.org/citation.cfm?id=1972457.1972490

[32] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed. Baltimore, Maryland: Johns Hopkins University Press, 1996.

[33] F. Guo *et al.*, "From chaos to qos: case studies in cmp resource management," *SIGARCH Comput. Archit. News*, vol. 35, no. 1, pp. 21–30, 2007.

[34] F. 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.

[35] J. L. Hennessy and D. A. Patterson, *Computer Architecture - A Quantitative Approach (5. ed.)*. Morgan Kaufmann, 2012.

[36] B. 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*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 22–22. Available: http://dl.acm.org/citation.cfm?id=1972457.1972488

[37] L. R. Hsu *et al.*, "Communist, utilitarian, and capitalist cache policies on cmps: caches as a shared resource," in *Proc. PACT '06*. New York, NY, USA: ACM, 2006, pp. 13–22.

[38] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing—degrees, models, and applications," *ACM Comput. Surv.*, vol. 40, no. 3, pp. 1–28, 2008.

[39] J. Huh *et al.*, "A nuca substrate for flexible cmp cache sharing," in *Proc. ICS '05*. New York, NY, USA: ACM, 2005, pp. 31–40.

[40] Intel Corp., "Intel 64 and ia-32 architectures optimization reference manual," June 2011.

[41] Intel Corp., "Intel 64 and ia-32 architectures software developer's manual," March 2012.

[42] M. Isard *et al.*, "Quincy: fair scheduling for distributed computing clusters," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 261–276. Available: http://doi.acm.org/10.1145/1629575.1629601

[43] R. Iyer, "Cqos: a framework for enabling qos in shared caches of cmp platforms," in *Proc. ICS '04*. New York, NY, USA: ACM, 2004, pp. 257–266.

[44] R. Iyer *et al.*, "Qos policies and architecture for cache/memory in cmp platforms," in *Proc. SIGMETRICS '07*. New York, NY, USA: ACM, 2007, pp. 25–36.

[45] A. Jaleel, "Memory characterization of workloads using instrumentation-driven simulation – a pin-based memory characterization of the spec cpu2000 and spec cpu2006 benchmark suites," VSSAD, Intel Corporation, Tech. Rep., 2007.

[46] S. Kamil, "Stencil probe," 2012, http://www.cs.berkeley.edu/~skamil/projects/stencilprobe/.

[47] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," in *Proc. ASPLOS-X*. New York, NY, USA: ACM, 2002, pp. 211–222.

[48] J. Kleinberg, Y. Rabani, and E. Tardos, "Fairness in routing and load balancing," in *J. Comput. Syst. Sci*, 1999, pp. 568–578.

[49] S. Kounev, R. Nou, and J. Torres, "Autonomic qos-aware resource management in grid computing using online performance models," in *Proc. ValueTools '07*. ICST, 2007, pp. 1–10.

[50] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*. Englewood Cliffs, NJ: Prentice Hall, 1974.

[51] Y. 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, pp. 1106–1115 vol.2.

[52] Y. Luo and R. Duraiswami, "Efficient parallel nonnegative least squares on multicore architectures," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2848–2863, 2011.

[53] Mathworks, "Matlab 2009b," http://www.mathworks.com/, 2009.

[54] J. Merino *et al.*, "Sp-nuca: a cost effective dynamic non-uniform cache architecture," *SIGARCH Comput. Archit. News*, vol. 36, no. 2, pp. 64–71, 2008.

[55] A. Merkel and F. Bellosa, "Task activity vectors: a new metric for temperature-aware scheduling," in *Proc. Eurosys '08*. New York, NY, USA: ACM, 2008, pp. 1–12. Available: http://portal.acm.org/ft_gateway.cfm?id=1352594

[56] M. N. Bennani and D. 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.

[57] K. J. Nesbit *et al.*, "Multicore resource management," *IEEE Micro*, vol. 28, no. 3, pp. 6–16, 2008.

[58] D. 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*, ser. CCGRID '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 124–131. Available: http://dx.doi.org/10.1109/CCGRID.2009.93

[59] H. Pan, B. Hindman, and K. Asanović, "Lithe: Enabling efficient composition of parallel libraries," in *HotPar09*, Berkeley, CA, 03/2009 2009. Available: http://www.usenix.org/event/hotpar09/tech/

[60] "Perfmon2 webpage," perfmon2.sourceforge.net/.

[61] A. Phansalkar, A. Joshi, and L. K. John, "Analysis of redundancy and application balance in the spec cpu2006 benchmark suite," in *ISCA*, 2007, pp. 412–423.

[62] M. K. Qureshi and Y. 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.

[63] R. Rajkumar *et al.*, "A resource allocation model for qos management," in *Proc. RTSS '97*. Washington, DC, USA: IEEE Computer Society, 1997, p. 298.

[64] R. Raman, M. Livny, and M. Solomon, "Matchmaking: An extensible framework for distributed resource management," *Cluster Computing*, vol. 2, no. 2, pp. 129–138, Apr. 1999. Available: http://dx.doi.org/10.1023/A:1019022624119

[65] F. Sironi *et al.*, "Metronome: operating system level performance management via self-adaptive computing," in *Proceedings of the 49th Annual Design Automation Conference*, ser. DAC '12. New York, NY, USA: ACM, 2012, pp. 856–865. Available: http://doi.acm.org/10.1145/2228360.2228514

[66] A. Snavely *et al.*, "A framework for performance modeling and prediction," in *SC*, 2002, pp. 1–17.

[67] Standard Performance Evaluation Corporation, "SPEC CPU 2006 benchmark suite," http://www.spec.org.

[68] G. E. Suh, L. Rudolph, and S. Devadas, "Dynamic partitioning of shared cache memory," *J. Supercomput.*, vol. 28, no. 1, pp. 7–26, 2004.

[69] G. E. Suh, S. Devadas, and L. 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.

[70] G. Tesauro, W. E. Walsh, and J. O. Kephart, "Utility-function-driven resource allocation in autonomic systems," in *Proc. ICAC '05*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 342–343.

[71] G. Tesauro *et al.*, "On the use of hybrid reinforcement learning for autonomic resource allocation," *Cluster Computing*, vol. 10, no. 3, pp. 287–299, 2007.

[72] G. Tesauro and J. O. Kephart, "Utility functions in autonomic systems," in *Proc. ICAC '04*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 70–77.

[73] T. 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*, ser. OSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 73–86. Available: http://dl.acm.org/citation.cfm?id=1855741.1855747

[74] T. Y. Yeh and G. Reinman, "Fast and fair: data-stream quality of service," in *Proc. CASES '05*. New York, NY, USA: ACM, 2005, pp. 237–248.

[75] M. 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*, ser. EuroSys '10. New York, NY, USA: ACM, 2010, pp. 265–278. Available: http://doi.acm.org/10.1145/1755913.1755940

[76] M. Zhang and K. 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.

[77] L. Zhao *et al.*, "Towards hybrid last level caches for chip-multiprocessors," *SIGARCH Comput. Archit. News*, vol. 36, no. 2, pp. 56–63, 2008.

[78] D. Zhu, D. Mossé, and R. Melhem, "Multiple-resource periodic scheduling problem: how much fairness is necessary?" in *Proceedings of the 24th IEEE International Real-Time Systems Symposium*, ser. RTSS '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 142–. Available: http://dl.acm.org/citation.cfm?id=956418.956616