Resource Management for Computer Operating Systems

Sarah Bird
PhD Candidate
Berkeley EECS

Burton Smith
Technical Fellow
Microsoft

OS Architecture Is Ancient

Most of it dates to the 60's and 70's

- CTSS (1961), VM/CMS (1968), UNIX (1969), VMS (1977)

It locks us in to old ways of doing things

- Using one or a few identical processor cores
- Using the OS kernel to assign cores to threads ("kernel threads")
- Using OS kernel operations to synchronize threads (typically by locking)
- Using interrupts to synchronize threads with devices
- Using traps to let threads invoke operating system services
- Using a variety of heuristics to allocate and schedule resources

These approaches don't cut it any more…

Hardware Has Changed

Available battery life limits user experiences

There isn't enough energy to run all of the hardware all of the time.
How can the state of battery charge be taken into account?

Processor cores are increasingly heterogeneous

Application software developers are trying to exploit them.
Can every core type access the OS services it needs?

New sensors and interfaces are emerging

Some of these require substantial or timely resource allocation.
How are dynamic resource allocations implemented by the OS?

Software Has Changed

Parallelism varies in both form and quantity

Even different phases of a single application can exhibit such diversity.

How can the OS adapt to varying application parallelism?

Media and games require responsiveness

Different platforms need differing resources for adequate performance.

How can appropriate allocations be done without over-provisioning?

Applications are built of distributed services

These services provide search, data access, connectivity, computation, *etc.*

How does application performance compose from its constituent services?

A Way Forward

- Resources should be allocated to processes
  - Cores of various types
  - Memory (working sets)
  - Bandwidths, *e.g.* to shared caches, memory, storage and interconnection networks
- The operating system should:
  - Optimize the responsiveness of the system
  - Respond to changes in user expectations
  - Respond to changes in process requirements
  - Maintain resource, power and energy constraints

What follows is a scheme to realize this vision

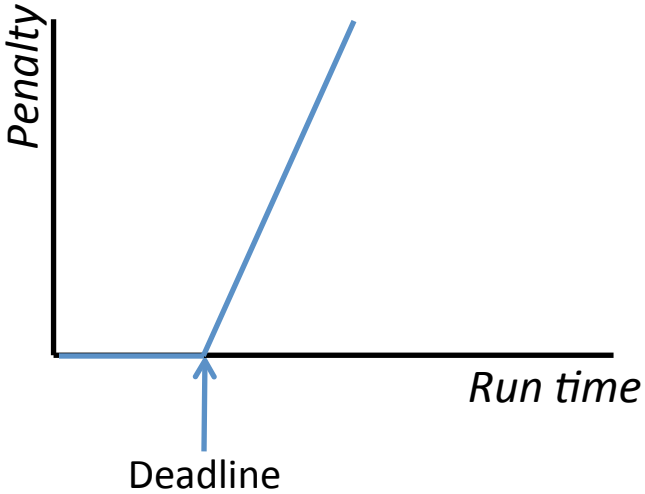
Responsiveness

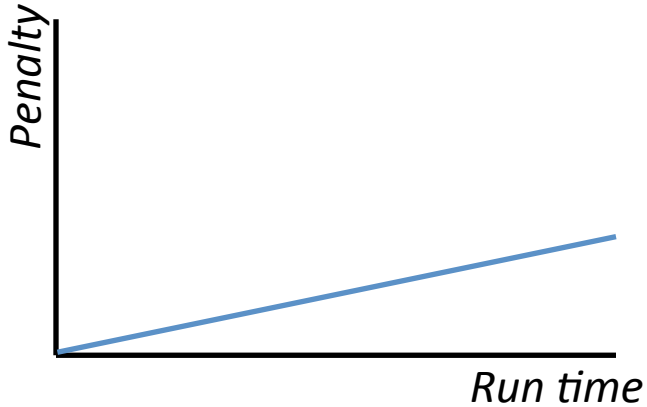
- Run times determine process responsiveness
  - The time from a mouse click to its result
  - The time from a service request to its response
  - The time from job launch to job completion
  - The time to execute a specified amount of work
- The relationship is usually a nonlinear one
  - Achievable responses may be needlessly fast
  - There is generally a threshold of acceptability
- Run time depends on the resources
  - Some resources will have more effect than others
  - Effects will often vary with computational phase

Penalty Functions

- The *penalty function* of a process defines how run times translate into responsiveness
  - Its shape expresses the nonlinearity of the relationship
  - The shape will depend on the application and on the current user interface state (*e.g.* minimized)
- We let the *total penalty* be the instantaneous sum of the current penalties of the running processes
  - Resources determine run times and therefore penalties
- Assigning resources to processes to minimize the total penalty maximizes system responsiveness

Typical Penalty Functions



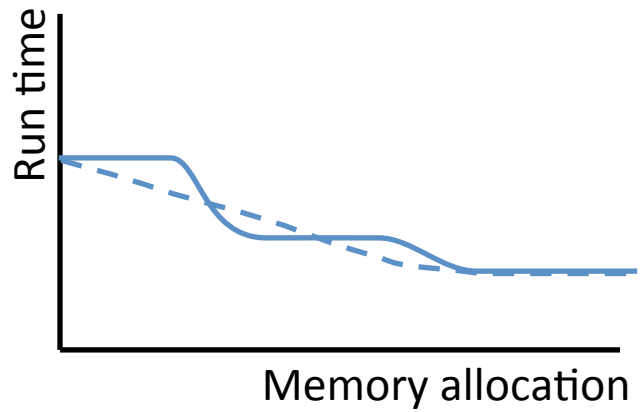


Adjusting Penalty Functions

- Penalty functions are like priorities, except:
  - They apply to processes, not kernel threads
  - They are explicit convex functions of the run time
- The operating system can adjust their slopes
  - Up or down, based on user behavior or preference
  - The deadlines can probably be left alone
- The total penalty is easy to compute
  - The objective is to keep minimizing it

Run Time Functions

- Generally, they exhibit "diminishing returns" as resources are added to a process
  - That is, run times are roughly convex functions
- They vary with input and must be measured
  - Sometimes run time increases with some resource
  - We may see "plateaus" or various types of "noise"



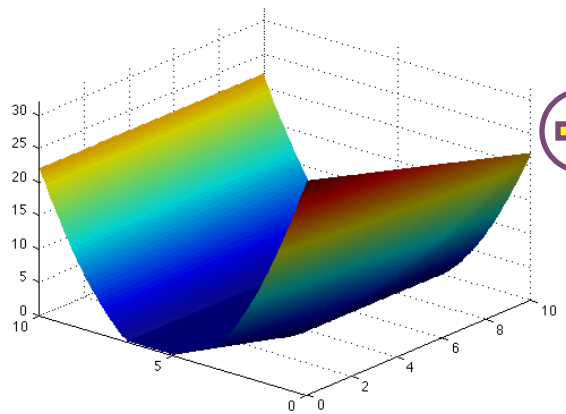
Resource Allocation As Optimization

Continuously minimize $\Sigma_{p \in P} \pi_p(\tau_p(r_{p,0}, \ldots r_{p,n-1})$ with respect to the resource allocations $r_{p,j}$, where

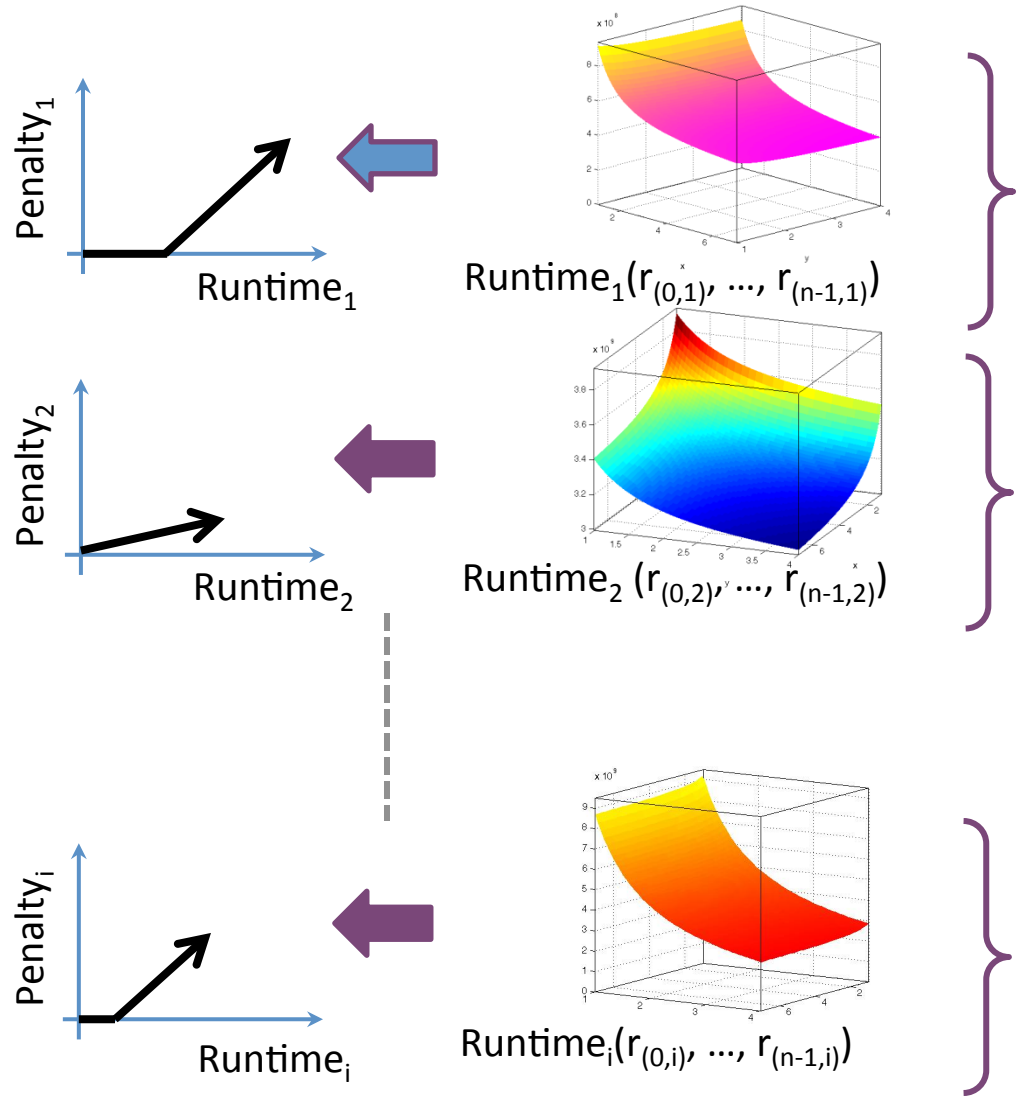
- P, π_p, τ_p, and the $r_{p,j}$ are all time-varying;
- P is the set of runnable processes;
- $r_{p,j} \geq 0$ is the allocation of resource j to process p;
- The penalty π_p depends on the run time τ_p;
- τ_p depends in turn on the allocated resources $r_{p,j}$;
- $\Sigma_{p \in P} r_{p,j} = A_j$, the available quantity of resource j.
  - All unused (slack) resources are allocated to process 0

The Optimization Picture

Continuously minimize the total penalty



Subject to the total available resources



Convexity and Concavity

- A function f is *convex* if for all x, y in $\Re^n$, λ in $\Re$, and $0 \le \lambda \le 1$, $f(\lambda x + (1-\lambda)y) \le \lambda f(x) + (1-\lambda)f(y)$
  - Chords of the function's graph must lie on or above it
- A function f is *concave* if for all x, y in $\Re^n$, λ in $\Re$, and $0 \le \lambda \le 1$, $f(\lambda x + (1-\lambda)y) \ge \lambda f(x) + (1-\lambda)f(y)$
  - Alternatively, f is concave if and only if $-f$ is convex
- Affine functions $Ax + b$ are both convex and concave
- If functions f_i are convex so is their sum $\Sigma_i f_i$
- If functions f_i are convex so is their maximum $\max_i f_i$
- If f is both convex and monotone non-decreasing and g is convex then $h(x) = f(g(x))$ is convex
- If f is convex, so is the set $C_\alpha = \{ x \in \mathrm{dom}(f) \mid f(x) \le \alpha \}$
  - C_α is called the *α-sublevel set* of f

Convex Optimization

- A convex optimization problem has the form:

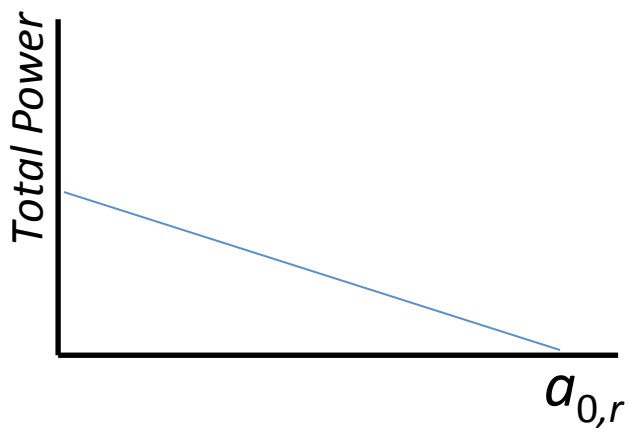
  Minimize $f_0(x_1, \ldots x_m)$

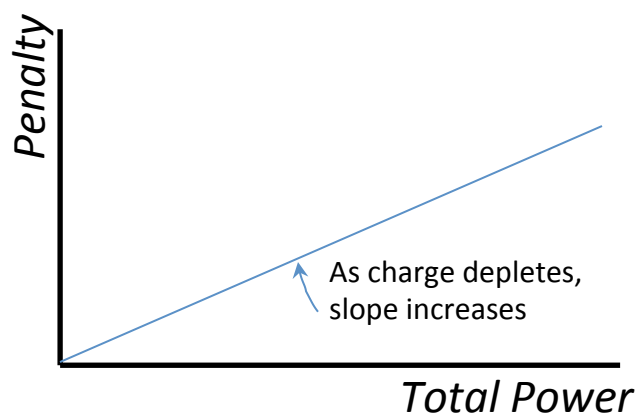
  subject to $f_i(x_1, \ldots x_m) \leq 0$, $i = 1, \ldots k$

  where the functions $f_i : \mathrm{R}^m \rightarrow \mathrm{R}$ are all convex
- Convex optimization has several virtues
  - It guarantees a single global optimal value
  - It is not much slower than linear programming
- In our formulation, resource management is a convex optimization problem

Managing Power and Energy

- Total system power can be limited by an affine resource constraint $\Sigma_j \mathrm{w}_j \cdot \Sigma_{p \neq 0} r_{p,j} \leq W$
- Total power can also be limited using π_0 and τ_0
  - Assume all slack resources $r_{0,j}$ are powered off
  - Instead of being a run time, τ_0 is total power
    - It will be convex in each of the slack resources $r_{0,j}$
  - π_0 has a slope that can depend on the battery charge
    - Low-penalty work loses to π_0 when the battery is depleted





Modeling Run Times

- Run time measurements can be used to construct a model, yielding several advantages
  - "Noise" can be removed while building the model
  - Interpolation among measurements is avoided
  - Rates of change with resources are computable
- We have constructed several types of model
  - Linear and quadratic (rate) models
  - Kernel Canonical Correlation Analysis
  - Genetically Programmed Response Surfaces
  - Convex non-negative least squares models

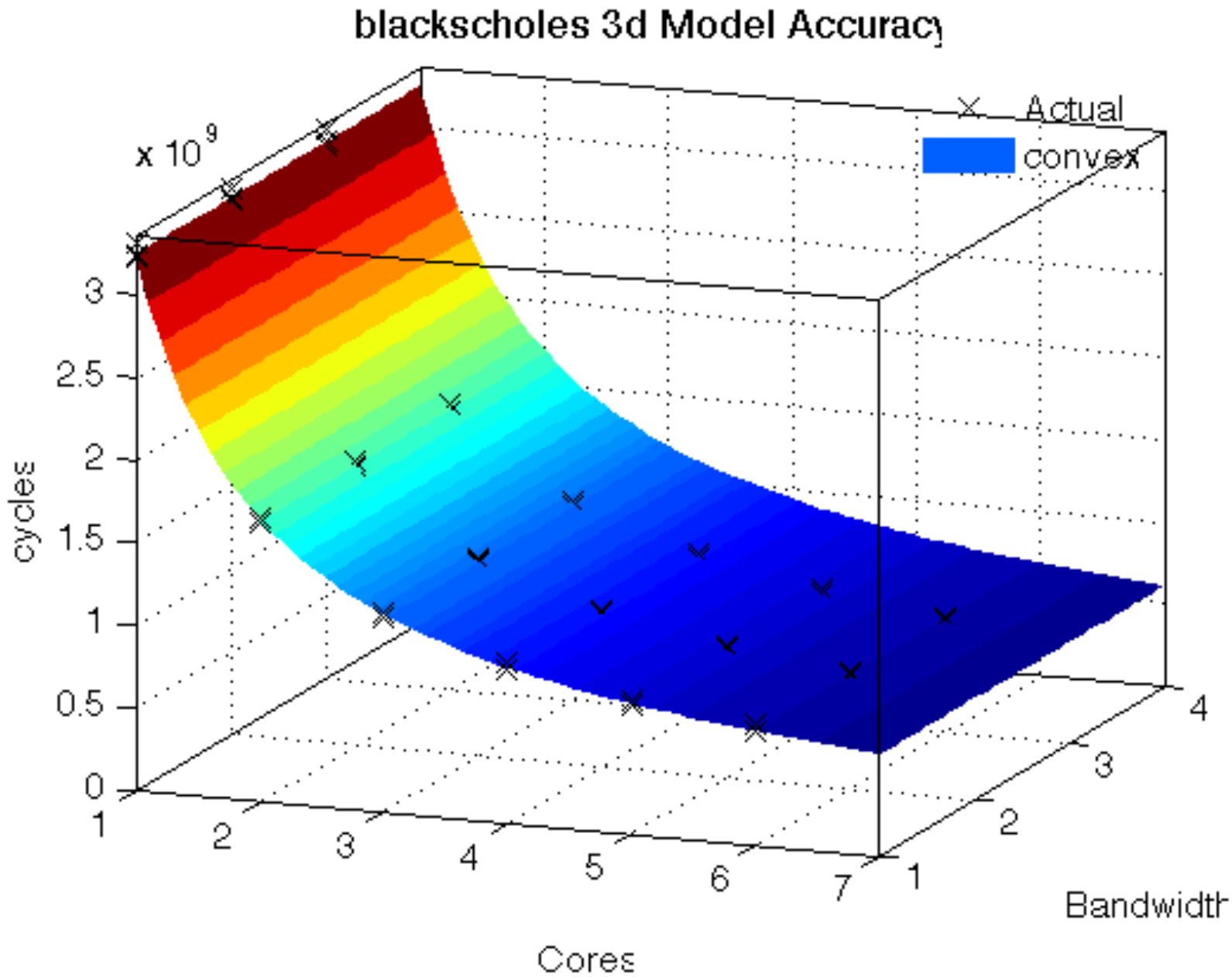
Convex Models

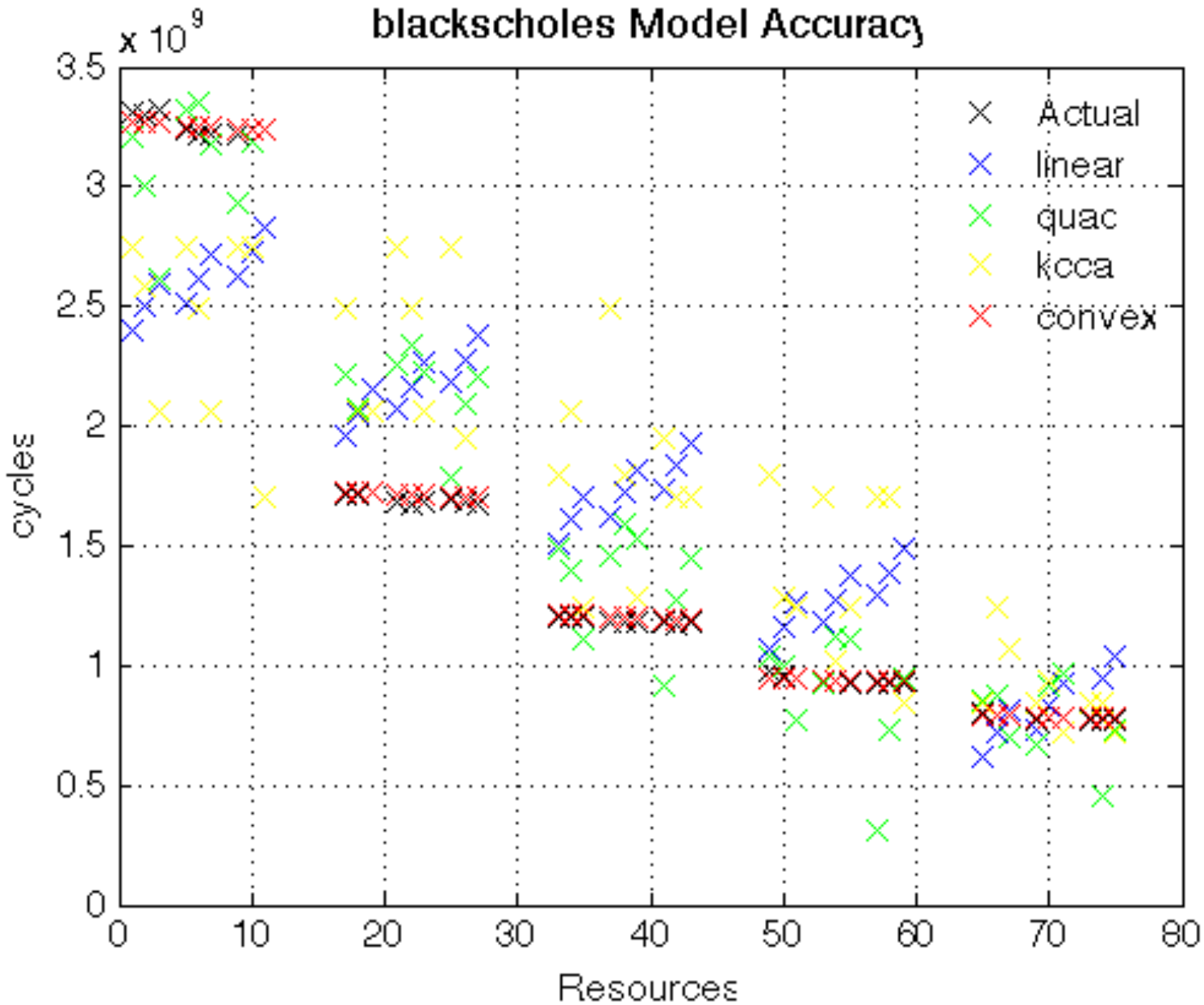
$$\tau(w,b) = T_0 + \sum_{i,j} \frac{w_{i,j}}{\sqrt{b_i * b_j}}$$

w: learned parameters
b: bandwidth allocations
α: bandwidth amplifications
m: memory allocations

$$\tau(w,b) = T_0 + \frac{w_1}{b_1} + \frac{w_2}{m_2} + \frac{w_3}{b_3}$$

$$\tau(w,b,\alpha,m) = T_0 + \sum_i \frac{w_i}{b_i \alpha_i(m_i)}$$

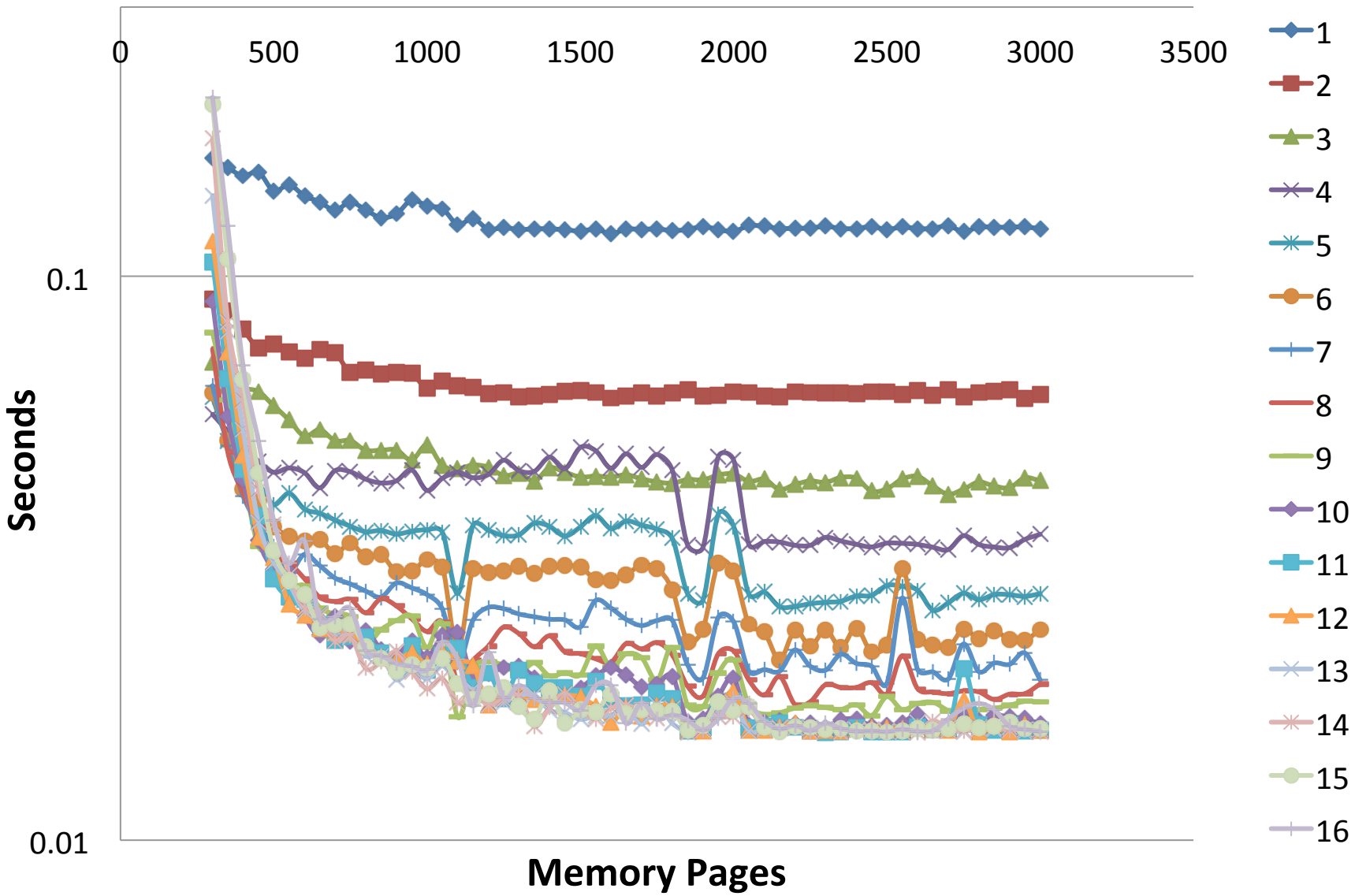

blackscholes 3d Model Accuracy
x 10^9
Actual
convex
cycles
3
2.5
2
1.5
1
0.5
0
1
2
3
4
5
6
7
Cores
1
2
3
4
Bandwidth


blackscholes Model Accuracy
x 10^9
3.5
3
2.5
2
1.5
1
0.5
0
cycles
0
10
20
30
40
50
60
70
80
Resources
Actual
linear
quac
kcca
convex

Modeling Results

- Non-negative least-squares applied to past run time data learns the model parameters
  - Rows of the QR factorization are updated and downdated to track changes in process behavior
  - Columns are also updated and downdated to remove and restore infeasible allocations
- Our convex models fit the data quite well
  - They work even better within the control loop
  - We have discovered some interesting artifacts

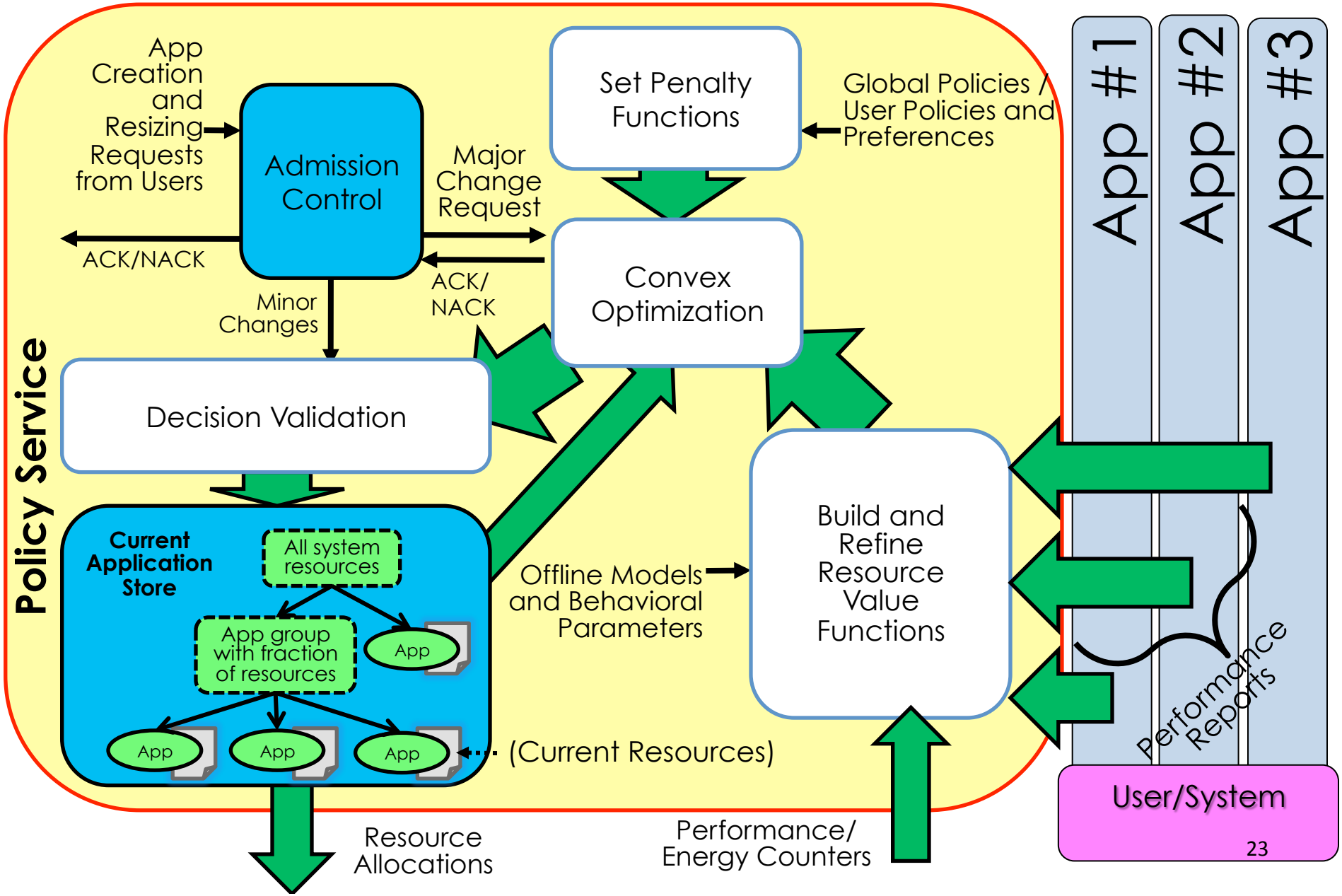
Nbodies Frame Time



PACORA

- PACORA stands for "Performance-Aware Convex Optimization for Resource Allocation"
  - See our paper in the poster session of HotPar '11
- It manages resource allocation policy in the Berkeley ParLab's Tesselation operating system
  - One of us (Bird) is currently demonstrating it
- Nearly all of the ideas presented here are being prototyped in PACORA and Tesselation
  - Perhaps they may be valuable elsewhere

PACORA in Tessellation



Acknowledgements

- Stephen Boyd and Lieven Vandenberghe, whose book taught us about convex optimization
- Krste Asanović, David Patterson, and John Kubiatowicz at Berkeley, for their strong support
- The Microsoft Windows developers, for giving us real applications and ways to measure them
- Colleagues at the UC Berkeley ParLab and at Microsoft Research, for helping us tell the story

Thanks!

Any questions?