Improving Karnak's Wait Time Predictions

Jungha Woo

Research Computing at Purdue University

Shava Smallen

San Diego Supercomputer Center

J.P. Navarro

Argonne National Laboratory

Overview

- Functionality
- Predictors of the wait times
- Prediction algorithm – decision tree
- Analysis on prediction accuracy
- Improvement ideas
- Current status and future work

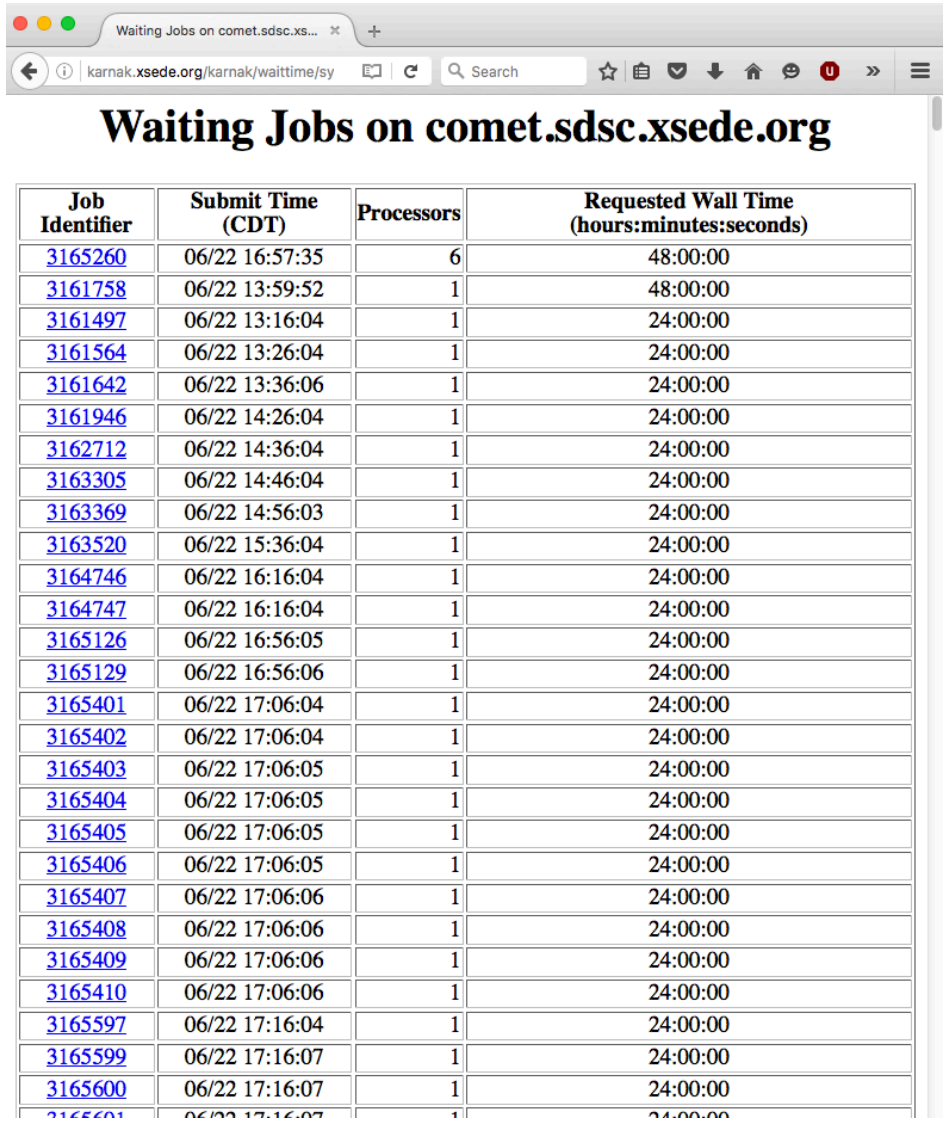
Karnak Prediction Service

- Functionality
  - Predict start time of hypothetical jobs
  - Predict start time of queued jobs
  - Provide information about current and recent jobs
- Interfaces
  - REST (XML and plain text), HTML
    - http://karnak.xsede.org
  - Command line programs
  - Java client library
- Available in XSEDE user portal

Slide modified from Warren Smith, with permission

Predicting Waiting Jobs

- A little information is presented about each job
- Jobs are presented in queue order
  - May not be schedule order
- This is accessible to anyone
  - Does not display
    - user names
    - project names



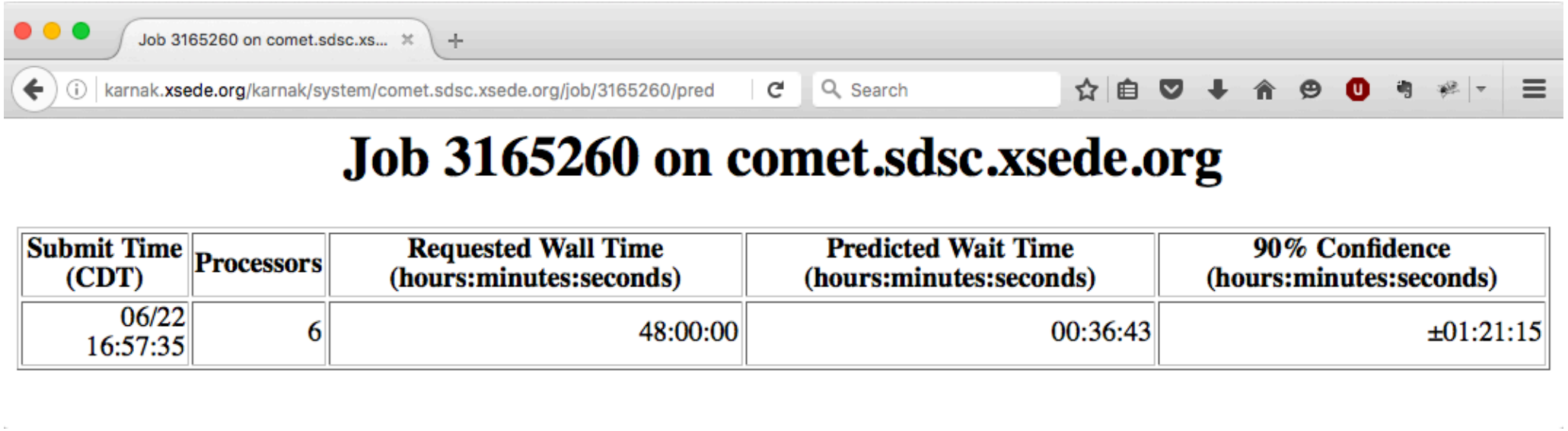
Waiting Jobs on comet.sdsc.xsede.org

Job Identifier	Submit Time (CDT)	Processors	Requested Wall Time (hours:minutes:seconds)
3165260	06/22 16:57:35	6	48:00:00
3161758	06/22 13:59:52	1	48:00:00
3161497	06/22 13:16:04	1	24:00:00
3161564	06/22 13:26:04	1	24:00:00
3161642	06/22 13:36:06	1	24:00:00
3161946	06/22 14:26:04	1	24:00:00
3162712	06/22 14:36:04	1	24:00:00
3163305	06/22 14:46:04	1	24:00:00
3163369	06/22 14:56:03	1	24:00:00
3163520	06/22 15:36:04	1	24:00:00
3164746	06/22 16:16:04	1	24:00:00
3164747	06/22 16:16:04	1	24:00:00
3165126	06/22 16:56:05	1	24:00:00
3165129	06/22 16:56:06	1	24:00:00
3165401	06/22 17:06:04	1	24:00:00
3165402	06/22 17:06:04	1	24:00:00
3165403	06/22 17:06:05	1	24:00:00
3165404	06/22 17:06:05	1	24:00:00
3165405	06/22 17:06:05	1	24:00:00
3165406	06/22 17:06:05	1	24:00:00
3165407	06/22 17:06:06	1	24:00:00
3165408	06/22 17:06:06	1	24:00:00
3165409	06/22 17:06:06	1	24:00:00
3165410	06/22 17:06:06	1	24:00:00
3165597	06/22 17:16:04	1	24:00:00
3165599	06/22 17:16:07	1	24:00:00
3165600	06/22 17:16:07	1	24:00:00

Slide modified from Warren Smith, with permission

Wait Time Prediction

- Two components in the prediction
  - Predicted wait time
  - A confidence interval
    - Provides information about the expected accuracy of the prediction



Slide modified from Warren Smith, with permission

Potential Job

- Describe a potential job
  - Systems & queues
  - Processing cores
  - Wall time
  - Confidence level

Select one or more systems and queues:

System	Queues
☑ comet.sdsc.xsede.org	☐ NFSmove ☐ NSGJRC ☐ check_node ☐ cipres_324 ☐ compute ☐ debug ☐ gpu ☐ gpu-shared ☐ jpg_341 ☐ kganamet_252 ☐ large-shared ☐ mahidhar_279 ☐ mahidhar_337 ☐ maint ☐ sch1 ☐ seagate_SSDs ☐ shared ☐ staging_compute ☐ sys-maint
☑ darter.nics.xsede.org	☐ batch ☐ capability ☐ dmover ☐ hpss ☐ large ☐ login ☐ medium ☐ serial ☐ small
☑ maverick.tacc.xsede.org	☐ gpu ☐ systest ☐ vis
☑ stampede.tacc.xsede.org	☐ designsafe2 ☐ development ☐ gpu ☐ gpudev ☐ large ☐ largemem ☐ normal ☐ normal-2mic ☐ normal-mic ☐ osu ☐ request ☐ serial ☐ sysdebug ☐ systest ☐ vis ☐ visdev

Describe your job:

Processing cores: 1

Requested wall time: 1 : 0 (hours:minutes)

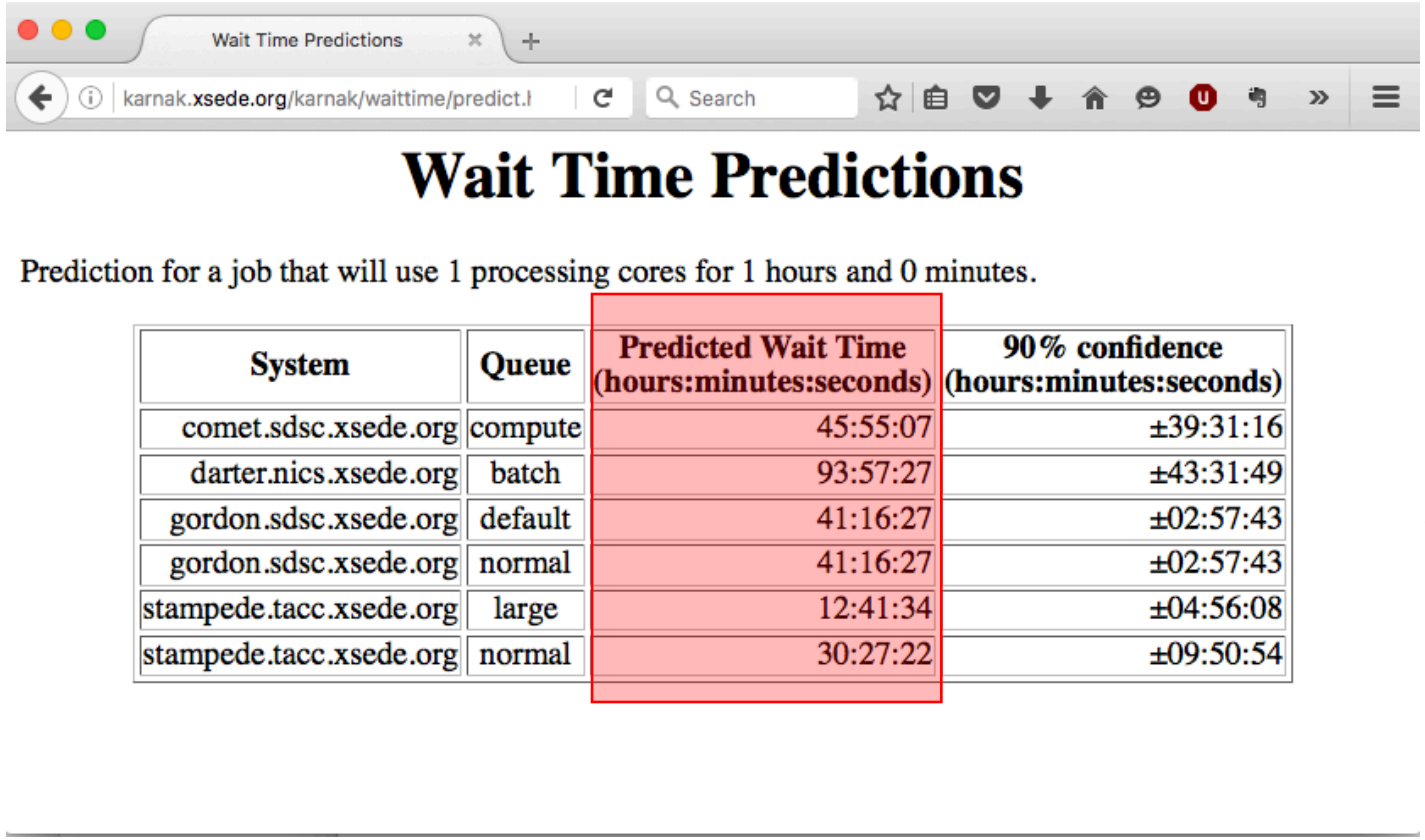
Confidence interval size: 90 %

Submit

Slide modified from Warren Smith, with permission

Prediction for a Potential Job

- Prediction provided for each system/queue
  - N/A if the job can't be submitted to a queue



Wait Time Predictions

Prediction for a job that will use 1 processing cores for 1 hours and 0 minutes.

System	Queue	Predicted Wait Time (hours:minutes:seconds)	90% confidence (hours:minutes:seconds)
comet.sdsc.xsede.org	compute	45:55:07	±39:31:16
darter.nics.xsede.org	batch	93:57:27	±43:31:49
gordon.sdsc.xsede.org	default	41:16:27	±02:57:43
gordon.sdsc.xsede.org	normal	41:16:27	±02:57:43
stampede.tacc.xsede.org	large	12:41:34	±04:56:08
stampede.tacc.xsede.org	normal	30:27:22	±09:50:54

Actual wait time statistics

Wait time(seconds)	Percentile
0	0%
0	25%
3	50%
11	75%
782645 (=9.05 days)	100%

Actual wait time percentile for comet.sdsc.xsede.org for 8 weeks (As of 06/15/2016)

Slide modified from Warren Smith, with permission

Alternative Prediction Formats

- Until now, the predictions are all numeric values
- Users may want simpler answers
  - Color : Blue, Yellow, Red,…
  - Speed: Fastest, medium, slow
  - Grouping: returns grouped results with group numbers such as 0,1,2,3,4,,..
  - Raw : Original format
- Good for fair utilization of systems

Group and LabeledGroup

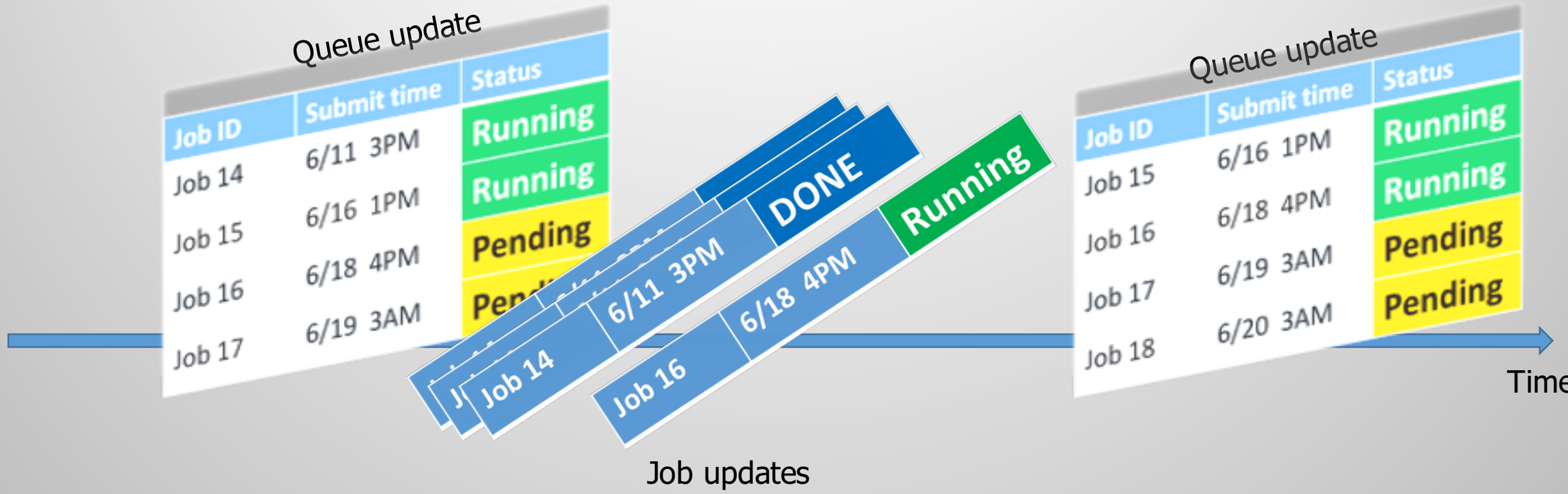
Wait Time/Start time Predictions

Group	Queue@System
1	systest@lonestar4.tacc.xsede.org normal-2mic@stampede.tacc.xsede.org serial@lonestar4.tacc.xsede.org
2	default@gordon.sdsc.xsede.org normal@stampede.tacc.xsede.org debug@gordon.sdsc.xsede.org
3	compute@comet.sdsc.xsede.org gpu@comet.sdsc.xsede.org

Wait Time/Start time Predictions

Group	Queue@System
Fast	systest@lonestar4.tacc.xsede.org serial@lonestar4.tacc.xsede.org normal-2mic@stampede.tacc.xsede.org
medium	default@gordon.sdsc.xsede.org debug@gordon.sdsc.xsede.org normal@stampede.tacc.xsede.org
slow	gpu@comet.sdsc.xsede.org compute@comet.sdsc.xsede.org

Job and Queue Updates from XSEDE Information Service (http://info1.dyn.xsede.org/)



Factors That May Affect Wait Times

- How busy are the target queues?
- How busy are the target systems?
- How demanding is your job?
  - How many CPUs did it ask for?
  - How long was the requested wall time?

- Assumptions
  - Karnak does not know systems' scheduling policy
  - A node can belong to multiple queues
  - A hypothetical job's wait time does not have user name

Wait Time Predictors

- Two key concepts
  - **countAhead**: number of pending jobs ahead of me
  - **workAhead** : remaining work of pending jobs ahead of me

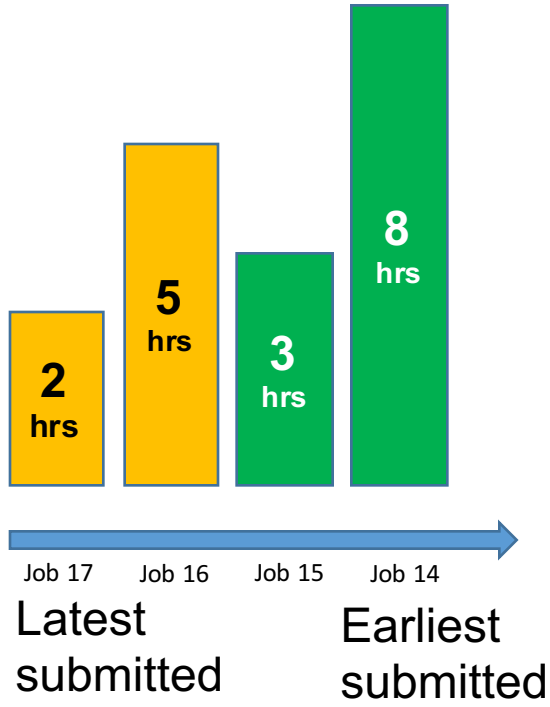
- Compute them for various group of jobs
  - Pending jobs : **countAhead, workAhead**
  - Running jobs : **countRunning, workRunning**
  - Same user jobs
  - Same queue jobs, other queue jobs
  - Same allocation jobs
  - Smaller work jobs

workAhead Computation

Head of queue

Job ID	Submit time	State	# CPU	Requested time	Start time
14	6/11 3PM	**Running**	**2**	**6 hrs**	**6/19 2 am**
15	6/16 1PM	**Running**	**3**	**2 hrs**	**6/19 3 am**
16	6/18 4PM	**Pending**	**5**	**1 hr**	
17	6/19 3AM	**Pending**	**1**	**2 hrs**	

Tail of queue



If a new job were submitted at 6/19 4am, its workAhead would be

workAhead = Sum of CPU hours
= 1 CPU x 2 Hrs + 5 CPU x 1 Hr
= 7 CPU Hrs

Features For Predicting Wait Times

- All features are computed from job and queue updates
  - Input features
    - 26 features for queued job prediction
    - 16 features for unsubmitted job prediction
  - Output feature : waitTime

Job state	Feature	Description	Coverage
	queue	Name of queue job submitted to	Job-specific
	processors	Number of CPUs requested by job	Job-specific
	requestedWallTime	User-specified maximum job execution time	Job-specific
Pending	countAhead	Number of jobs pending ahead of this job	System-wide
Pending	workAhead	Amount of work requested by countAhead jobs	System-wide
Running	countRunning	Number of jobs running ahead of this job	System-wide
Running	processorsRunning	Sum of requested processors of currently running jobs	System-wide
Running	workRunning	Number of remaining hours of running jobs	System-wide

Features For Predicting Wait Times

Pending

Feature	Description	
countAheadQueue	countAhead for the jobs submitted to same queue	Queue-specific
workAheadQueue	workAhead for the jobs submitted to same queue	
countAheadOtherQueues	countAhead for the jobs submitted to other queues	
workAheadOtherQueues	workAhead for the jobs submitted to other queues	
countAheadLessEqualProcs	countAhead for the jobs asked for less or equal processor than this job	Job-specific
workAheadEqualProcs	workAhead for the jobs asked for less or equal processor than this job	
countAheadLessEqualWork	countAhead for the jobs whose remaining work is less than equal to this job	
workAheadLessEqualWork	workAhead for the jobs whose remaining work is less than equal to this job	

Features For Predicting Wait Times

	Feature	Description	
Pending	countAheadUser	countAhead for the jobs submitted by same user	User-specific
	workAheadUser	workAhead for the jobs submitted by same user	
Running	countRunningUser	countRunning for the jobs submitted by same user	
	processorsRunningUser	processorsRunning for the jobs submitted by same user	
	workRunningUser	workRunning for the jobs submitted by same user	
Pending	countAheadProject	countAhead for the jobs having same project number	Project specific
	workAheadProject	workAhead for the jobs having same project number	
Running	countRunningProject	countRunning for the jobs having same project number	
	processorsRunningProject	processorsRunning for the having same project number	
	workRunningProject	workRunning for the jobs having same project number	

Features in this box can only be computed for queued jobs

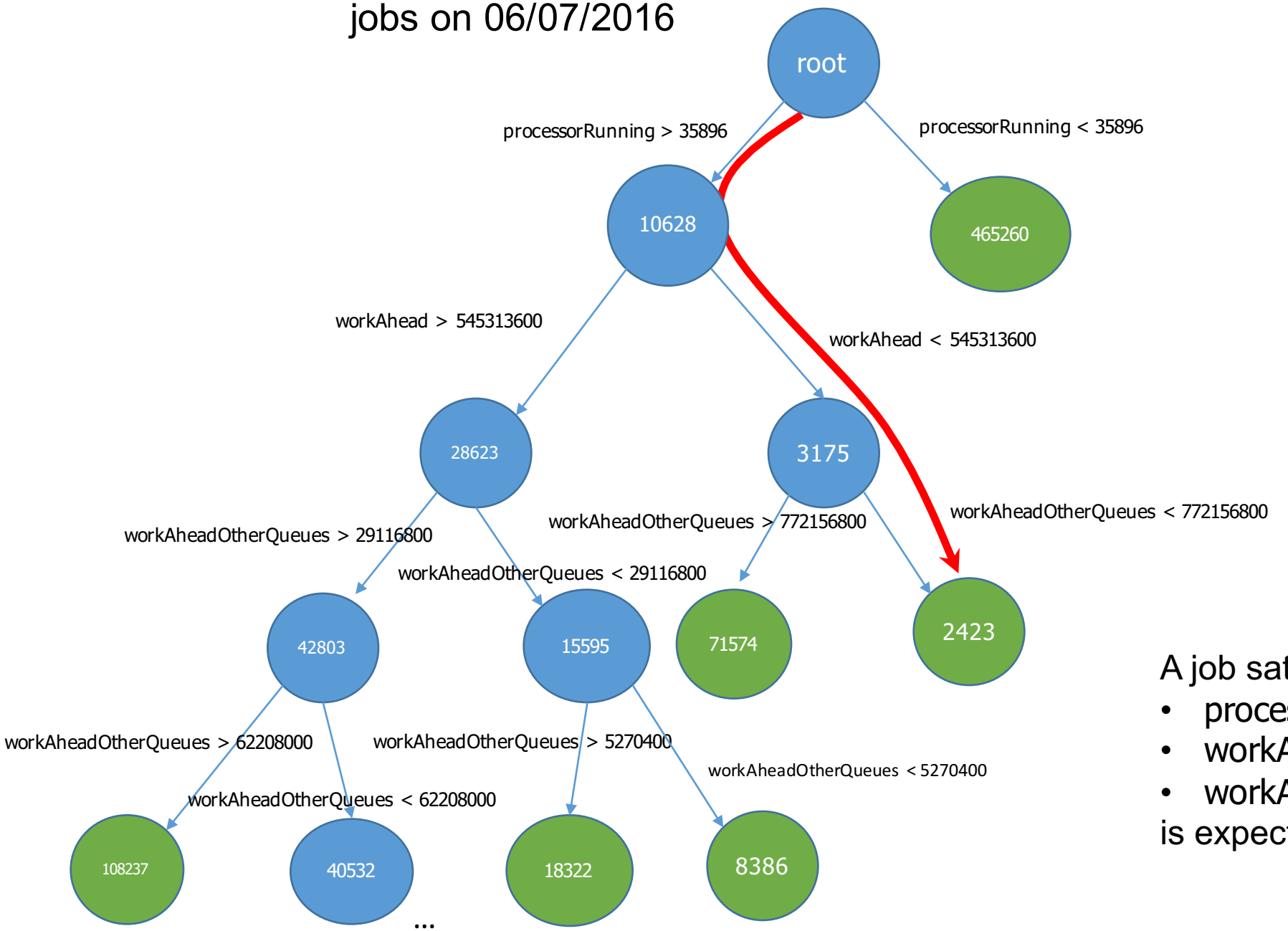
Karnak Uses Decision Trees

- Decision tree
  - Decision tree builds regression or classification models in the form of a tree structure
  - Breaks down a dataset into smaller and smaller subsets having similar wait times
  - At each depth, choose the best feature that reduces the standard deviation
  - Final result is a tree with decision nodes and leaf nodes
- How to use
  - Construct a tree using training data (= past experiences) per system
  - Traverse the tree using a query, leaf node contains the prediction

Decision Tree

- Advantages
  - Simple to understand, and easy to visualize
  - Little data preparation needed
  - The cost of using tree is logarithmic in the number of data points
  - Handle both categorical and numerical data
- Disadvantages
  - Overfitting (= do not generalize data well)
  - Decision tree can be unstable with small variations in the data
  - Learning optimal decision tree is NP-Complete

A decision tree for comet.sdsc.xsede.org with 4 hours of window size jobs on 06/07/2016



Decision node

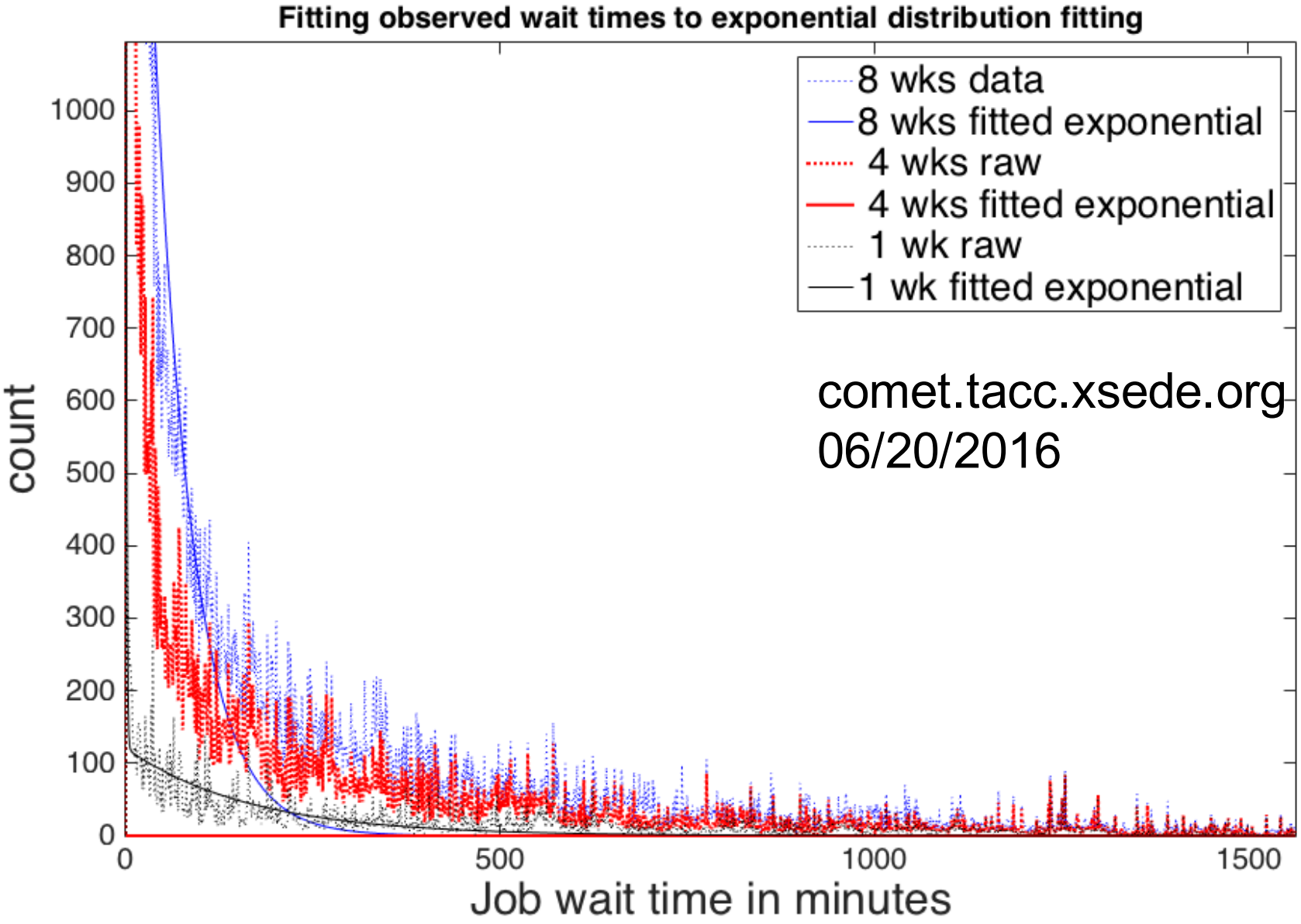
Leaf node

A job satisfying

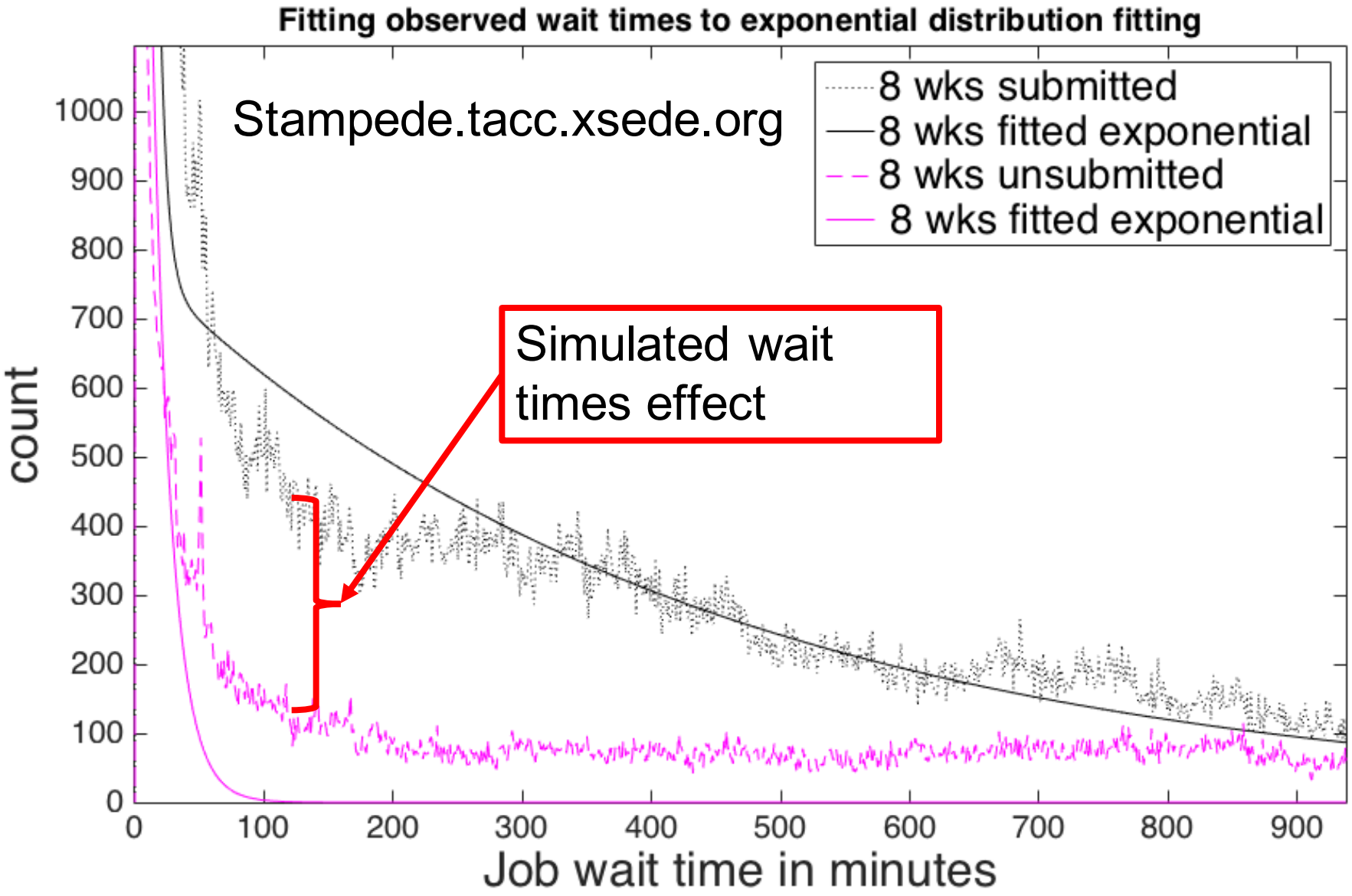
- processorRunning > 35896
- workAhead < 545313600
- workAheadOtherQueues < 772156800

is expected to start in 2423 seconds

Empirical Wait Time Distributions for Various Windows

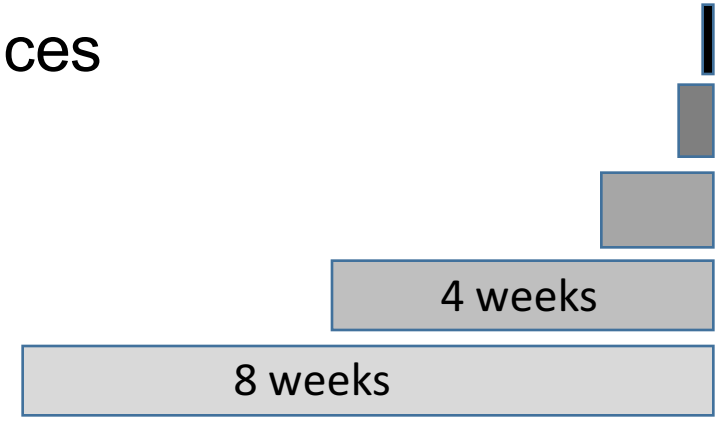


Fitting Wait Times Into Sum Of Exponentials



Bagging

- Make a prediction by averaging predictions of multiple decision tree
- Karnak builds five decision trees per system
  - Different lookback window size for experiences
    - 4 hours
    - 1 day
    - 1 week
    - 4 weeks
    - 8 weeks
  - Their experiences' distributions are not independent

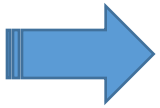


Why Are Predictions So Big ?

- Huge wait times
- Decision trees built upon larger dataset tend to output bigger predictions
- Creating multiple experiences per observation inflates the predictions of other leaves

Larger # of Experiences Lead to Larger Predictions

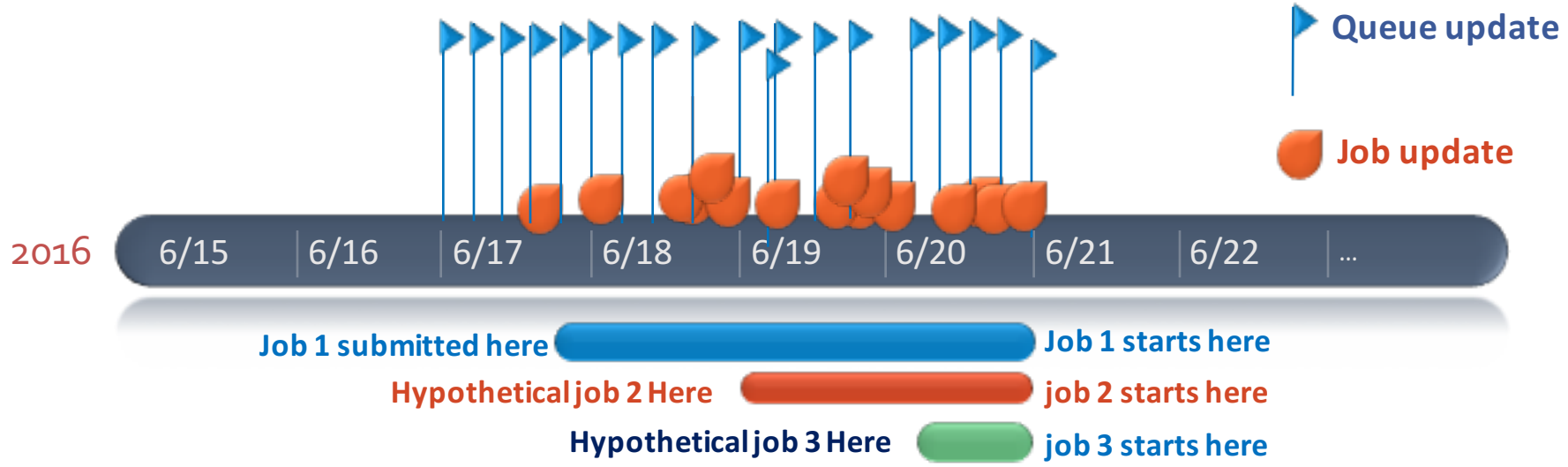
Feature	Value
processors	777
requestedWallTime	240
countAhead	257
workAhead	1193362560
countAheadQueue	256
workAheadQueue	1193333760
workRunning	1367840807
countAheadOtherQueue	1



Window size	Prediction (seconds)	Confidence interval (seconds)
4 hours	31097	(27154, 35040)
1 day	70848	(67312, 74384)
1 week	181974	(170804, 193143)
4 weeks	163391	(157357, 169425)
8 weeks	470820	(391119, 550521)

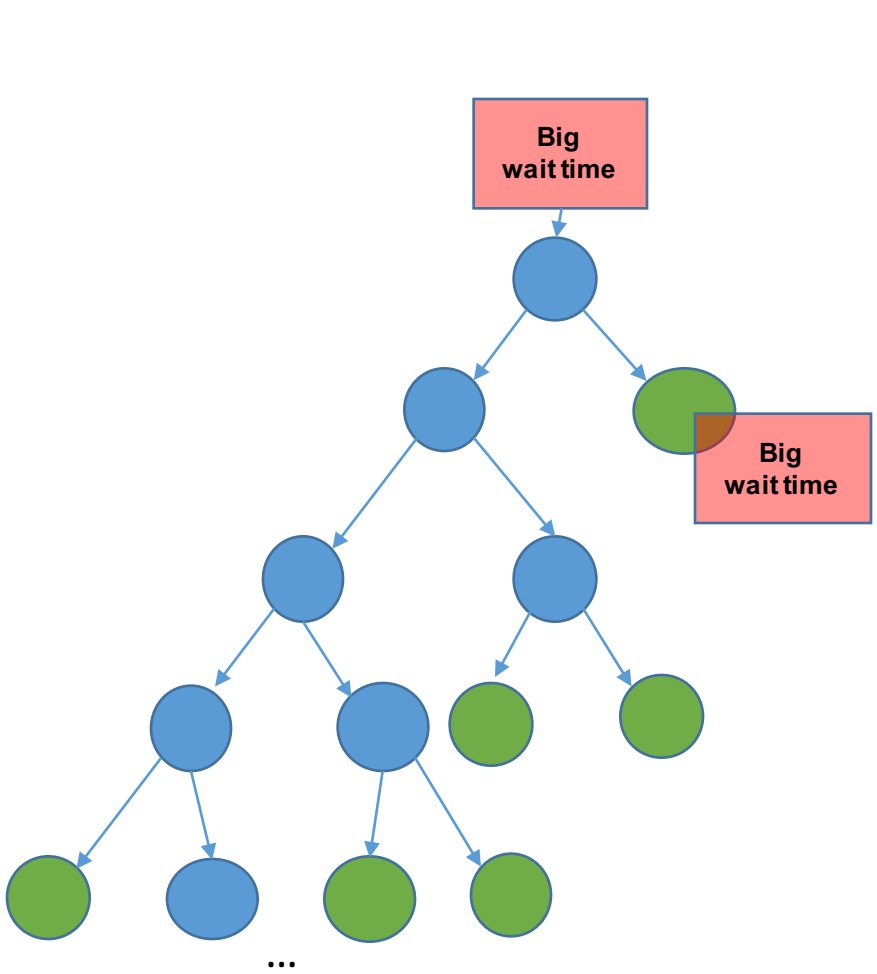
Query/prediction results for Gordon.sdsc.xsede.org on 06/23/2016

Simulated Wait Times

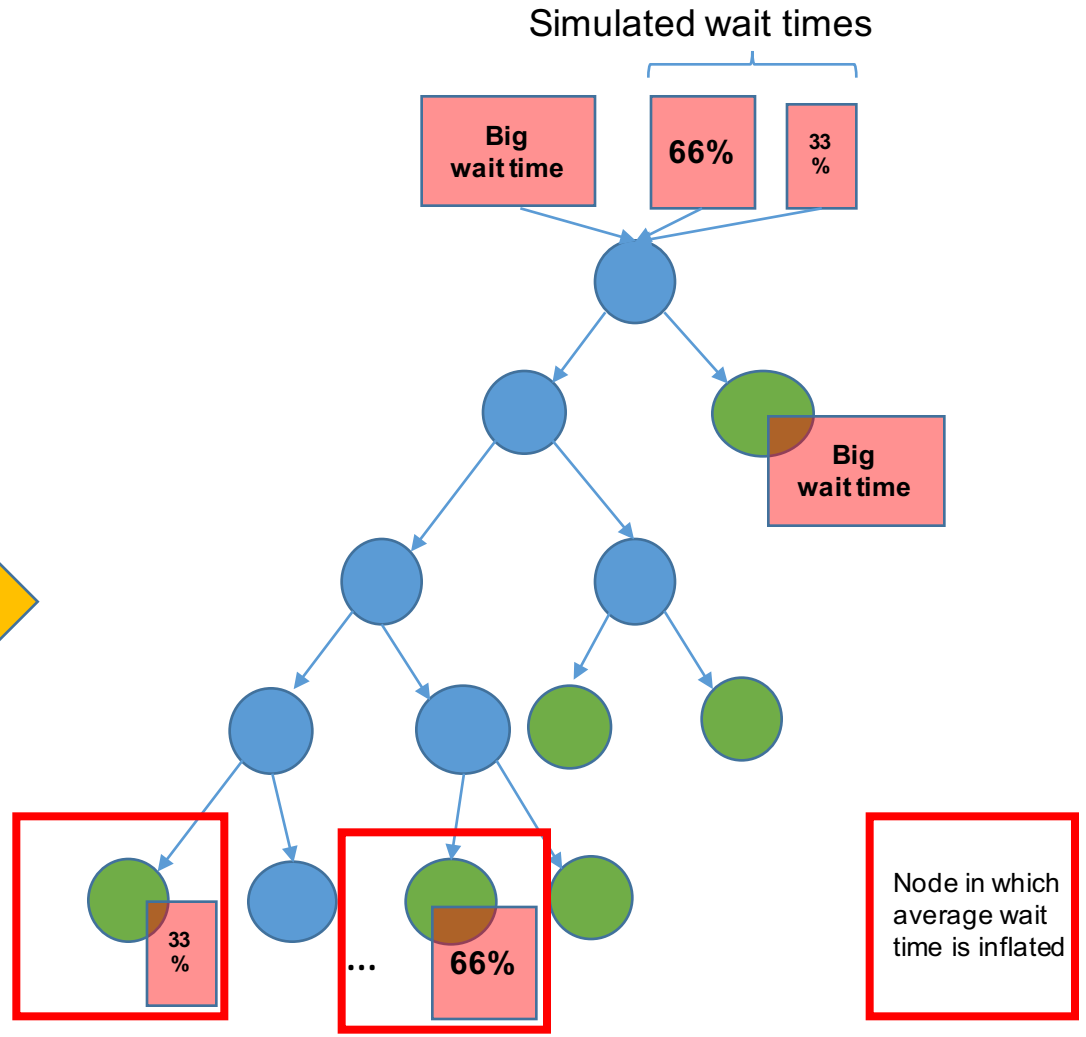


- We know the submit time and start time of Job 1
- Can create arbitrary number of experiences satisfying
  - Submitted between (submit, start) of Job 1
  - Then it will start at the same time as Job 1
- Karnak creates two additional experiences of wait times x/3, 2x/3 out of a wait time x

Simulated Wait Times Increase Predictions



Decision tree built without simulated wait times



Decision tree built with simulated wait times

Discussion

- Selecting best features
  - workAhead may not be accurate
    - Not all job run for maxWallTime
    - Hard-coded maxWallTime
  - Proxy variables for scheduling policy
    - Policy updates
- Limitation of decision tree
  - Each feature has equal weight vs. variable weights on linear regression coefficients
  - Weak to outliers

Current Status and Future Work

- What remains to be done
  - Implementation
    - Give up ensemble method
    - Do not create simulated wait times under exponential wait time distribution
  - Back-testing and bench marking: comparing error rates between old and new decision trees
- Potential future work
  - Improving the confidence interval
  - Multiple linear regression
  - Neural networks

Questions?

Jungha Woo
wooj@purdue.edu

Backup Slides

Service Interfaces

- REST-style interactions
- HTTP is the base protocol
- A few options for encoding data
  - HTML
    - Simple web interface described previously
  - Plain text
    - Supports thin command line clients
  - XML
    - Supports integration with tools

Slide from Warren Smith

Command Line Programs

- Roughly follows the web pages
  - `ksystems` – system summary
  - `ksystem` – jobs in queues on a system
  - `kqueue` – current and historical job information
  - `kjobs` – info about waiting jobs
  - `kwait`, `kstart` – predict waiting job
  - `kwouldwait`, `kwouldstart` – predict potential job
- Implemented in Python
- Provides output as HTML, XML, or text
- Downloadable via the XSEDE Karnak web page
  - http://karnak.xsede.org/karnak/index.html

Slide modified from Warren Smith, with permission

Potential Job Performance

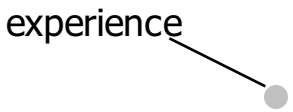
- Post-analysis of performance
  - Event driven simulation
  - Job data recorded from XSEDE systems
- Workload has 2 event types
  - Request a prediction for each job using system state just before it is submitted
  - Insert information about a job wait time as the job starts
- Process:
  - Perform insertions only for X days
  - Perform insertions and predictions for the next Y days
  - Output performance
- Optimized prediction parameters

Slide modified from Warren Smith, with permission

Example GLUE2 Job message


```
{
  "ID": "urn:glue2:ComputingActivity:7188907.stampede.tacc.xsede.org",
  "ResourceID": "stampede.tacc.xsede.org",
  "Name": "code.p",
  "CreationTime": "2016-06-15T18:44:52Z",
  "EntityJSON": {
    "Associations": {
      "UserDomainID": null,
      "ResourceID": null,
      "ActivityID": [],
      "ShareID": "urn:glue2:ComputingShare:normal.stampede.tacc.xsede.org",
      "EndpointID": null
    },
    "LocalOwner": "saikisha",
    "Name": "code.p",
    "Extension": {
      "Priority": 1358,
      "LocalAccount": "A-aoag"
    },
    "WaitingPosition": 856,
    "CreationTime": "2016-06-15T18:44:52Z",
    "ComputingManagerSubmissionTime": "2016-06-15T15:04:49Z",
    "RequestedTotalWallTime": 1382400,
    "Queue": "normal",
    "LocalIDFromManager": "7188907",
    "State": [
      "ipf:pending",
      "slurm:PENDING"
    ],
    "SubmissionTime": "2016-06-15T15:04:49Z",
    "Owner": "unknown",
    "RequestedSlots": 16,
    "ID": "urn:glue2:ComputingActivity:7188907.stampede.tacc.xsede.org"
  }
},
```

Experience

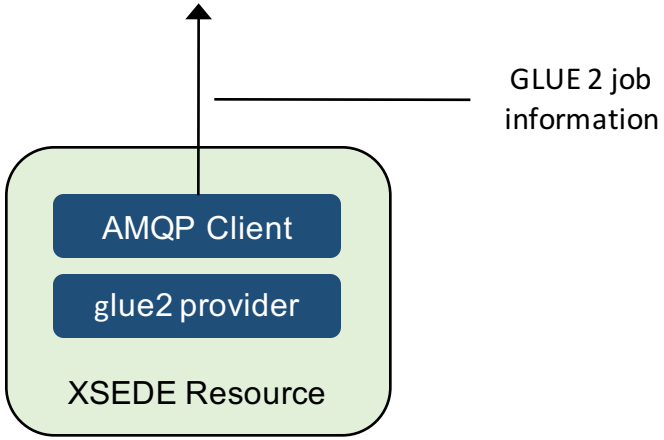


- Describes something that has happened in the past
- Consists of input features and output features
  - Input features describe conditions
  - Output features describe what happened
  - Each feature has a name and value
  - Values can be linear (numbers) or nominal (strings)
- For example, a job that ran on a cluster yesterday
  - Input features: username, queue, #CPUs
  - Output feature: run time

Slide from Warren Smith

Implementation

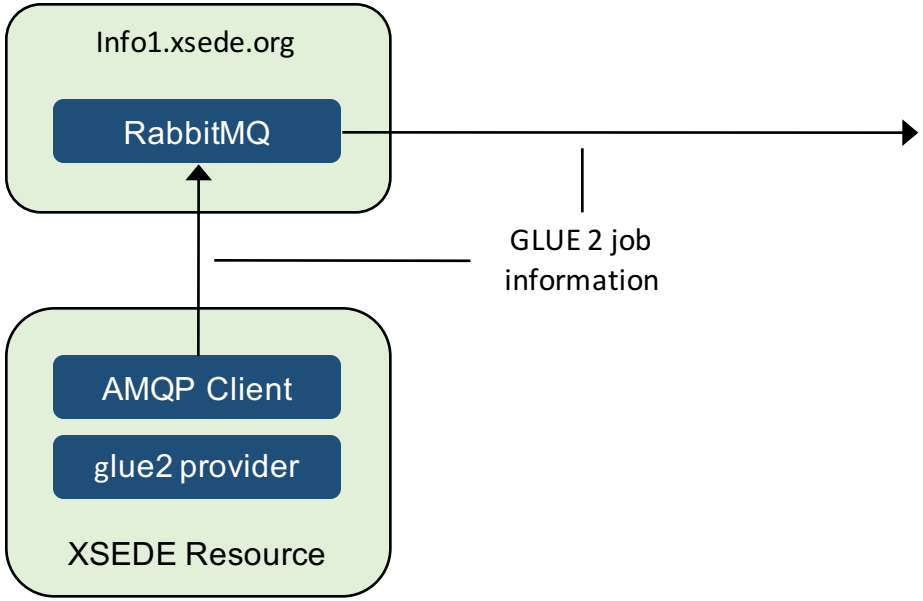
- glue2 provider generates job information
  - Periodic snapshots of queued and running jobs
  - XML in a XSEDE realization of the GLUE 2 schema
- RabbitMQ picks up this information
- No GLUE 2 job information, no predictions



Slide modified from Warren Smith, with permission

Implementation

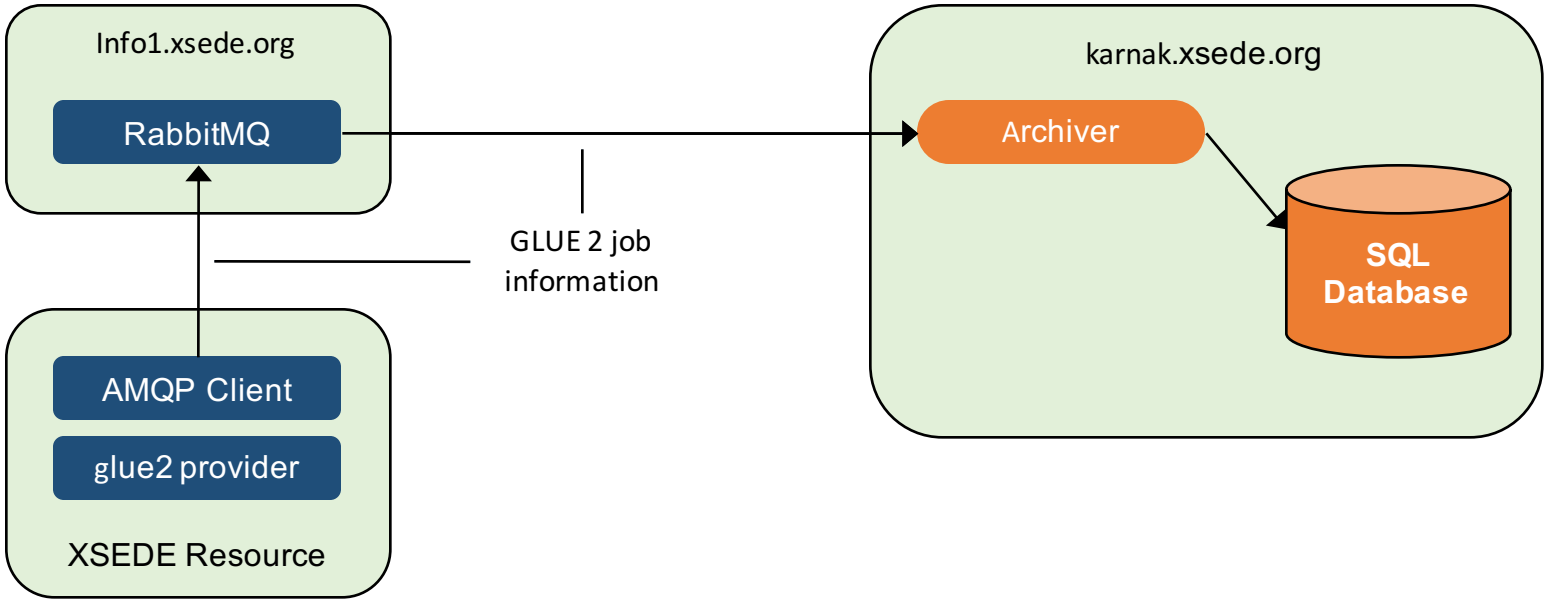
- GLUE 2 job information published to centralized RabbitMQ
  - Only authenticated access
  - Short list allowed to access



Slide modified from Warren Smith, with permission

Implementation

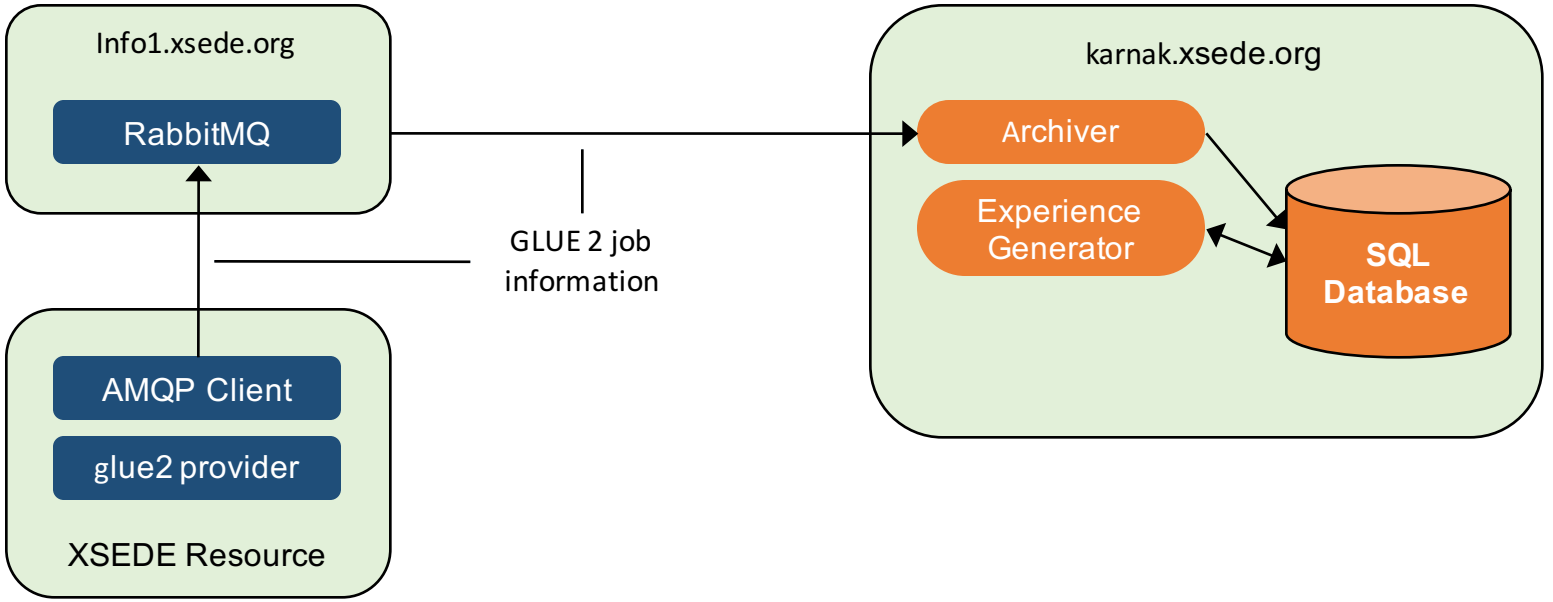
- Archiver process
  - Subscribes RabbitMQ events for job, and queue information
  - Stores job ordering
  - Stores job information (cores, req. wall time, user, …)
  - Generates events for each job (submit, start, complete)



Slide modified from Warren Smith, with permission

Implementation

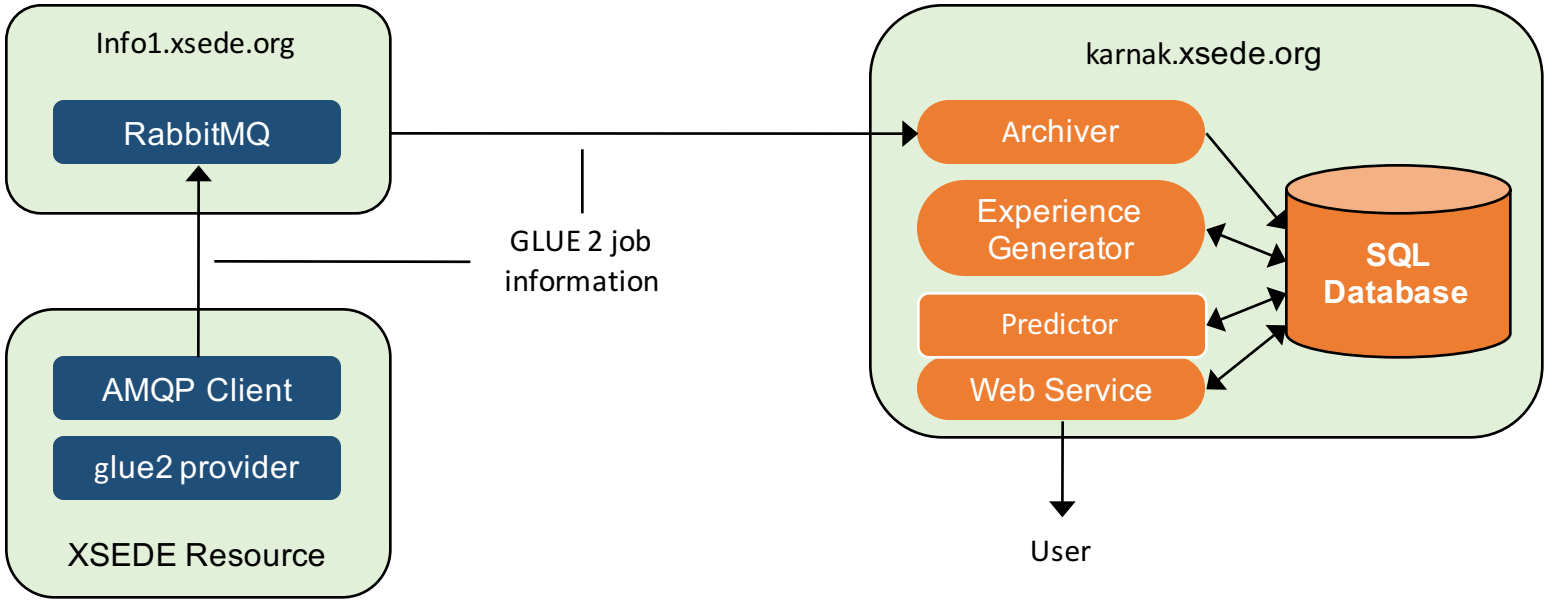
- Experience Generator
  - Performs scheduling simulations and stores results
  - Constructs experiences
    - job description, queue position, simulated start time



Slide modified from Warren Smith, with permission

Implementation

- Web service interacts with users
  - Job information from database
  - Embedded predictor to form predictions



Slide modified from Warren Smith, with permission

Creating a Query to Get Prediction

Job ID	Submit time	State	# CPU	Requested time	Start time
14	6/11 3PM	Running	2	6 hrs	6/19 2 am
15	6/16 1PM	Running	3	2 hrs	6/19 3 am
16	6/18 4PM	Pending	5	1 hr	
17	6/19 3AM	Pending	1	2 hrs	

Submitted job

16	6/18 4PM	Pending	5	1 hr	

Job ID	Submit time	State	# CPU	Requested time	Start time
14	6/11 3PM	Running	2	6 hrs	6/19 2 am
15	6/16 1PM	Running	3	2 hrs	6/19 3 am
16	6/18 4PM	Pending	5	1 hr	
17	6/19 3AM	Pending	1	2 hrs	

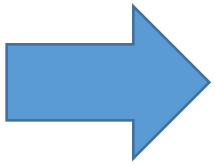
Unsubmitted job

# CPU	Requested walltime
4	1200

Feature
queue
processors
requestedWallTime
countAhead
workAhead
countRunning
processorsRunning
workRunning
countAheadUser
workAheadUser
countRunningUser
processorsRunningUser
workRunningUser

Creating Experiences to Build Trees

Job ID	Submit time	Status	# CPU	Request time	Start time
12	6/8 3PM	Done	24	3 hrs	6/12 2 am
13	6/10 1PM	Done	3	4 hrs	6/10 3 pm
14	6/11 3PM	Running	2	6 hrs	6/19 2 am
15	6/16 1PM	Running	3	2 hrs	6/19 3 am



Feature
queue
processors
requestedWallTime
countAhead
workAhead
countRunning
processorsRunning
workRunning
countAheadUser
workAheadUser
countRunningUser
processorsRunningUser
workRunningUser

Only started jobs have wait times

How To Build a Decision Tree

- Top-down, greedy way to build a decision tree

1. Start from the root
2. Compute the standard deviation for all data
3. Split the dataset with each feature. Repeat for 26 features
4. Choose the feature that reduces standard deviation most
5. Divide the dataset based on the selected feature
6. Keep splitting if there is standard deviation reduction
7. Run recursively on the non-leaf nodes, until all data is processed

An excellent decision tree example is available at http://www.saedsayad.com/decision_tree_reg.htm

Example of Karnak decision tree