Forecasting Storms in Parallel File Systems

Student: Ryan McKenna[1,2]
Advisors: Todd Gamblin[2], Adam Moody[2], Bronis de Supinski[2], Michela Taufer[1]
[1]University of Delaware, [2]Lawrence Livermore National Laboratory

Motivation

- Large-scale scientific applications rely on the parallel file system (PFS) to store checkpoints and outputs
- The PFS is a shared resource, and many applications need to read and write to it concurrently, but the performance degrades as more applications read/write more data
- These types of PFS 'storms' could be avoided if the scheduler was aware of the IO workload of jobs before they start

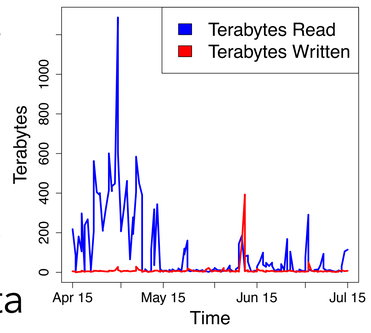
Goals

- Mitigate PFS resource contention by identifying IO-intensive jobs before they start
- Predict the per-job runtime statistics and IO counters using historical data
- Forecast PFS 'weather' using IO and runtime predictions for upcoming jobs

Background

- A PFS is a file system that can efficiently handle big data and concurrent requests by distributing data across many storage nodes
- The majority of file system use comes from simulation *checkpoints*: jobs save their state every few hours so they can be restarted from a stable state in the event of a failure
- A scheduler is a program that determines how to allocate resources (nodes in a cluster) to jobs
  - Schedulers make decisions based on the size of the job (number of nodes + requested time)
- Sometimes the scheduler allocates resources for multiple IO-intensive jobs at once
  - In these situations, the PFS becomes overloaded and potentially unresponsive

Parallel File System Traffic



This figure shows how many terabytes the PFS handles over time. Spikes in the graph correlate with poor file system performance.

Job Data

- Our data comes from two sources:
  - Job logs contain user-submitted information (job-scripts) and summary statistics about the job
  - IO reports summarize calls to the file system over the course of the job
- Job scripts contain a wealth of data for understanding the behavior of jobs
- Jobs with similar input features often have similar output features

Input Features	Output Features	
workdir	*act_time*	exitcode
jobid	send_bytes	alloc-inode
submit_time	create	rename
req_time	duration	endtime
machine	mkdir	mknod
user	mmap	rmdir
group	mdt_cancel	mdt_grant
account	osc_read_count	osc_read
jobname	write_count	*write_bytes*
req_tasks	osc_write_count	osc_write
req_nodes	osc_cancel	ost_grant
submitdir	recv_count	recv_bytes
outfile	read_count	*read_bytes*

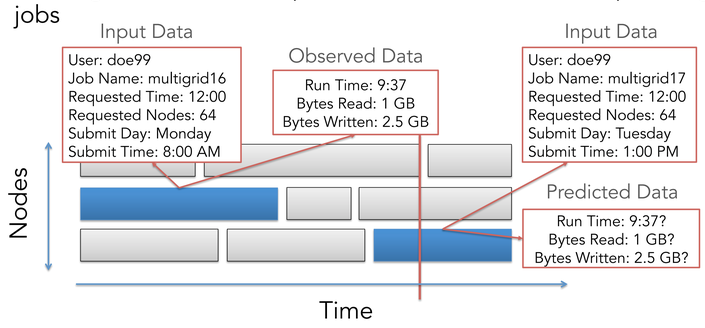
The input features are derived from the job script and the output features are reported at job completion. We focus on predicting the boxed output features.

Methodology

- Scientists are predictable – they often run many versions of the same code
- It's likely that these different versions share runtime statistics (amount of data read/written, runtime, etc.)

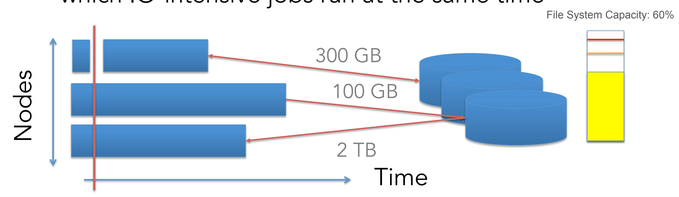
This diagram shows the design process for building scientific applications

- Parse job scripts and build a feature set
  - For text based features such as job-name, split word along alphanumeric boundaries to extract keywords
- Build a regression model using available job history
  - Regression models are specialized to handle continuous valued output, such as runtime and bytes read/written
  - We investigate three regression models: decision trees, random forests, and k-nearest neighbors
- Use regression model to predict runtime statistics for upcoming jobs



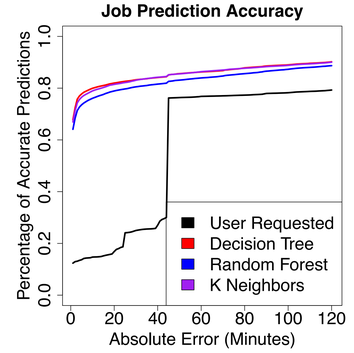
Historical IO patterns (behind red bar) are used to predict future IO patterns

- With both runtime predictions and IO workload predictions, we can simulate the queue to determine which IO-intensive jobs run at the same time



Results

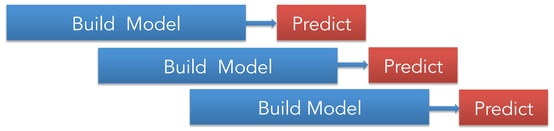
Job Prediction Accuracy



- We accurately predict the runtime (within 10 mins) of 80% of jobs – a 65% improvement over the user requests
- Requested wall time, job name, and submit time are the strongest predictors of actual runtime

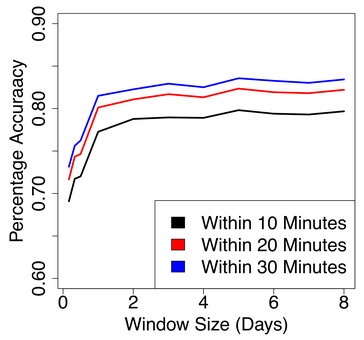
Error	User Requested	Decision Tree	Random Forest	K Neighbors
< 10 Mins	14.7%	80.0%	76.1%	78.9%
< 20 Mins	16.9%	82.0%	78.9%	81.5%
< 30 Mins	25.0%	83.2%	80.5%	83.1%
< 40 Mins	27.1%	84.1%	81.6%	84.2%
< 50 Mins	76.4%	85.6%	83.1%	85.5%
< 60 Mins	76.8%	86.2%	83.9%	86.4%

By only training on the most recent history, we save time and space by building a smaller model and storing less data

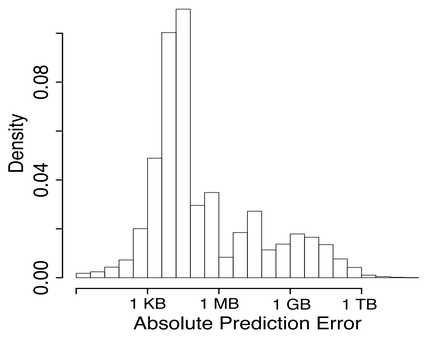


- With a window size of five days, 79.8% of predictions are accurate to within 10 minutes – only 0.2% below the accuracy attained from using all available history

Window Size vs. Prediction Accuracy



Bytes Written Accuracy



- We accurately predict the file system traffic by application for the majority of jobs

Error	Bytes Written	Bytes Read
< 1 MB	74.9%	63.7%
< 16 MB	79.8%	68.6%
< 256 MB	86.6%	82.7%
< 4 GB	92.0%	90.0%
< 64 GB	97.4%	97.0%

Conclusion

- Training machine learning models on job scripts is a viable solution to predict application IO patterns
- These results can help schedulers avoid PFS resource contention
- For future work, we would like to:
  - Build a model that predicts IO patterns over the course of a job
  - Extract more features from the job script to allow the regression models to detect more correlations

This work performed under the auspices of the U.S. Department of Energy and an appointment to the Office of Science, Science Undergraduate Laboratory Internship (SULI) Program at the Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.
IM #: LLNL-POST-675538 National Science Foundation CCF #1318445