PACORA: Performance Aware Convex Optimization for Resource Allocation

Sarah L. Bird, University of California-Berkeley <slbird@eecs.berkeley.edu>

Burton J. Smith, Microsoft <burtons@microsoft.com>

Abstract

Resource allocation is the dynamic allocation and de-allocation of processor cores, memory pages, and various categories of bandwidth to client sub-computations (*e.g* processes within an operating system) that compete for those resources. Historically, resource allocation has been rather unsystematic, and now the assumptions underlying the traditional strategies no longer hold. The existence of parallel software and multiple heterogeneous processor cores, the requirement to maintain quality of service agreements and responsiveness, and the need to limit power and energy consumption (especially in mobile systems) are inadequately addressed by the *status quo.* A new approach to resource allocation is described that addresses all of these needs and that can be framed as a *convex optimization problem*; the result is that an optimal solution will exist and be unique with no local extrema. As a result, rational, fast, and fully incremental solutions to the resource allocation problem become feasible.

1 Introduction

Resource allocation is one of the primary functions for operating system software and has becoming increasingly important for client systems as well as servers and data centers due to an increased emphasis on energy efficiency, stringent user expectations of application responsiveness, and a growing diversity of resources. "Resource allocation" as the term is used here means the dynamic allocation and de-allocation by an operating system of processor cores, memory pages, cache, and various categories of bandwidth to computations that compete for those resources. Given a finite set of available resources, the operating system must decide how best to allocate resources to minimize a metric of responsiveness. (Some systems may use other optimization criteria, e.g. maximizing throughput, but for this work we concentrate on responsiveness.)

This definition naturally establishes resource allocation as a type of *constrained optimization problem*; however, historically resource allocation solutions have been rather unsystematic [Corb, RuSI]. 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. Memory allocation has usually employed independent machinery, and other resources such as I/O or network bandwidth have been deemed so abundant as to require no explicit management at all.

The assumptions underlying allocation strategies of this sort no longer hold, especially for emerging client systems. First, applications increasingly differ in their ability to exploit multiple processor cores and other resources, and these differences are independent of their relative responsiveness requirements. Second, the value of application responsiveness is highly nonlinear for an increasing variety of "Quality-Of-Service" (QOS) applications like streaming media or gaming; for these applications, responsiveness is approximately two-valued depending on whether performance is adequate or not. Third, power and battery energy have become key resources, *e.g.* in mobile computing, and available battery energy is itself a component of responsiveness and the user experience.

We present PACORA, a resource allocation system that constructs the resource allocation problem as a convex optimization problem. PACORA creates simplified models of application performance through measurement and uses these models along with information about the applications' responsiveness requirements and importance and the system battery life to determine the optimal resource allocation.

This research was supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, NEC, Nokia, NVIDIA, and Samsung.

In the following sections, we describe PACORA in greater detail. We will focus primarily on PACORA for client systems rather than servers or data centers, but all of the principles here are largely applicable in these other domains as well.

2 PACORA Overview

The PACORA framework is a general framework for allocating a variety of different resources to applications in the system. Resources can be bandwidth, the quantity of cache, memory pages, or different execution units and cpus. PACORA will perform best on systems with strong performance isolation among resources; however, it is still applicable to current, more limited systems for making allocation decisions.

The resource allocation approach taken by PACORA attempts to address the resource management problem as follows:

1. The *process* is the entity to which the operating system allocates resources. Micro-management of the resources within a process is generally application dependent and should be under the control of components of the runtime environment such as a user-mode work scheduler for processor cores or a memory garbage collector for memory pages.
2. The objective function to be minimized is the total *penalty*, which is the sum of the penalties of the runnable processes *p*. Penalty minimization is done on-line and continuously.
3. The penalty of a process is described by a function rather than a single value, and its argument, the *runtime*, is an appropriate measure of process responsiveness. For example, the runtime of a process might be: 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; or the time to execute a specified amount of work.
4. The runtime of a process is measured or predicted from its history of resource usage.

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

Minimize $\Sigma_{p\in P}\pi_p(\tau_p(a_{p,1}\ldots a_{p,n}))$
Subject to: $\Sigma_{p\in P}\ a_{p,r} \leq A_r$ for $r = 1\ldots n$
$a_{p,r} \geq 0$ for all $p\in P$ and $r = 1\ldots n$

Here π_p is the penalty function for process p, τ_p is its runtime function, A_r is the total amount of resource r available, and $a_{p,r}$ is the allocation of resource r to process p. The optimization problem described above is in fact *convex*, making its use for operating system resource management entirely practical.

3 Resource Management as Convex Optimization

If the penalty functions, runtime functions, and resource constraints are 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 [BoVa], two benefits accrue:

1. An optimal solution will exist and be unique, with no local extrema;
2. 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 points are *convex functions.*

A convex optimization problem can be expressed in this form:

Minimize $F_0(x_1, x_2, \ldots x_m)$
Subject to $F_i(x_1, x_2, \ldots x_m) \leq 0$, $i = 1,\ldots k$
Where $F_0, F_1, \ldots F_k : \mathbf{R}^m \rightarrow \mathbf{R}$ are all convex.

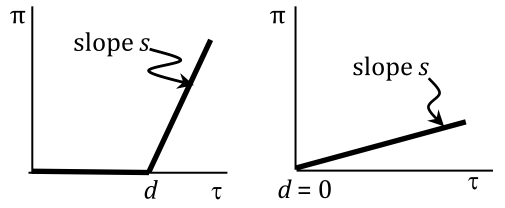
A few more facts about convex functions will be useful in what follows. First, a *concave* function is one whose negative is convex. Clearly, maximization of a concave function is equivalent to minimization of its convex negative. An *affine* function, one whose graph is a straight line in two dimensions or a hyperplane in *n* dimensions, is both convex and concave. A non-negative weighted sum or pointwise 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 $m = |P|\cdot n$ variables $a_{p,r}$ as long as the penalty functions π_p are convex non-decreasing and the runtime functions τ_p are convex. Note that the resource constraints are all affine and can be rewritten as $\Sigma_{p\in P}\, a_{p,r} - A_r \leq 0$, $r = 1, \ldots, n$, and $-a_{p,r} \leq 0$.

4 Penalty Functions

Penalty functions are generically defined as members of a family of such functions so that user preferences for a process *p* (an implicit parameter 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 parametrically modified to effect the change. 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.

One possible generic penalty function idea is to define a family of piecewise linear functions of the form $\pi(\tau) = \max(s \cdot (\tau - d), 0)$. Two representative graphs are shown below.



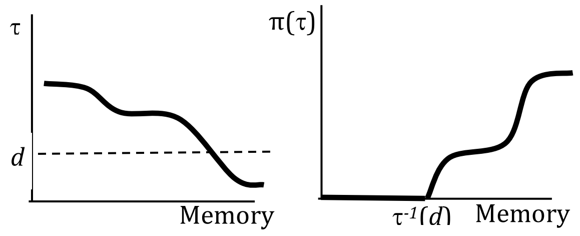
The two parameters d and s define the penalty function. To guarantee π is convex and non-decreasing, s must be non-negative. The runtime τ is of course non-negative, and it may be sensible (if not strictly necessary) to convene that d is also. A service-constrained process has a marked change in slope, namely from 0 to s, at the point $\tau = d$. This means that any runtime not exceeding d is as satisfactory as any other, but a runtime exceeding d contributes a penalty. 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 service constraints one can set $d = 0$ so that $\pi(\tau) = s{\cdot}\tau$. This defines linear behavior with s as the rate of penalty increase with runtime.

The gradient of process penalty with respect to its resource allocations a_r is useful in controlling the optimization process. Since the objective function is convex, descent along the gradient leads toward the global minimum. By the chain rule, each term in the gradient $\partial\pi/\partial a_r = d\pi/d\tau \cdot \partial\tau/\partial a_r$. The first factor is well-defined but discontinuous at $\tau = d$ with $d\pi/d\tau =$ if $(\tau - d) \leq 0$ then 0 else s. The problem of estimating the partial derivatives $\partial\tau/\partial a_r$ is discussed below.

5 Runtime Functions

Unlike penalty functions, which describe user preference, runtime functions measure performance as a function of the resource assignment. Runtime will commonly vary with time as a process changes "phase" and can make better or worse use of certain resources. To guarantee the objective function is convex τ must be also, and this is at first glance a plausible requirement akin to the proverbial "Law of Diminishing Returns". An equivalent statement is that incrementally changing the allocated quantity of a resource results in a runtime that is never smaller than one extrapolated from the rate of change of the runtime with resources at the current resource allocation.

There are examples of runtime behavior that violate convexity. One example can occur in memory allocation, where "plateaus" can be seen:



Typically, these plateaus are caused by algorithm adaptations within the application to accommodate variable memory availability. The runtime becomes the minimum of two or more functions, one for each range of algorithm applicability, and the minimum fails to preserve convexity. The effect of the plateaus will be corresponding inflections of penalty as shown in the right-hand figure above, and multiple solutions to the optimization problem will result.

There are a few ways to sidestep this issue. One is based on the observation that such runtime functions will be at least *quasiconvex.* A variation on this idea is to use additional constraints to exclude values for memory resource allocation spanning multiple plateaus. Finally, one can model the measured runtime by a function which is convex and does not distort the runtime behavior seriously.

The partial derivatives $\partial\tau/\partial a_r$ or approximations to them are useful to estimate the relative runtime improvement from each type of resource. A user-level runtime that manages allocation internal to the process may be a good source of these data. Additionally, the resource manager can allocate a modest amount of a resource and measure the change in runtime. Finally, a parameterized analytic model can be constructed and the partial derivatives evaluated directly. This approach is developed more fully below.

6 Managing Power and Battery Energy

It is useful to designate a "process" to receive allocations of all resources that are powered off. Ideally, this would include all resources not allocated elsewhere. Process 0 will play this role in what follows. τ_0, the measure of runtime for process 0, is artificially defined to be the *total system power consumption*. This function is linear and monotone decreasing in its arguments $a_{0,r}$, *i.e.* the resources assigned to process 0. 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 it. Alternatively, the slope s_0 can be adjusted to reflect the current battery charge state: as the battery depletes, s_0 can increase and force processes with lesser penalty to slow or even cease execution as they yield resources to be powered off.

7 Runtime Function Modeling

While it might be possible to model runtimes by recording and interpolating among the values that result from assignments of resources to processes, this idea has serious potential shortcomings:

1. The size of the multidimensional runtime function tables may be large;
2. Interpolation in many dimensions is expensive;
3. The runtime measurements will be "noisy" and require smoothing;
4. Convexity in the resources may be violated;
5. Gradient estimation may be infeasible.

An alternative approach is to model the runtime functions using analytic expressions that are convex by construction. For example, a runtime might be modeled as a weighted sum of terms, one per bandwidth resource, where each term is the amount of application work needing the resource divided by the allocated bandwidth. One term might represent the number of instructions divided by allocated instruction execution rate; another might be number of storage accesses divided by allocated storage bandwidth and so forth. Such a model will automatically be convex in the bandwidth allocations because $1/b$ is convex for positive b and because a positively-weighted sum of convex functions remains convex.

Asynchrony and runtime tolerance may make the constituent runtimes 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 on the terms 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 (max) describing full overlap.

The "non-bandwidth" resources are not part of this model, notably the quantity of memory. Cache allocations, when and if they are technically feasible, are also excluded. The effect of these memory resources on runtime is largely indirect, namely to exploit temporal locality and thereby reduce the consumption of bandwidth. For example, additional memory may reduce the need for storage or network bandwidth, and of course more cache capacity may reduce the need for memory bandwidth. The effectiveness of memory in reducing the need for bandwidth has been studied by H. T. Kung [Kung], who developed tight asymptotic bounds for certain applications on the storage bandwidth amplification resulting from a quantity of memory m. His results apply equally well to memory bandwidth amplification due to cache size. For dense matrix factorization and eigenproblems, the factor is $\alpha(m)= m^{1/2}$; for explicit PDE solution on d-dimensional meshes, $\alpha(m)= m^{1/d}$; for comparison-based sorting and FFTs, $\alpha(m)= \log m$; and for data-intensive computations with $\Theta(1)$ work per data element, $\alpha(m)= 1$, *i.e.* there is no amplification of bandwidth.

A process may exhibit diminished bandwidth amplification manifested by reduced improvement as memory allocation exceeds the limit of usability. This kind of behavior could be modeled by setting $\alpha(m) = \min(C_1\ \alpha_1(m),\ C_2\ \alpha_2(m))$ for suitable positive constants C_1 and C_2, "flattening" the amplification of bandwidth for large m.

Each bandwidth amplification factor might be modeled by one of the functions listed above and included in the denominator of the appropriate term in the runtime function model. For example, the storage term for the model of an out-of-core sort might be the quantity of storage data accessed divided by the product of the storage bandwidth allocation and $\log m$, the amplification factor associated with sorting. Amplification factors for each application could be learned from runtime measurements by observing the effect of varying the memory resource argument while keeping the others constant.

It remains to show that the model including bandwidth amplification functions is convex in the bandwidth and memory resources b_j and m_j given any of the various α_j possibilities above. Since norms preserve convexity, this reduces to proving each term in the norm is convex. Notice further that $w/(b\cdot\min(c_1\alpha_1(m),\ c_2\alpha_2(m))) = \max(w/(b\cdot c_1\alpha_1(m)),\ w/(b\cdot c_2\alpha_2(m)))$ because all quantities are positive; since maximum and scaling by a positive constant both preserve convexity, it only remains to show $1/(b\cdot\alpha(m))$ 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 of two log-convex functions is the sum of their logs, each of which is convex by hypothesis. Now b^{-1} is log-convex for positive b because –log b is convex, and log $\alpha(m)^{-1}$ = log $m^{-1/d}$ = $-1/d$ log m is convex in a similar vein. Finally, log $\alpha(m)^{-1}$ = (log (log $m)^{-1}$) is convex because its second derivative is positive for $m \geq 1$.

Summarizing, a runtime function for a process might be modeled by the convex function

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

Here the w_i are learned parameters of the model (the "quantities of work"), the b_i are the allocations of the bandwidth resources, the α_i are the (learned) bandwidth amplification functions, and the m_i are the allocations of the memory or cache resources that lead to the corresponding amplifications.

These assumptions allow the runtime to be modeled as the p-norm of the component-wise product of a vector $\boldsymbol{d}$, computed from the resource allocation that resulted in the given runtime, with the vector of learned parameters $\boldsymbol{w}$.

8 Related Work

Doug Jensen first introduced Time-Utility Functions (TUFs) to express time-dependent utility in real-time scheduling [Jens]. In his approach there is a single processor per process and scheduling decisions do not depend directly on resource allocations. Nevertheless, the successes of his ideas support the validity of our approach, perhaps even for real-time scheduling problems.

Nesbit *et al.* explore performance and power modeling and allocation based on a Virtual Private Machine (VPM) abstraction [NMCR]. Feedback from global policy to local applications drives satisfaction of total resource constraints. The techniques we discuss might help guide this configuration pruning process in the VPM approach.

Stephen Boyd and his colleagues and students have extended the applicability of convex optimization to many arenas not normally thought of as practical candidates for optimization at all. Especially relevant is recent work [MaBo] on embedded real-time convex optimization performed continuously and incrementally, as we propose here.

9 Discussion and Conclusion

PACORA builds models of application performance and attempts to allocate resources to ensure the performance is adequate. We believe that this approach matches well with future computing environments. While the runtime models may not be completely faithful to application performance in all cases, it is unlikely that another approach would do any better. Applications are not always written to be deterministic and even deterministic applications have sources of non-determinism such as varying input data and unpredictable user interactions. Hardware also is a source of performance variability due to resource sharing and variable core performance. The models provide the operating system with an appropriate level of detail about application performance, giving good insight into how applications respond to resources.

With so many potential sources of variability it is uneconomical to design a system to make hard response-time guarantees. Furthermore, for most systems it is unnecessary. Users of both client systems and cloud services expect a level of responsiveness and consistency, but they may not notice if an occasional frame is dropped or a query responds a little slowly. We believe that the penalty functions strike a good balance between these two realities and treat responsiveness as a first class citizen by characterizing how costly it is for a deadline to be missed.

We have presented PACORA, a resource allocation framework for manycore systems, which treats resource allocation as a convex optimization problem. PACORA has several strengths as a resource allocation framework. First, convex optimization is relatively inexpensive and has a single extreme point, which makes it affordable to do frequently. Furthermore, the system does not have to perform a full convex optimization each time, but instead can do a partial gradient descent and make incremental steps towards the optimal solution. Penalty function slopes allow the system to express relative importance of application deadlines, and the runtime intercept encapsulates QoS requirements. An additional process can be used to represent power and manage battery energy. We believe PACORA's ability to handle a diversity of resources, represent QoS requirements, and express the importance of battery life make it a good match for future systems.

10 Acknowledgements

We would like to thank Juan Colmenares, John Kubiatowicz, Dave Patterson, Krste Asanovic and the rest of the ParLab Architecture and Operating Systems groups for their continual support and guidance as we develop and refine the ideas in PACORA. We would also like to thank the many Microsoft Windows operating system developers who have shared their knowledge with us.

References

[BoVa] Boyd, Stephen and Lieven Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004. http://www.stanford.edu/~boyd/cvxbook/.

[Corb] Corbató, F. J. *et al.*, *The Compatible Time-Sharing System: A Programmer's Guide.* MIT Press, 1963.

[Jens] Jensen, E. Douglas, Utility Accrual Real-Time Scheduling Under Variable Cost Functions. *IEEE Transactions on Computer*s 56(3):385-401, 2007.

[Kung] Kung, H. T. Memory Requirements for Balanced Computer Architectures. *Proceedings of the 13th International Symposium on Computer Architecture* , pp.49-54, 1986.

[MaBo] Mattingley, Jacob and Stephen Boyd, Automatic Code Generation for Real-Time Convex Optimization. To appear in *Convex Optimization in Signal Processing and Communications,* Y. Eldar and D. P. Palomar, Eds. Cambridge University Press, 2009.

[NMCR] Nesbit, K. J., M. Moreto, F. J. Cazorla, A. Ramirez, M. Valero, and J. E. Smith. Multicore Resource Management. *IEEE Micro* 28(3):6–16, 2008.

[RuSI] Russinovich, Mark and David Solomon with Alex Ionescu, *Windows Internals, Fifth Edition.* Microsoft Press, 2009.