Jockey: Guaranteed Job Latency in Data Parallel Clusters

Microsoft Research

Andrew D. Ferguson
adf@cs.brown.edu

Peter Bodík
peterb@microsoft.com

Srikanth Kandula
srikanth@microsoft.com

Eric Boutin
eric.boutin@microsoft.com

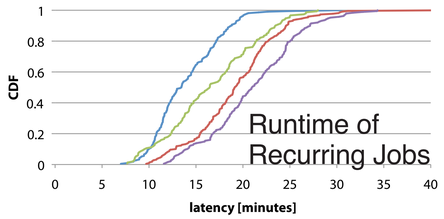
Rodrigo Fonseca
rfonseca@cs.brown.edu

BROWN

Deadlines and Varying Latency

- Users of data parallel clusters now demand predictable latency
- Predictable latency can be required for deadlines with business partners; missing a deadline can have financial consequences

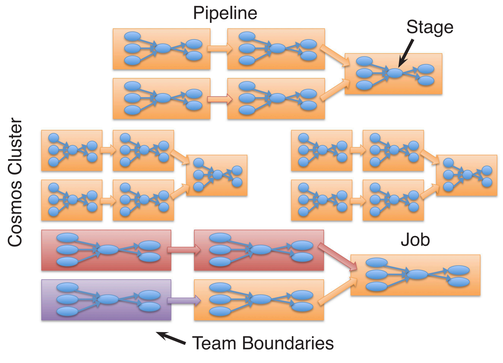
It would be easy to provide deadlines if job latency had low variance; unfortunately, it does not.



Why does latency vary?

- Pipeline complexity: Users develop multi-stage pipelines of dependant jobs, variance in earlier jobs impacts later ones
- Noisy environment: Simultaneous data parallel jobs compete for highly utilized shared resources, which can also fail

The Cosmos Environment



Cosmos Components

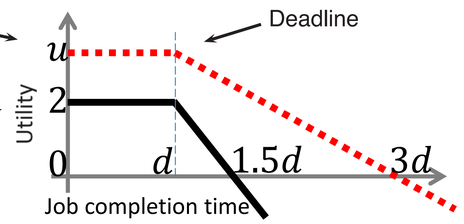
- CosmosStore: distributed storage layer
- Dryad: data-parallel execution engine
- SCOPE: SQL-like query language for Dryad

- Cosmos is Microsoft's data parallel processing environment
- It primarily supports Bing, Microsoft AdCenter, and MSN
- Cosmos clusters contain 1000s of commodity servers, each running multiple tasks for many jobs
- Resources are managed by granting *tokens* to tasks
- Tokens are de-normalized weights in the scheduler and guarantee a fixed slice of CPU and memory

Expressing Performance Targets

For single jobs, scale doesn't matter

For multiple jobs, use financial penalty

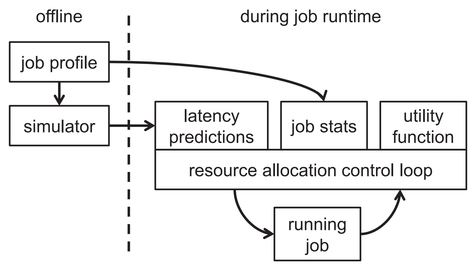


Users provide utility curves to express performance targets

Our Goal:
Maximize utility, while **minimizing** resources by dynamically **adjusting** the allocation

Jockey

Conceptually, Jockey is
1) a function from *progress* and *allocation* to *remaining run time*
2) a control-loop which dynamically adjusts the resource allocation



Our paper provides details and evaluation of each component

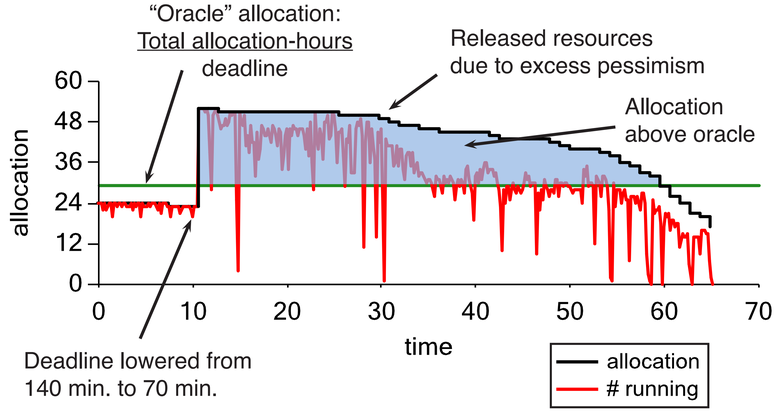
Conclusion

Problem	Solution
Pipeline complexity	Use a simulator
Noisy environment	Dynamic control

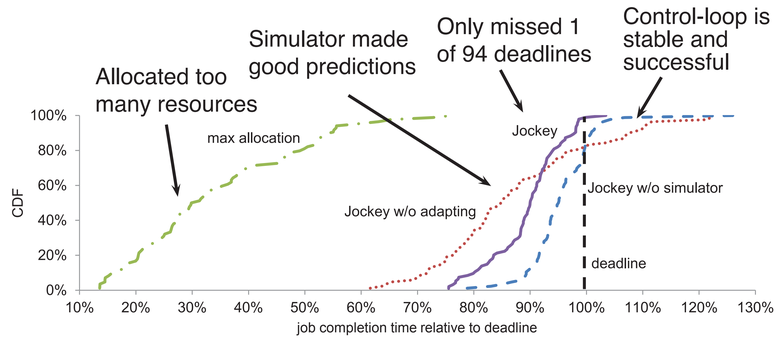
- Jockey works without requiring latency guarantees from individual cluster components
- When a shared environment is underloaded, guaranteed latency brings predictability to the user experience
- When a shared environment is overloaded, utility-based resource allocation ensures jobs are completed by importance

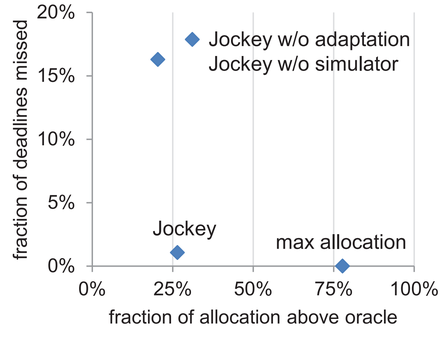
Evaluation

How Jockey Managed a Real Job in a Production Cluster



Jockey's Performance Relative to other Allocation Schemes





- Compared with a naive max allocation scheme, and simulator and control-loop independently
- 21 jobs in production cluster, CPU use: ~80%
- Two metrics: Did jobs complete before deadline? Minimized impact on the rest of the cluster?