Using Machine Learning to Improve Job Scheduling in Datacenters

Eli Sherman, Yash Kumar Lal, Brian Choi, Avais Pagarkar

Department of Computer Science

Johns Hopkins University

Motivation

Background:

- Scheduling algorithms within large-scale systems are necessary to manage resources and jobs.
  - Shortest Job First
  - Dominant Resource Fairness

Problem:

- SJF-type algorithms require prior prediction of runtimes

Proposed Solution: develop a machine learning-based SJF scheduling algorithm (ML-SJF), which predicts job runtimes based on the characteristics of jobs submitted to a datacenter framework

Related Work

- Hidden markov models to predict job completion times using data extracted from supercomputing cluster logs. Chen et al. (2013)
- Exploration of various ML techniques to model homogenous job scheduling. Helmy et al. (2015)
- Fuzzy rule-based system built over job history to predict next CPU burst time Pourali and Rahmani (2009)

Testbed System

Simulate Datacenter based on Mesos Scheduling Framework

- Abstraction with single master node communicating with agent node via Python sockets
- System receives and processes jobs
- Use results of job runs to train ML predictor for future job runs

Agent Workflow

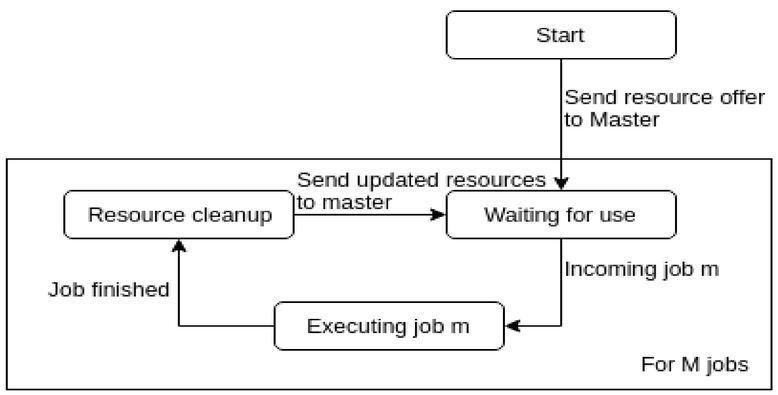

Figure: Agent

- Agent maintains list of resources
- Executes jobs from master using os system call commands
- Sends updates to master when job has finished

Master Workflow

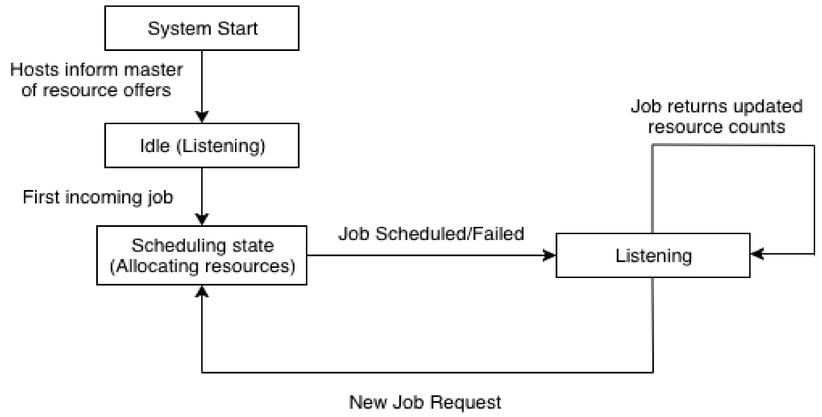

Figure: Master

- Master maintains available resources (CPU, RAM, etc..) and list of jobs to complete
- Master schedules job requests of varying types (Flask, M/R, etc..)
- Sends jobs to run on agents after scheduling

Types of Jobs

Web-Based

- Flask Jobs: Insertion Sort, Bubble Sort, Bogo Sort, etc...

Machine Learning Jobs

- scikit-learn Jobs: Pre-installed Datasets

Distributed Computation

- Map Reduce Jobs: Word Count on Book Excerpts

Machine Learning

Overall Idea

- Generate 1000 jobs of each type with random features
- Run jobs in Mesos system to collect runtimes
- Train ML model
- Use ML model to predict future job runtimes

Input Features

- Web-Based: size of work file, type of operation (sorting algorithm)
- ML Jobs: number of rows and columns in training set, number of target classes
- MapReduce Jobs: size of work files, number mappers, number reducers

Model Learning

- Support Vector Regression (SVR) is highly flexible ML algorithm
- Capable of learning relationships in high dimensional data
- Train each regressor with 5-fold CV on input data, keep best regularization parameter

Experiments

- Generate additional 600 random jobs as test set
- Comparison of all scheduling algorithms
  - Schedule jobs in test set using different algorithms and execute
  - Log throughput and average waiting time
- Analysis of runtime trends for each framework
  - Replace randomly generated predicted runtimes for test jobs with average runtime of that framework's job
  - Schedule each framework's jobs using SJF and execute
  - Log throughput and average waiting time

Results (Experiment 1)

Algorithm	Throughput	Avg Wait Time
DRF	0.3522	835.95 secs
SJF	0.4735	643.31 secs
ML-SJF	0.1561	1713.26 secs
Hard-coded	0.1689	1734.43 secs

Table: Performance of algorithms overall

- ML-SJF has a lower throughput than the conventional SJF and DRF implementations, contrary to our hypothesis.
- In ML-SJF, the three types of jobs generally are in three different ranges of runtime
- MapReduce jobs are almost always executed in parallel with other MapReduce jobs in case of ML-SJF, reducing the throughput.

Results (Experiment 2)

Job type	Algorithm	Throughput	Avg Wait Time
MR	SJF	0.1371	310.875 secs
MR	ML-SJF	0.1375	273.97 secs
ML	SJF	0.1675	897.045 secs
ML	ML-SJF	0.3324	437.83 secs
Flask	SJF	0.1494	568.78 secs
Flask	ML-SJF	0.1393	554.29 secs

Table: Performance of algorithms with one framework

- For homogeneous jobs,ML-SJF outperforms SJF
- This is due to SJF not having the advantage of scheduling different types of jobs together.

References

X. Chen, C. Lu, and K. Pattabiraman. Predicting Job Completion Times Using System Logs in Supercomputing Clusters. In *43rd IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)*, 2013.

T. Helmy, S. Al-Azani, and O. Bin-Obaidellah. A machine learning-based approach to estimate the cpu-burst time for processes in the computational grids. In *3rd International Conference on Artificial Intelligence, Modelling and Simulation (AIMS)*, 2015.

A. Pourali and A. M. Rahmani. A fuzzy-based scheduling algorithm for prediction of next cpu-burst time to implement shortest process next. In *International Association of Computer Science and Information Technology - Spring Conference*, 2009.