Runtime Resource Management

Sarah L. Bird
Microsoft Research
New York, New York
USA

Burton J. Smith
Microsoft Research
Redmond, Washington
USA

Computer Resource Management

- RM is the principal job of an operating system
  - Allocating resources to processes
  - Sharing resources among processes
- RM meets user needs given system constraints
  - There are usually multiple processes and/or users
  - Their number and characteristics vary dynamically
- Operating system R&D is close to nonexistent
  - Everyone seems to want the OS “out of the way”
  - This may be because it is generally “in the way”

New Challenges for RM

- Heterogeneous processors
  - Which and how many should an application use?
- Quality of Service (QoS) requirements
  - How can response time requirements be met?
- Energy efficiency given QoS requirements
  - Which feasible allocation is most efficient?
- Cloud computing
  - How should cloud resources be managed?

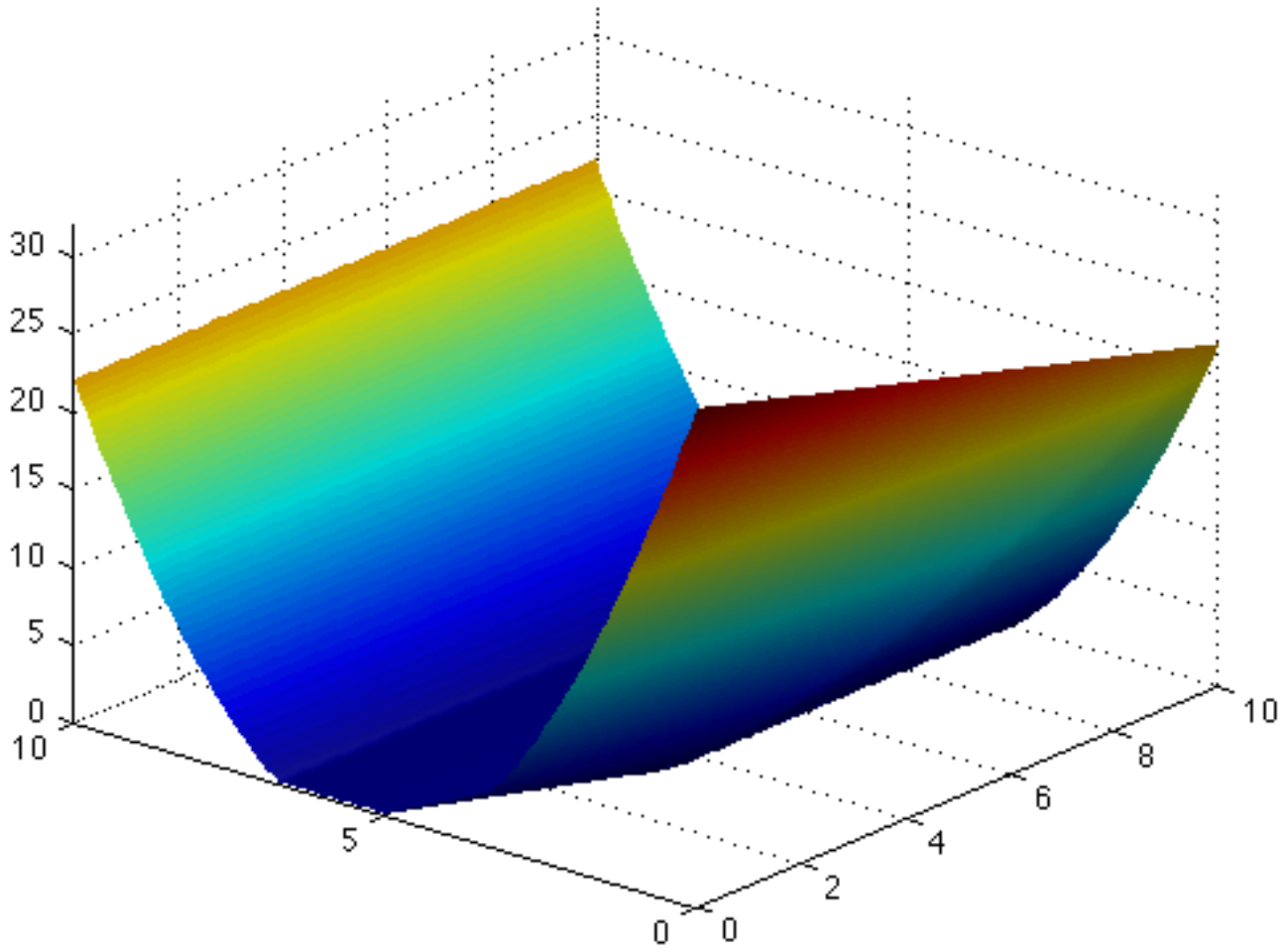
The Current State of Affairs

- Client operating system RM uses heuristics
  - Many of the concepts date from the 1960's
  - Hardware is adding more heuristics to the mix
- Server operating system RM is manual
  - The "OS" is merely a set of virtual machines
  - Reallocation is seldom and done by humans
- In either case, the consequence is erratic performance, wasted power, or both

Improving Resource Management

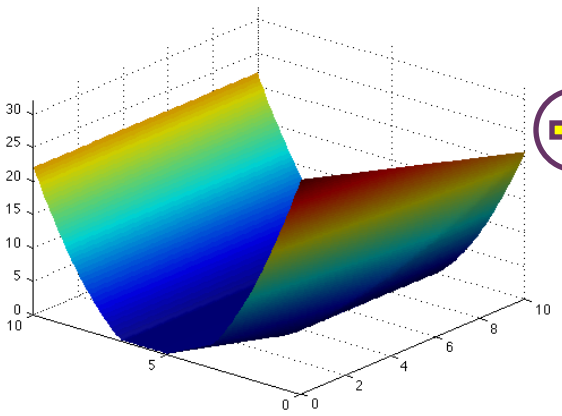
- We need a more principled approach to RM
  - One way: treat RM as an *optimal control problem*
- PACORA is our implementation of this idea
  - PACORA stands for "Performance-Aware Convex Optimization for Resource Allocation"
  - It optimizes allocation of resources to minimize:
    - the cost of failing to meet service requirements plus
    - the cost of powering the allocated resources
  - subject to the total resources available
- Because the convex optimization is fast, PACORA can continuously adapt to changes in workload

A Convex Function

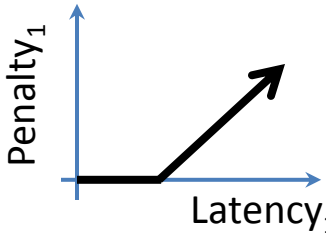


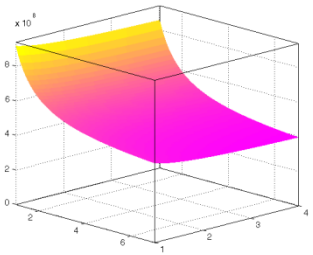
Client PACORA

Continuously minimize the total penalty

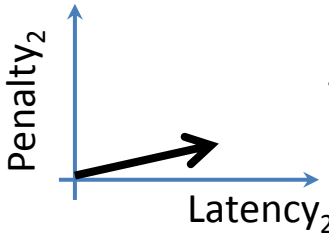


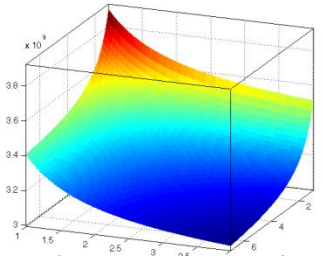
Subject to the total available resources



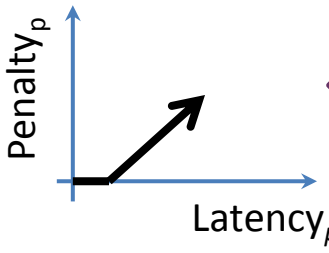


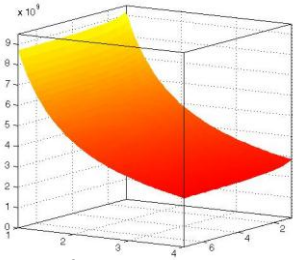
$Latency_1(a_{1,0}, \ldots, a_{1,n-1})$





$Latency_2\ (a_{2,0}, \ldots, a_{2,n-1})$





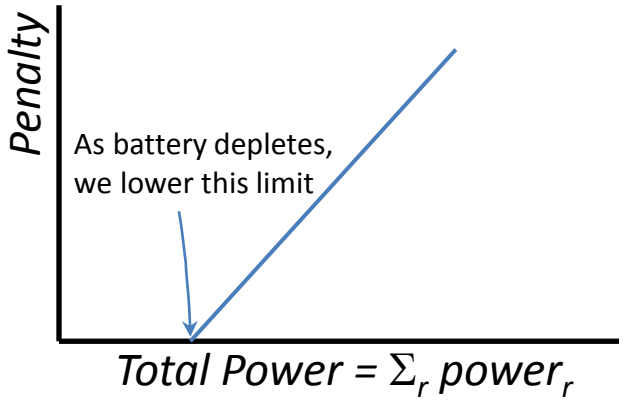
$Latency_{m-1}(a_{m-1,0}, \ldots, a_{m-1,n-1})$

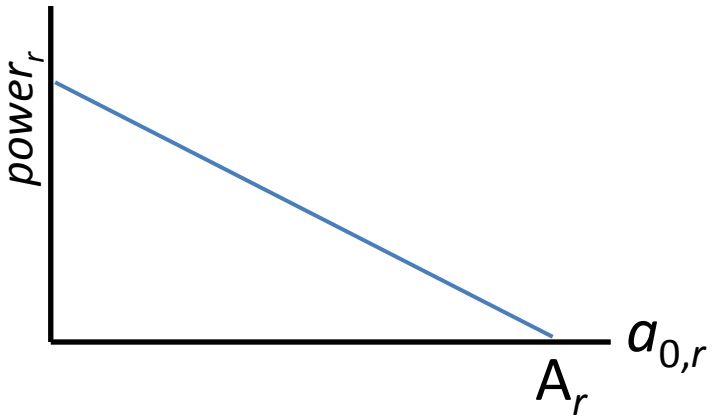
Who Does What

- The OS models the latency functions
  - Non-Negative Least Squares (NNLS) in PACORA
  - It needs to know what latencies to measure
- The user adjusts the penalty functions
  - A surrogate like the OS shell may do this
- The OS continuously optimizes the cost

Optimizing Power and Energy

- Total system power can be limited by a convex resource constraint like $\Sigma_r (w_r \Sigma_p a_{p,r}) \leq W$
- It can also be given its own penalty function
  - Assume the *slack resources* $a_{0,r}$ are powered down
  - Let the "latency" for process 0 be total system power
    - It is convex (actually affine) in the slack resources $a_{0,r}$
  - The penalty function can change with battery energy
    - Low-penalty work will slow when the battery is depleted





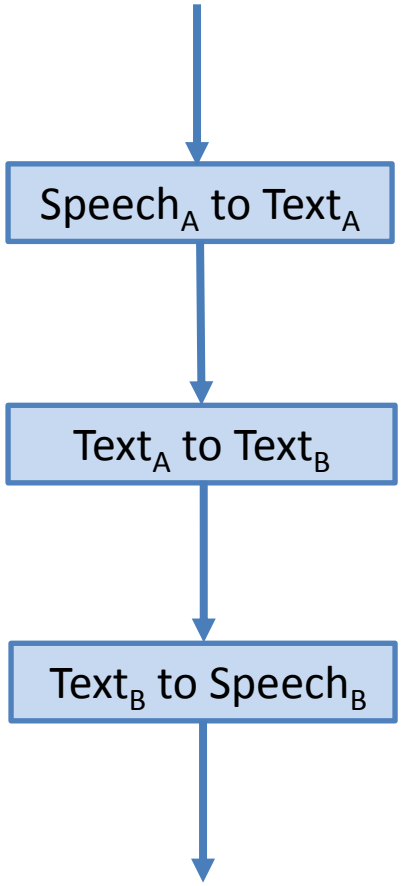
Client PACORA Algorithms

- Latency functions are modeled using non-negative least-squares (NNLS) which incrementally updates and downdates a QR factorization:
  - By rows, to add recent data and delete old or outlier data while preserving the rank of R
  - By columns, to remove negative parameters and restore them if and when they become positive
- Objective functions are piecewise linear and can be optimized by an interior point method
  - Currently, we use Stephen Boyd's ADMM algorithm which is well-suited to distributed control problems

Cloud PACORA

- Cloud services need RM for several reasons:
  - To adapt to workload changes (bandwidth)
  - To regulate responsiveness (latency)
  - To minimize costs, including power
- Frequently, services will be pipelined
  - For example, if they are built from other services
  - The sub-services may or may not be co-located
- Asynchronous pipelines can be hard to control
  - Ask a highway traffic engineer!

An On-line Service Example



The service must:

1. Accommodate variations in the incoming load bandwidth
2. Bound the total latency based on the Service Level Objective (SLO)
3. Minimize operating cost
4. Stay within available resources

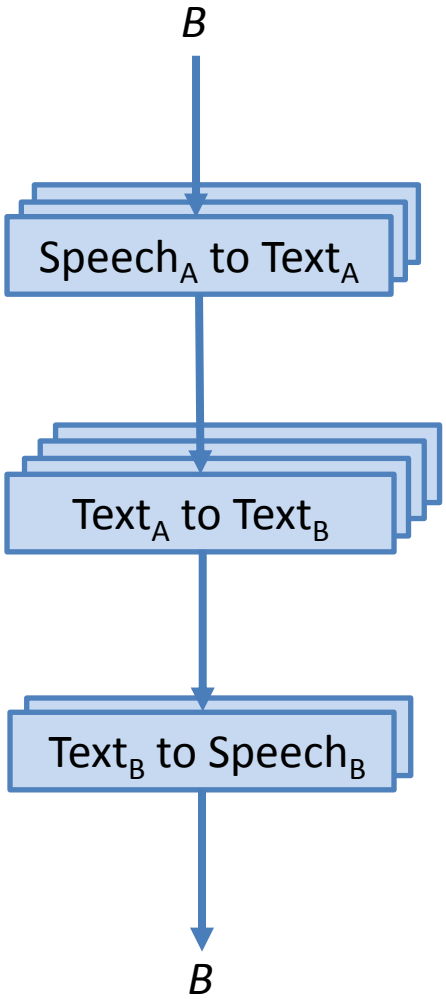
No problem, right?

Accomodating Workload Variations

- A pipeline stage comprises a set of *instances*
  - Stage bandwidth is the sum of instance bandwidths
  - The number of instances varies to handle the load
- Bandwidth per instance will vary
  - From resource adjustments for latency, for example
  - The instance bandwidth needs to be modeled
- Instead of instance bandwidth, we prefer its reciprocal: Valiant's *gap* from the LogP model
- Instance gaps, like latency, will be (roughly) convex in each instance's resource allocation

See Valiant, L. G., "A Bridging Model for Parallel Computation", Comm. ACM **33**, No. 8, Aug. 1990, pp. 103–111.

Incorporating Instances and Gaps



$\text{gap}_1 = \gamma_1(\ldots a_{1,r}\ldots)$

$n_1 = \lceil B \cdot \text{gap}_1 \rceil$

$\text{gap}_2 = \gamma_2(\ldots a_{2,r}\ldots)$

$n_2 = \lceil B \cdot \text{gap}_2 \rceil$

$\text{gap}_3 = \gamma_3(\ldots a_{3,r}\ldots)$

$n_3 = \lceil B \cdot \text{gap}_3 \rceil$

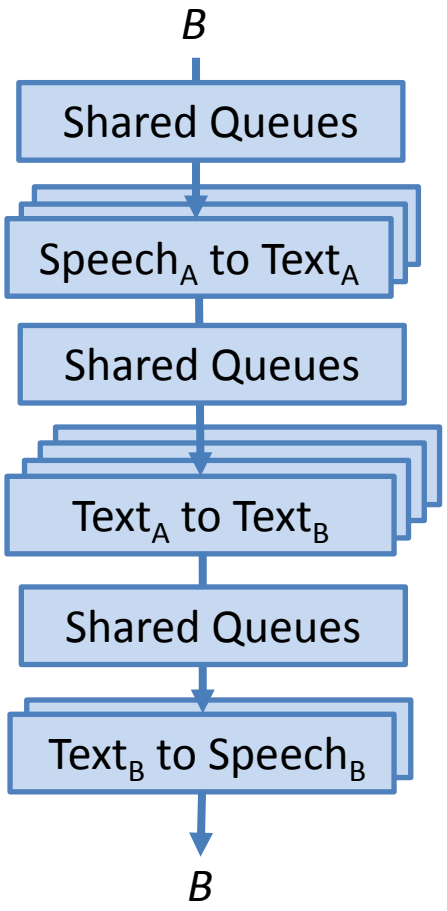
The bandwidth B has units of items/second

The gaps $\gamma_i(\ldots a_{i,r}\ldots)$ have units of instance-seconds/item

Queue Management

- Instances should not use internal queues
  - If they do, how can one distinguish between computational latency and queueing delay?
  - Also, how are queueing blockades shrunk?
- If shared external queues are used instead:
  - The sharing permits dynamic work distribution
  - The queue discipline can give priority to the *oldest* work, reducing the latency variance
  - Queueing delays can be kept small by ensuring that n_i slightly exceeds $B \cdot \text{gap}_i$ for all i

Shared External Queues



- To reduce communication and improve scaling, the queues can be distributed and *work stealing* used
- The shared queues can be placed in a shared address space, *PGAS* for example
- Instances and queues can share cores and caches across stages for better streaming locality

Latency Optimization

- A typical latency objective (SLO) specifies a time limit and the probability it is not exceeded
  - Resources per instance are adjusted to suit
- One could build a latency function model for the pipeline using some kind of regression
  - This would permit inter-stage resource tradeoffs
- Unfortunately, the usual kinds of regression describe the mean or median of the latency
  - What is needed here is *quantile regression*

Quantile Regression

- Given a sequence of resource allocations A and latencies t, quantile regression finds a model w that makes $t < Aw$ with relative frequency p
- Maintenance of the latency model w involves solving a sequence of linear programs
- Interior-point methods perform this well, even compared to traditional least-squares methods

See Portnoy, S. and R. Koenker, "The Gaussian Hare and the Laplacian Tortoise", *Statistical Science* **12**, No. 4, Nov. 1997, pp. 279–296.

Minimizing Cost

- The objective function sums one penalty function per SLO plus one for power
  - SLO violations will usually incur penalties
  - The power penalty can resemble the client case

Runtime Considerations

- Virtual memory should be shared by all cores
- The OS interface should be asynchronous
  - For cores that don't (or can't) run the OS
  - For cores that do, to retain their hardware threads
  - User-level task scheduling hides the asynchrony
- Resource allocations will vary dynamically
  - Some applications may have difficulty handling this
  - Enterprise data bases are good at it
- The runtime can supply rate-of-change data
  - This can be used to accelerate model adaptation