dV/dt Accelerating the Rate of Progress towards Extreme Scale Collaborative Science

Bill Allcock (ANL)
Douglas Thain (ND)
Ewa Deelman (USC)
Frank Wuerthwein (UCSD)
Miron Livny (UW)

USC Viterbi
School of Engineering
Information Sciences Institute

Thesis

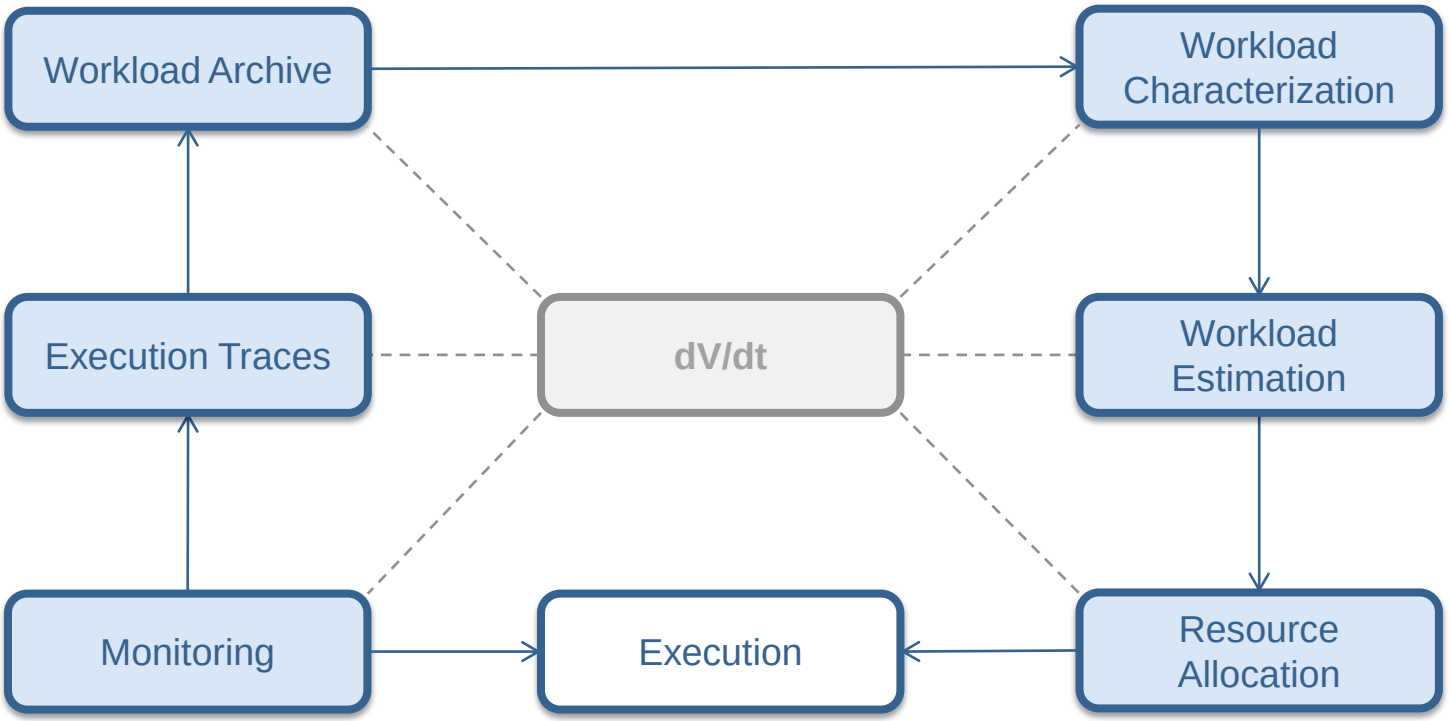
- Researchers band together into dynamic collaborations and employ a number of applications, software tools, data sources, and instruments
- They have access to a growing variety of processing, storage and networking resources
- Goal: “make it easier for scientists to conduct large-scale computational tasks that use the power of computing resources they do not own to process data they did not collect with applications they did not develop”

Challenges today

- Estimate the application resource needs
- Finding the appropriate computing resources
- Acquiring those resources
- Deploying the applications and data on the resources
- Managing applications and resources during run
- Make sure the application actually finishes successfully!

- **Approach:** Develop a framework that encompass the five phases of collaborative computing—estimate, find, acquire, deploy, and use

Overview of the Resource Provisioning Loop



Measuring and recording HPC applications

Argon Leadership Facility

- Argonne is working on techniques for gathering job data that work in a Blue Gene environment
  - Job run times
    - Use scheduler data for both scheduler and individual task data.
  - Disk I/O
    - Use Darshan (joint development between ALCF and MCS)
    - Users can opt out, but we can assist with user issues if needed
    - Gaining traction at other sites, so could become a more general solution
  - Peak RAM usage
    - Using built in HW performance counters (AutoPerf)
    - Users can take control of performance counters preventing this from working
- Summer Students prototyped a database for storing / querying this data
  - First version was specific to DVDT requested data
- On-going Work
  - One of the students will be working with ALCF staff part time during the school year to generalize the schema.
  - This will also be used internally for troubleshooting / optimization and the goal is to make it available to users.
  - Push the results to the archive

Darshan overview

- Runtime library for characterization of application I/O
  - Instrumentation is inserted at build time (for static executables) or at run time (for dynamic executables)
  - Captures POSIX I/O, MPI-IO, and limited HDF5 and PNetCDF functions
- Minimal application impact
  - Bounded memory consumption per process
  - Records strategically chosen counters, timestamps, and histograms
  - Reduces, compresses, and aggregates data at MPI_Finalize() time
- Compatible with IBM BG, Cray, and Linux environments
  - Deployed system-wide or enabled by individual users
  - Instrumentation is enabled via software modules, environment variables, or compiler scripts
  - No source code modifications or changes to build rules
  - No file system dependencies
  - Currently beta testing Cray PE 2.x support for XC30 systems

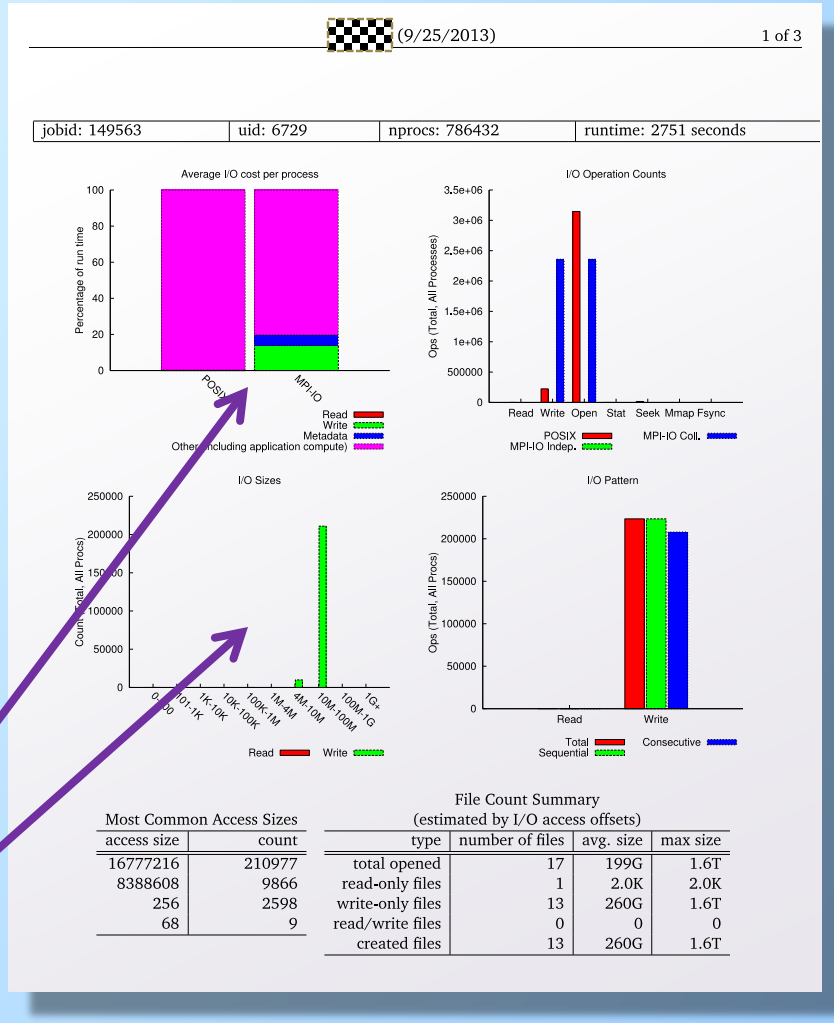
Darshan analysis tools

- Each job instrumented with Darshan produces a compact characterization log file
- Darshan command line utilities can be used to analyze these log files
- Example: Darshan-job-summary.pl produces a 3-page PDF file summarizing various aspects of I/O performance

- This figure shows the I/O behavior of a 786,432 process turbulence simulation (production run) on Mira
- Application is write intensive and benefits greatly from collective buffering

Example measurements: % of runtime in I/O

access size histogram



Most Common Access Sizes

access size	count
16777216	210977
8388608	9866
256	2598
68	9

File Count Summary
(estimated by I/O access offsets)

type	number of files	avg. size	max size
total opened	17	199G	1.6T
read-only files	1	2.0K	2.0K
write-only files	13	260G	1.6T
read/write files	0	0	0
created files	13	260G	1.6T

Characterization of a HTC workload:
The Compact Muon Solenoid (CMS) Experiment

University of Southern California
University of Wisconsin–Madison

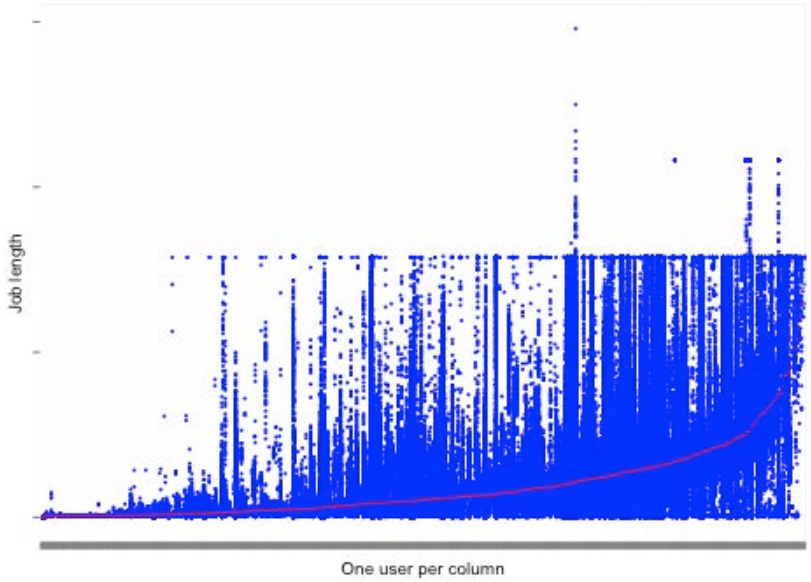
Workload Characteristics

Characteristics of the CMS workload for a period of a month (Aug 2014)

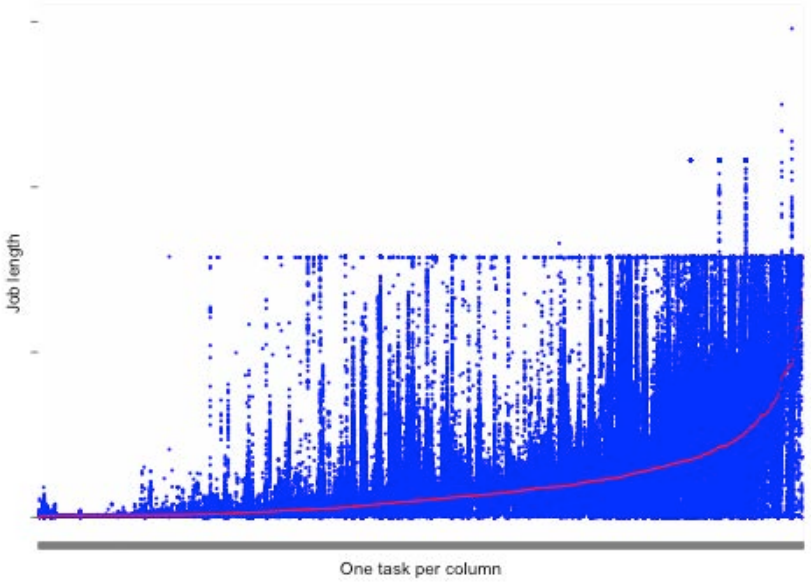
Characteristic	Data
General Workload	
Total number of jobs	1,435,280
Total number of users	392
Total number of execution sites	75
Total number of execution nodes	15,484
Jobs statistics	
Completed jobs	792,603
Preempted jobs	257,230
Exit code (!= 0)	385,447
Average job runtime (in seconds)	9,444.6
Standard deviation of job runtime (in seconds)	14,988.8
Average disk usage (in MB)	55.3
Standard deviation of disk usage (in MB)	219.1
Average memory usage (in MB)	217.1
Standard deviation of memory usage (in MB)	659.6

Workload Execution Profiling

- The workload shows similar behavior to the workload analysis conducted in [Sfiligoi 2013]
- The magnitude of the job runtimes varies among users and tasks

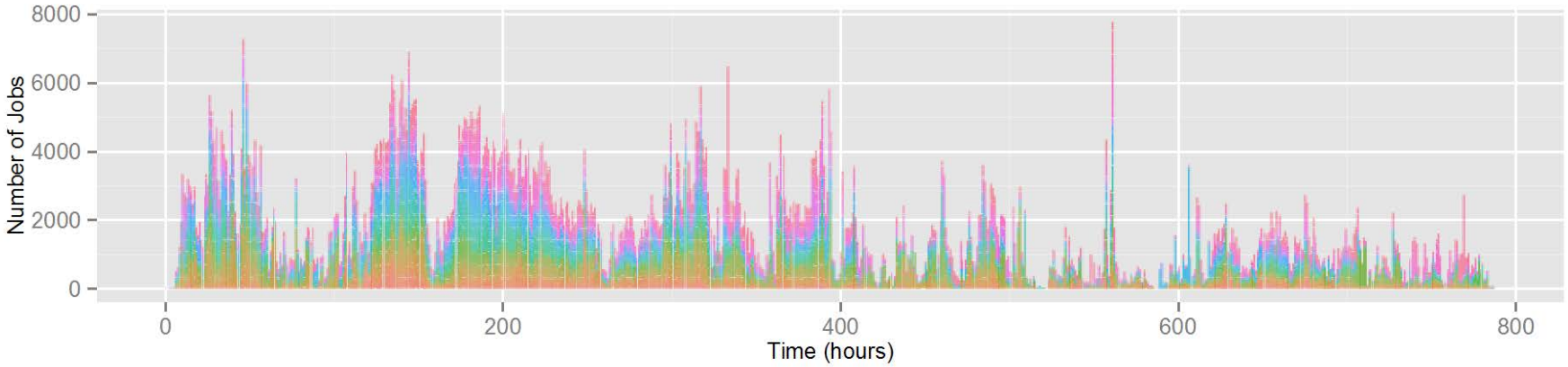


Job runtimes by user
sorted by per-user mean job runtime



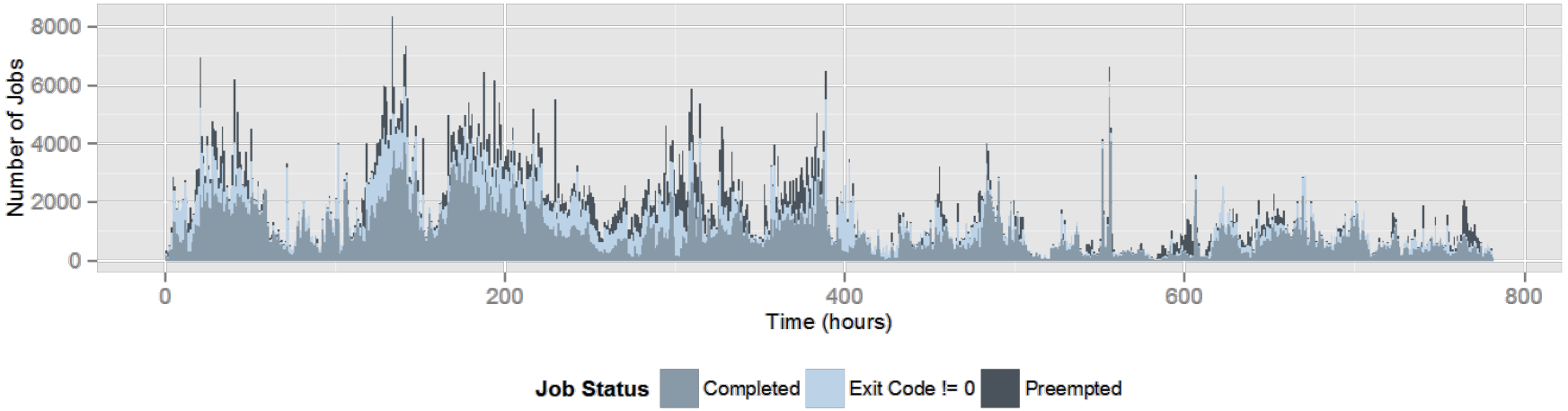
Job runtimes by task
sorted by per-task mean job runtime

Workload Execution Profiling (2)



Job start time rate

Colors represent different execution sites – job distribution is relatively balanced among sites

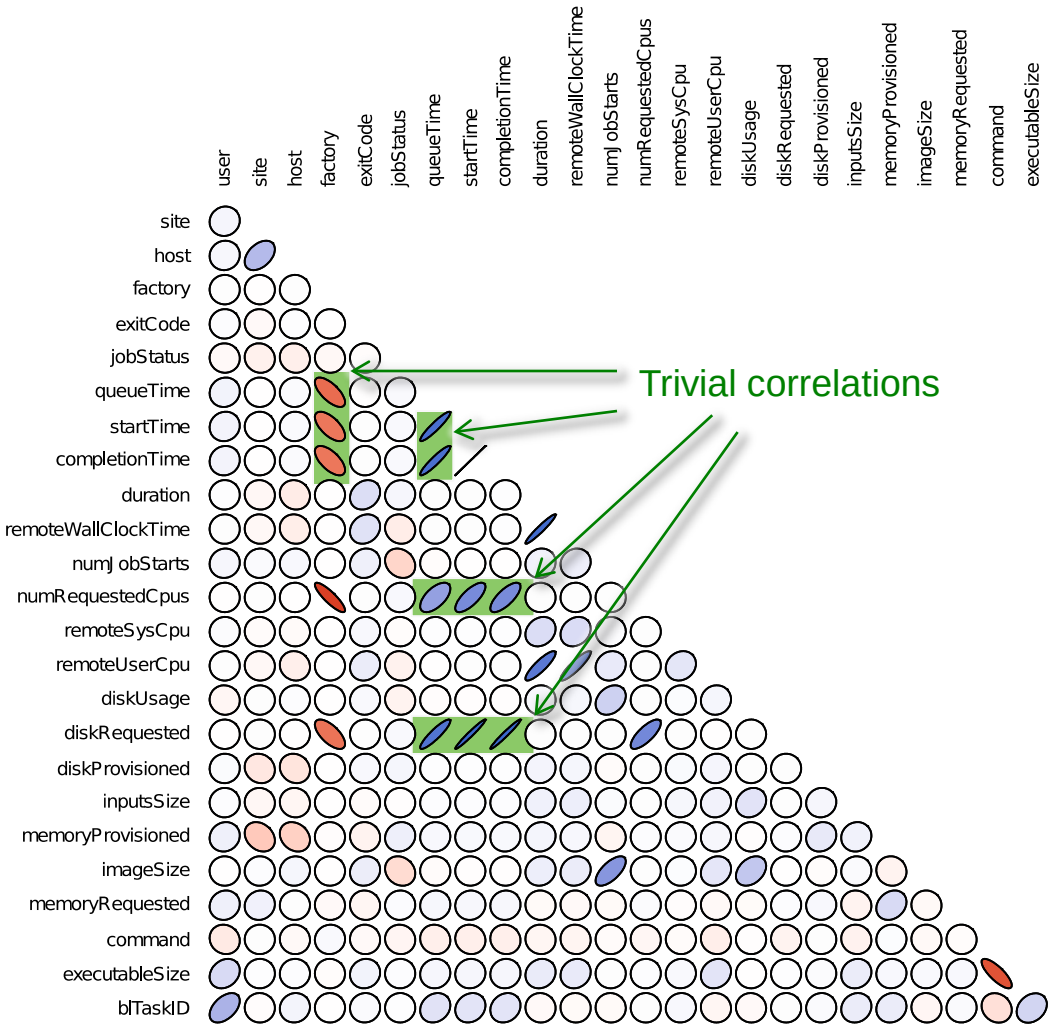


Job completion time rate

Colors represent different job status

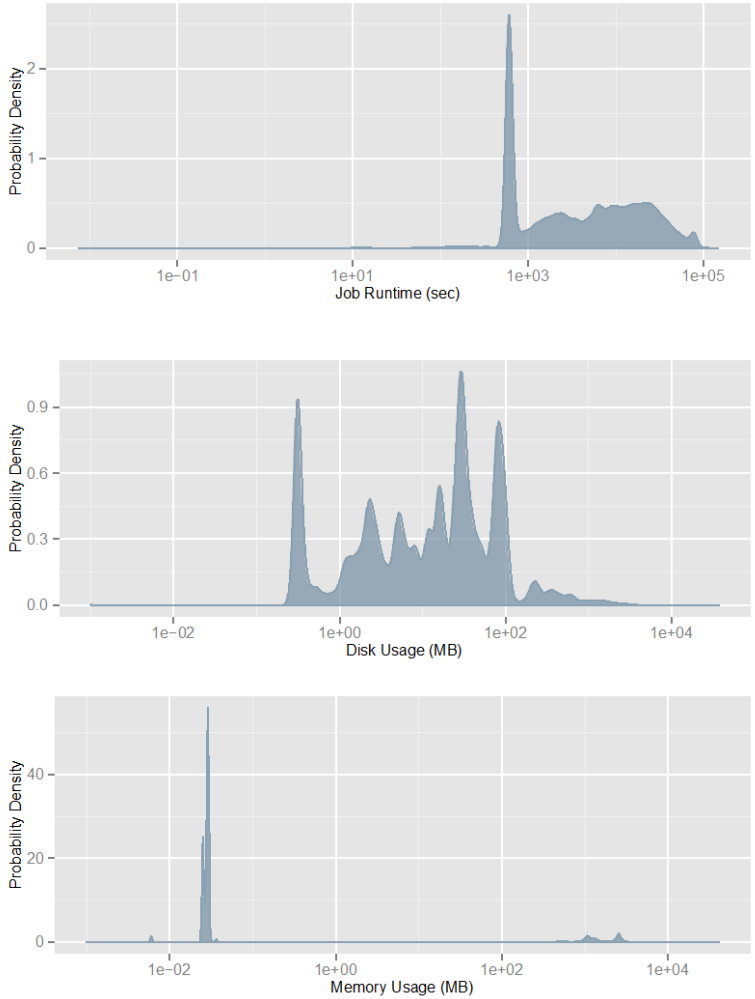
Workload Characterization

- Correlation Statistics
  - Weak correlations suggest that none of the properties can be directly used to predict future workload behaviors
  - Two variables are correlated if the ellipse is too narrow as a line



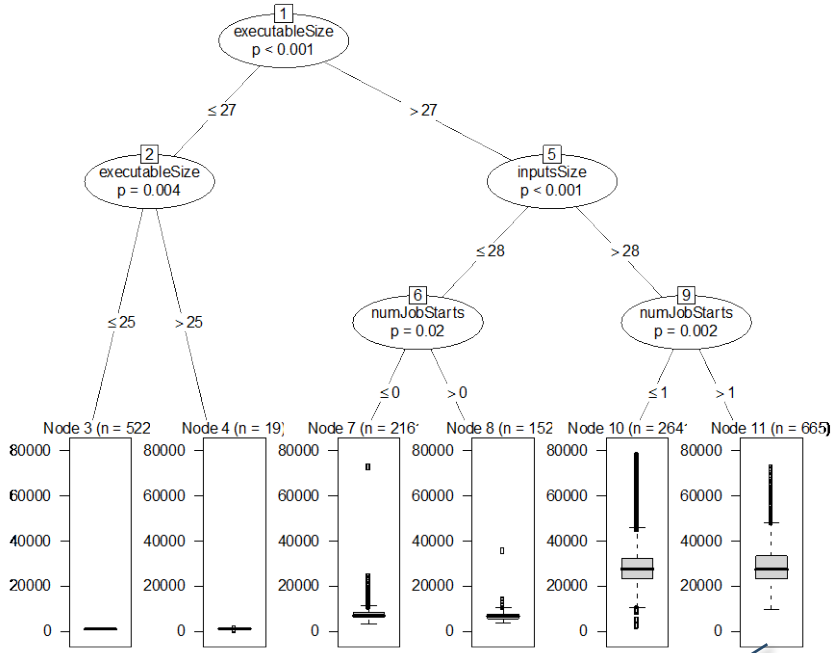
Workload Characterization (2)

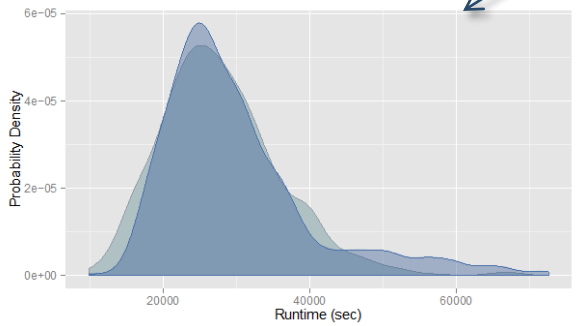
- Correlation measures are sensitive to the data distribution
- Probability Density Functions
  - Do not fit any of the most common families of density families (e.g. Normal or Gamma)
- Our approach
  - Recursive partitioning method to combine properties from the workload to build Regression Trees



Regression Trees

- The recursive algorithm looks for PDFs that fit a family of density
  - In this work, we consider the Normal and Gamma distributions
  - Measured with the Kolmogorov-Smirnov test (K-S test)





The PDF for the tree node (in blue) fits a Gamma distribution (in grey) with the following parameters:

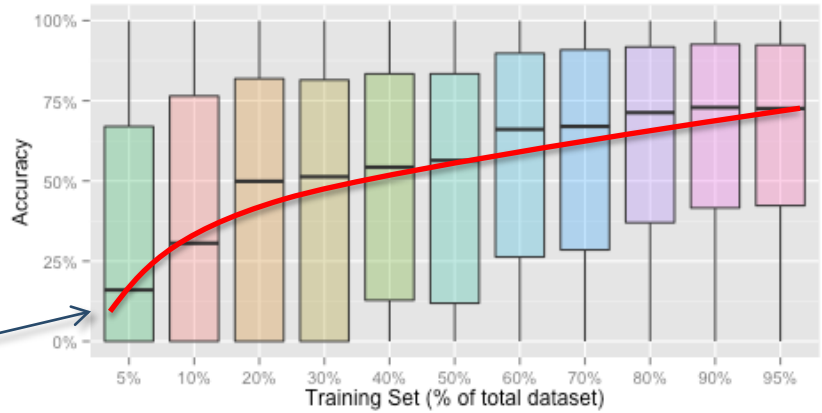
Shape parameter = 12
Rate parameter = 5×10^{-4}
Mean = 27414.8
p-value = 0.17

Job Estimation: Experimental Results

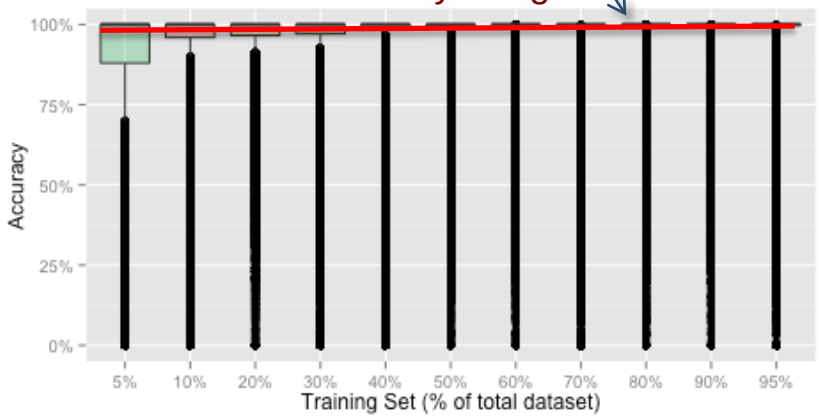
- Based on the regression trees
  - We built a regression tree per user
  - Estimates are generated according to a distribution (Normal or Gamma) or a uniform distribution

The median accuracy increases as more data is used for the training set

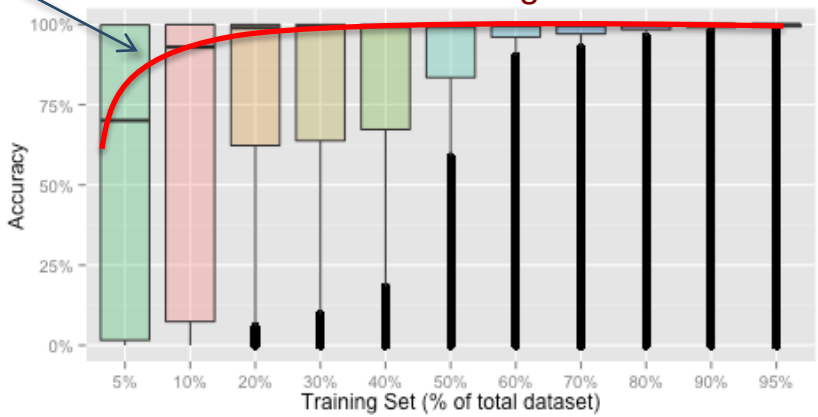
Job Runtime



Memory Usage



Disk Usage



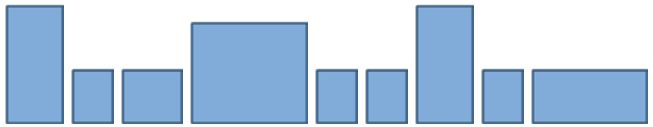
Average accuracy of the workload dataset
The training set is defined as a portion of the entire workload dataset

Resource Allocation

University of Notre Dame

Introduction

- Tasks have different sizes (known at runtime) while computation nodes have fixed sizes



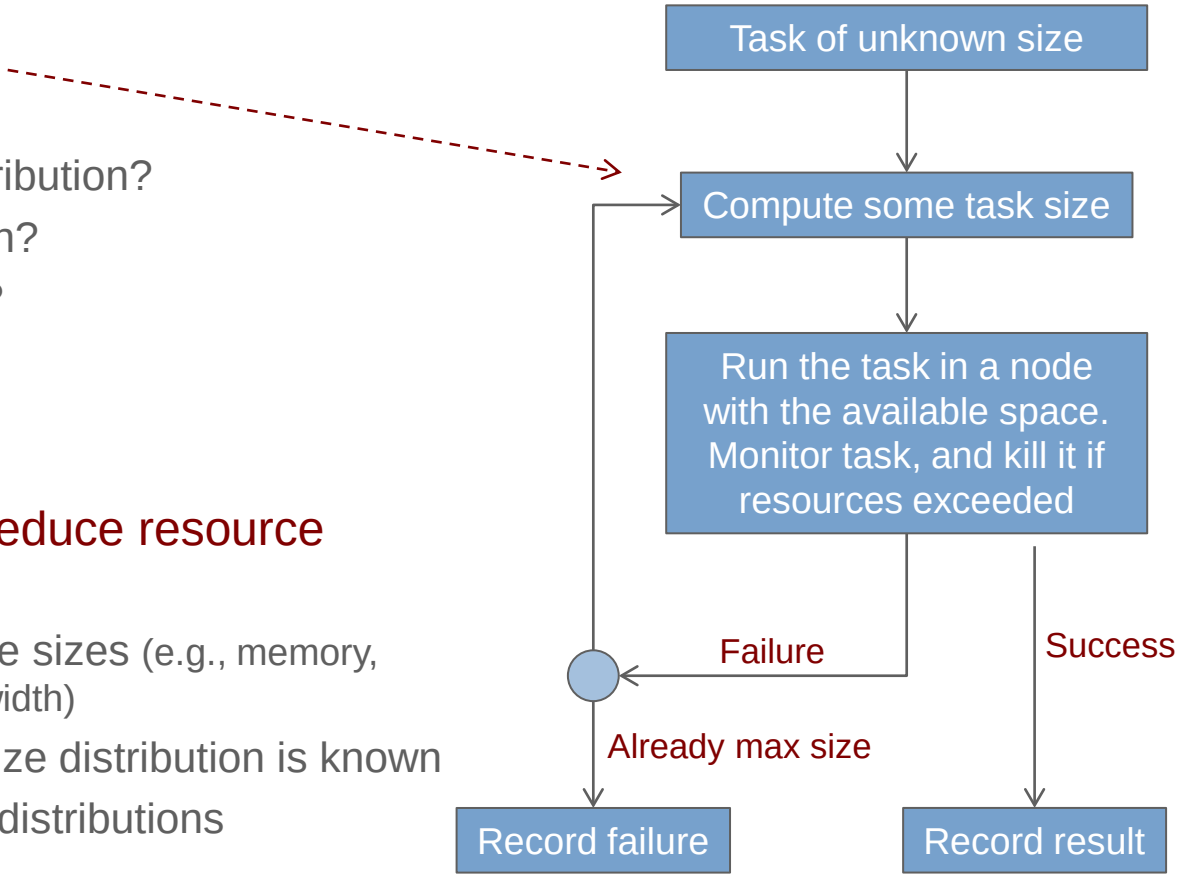
Tasks

Computation Nodes

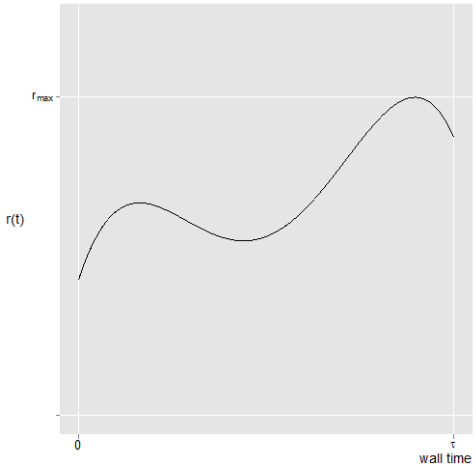
- Resource allocation strategies
  - One task per node
    - Resources are underutilized
    - Throughput is reduced
  - Many tasks per node
    - Resources are exhausted
    - Throughput is reduced

General Approach

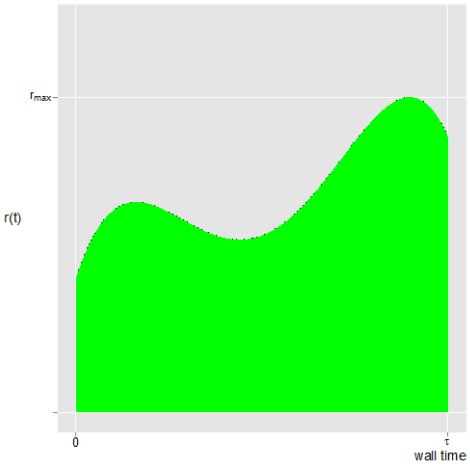
- Setting tasks
  - What do we know?
    - Maximum size?
    - Size probability distribution?
    - Empirical distribution?
    - Perfect information?
- Our approach
  - Setting task sizes to reduce resource waste
    - Modeling of resource sizes (e.g., memory, disk, or network bandwidth)
    - Assumes the task size distribution is known
    - Adapts to empirical distributions



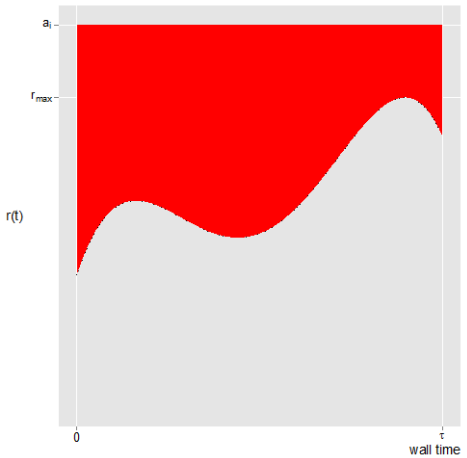
Resource Waste Modeling



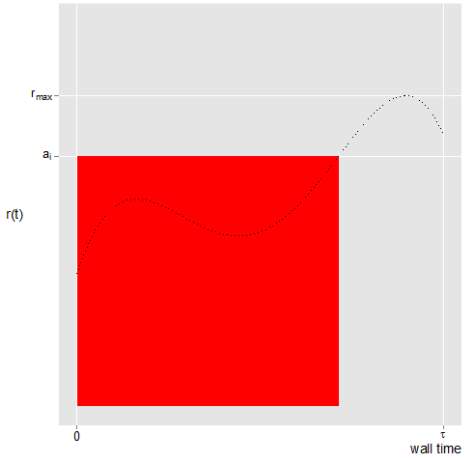
Model the task resource as a function of time



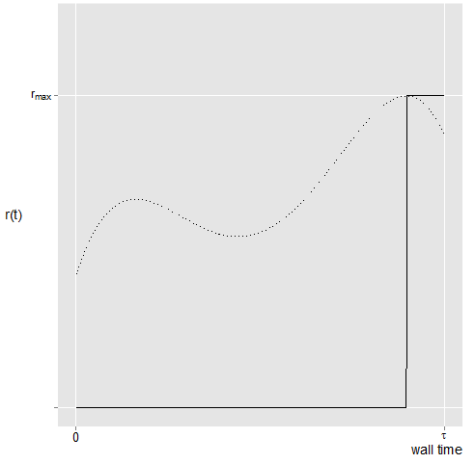
Model the task resource usage as resource x time (area below the curve)



Overestimating size
(waste is the area above the curve)



Underestimating size
(waste is resource x time until resource exhaustion)



Single Peaks Model
Simplifying assumption: any resource exhaustion only happens at time of maximum peak
(i.e., resource usage looks like a step function)

Finding Task Sizes to Minimize Waste

- Random Variables for the Single Peaks Model
  - Maximum peak size
  - Time for successful execution
  - Time at which maximum peak occurs
- Task sizes (allocations) to be tried are found by solving:

$$\arg\min_A(\mathrm{E}[\mathrm{waste}(\mathrm{peak}, \mathrm{time}, \mathrm{time\ to\ peak}), \mathrm{A}])$$

on which:

$$A = [a_1, a_2, \ldots, a_m]$$

is the sequence of task sizes to be computed (that is, the task sizes to be tried).

Example: Two-step sequence with “Slow Peaks”

- Worst-case
  - Peak occurs at the end of execution

$$A = [a_1, a_m]$$ ← a_m given by max size of computation node, only a_1 is unknown.

$$\text{argmin}_A(\text{E}[\text{waste}(\text{peak}, \text{time}), \text{A}])$$

  - Optimality Condition Found

$$p(a_1) = \frac{1}{a_m}$$

Applied to the exponential distribution

$$p(a) = \lambda \exp(-\lambda a)$$

$$a_1 = \frac{1}{\lambda} \ln(\lambda a_m)$$

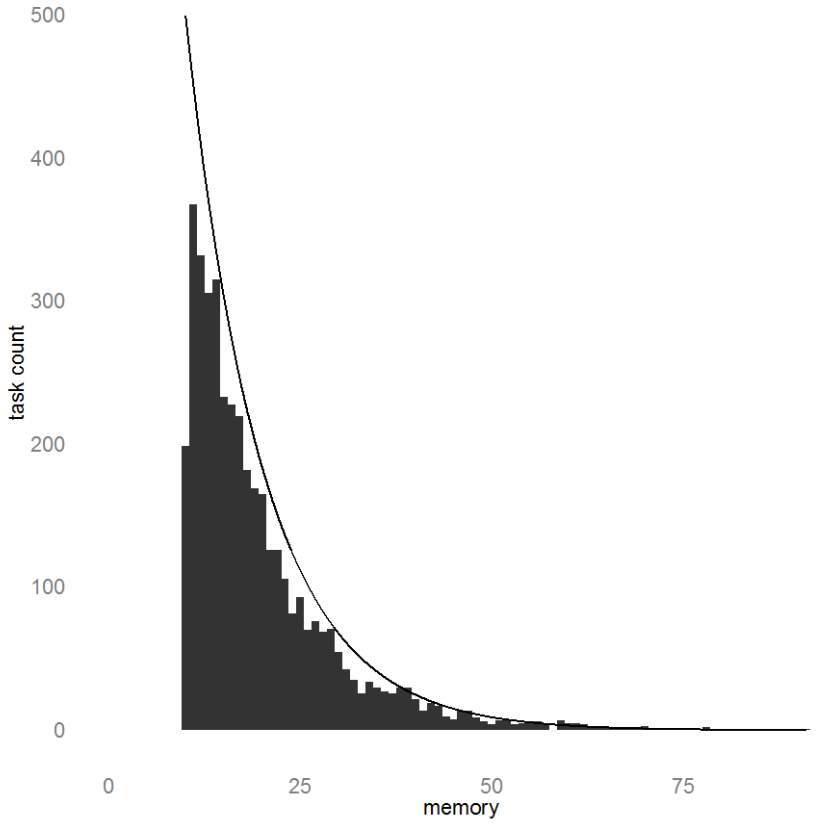
Adapted to observations gathered at runtime

$$\text{argmin}_{a_1}(a_1 + a_m P(a_1))$$

↑ can be approximated with histograms at runtime

Synthetic Workflow Experiment

- Exponential Distribution
  - 5000 Tasks
  - Memory according to an exponential distribution
    - Shifted min 10 MB, truncated max 100 MB, average 20 MB
  - Tasks run anywhere from 10 to 20 seconds
  - 100 computation nodes available, from ND Condor pool
  - Each node with 4 cores and a limit of 100 MB of memory



Synthetic Workflow Experiment (2)

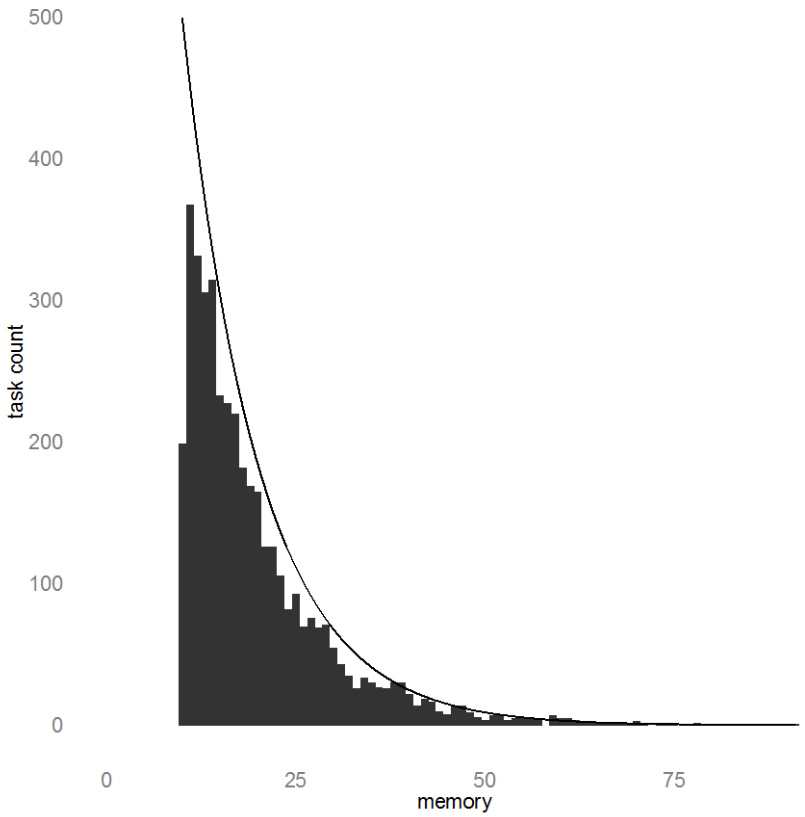
- Exponential Distribution

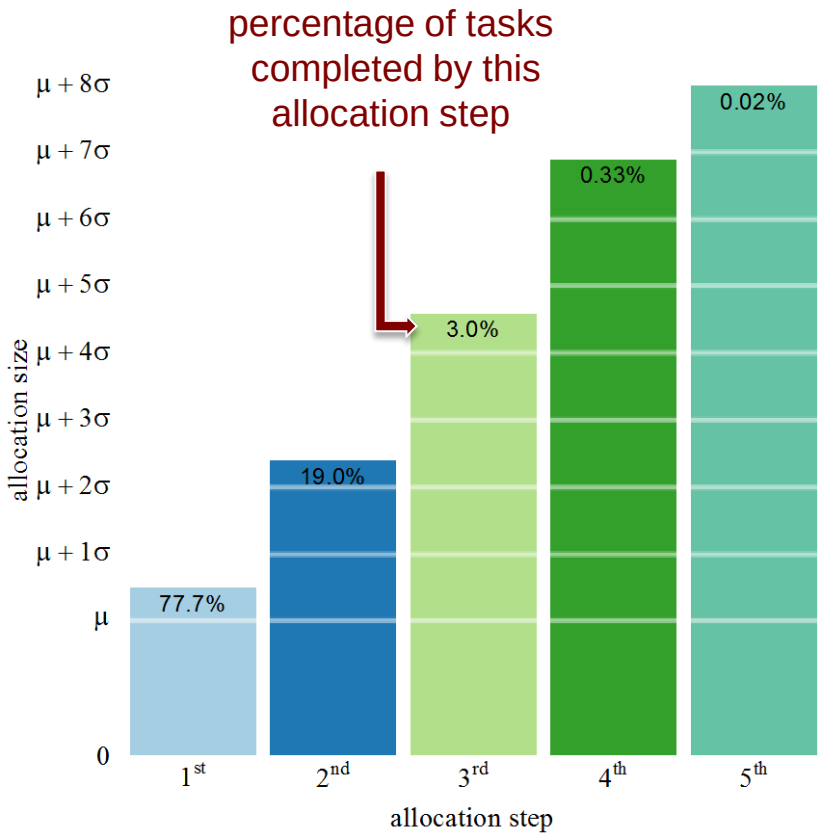
Mode	First allocation	Tasks retried	Throughput (tasks/sec)
Perfect knowledge	*	0	26.46
Fixed (maximum)	100	0	9.03
Fixed (maximum/2) + 1	51	94	8.59
Fixed (maximum/2)	50	101	15.97
Optimal $a_{min} + (1/\lambda)\ln(\lambda(a_{max} - a_{min})$	32	518	19.61
Dynamic	31 (Found after 150 tasks)	560	18.32
Fixed (average + std.dev.)	30	618	17.01
Fixed (average)	20	1680	13.93

from known distribution

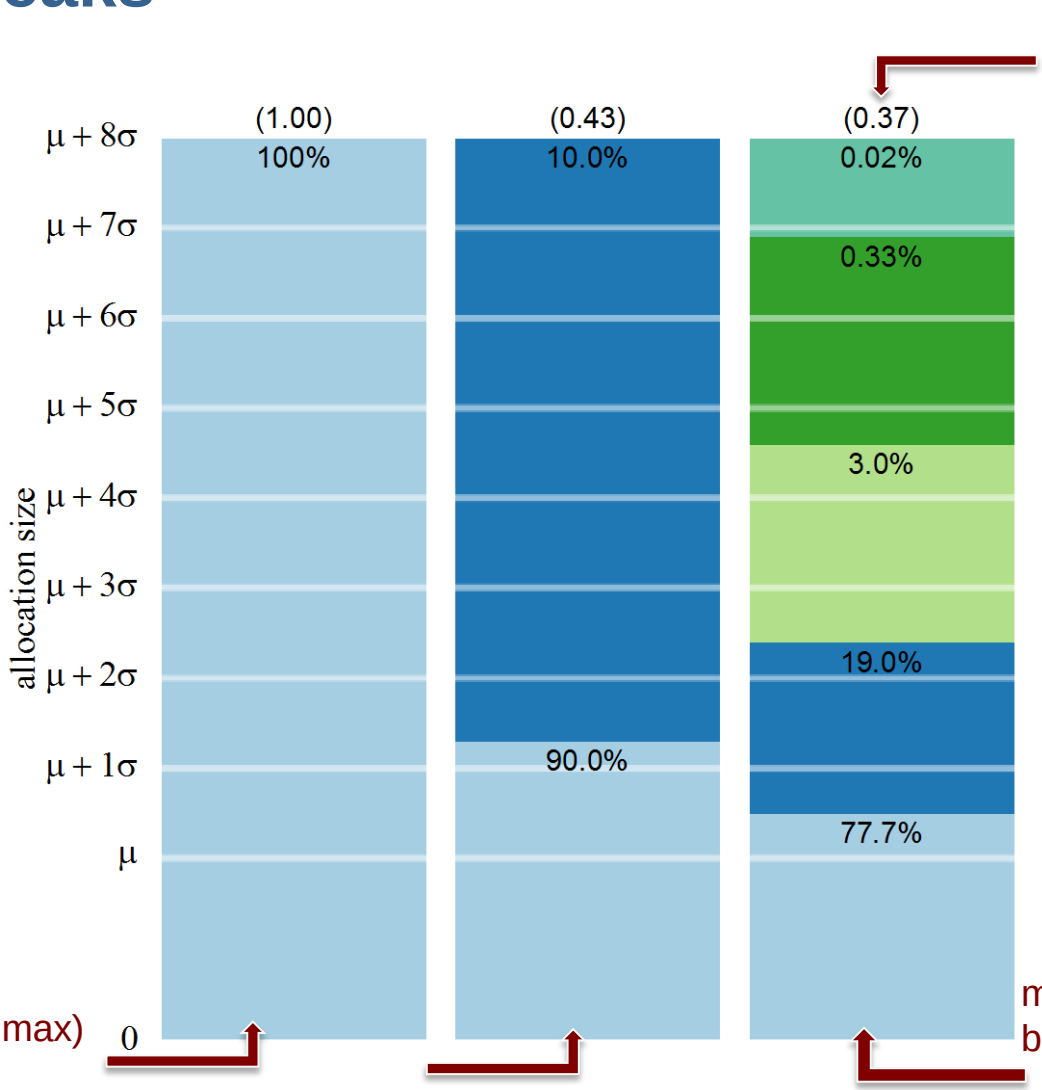
from empirical distribution

Example: Multi-step sequence with “Slow Peaks”

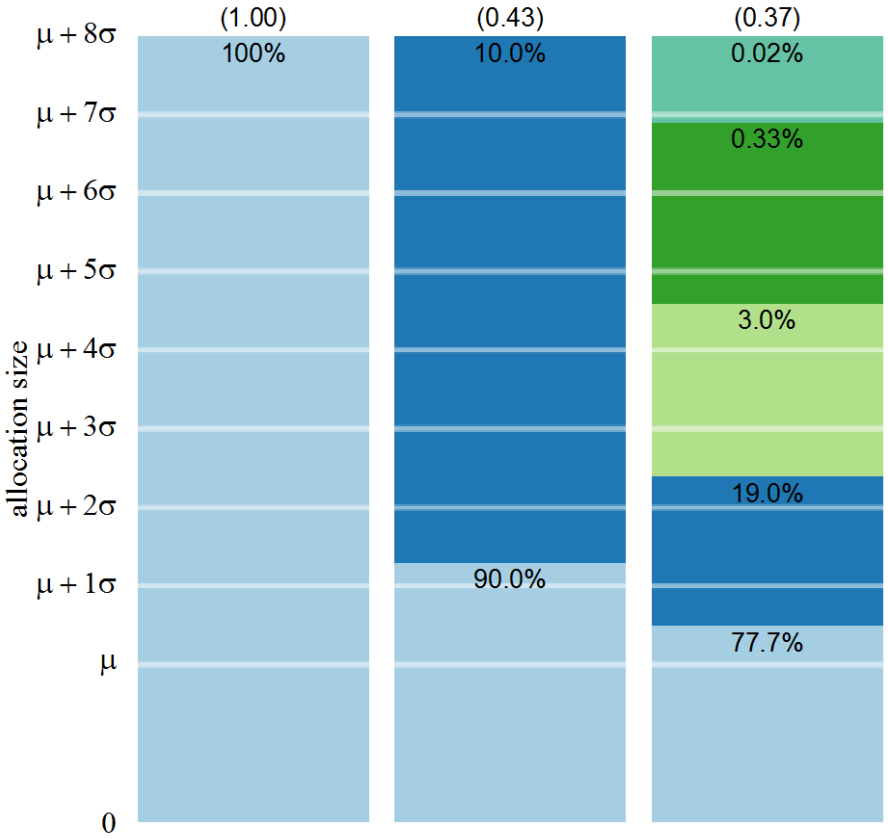




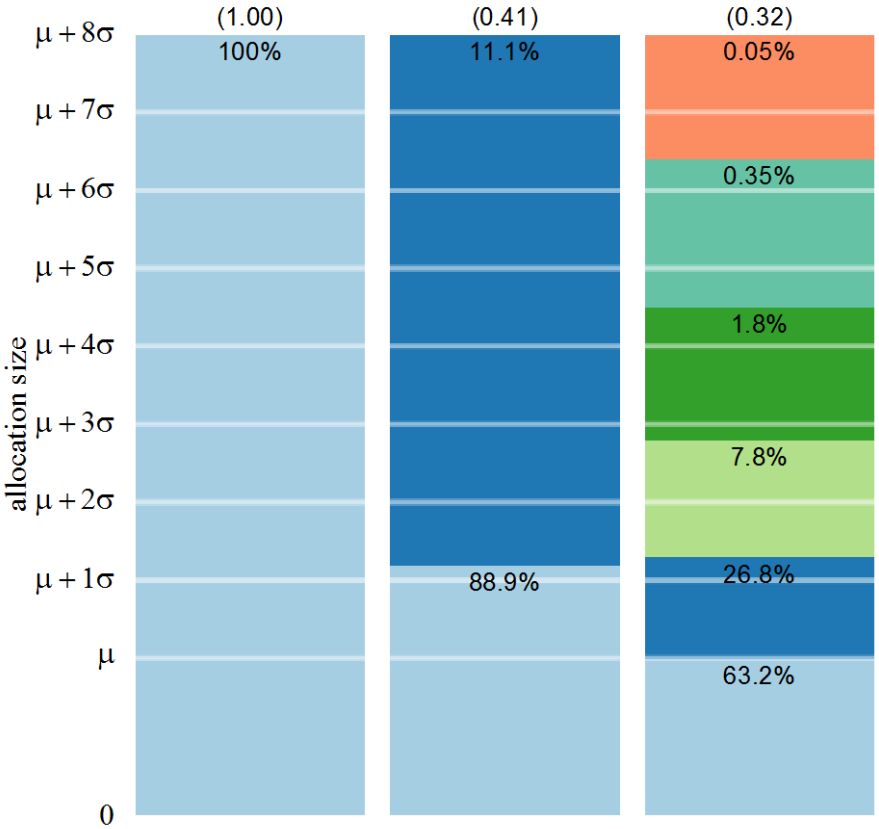
Example: One,Two and Multi-step sequences with "Slow Peaks"



One, Two, and Multi-step sequence with “Slow Peaks” v.s. “Uniform Peaks”



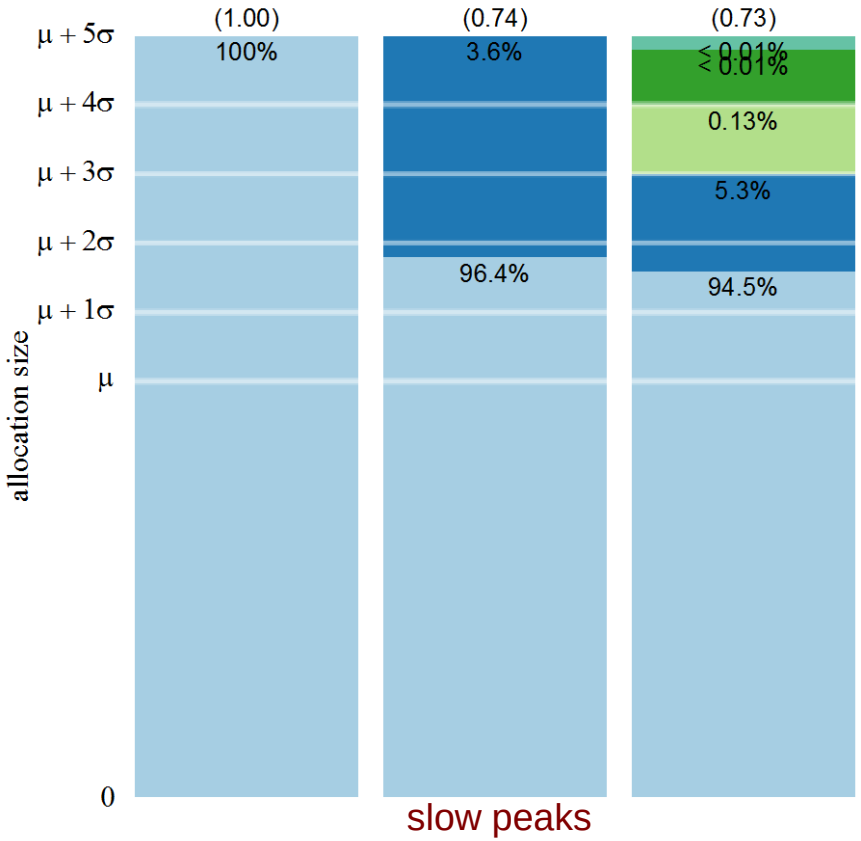
slow peaks (resource peaks at end of execution, as previous slide)



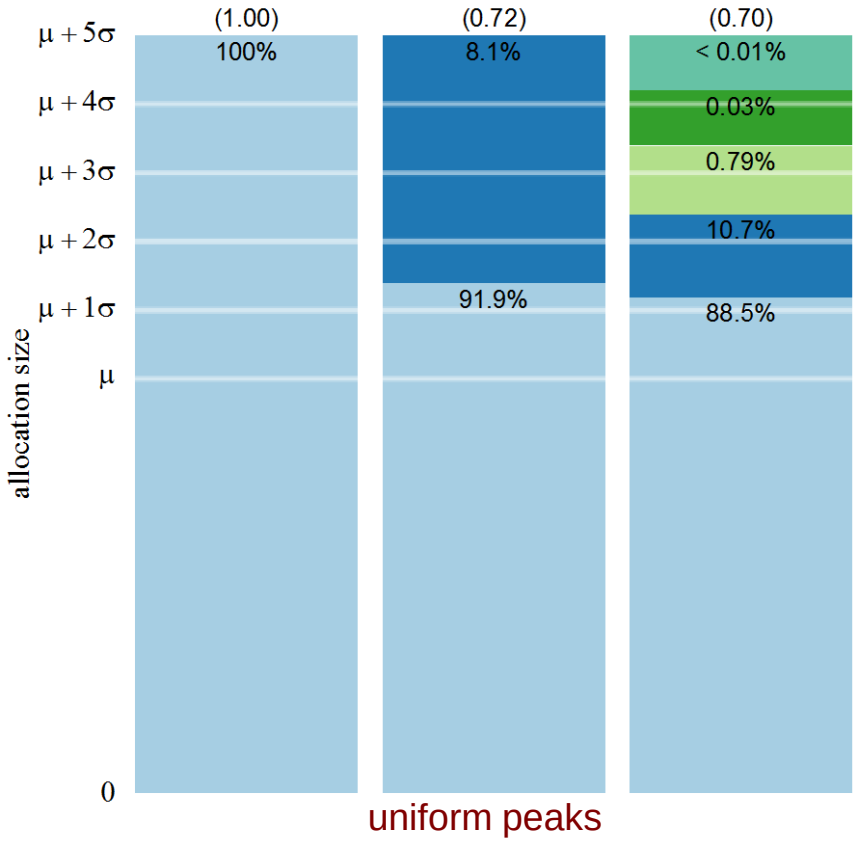
uniform peaks (resource peaks occur at any time during execution)

One, Two, and Multi-step sequence with “Slow Peaks” v.s. “Uniform Peaks”

- Resource peaks with Gaussian distribution
  - mean=60, sd=10, min=20, max=110



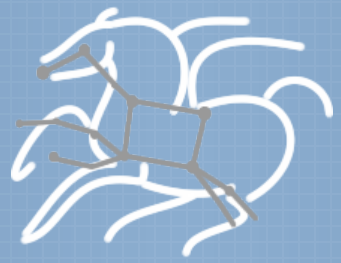
slow peaks



uniform peaks

What question does our research motivate us to now ask?

- How to measure, profile and account for the consumption of hidden/shared resources?
- How to minimize the impact of the monitoring process on the operational aspects of production systems?
- How to manage private data collected by the monitoring system?
- How to uniquely identify applications acros sites and users?
- What is the right tradeoff between machine functionality and machine performance?



dV/dT: Accelerating the Rate of Progress Towards Extreme Scale Collaborative Science

Thank you.

deelman@isi.edu

http://pegasus.isi.edu