PACORA: Optimizing Resource Allocations for Dynamic Interactive Workloads

Authors Redacted

Abstract

Users have an insatiable appetite for responsive user interfaces and high-quality multimedia with stringent real-time guarantees on their multicore devices, and they expect better performance or responsiveness as the core count increases. Meeting these expectations requires not only parallelizing client applications but striking the right balance of resources among competing software components. 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 QoS requirements is a complex optimization problem, and modern applications frequently extend into the cloud, requiring responsiveness and performance predictability on a global scale, which adds additional complexity. We present PACORA, a resource allocation framework for general-purpose operating systems, which is designed to provide responsiveness guarantees to a simultaneous mix of high-throughput parallel, interactive, and real-time applications in an efficient, scalable manner.

1. Introduction

Users have an insatiable appetite for responsive user interfaces and high-quality multimedia with stringent real-time guarantees on their multicore devices, and they expect better performance or responsiveness as the core count increases. Meeting these expectations requires not only parallelizing client applications but striking the right balance of resources among competing software components. Additionally, modern applications frequently extend into the cloud, requiring responsiveness and performance predictability on a global scale, which adds complexity.

Ideally, applications with strict performance requirements should be given just enough system resources (*e.g.,* processor cores, cache slices, memory pages, various kinds of bandwidth) to meet these requirements consistently, without unnecessarily siphoning resources from other applications. However, executing multiple parallel, real-time applications while satisfying *Quality-of-Service* (QoS) requirements is a complex optimization problem.

Consequently, predictability has traditionally been obtained at a significant expense by designing for the worst case and over-provisioning. Evidence of this behavior can be found in current mobile and cloud systems. In order to preserve responsiveness and battery life, some mobile systems have gone so far as to limit which applications can run in the background [2], despite the obvious concerns this raises for user experience. Cloud computing providers routinely utilized their clusters at only 10% to 50% to keep the system responsive despite the additional operational costs of consuming electricity and the significant impact to the capital costs of the infrastructure [3, 26].

Historically operating systems have not provided useful mechanisms that implement stronger performance guarantees, so developers have been left with few alternatives to over-provisioning. Resource allocation has been rather unsystematic making it difficult to reason about the expected response time of an application. Responsiveness has been described by a single value (usually called a *priority*) associated with a thread of computation and adjusted within the operating system by a variety of ad-hoc mechanisms. Other shared resources either employ independent machinery (*e.g.,* memory, caches), or are deemed so abundant as to require no explicit management at all (*e.g.,* I/O, network bandwidth).

The assumptions underlying strategies of this sort no longer hold, especially for emerging client systems. The value of application responsiveness is highly nonlinear for an increasing variety of applications like streaming media or gaming; for these real-time applications, performance is measured as sufficient if the deadline is met and insufficient otherwise. Priority approaches have no mechanism to understand deadlines or the resources required to meet a deadline and as such must run the highest priority applications as fast as possible on all the resources requested. As a result, interactive and realtime applications are often run needlessly fast with significantly over-provisioned resources –wasting power and energy and preventing other applications from using the resources.

Alternatively, by understanding and effectively adapting to application requirements, the OS can provide predictable behavior without over-provisioning, allowing excess resources to be turned off or to be used opportunistically –gaining efficiency. In this paper, we present PACORA, a resource allocation framework for general-purpose operating systems, which is designed to provide responsiveness guarantees to a simultaneous mix of high-throughput parallel, interactive, and real-time applications in an efficient, scalable manner. PACORA leverages convex optimization and application performance models to determine the optimal amount of each resource to give each application, enabling the OS to make trade-offs between application QoS/responsiveness, system performance, and energy efficiency.

Add Performance Numbers

2. PACORA Architecture

PACORA, which stands for Performance-Aware Convex Optimization for Resource Allocation, is a framework designed to determine the proper amount of each resource type to assign to each application. For our purposes an application is the entity to which the operating system allocates resources: it can viewed as a complete application (*e.g.,* a video player) or a component of an application (*e.g.,* a music synthesizer) or a process (*e.g.,* indexing).

PACORA is designed to work in systems with hierarchical scheduling. In a client system, this design just looks like classic two-level scheduling. PACORA makes allocation decisions about resources (*e.g.,* cores and memory pages), and micro-management of the resources within an application is left to user-level runtimes such as User Mode Scheduling in Windows [**?**] or Lithe [45] and memory managers. In a cloud environment, PACORA allocates resources (*e.g.,* nodes and storage) to jobs, and scheduling is left to other entities such as the MapReduce framework [16] or the operating system on the node. Separating resource allocation from scheduling enables the use of application-specific scheduling policies, which have the potential to easier to design and more efficient than a general-purpose scheduler that has to work for everything.

PACORA formulates resource allocation as an optimization problem built from two types of application-specific functions. **why?** The following sections describe each of the functions and the optimization construction.

Response Time Functions

Response Time Functions are built from data on

the *response time*, is an appropriate measure of the performance of the process. For example,the response time of a process might be one of these:

The time from a mouse click to its result;

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 response time of a process is a function of the allocated resources and is predicted from its history of resource usage.

Penalty Functions

The penalty of a process is a function rather than a single value and the argument of each penalty function

Resource Allocation As Optimization

PACORA formulates resource allocation as an optimization problem

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

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

Here π_p is the penalty function for process p, τ_p is its response time function, $a_{p,r}$ is the allocation of resource r to process p, and A_r is the total amount of resource r available.

Convex Optimization

If the penalty functions, response time functions, and resource constraints were arbitrarily, little could be done to optimize the total penalty beyond searching at random for the best allocation. However, if resource management can be framed as a *convex optimization problem* [9], two benefits will accrue: an optimal solution will exist without multiple local extrema and fast, incremental solutions will become feasible.

A constrained optimization problem will be 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 θ 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 θ between 0 and 1. If F is differentiable, it is convex if its domain is an open convex set and $F(y) \geq F(x) + \nabla F^T(y-x)$ where ∇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:

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

A few more 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).

As a consequence, the resource management problem posed above can be transformed into a convex optimization problem

in the $m = |P| \cdot n$ variables $a_{p,r}$ as long as the penalty functions π_p are convex non-decreasing and the response time functions τ_p are convex. Note that the resource constraints are all affine and can be rewritten as $\sum_{p \varepsilon P} a_{p,r} - A_r \leq 0$ and $-a_{p,r} \leq 0$.

Resources

3. Application Functions

Penalty Functions

Penalty functions are generically defined as members of a family of such functions so that user preferences for a process p (elided in the discussion below) can be implemented by assigning values to a few well-understood parameters. As a process grows or diminishes in importance, its penalty function can be modified accordingly. In a client operating system,the instantaneous management of penalty function modifications should be highly automated by the system to avoid unduly burdening the user. PACORA's penalty functions are non-decreasing piecewise linear functions of the form $\pi(\tau) = \max(0, s(\tau - d)$. Two representative graphs of this type appear in Figures 1 and 2.

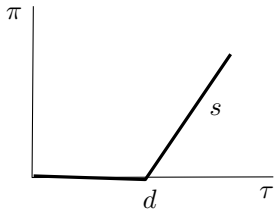

Figure 1: A penalty function with a response time constraint.

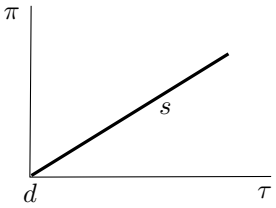

Figure 2: Penalty function without such a constraint.

The two parameters d and s define the penalty function. To guarantee it is convex and non-decreasing, s must be non-negative. The response time τ is of course non-negative, and it may be sensible (if not strictly necessary) to convene that d is also. A response time constrained process has a marked change in slope, namely from 0 to s, at the point $\tau = d$. In the most extreme case $s = \infty$ (implying infinite penalty for the system as a whole when $\tau > d$). "Softer" requirements will doubtless be the rule. For processes without response time constraints one can set $d = 0$. This defines linear behavior with s as the rate of penalty increase with response time.

The gradient of process penalty with respect to its resource allocations is useful in controlling the optimization algorithm. By the chain rule, $\partial\pi/\partial a_r = \partial\pi/\partial\tau \cdot \partial\tau/\partial a_r$. The first term is well-defined but discontinuous at $\tau = d$ with $\partial\pi/\partial\tau =$ if $(\tau - d) \leq 0$ then 0 else s. The problem of estimating the partial derivatives $\partial\tau/\partial a_r$ is dealt with below.

Response Time Functions

Unlike penalty functions, which describe user preference, a response time function describes process performance based on its resource assignments. Response time will commonly vary with time as a process changes phase and makes better or worse use of its resources. To guarantee the objective function is convex, the response time must be also; this is a plausible requirement akin to the proverbial "Law of Diminishing Returns".

Besides the value of the response time function, its gradient or an approximation to it is useful to estimate the relative response time improvement from each type of resource. A user-level runtime scheduler that schedules work internal to the process may be a good source of data. Additionally, the resource manager can allocate a modest amount of a resource and measure the change in response time. Instead of these, PACORA maintains a parameterized analytic response time model with the partial derivatives evaluated from the model *a priori*.

There are examples of response time versus resource behavior that violate convexity. One such example sometimes occurs in memory allocation, where "plateaus" can sometimes be seen as in Figure 3. Such plateaus are typically caused by algorithm adaptations within the process to accommodate variable resource availability. The response time is really the *minimum* of several convex functions depending on allocation and the point-wise minimum that the process implements fails to preserve convexity. The effect of the plateaus will be a non-convex penalty as shown in Figure 4 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*. 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 ℓ and replacing the objective function with a convex constraint function whose sublevel set S_ℓ is the same as that of f. Next, one determines whether there is a feasible solution for that particular threshold ℓ. Repeated application with a binary search on ℓ will reduce the level of feasibility until the solution is approximated well enough.

Another idea is to use additional constraints to explore convex sub-domains of τ. For example,the affine constraint

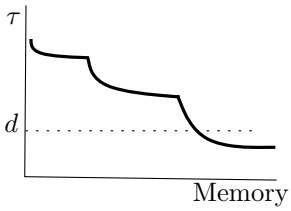

Figure 3: Response time function with some resource "plateaus".

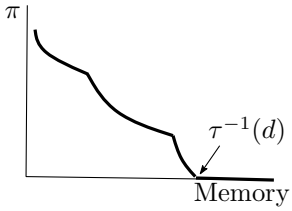

Figure 4: Net effect of the resource plateaus on the process penalty.

$a_{p,r} - \mu \leq 0$ excludes process p from any assignment of resource r exceeding μ. Similarly, $\mu - a_{p,r} \leq 0$ exludes the opposite possibility. A binary (or even linear)search of such sub-domains could be used to find the optimal value.

PACORA adopts a simpler idea, modeling response times by functions that are convex by construction and do not distort response time behavior too much. This approach is developed more fully below.

Power and Battery Energy

It is useful to designate a "process" to receive allocations of all resources that are not used elsewhere and are therefore to be powered off if possible. In PACORA, process 0 plays this role. The "response time" for process 0, τ_0, is artificially defined to be the total system power consumption. This response function is affine and monotone nonincreasing in its arguments $a_{0,r}$.

The penalty function π_0 can now be used to keep total system power below the parameter d_0 to the extent the penalties of other processes 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. As the battery depletes, π_0 can be used to force other processes to slow or cease execution. In any event, the slope s_0 establishes a system tradeoff between power and performance that will determine which processor cores are used for each process and which cores are left idle.

This introduction of *slack* resource allocations into Process 0 turns the resource bounds into equalities:

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

Response Time Modeling

While it might be possible to model response times by recording past values and interpolating among them, this idea has serious shortcomings:

- The size of the multidimensional response time function tables will 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 difficult.

An alternative approach is to model the response time functions using parameterized expressions that are convex by construction. For example, the response time might be modeled as a weighted sum of component terms, one per bandwidth resource, where each term w_r/a_r is the amount of work $w_r \geq 0$ divided by a_r, the allocation of that bandwidth resource [51]. For example, one term might model the number of instructions executed divided by total processor MIPS, another might model storage accesses divided by storage bandwidth allocation and so forth. Such models will automatically be convex in the allocations because $1/a$ is convex for positive a and because a positively-weighted sum of convex functions is convex.

It is obviously important to guarantee the positivity of the resource allocations. This can be enforced as the allocations are selected during penalty optimization, or the response time model can be made to return ∞ 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 $\Re^n$.

Asynchrony and latency tolerance may make response time components overlap partly or fully; if the latter, then the maximum of the terms might be more appropriate than their sum. The result will still be convex, though, as will any other norm including the 2-norm, *i.e.* the square root of the sum of the squares. This last variation could be viewed as a "partially overlapped" compromise between the 1-norm (sum) describing no overlap and the ∞-norm (maximum) describing full overlap.

This scheme also accommodates non-bandwidth resources such as memory, the general idea being to roughly approximate "diminishing returns" in the response time with increasing resources. For clarity's sake, rather than using a_r indiscriminately for all allocations, we will denote an allocation of a bandwidth resource by b_r and of a memory resource by m_r.

Sometimes a response time component might be better modeled by a term involving a combination of 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}. Such a model could use the geometric mean of the two allocations in the denominator, *viz.* $w_{r1,r2}/\sqrt{b_{r1} \cdot m_{r2}}$, without compromising convexity.

This begs the question of how memory affects the response time. The effect is largely indirect. Memory permits exploitation of temporal locality and thereby *amplifies* 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. The effectiveness of cache in reducing bandwidth was studied by H. T. Kung [35], who 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

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

For these expressions to make sense, the argument of α should be dimensionless and greater than 1. Ensuring this might be as simple as letting it be the number of memory resource quanta (*e.g.* hundreds of memory pages) assigned to the process. If a process shows diminishing bandwidth amplification as its memory allocation increases, this can be accommodated:

$$\alpha(m) = \min(c_1\alpha_1(m), c_2\alpha_2(m)),\ c_1, c_2 \geq 0$$

Each bandwidth amplification factor might be described by one of the functions above and included 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 process 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.

The gradient $\nabla\tau$ is needed by the penalty optimization algorithm. Since τ is analytic, generic, and symbolically differentiable it is a simple matter to compute the gradient of τ once the model is defined.

Response Time Model Convexity

We now show that the response time model including the various bandwidth amplification functions is convex in both the bandwidth and memory resources b_r and m_r given any of the possibilities listed above. Since norms preserve convexity, this reduces the question to proving each term in the norm is convex. Since all quantities are positive and both maximum and scaling by a positive constant preserve convexity,

$$\begin{aligned} & w/(b\cdot\min(c_1\alpha_1(m),c_2\alpha_2(m))) \\ = \;& \max(w/(b\cdot c_1\alpha_1(m)),w/(b\cdot c_2\alpha_2(m))). \end{aligned}$$

It only remains to show that $1/(b\cdot\alpha(m))$ is convex in b and m.

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 $1/b$ is log-convex for $b>0$ because $-\log b$ is convex on that domain. In a similar way, $\log(1/\sqrt{b\cdot m}) = -(\log b+\log m)/2$ and $\log m^{-1/d} = -(\log m)/d$ are convex. Finally, $\log(1/\log m)$ is convex because its second derivative is positive for $m>1$:

$$\begin{aligned} \frac{d^2}{dm^2}\log(1/\log m) &= \frac{d^2}{dm^2}(-\log\log m) \\ &= \frac{d}{dm}\left(\frac{-1}{m\log m}\right) \\ &= \frac{1+\log m}{(m\log m)^2}. \end{aligned}$$

Summing up, a response time function for a process might be modeled by the convex function

$$\begin{aligned} \tau(w,b,\alpha,m) &= \sqrt[p]{\sum_j\left(\frac{w_j}{b_j\cdot\alpha_j(m_j)}\right)^p} \\ &= \|d\cdot w\|_p \end{aligned}$$

where the w_j are the parameters of the model (the "quantities of work") to be learned, the components of d satisfy $d_j = 1/(b_j\cdot\alpha_j(m_j))$, the b_j are the allocations of the bandwidth resources, the α_j are the bandwidth amplification functions (also to be learned), the m_j are the allocations of the memory or cache resources that are responsible for the amplifications. This formulation allows the process response time τ to be modeled as the p-norm of the component-wise product of a vector d that is computed from the resource allocation and a learned vector of work quantities w.

4. Dynamic Optimization

On-line Response Time Modeling

For the moment, assume the model norm $p=1$ and suppose several response time measurements have already been made using a variety of resource allocations to begin optimizing the application response time. After enough measurements, discovery of the model parameters w that define the function τ 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 process and D is a matrix whose ith row $D_{i,*}$ contains the reciprocals of the amplified bandwidth allocations that generated the corresponding response time measurement t_i. Estimating w is relatively straightforward: a least-squares solution accomplished via *Q-R factorization* [22] of D will determine the w that minimizes the *r*esidual error $\|t-Dw\|_2 = \|\varepsilon\|_2$. The solution proceeds as follows:

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

It is not always necessary to materialize the orthogonal matrix $Q^T=Q^{-1}$; the individual elementary orthogonal transformations (Householder reflections or Givens rotations) that triangularize R by progressively zeroing out partial columns of D can simultaneously be applied to t. The elements of the resulting vector Q^Tt that correspond to zero rows in R comprise $Q^T\varepsilon$. Since Rw exactly equals the upper part of Q^Tt, 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.

Suppose a different model norm p is desired. If $p=2$, we might 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. This is not the same as minimizing the 4-norm; what is being minimized is $\|t^2-D^2w^2\|_2$.

Incremental Least Squares

As resource allocation continues, more measurements will become available to augment t and D. Moreover, older data

may become a poor representation of the current behavior of the process if its characteristics have changed, presumably as reflected in $Q^T\varepsilon$. 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 in the way described in Section 12 of [22] 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 maintained at 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.

Downdating makes an instructive example. A downdate 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 is still orthogonal, the non-diagonal elements of row dd must also be zero 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 had on Q^T.

It is crucial to select the row pairs and the order of rotations that will preserve the upper triangular structure of R while zeroing almost all of a column of Q^T. Since dd is below the diagonal of R it initially will contain only zeros. It therefore suffices 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 the top down through dd of the modified matrices $Q^T t$ and R and both the rows and columns of the new Q^T are circularly shifted one position down, moving row dd to the top (and column dd of Q^T to the left edge). The following picture 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 process 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 $\tilde{R}$). These rotations are applied not just to R $(= Q^T D)$ but also to $Q^T t$ and of course to Q^T itself.

Non-negativity

To guarantee convexity of the response time model, the solution w to $t \approx QRw$ must have no negative components. If a resource allocation is associated with more than a single w_j or if measured response time increases with the allocation then negative w_j may occur.

The need for non-negative solutions to least-squares linear algebra problems is common, so much so that it has a name: *Non-Negative Least Squares*, or NNLS. There are several well-known techniques [12], but since the method proposed here for 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 complementary scheme that downdates and updates the *columns* of R incrementally, somewhat in the style of Algorithm 3 in [38]. The scheme is too complex to be described here.

Model Rank Preservation

If care is not taken in the allocation process, the rows of R may become linearly dependent to such an extent that its rank is insufficient to determine w. This might be the result of repetitions in resource assignment updates. There are several possible ways to avoid this *rank-deficiency* problem. The characteristics of R depend on both the resource optimization trajectory and the choices made in the downdate-update algorithm. In particular, deciding whether to downdate the bottom row of R or some "younger" row will depend on whether the result would become rank-deficient. This approach decouples allocation optimization from performance model maintenance and places responsibility upon the latter to always keep enough history to determine a w.

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 is particularly easy to discover because dd, the 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.

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 τ and the measured ones t by a factor that increases exponentially with the age $g(i)$ of the ith error ε_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 ith row is then $\eta^{g(i)}$ where η is a constant somewhat greater than 1. The candidate row to downdate is the row with the largest weighted error, *i.e.*

$$dd = \arg\max_i |\varepsilon_i| \cdot \eta^{g(i)}$$

Penalty Optimization

Convex optimization is simplest when it is 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 Process 0 for slack resources turns the affine inequalities into equalities:

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

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 τ_0 to be the whole resource allocation matrix. The definition of τ_0 might take the form

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

where Δ_r is the (constant) power dissipation associated with resource r. However,if any of the allocations $a_{0,r}$ is negative then τ_0 should instead return the value $+\infty$. This modification of the objective function transforms the resource allocation problem to unconstrained convex optimization.

The penalty optimization algorithm used in PACORA is descent via backtracking line search along the negative gradient direction [9]. 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 π 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. Since the only constraint is variable positivity then either the variable or its gradient component will be zero at a solution point; see [9], page 142.

The rate of convergence of gradient descent depends on how well the sublevel 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 on t similar. For example, when compared with processor allocation units, memory allocation units of 4MB 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.

5. Evaluation

[37] [52] [6] [7] [45] [39] [15] [5]

PULSE (Preemptive User-Level SchEduling) Lithe (LIquid THrEads)

5.1. Experimental Methodology

To evaluate PACORA, we use two different implementations of the framework on the same hardware platform. Our offline framework was used to experiment with the accuracy of different types of models and test

5.1.1. Experimental Platform We use a prototype version of Intel's Sandy Bridge x86 processor to collect results on resource allocation and application co-scheduling. By using a real hardware prototype, we are able to run full applications for realistic time scales and workload sizes, while running a standard operating system. The processor is similar to the commercially available client chip, but with additional hardware to support way-based cache partitioning in the LLC.

The Sandy Bridge client chip has four quad-issue out-of-order superscalar cores, each of which supports two hyperthreads using simultaneous multithreading [**?**]. 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. Larger server versions of the same processor family have up to 15 MB of LLC capacity.

The cache partitioning mechanism is way-based and modifies the cache-replacement algorithm. Each core can be assigned a subset of the 12 ways in the LLC. Although all cores can hit on data stored in any way, a core can only replace data stored in one of its assigned ways. Allocation of ways among cores can be completely private, completely shared, or overlapping. Data is not flushed when the way allocation changes; newly fetched data will just be written into one of the assigned ways according to the updated allocation configuration.

We use a customized BIOS that enables the cache partitioning mechanism,

To measure on-chip energy, we use the energy counters available on Sandy Bridge to collect the energy used by 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.

5.1.2. Offline Implementation and run unmodified Linux-2.6.36 for all of our experiments. We use the Linux `taskset` command to pin each application to subsets of the available HyperThreads.

To measure application performance, we use the `libpfm` library [**?**, **?**], built on top of the `perf_events` infrastructure introduced in Linux 2.6.31, to access various performance-monitoring counters available on the machine [**?**].

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

5.2. Description of Workloads

We built our workload using a wide range of codes from three different popular benchmark suites: SPEC CPU 2006 [52], DaCapo [**?**], and PARSEC [**?**]. We included some additional application benchmarks to broaden the scope of the study, and some microbenchmarks that exercise certain features of the system.

The **SPEC CPU 2006** benchmark suite [52] 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. [46], 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 [**?**], we also pick 4 extra floating-point benchmarks that stress the LLC: GemsFDTD, leslie3d, soplex and sphinx3. When multiple input sets and sizes are available, we pick the single *ref* input indicated by Phansalkar et al. [46]. SPEC is the only benchmark suite used in many previous characterizations of LLC partitioning [**?**, **?**, **?**].

The **DaCapo** benchmark suite is intended as a tool for Java benchmarking, consisting of a set of open-source, real-world applications with non-trivial memory loads, including both client and server-side applications. We used the latest 2009 release. The managed nature of the DaCapo runtime environment has been shown to make a significant difference in some scheduling studies [**?**], and is also representative of the increasing relevance of such runtimes.

The **PARSEC** benchmark suite is intended to be representative of parallel real-world applications [**?**]. PARSEC programs use various parallelization approaches, including data- and task-parallelization. We use the version of the benchmarks parallelized with the `pthreads` library, with the exception of `freqmine`, which is only available in OpenMP. We used the full native input sets for all the experiments. Past characterizations of PARSEC have found it to be sensitive to available cache capacity [**?**], but also resilient to performance degradation in the face of intra-application sharing of caches [**?**].

We added four **additional parallel applications** to help ensure we covered the space of interest: *Browser_animation* is a multithreaded kernel representing a browser layout animation; *G500_csr* code is a kernel performing breadth-first search of a large graph for the Graph500 contest, based on a new hybrid 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 through a fluid using a parallel stencil kernel over a regular grid [**?**].

5.2.1. Manycore OS Implementation

5.3. Response-Time Functions

5.4. Resource Allocation Decisions

6. Discussion

Program phase detection has targeted hardware or software reconfiguration. The overview [17] evaluates three measurement-based predictors, and while performance variation is not one of the three, the authors note that small performance variation is an indicator of correct phase identification. A similar conclusion is reached in [49].

7. Related Work

Resource Allocation for Autonomic Data Centers using Analytic Performance Models [42]

Autonomic QoS-aware resource management in grid computing using online performance models [33]

On the use of hybrid reinforcement learning for autonomic resource allocation [56]

Utility-Function-Driven Resource Allocation in Autonomic Systems [55]

Redline: First Class Support for Interactivity in Commodity OSs [59] Paper presented in OSDI'08.

AcOS: an Autonomic Management Layer Enhancing Commodity Operating Systems [4] Work at the Politecnico di Milano, Italy. Master Thesis Paper and slides at the CHANGE 2012 Workshop (DAC)

Metronome: OS Level Performance Management via Self Adaptive Computing [50] Work related to AcOS, done by peo-

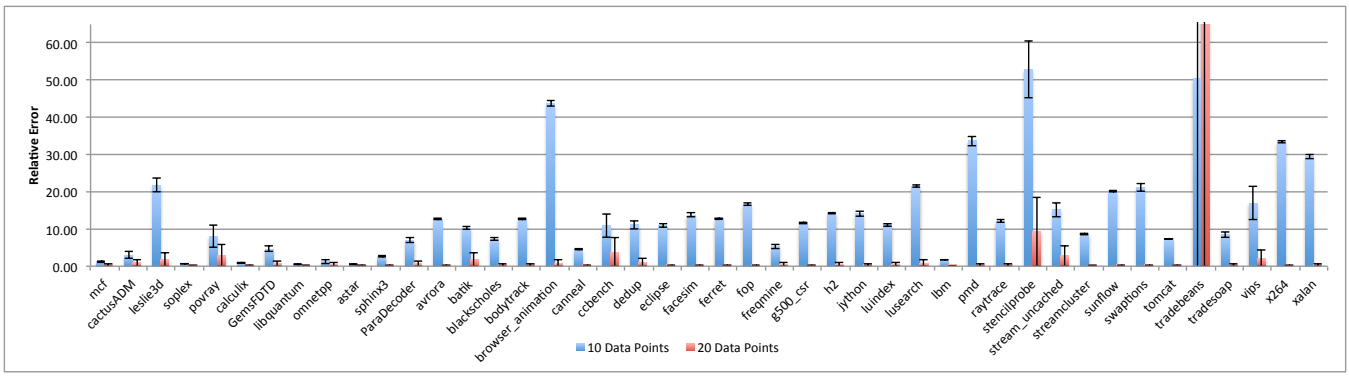

Figure 5: L1 Norm of Relative Error of Response-Time model predicted runtime vs. actual runtime. Data is from 3 complete trials.

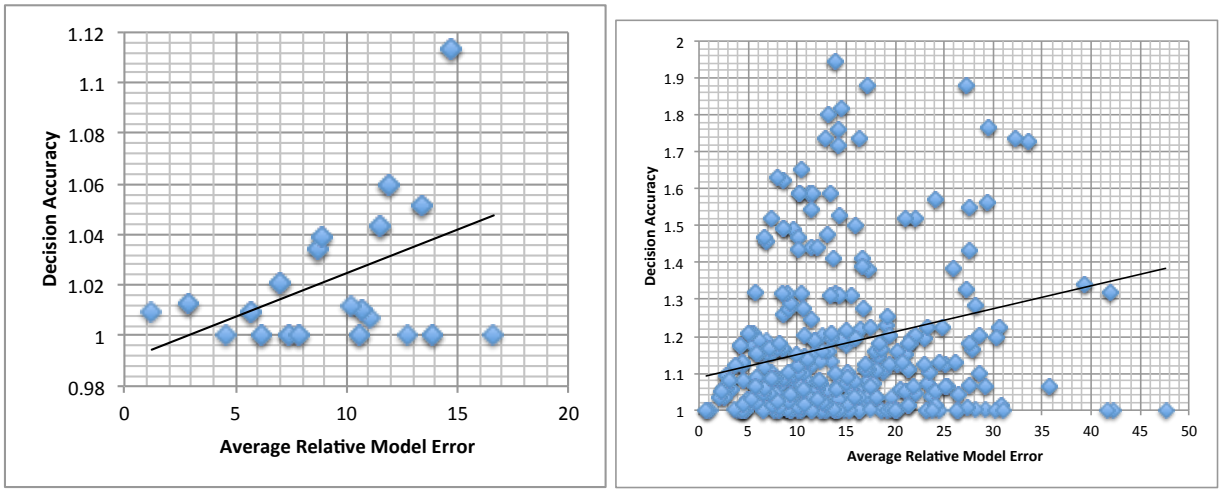

Figure 6

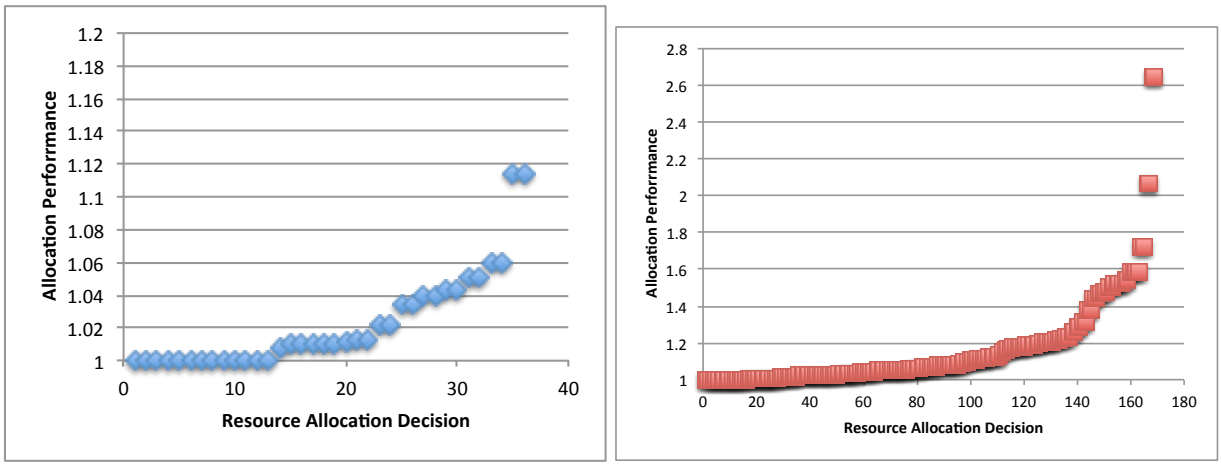

Figure 7

ple from the Politecnico di Milano (Italy), MIT and Harvard. Paper presented in DAC 2012.

Jockey: Guaranteed Job Latency in Data Parallel Clusters [19] Work from Microsoft about the system they built on top of their Cosomos cluster (Dryad???) Discusses how they do the dynamic resource allocation for jobs running in a cluster Paper from Eurosys 2012

Automatic Exploration of Datacenter Performance Regimes [8]

A resource allocation model for QoS management [48]

CQoS: a framework for enabling QoS in shared caches of CMP platforms [30]

7.1. Partitioning Mechanisms

Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches [47]

Most hardware partitioning mechanism work has looked at shared cache structures and provided mechanisms to partition them according to a varied set of goals. Suh et al. [53, 54] and Qureshi and Patt [47] monitor individual applications' cache performance and use this monitoring to inform their partitioning mechanism in an attempt to reduce the total amount of cache misses and off-chip memory traffic. A wide variety of proposals exist for multicore last-level cache structures that partition the spatial resources between private and shared data, in an attempt to create a manageable trade-off between capacity for shared data and low latency for private data [11, 14, 18, 29, 40, 61, 62].

Further cache partitioning work has focused on providing QoS guarantees to applications. Early work focused on providing adaptive, fair policies that ensure equal performance degradation [32, 60], while more recent proposals have incorporated more sophisticated policy management [23, 24, 27, 31]. Other partitioning work has focused on interconnect bandwidth QoS [36] or partitioning cache capacity and bandwidth simultaneously [43]. In general, these papers focus on designing and proving the effectiveness of particular mechanisms for particular goals, without a concrete notion of a general framework in which a variety of application-specific QoS requirements can be communicated to an all-purpose resource allocator and scheduler.

7.2. Resource Allocation Frameworks

Guo et al. [24] point out that most 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. The framework they present incorporates a scheduler that supports multiple execution modes.

Nesbit et al. [44] introduce Virtual Private Machines (VPM), a framework for resource allocation and management in multicore systems. A VPM consists of a set of virtual hardware resources, both spatial (physical allocations) and temporal (scheduled time-slices). Unlike traditional virtual machines that only virtualize resource functionality, VPMs virtualize a system's performance and power characteristics, meaning that a VPM has the same performance and power profile as a real machine with an equivalent set of hardware resources.

They break down the framework components into policies and mechanisms which may be implemented in hardware or software. Critical software components of the VPM framework are VPM *modeling*, which maps high-level application objectives into VPM configurations, and VPM *translation*, which uses VPM models to assign an acceptable VPM to the application while adhering to system-level policies. A VPM scheduler then decides if the system can accommodate all applications or whether resources need to be revoked.

The VPM approach and terminology mesh well with our study, which can be seen as a specific implementation of several key aspects of the type of framework they describe (i.e. VPM modeling and translation). Nesbit et al. did not perform any evaluations of the modeling, translation, or scheduling processes suggested in their paper.

[1] [34] [21] [41]

Ganapathi et al. have had success using machine learning to model application performance and select the best performing configuration in [20].

Calandrino *et al.* [10] use working set sizes to make co-scheduling decisions and enhance soft real-time behavior.

[57] utility functions

[25, 28, 58]

Scheduling threads for constructive cache sharing on CMPs [13]

8. Conclusion

References

[1] M. Aiken *et al.*, "Deconstructing process isolation," in *Memory Systems Performance and Correctness*, 2006.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

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

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

[10] 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.

[11] 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.

[12] 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.

[13] S. Chen *et al.*, "Scheduling threads for constructive cache sharing on CMPs," in *Proc. SPAA '07*. New York, NY, USA: ACM, 2007, pp. 105–115.

[14] 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.

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

[16] 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

[17] 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.

[18] 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.

[19] 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

[20] A. Ganapathi *et al.*, "A case for machine learning to optimize multicore performance," in *HotPar09*, Berkeley, CA, 3/2009 2009. Available: http://www.usenix.org/event/hotpar09/tech/

[21] D. Genbrugge and L. Eeckhout, "Statistical simulation of chip multiprocessors running multi-program workloads," in *ISCA*, 2007.

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

[23] 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.

[24] 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.

[25] J. L. Hellerstein *et al.*, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

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

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] R. Kumar *et al.*, "Single-isa heterogeneous multi-core architectures for multithreaded workload performance," in *ISCA '04: Proceedings of the 31st annual international symposium on Computer architecture*. Washington, DC, USA: IEEE Computer Society, 2004, p. 64.

[35] H. T. Kung, "Memory requirements for balanced computer architectures," in *International Symposium on Computer Architecture*, 1986, pp. 49–54.

[36] J. W. Lee, M. C. Ng, and K. 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.

[37] R. Liu *et al.*, "Tessellation: Space-time partitioning in a manycore client os," in *HotPar09*, Berkeley, CA, 03/2009 2009. Available: http://www.usenix.org/event/hotpar09/tech/

[38] 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.

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

[40] 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.

[41] 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&type=pdf&coll=GUIDE&dl=GUIDE&CFID=35811817&CFTOKEN=17839388

[42] 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.

[43] K. J. Nesbit, J. Laudon, and J. E. Smith, "Virtual private caches," in *Proc. ISCA '07*. New York, NY, USA: ACM, 2007, pp. 57–68.

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

[45] 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/

[46] 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.

[47] 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.

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

[49] T. Sherwood, S. Sair, and B. Calder, "Phase tracking and prediction," *SIGARCH Comput. Archit. News*, vol. 31, no. 2, pp. 336–349, 2003.

[50] 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

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

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

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

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] L. Wasserman, *All of Nonparametric Statistics (Springer Texts in Statistics)*. Se- caucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[59] 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

[60] 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.

[61] 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.

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