Job Throughput Analysis of Data Center Scheduling Algorithms

GERARDO RAVAGO

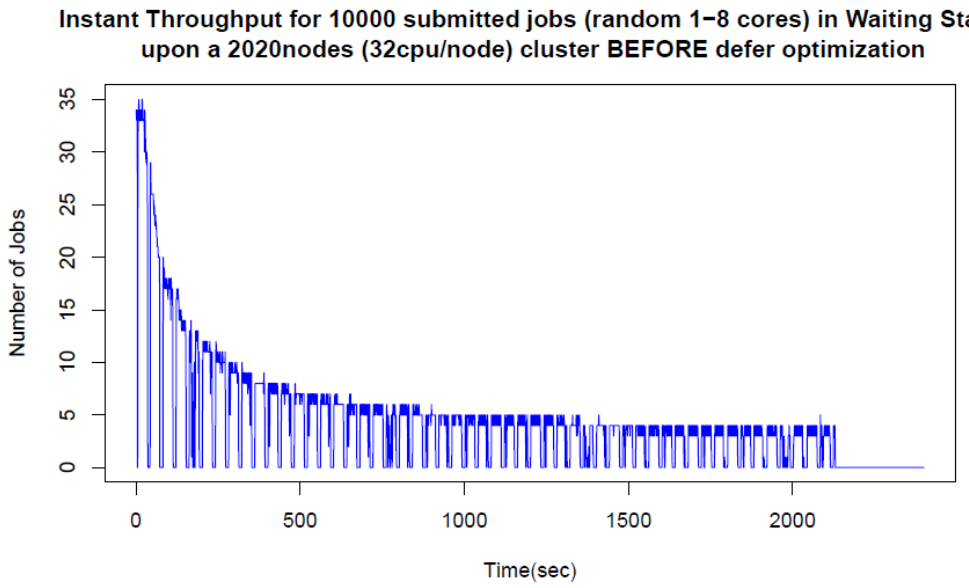
ANKIT SHARMA

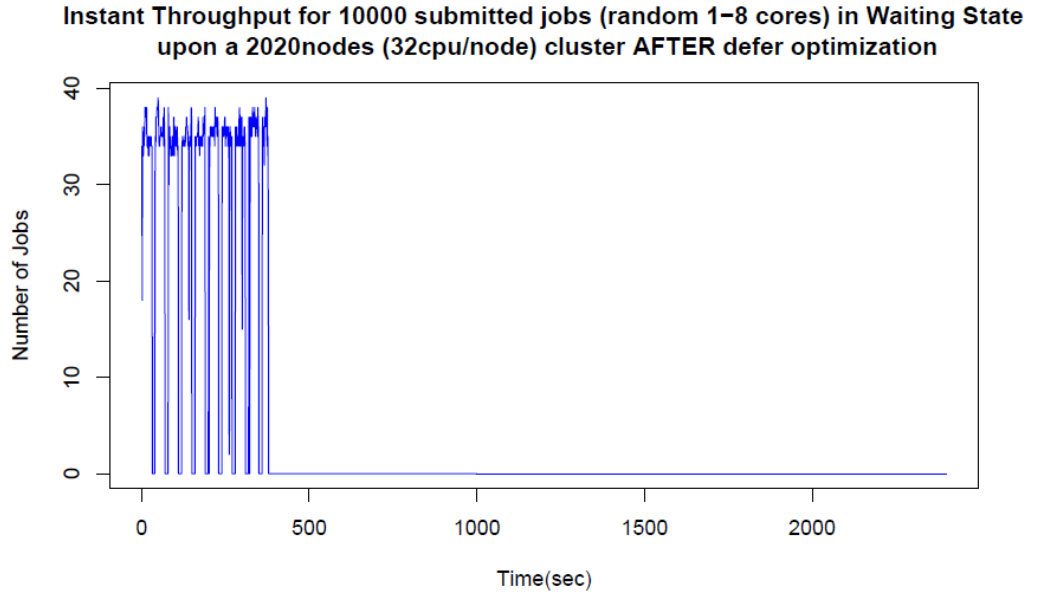
The Problem

The need to empirically evaluate the relative performance of job scheduling algorithms on resource and job management systems (RJMS).

Motivation

The configuration of the RJMS can be just as critical to performance as the hardware it manages. A way to empirically analyze the performance of the software agnostic is essential to tuning for the optimum performance. In particular, a bad scheduler can result in poor utilization and increase the overall run time.



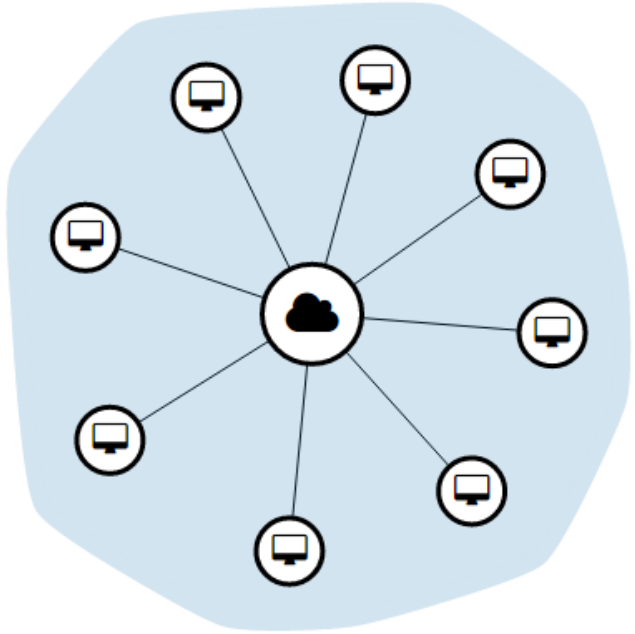


What We're Doing

1. Recreate common data center benchmarks from literature review
2. Create a small scale data center using a production RJMS
3. Run benchmarks on data center with different scheduling algorithms
4. Empirically analyze the tradeoffs of each algorithm

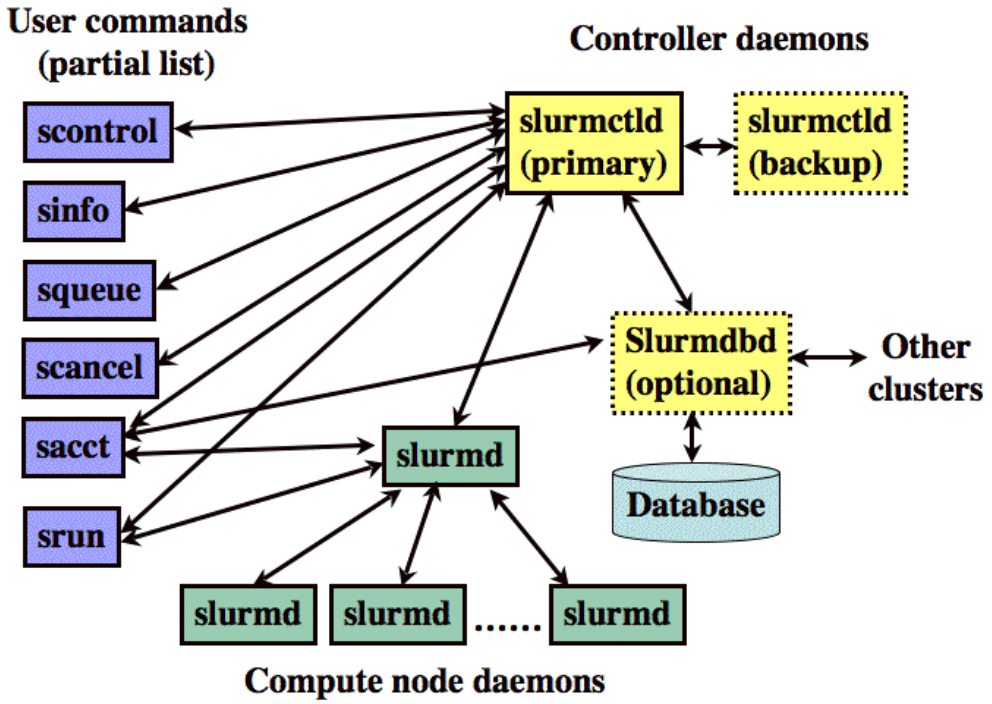
Small Scale Data-Center

- Use the cloud to create a small compute cluster
- Allows us to test RJMS software on a real cluster without the cost overhead of real hardware.
- Based on Massachusetts Open Cloud (MOC) and Azure
- Homogeneous 2-core systems



SLURM Workload Manager

- Production RJMS software for real world HPC clusters
  - Powers 60% of the top 500 supercomputers
  - Tianhe-2, most powerful HPC cluster uses SLURM
- Open source and actively developed
  - Source code changes were being pushed while we were setting up
- Simple and scalable architecture
  - Single SLURM controller manages job scheduling on the many compute nodes



Scheduling Algorithms

- Built-In FIFO
  - Jobs are assigned priority in first in first out fashion
  - The job with the highest priority is scheduled as soon as the resources required is available
  - Does not allow for preemption
- Backfill algorithm
  - Starts out with an initial FIFO priority
  - Increases priority of tasks that will not negatively affect the expected start time of higher priority tasks
  - Depends on reasonable estimates of execution time

Benchmarks

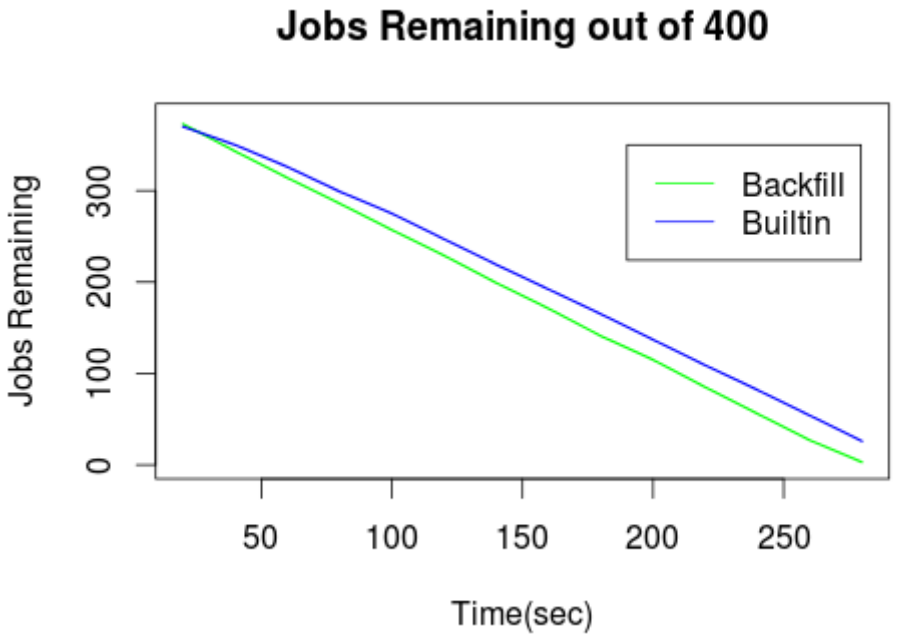
- PARSEC
  - Familiar workload from class projects
  - Provides a reasonable variety of multi-core workloads
  - Used blackscholes, streamcluster, canneal, and facesim to represent jobs with different execution times and resource requirements.
- ESP – Effective System Performance
  - Set of jobs that exist to occupy resources in the simplest way possible
  - Job simply reserves nodes and sleeps for a fixed time
  - Scaled down cores to fit our cluster
  - Scaled down execution times to complete in a reasonable time
- IMB – Intel MPI Benchmarks (Time permitting)
  - Real world jobs that require multiple nodes
  - Makes use of SLURM MPI plugins

What's Been Done

- Setup an 8 node cluster on MOC
  - Homogeneous datacenter consisting of medium 2 core processors running Ubuntu 14.04
- Installed SLURM
  - 1 SLURM controller node and 7 compute nodes
- Installed and built PARSEC benchmarks
- Obtained initial PARSEC performance data
  - Script scheduled 1000 PARSEC jobs and collected data for both scheduling algorithms

Early Results

- Scheduled 1000 PARSEC jobs at same time
- Figure illustrates the number of remaining jobs on cluster for each minute
- Note how Backfill algorithm performs better
  - This is due to how Backfill is able to take better advantage of idle time on compute nodes
  - For example if a job finishes much earlier than its expected end time backfill will find a job that fits that extra time



Demo Time

Work Remaining

- Finish PARSEC data collection with workload scalability analysis
- Implement a scaled down version of the ESP benchmark
- Microsoft Azure based compute nodes for inter-data center scheduling analysis
- Time Permitting
  - Get Open MPI working with SLURM
  - Implement Intel MPI Benchmark