Optimized Hardware Configuration for High Performance Computing Systems

Scott Hutchison, Daniel Andresen, William Hsu, Mitchell Neilsen, and Benjamin Parsons

scotthutch@ksu.edu, dan@ksu.edu, bhsu@ksu.edu, neilsen@ksu.edu, ben.s.parsons@erdc.dren.mil

September 13, 2023
About Scott Hutchison

- BS in Computer Science
  Texas A&M University, 2005
- MS in Cyber Operations
  Air Force Institute of Technology, 2015
- PhD Student at Kansas State University since 2021
- Research Interests:
  - HPC scheduling, optimization, infrastructure, and design.
  - Machine Learning for HPC applications, reinforcement learning, recommender systems.
Outline

Introduction and Problem Statement

Background and Related Work

Methodology

Results

Discussion

Conclusions

Bibliography
Introduction

- HPC administrators face tough decisions when faced with upgrading or replacing HPC systems
- Server capabilities (memory, cpus, gpus, etc.) greatly effect cost
- Requirements drive optimal server package (e.g. Do HPC users often submit GPU accelerated workloads?)
- What server package should be purchased for a given budget?
- No one right answer, but how can we get the “best bang for our buck”?
- Procurement decisions often made by administrator intuition or preference.
- Can data help inform this procurement decision?
Problem Statement

For a planned HPC expansion, can experimental simulation provide an optimal set of hardware under a given budget which will minimize job wait time?
Related Work

Hardware Optimization Related Research:

- Evans et. al [1] used benchmarks for various software/hardware combinations to optimize CPU/GPU ratios
  - Application runtime on HPC systems does not account for queue times in HPC environment
- Kutzner et. al [2] optimized for a particular application (GROMACs)
  - Application specific optimizations work best for systems with many homogeneous jobs
Related Work (cont.)

HPC Workloads and Scheduling Algorithms:
- Various public HPC workloads exist [3], but fail to include jobs requesting GPUs.
  - Used log data from submitted jobs from our local HPC system
- Many different scheduler options (Slurm, PBS, HTCondor, etc.), but scheduling algorithms are highly customizable and/or proprietary
- Various HPC simulators also exist (SimGrid, GridSim, Alea, etc.), but were deemed either overly complex or failed to allow for three job constraints (CPUs, Memory, GPUs).
  - Used Best Fit Bin Packing (BFBP) algorithm and a self implemented HPC Discrete Event Simulator
Related Work (cont.)

- New hardware may perform better/worse than current HPC hardware
- Job running duration was scaled according to the SPEC CPU2017 CPU benchmark [4]
- Sharkawi et al. [5] successfully used a similar SPEC benchmark to estimate the performance projections of HPC applications.
- Wang et al. [6] have pointed out that these benchmarks fail to account for all the variables affecting job resource utilization and should be avoided.
- Code written to easily allow different or no scaling
Contributions of this work

1. A Discrete Event Simulator for modeling HPC scheduling
   ➤ https://github.com/shutchison/hpc-discrete-event-simulator

2. A data set consisting of almost 12,700 HPC scheduling simulations, each with a different HPC server set

3. An optimized XGBoost regression model for predicting \textit{AvgWaitTime} when given a composition of servers

4. A recommender system with precision@50=92% which can inform hardware procurement decisions when expanding or replacing a HTC or HPC system
   ➤ https://github.com/shutchison/Optimal-Hardware-Procurement-for-a-HPC-Expansion
Methodology

1. Receive vendor quotes with potential server options.
2. Generate potential server combinations to purchase under the specified budget which meet our procurement requirements.
3. Identify a typical set of jobs representing the workloads typically submitted to our HPC system.
4. Conduct simulations using a subset of the server packages to schedule the representative job set and compute metrics to determine their performances.
5. Use machine learning to train and refine a model that can predict the performance of un-simulated server combinations.
6. Develop a recommender system using the machine learning model.
7. Subjectively evaluate the recommended server packages and make a more informed procurement decision.
Methodology

1. All server combos (~127,000)
2. Every 10th server combo (~12,700)
3. Label AvgWaitTime with DES
4. Summarize into package: TotalCPUs, TotalMemory, TotalGPUs,
5. XGBoost Model
6. Quantitative Evaluation
7. Subjective Evaluation
Computing Server Combinations

- Received two vendor quotes, separate 21 possible servers into three different server categories:
  - Compute nodes, big memory nodes, GPU nodes

<table>
<thead>
<tr>
<th></th>
<th>Number of Server Types in Category</th>
<th>Range of Memory per Node</th>
<th>Range of CPUs per Node</th>
<th>Range of GPUs per Node</th>
<th>Cost Range per Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute Nodes</td>
<td>4</td>
<td>256-512 Gb</td>
<td>24-64 cores</td>
<td>0 GPUs</td>
<td>$6,000 - $10,000</td>
</tr>
<tr>
<td>Big Memory Nodes</td>
<td>2</td>
<td>1024 Gb</td>
<td>24-64 cores</td>
<td>0 GPUs</td>
<td>$11,000 - $13,000</td>
</tr>
<tr>
<td>GPU Nodes</td>
<td>15</td>
<td>256-1024 Gb</td>
<td>24-64 cores</td>
<td>1-8 GPUs</td>
<td>$14,000 - $100,000</td>
</tr>
</tbody>
</table>

- Fix budget at $1 million
- Generate all server combinations:
  1. For each combination of one compute node, one big memory node, and one GPU node:
  2. Purchase as many nodes as possible under the $1 million budget such that we cannot purchase another server AND
  3. Each combination contains at least 1 GPU node.
- Produced approx. 127,000 combinations of affordable server packages
Simplifying Assumptions

1. No budget for additional server infrastructure
   - Considered a “fixed cost” across all server packages
   - Could reduce budget and apply same procedure

2. Existing HPC infrastructure not included in server package for simulations
   - Considered a “fixed benefit” which each server composition would benefit equally from

Server package example:

<table>
<thead>
<tr>
<th>ComputeNode1, $6,960</th>
<th>BigMemNode1, $11,112</th>
<th>GPUNode1, $14,730</th>
<th>…</th>
<th>Package Cost</th>
<th>Money Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>141</td>
<td>0</td>
<td>1</td>
<td>…</td>
<td>$996,090</td>
<td>$3,910</td>
</tr>
<tr>
<td>139</td>
<td>1</td>
<td>1</td>
<td>…</td>
<td>$993,282</td>
<td>$6,718</td>
</tr>
<tr>
<td>138</td>
<td>2</td>
<td>1</td>
<td>…</td>
<td>$997,434</td>
<td>$2,566</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>67</td>
<td>…</td>
<td>$998,022</td>
<td>$1,978</td>
</tr>
</tbody>
</table>
Job Selection

- One typical day of jobs (∼16,000 jobs) selected from local HPC log data
- Jobs were “bursty” and requested a variety of CPU/memory/GPU resources
- Job duration scaled:
  \[
  \text{New duration} = \frac{\text{logged duration} \times \text{logged processor performance}}{\text{new processor performance}}
  \]
HPC Discrete Event Simulator

- Jobs and Machines are loaded from a CSV file
- Machines have 3 limiting resources: CPUs, Memory, and GPUs
- Jobs are specified with: submit time, actual duration, and requested duration, memory, CPUs, and GPUs. Jobs track start time and end time.
- Used BFBP scheduling (could use FIFO, SJF, etc.)

Algorithm Best Fit Bin Packing Scheduling

1: while The simulation is incomplete do
2:   if Some job in the queue can be executed on some machine then
3:     Find the (job, machine) pairing which results in the fewest remaining resources for some machine. Begin executing that job on that machine.
4:   else
5:     Advance simulation time until a new job is submitted or a running job ends, whichever is sooner.
6:     Queue submitted jobs and stop ending jobs.
7:   end if
8: end while
Machine Learning

- Simulating every server composition would take too long (\(\sim 30\) minutes per simulation)
  - Accomplished using HPC resources
- Tried various regression techniques and chose the best performing
Model Development and Recommender System

- Aggregate all simulated data (approx. 12,700 simulations)
- Shuffle and split into 90% training and 10% test
- Apply various machine learning regression techniques using five fold cross validation on training data, evaluating with the test data
- XGBoost [7] produced the lowest RMSE
- Randomized grid search for hyper parameter optimization
- Sort by AvgWaitTime predictions made by trained model to power recommender system. Take top k predictions and evaluate using precision@k, recall@k, and f1@k on test data
Recommender System

- "Hits" defined as server sets with the lowest 5% AvgWaitTime
- Precision@k - User requests $k$ items, what percentage of them are hits?
- Recall@k - User requests $k$ items, what percentage of the total hits are retrieved?
  - Unfairly penalizes when: $k \ll$ total number of hits
- F1@k - Harmonic mean of Precision@k and Recall@k
Evaluation

- Root Mean Squared Error (RMSE) used for model comparison
- AvgQueueTime measured from simulator log data

For N jobs:

$$\text{AvgWaitTime} = \frac{\sum_{i=0}^{N} (\text{Start Time}_i - \text{Submit Time}_i)}{N}$$

$$\text{RMSE} = \sqrt{\frac{\sum_{i=0}^{N} (\text{actual wait time}_i - \text{predicted wait time}_i)^2}{N}}$$
Evaluation (cont.)

- Recommender system evaluated with precision@k, recall@k, and f1@k
  - Top 5% of simulated data set with the lowest AvgWaitTime
  - 632 server configurations “hits” from the test data set

Precision@k = \( \frac{\text{(# of recommended items @k that are relevant)}}{\text{(total # of relevant items)}} \)

Recall@k = \( \frac{\text{(# of recommended items @k that are relevant)}}{\text{(total # of relevant items)}} \)

F1@k = \( \frac{2 \times \text{precision@k} \times \text{recall@k}}{\text{precision@k} + \text{recall@k}} \)
Feature Correlation

- Pearson Correlation coefficients closer to 1 or -1 indicate stronger correlation between variables.
- TotalCPUs was strongly correlated to AvgWaitTime with a correlation coefficient.
  - Implies that, for the chosen jobs, the number of CPUs in the package was the limiting factor.
- Positive correlation of TotalGPUs implies the more GPUs we purchase, the fewer CPU nodes we can afford.

<table>
<thead>
<tr>
<th></th>
<th>TotalMem</th>
<th>TotalCPUs</th>
<th>TotalGPUs</th>
<th>AvgWaitTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>TotalMem</td>
<td>1.00</td>
<td>0.14</td>
<td>-0.54</td>
<td>-0.23</td>
</tr>
<tr>
<td>TotalCPUs</td>
<td>0.14</td>
<td>1.00</td>
<td>-0.42</td>
<td>-0.70</td>
</tr>
<tr>
<td>TotalGPUs</td>
<td>-0.54</td>
<td>-0.42</td>
<td>1.00</td>
<td>0.44</td>
</tr>
<tr>
<td>AvgWaitTime</td>
<td>-0.23</td>
<td>-0.70</td>
<td>0.44</td>
<td>1.00</td>
</tr>
</tbody>
</table>
XGBoost Model Performance

- XGBoost RMSE = 150.13 seconds
- TotalCPUs, TotalMemory, TotalGPUs does a good job at predicting AvgWaitTime
Recommender System Performance (Quantitative)

- Threshold was the top 5% server compositions with the lowest AvgWaitTime (632 “hits” in the test data set)
- Higher values of k have good recall@k
- Precision@50 = 92% indicates that if the recommender system returns 50 results 46 of them will be in the top 5% of performing combinations

<table>
<thead>
<tr>
<th>k value</th>
<th>Precision@k</th>
<th>Recall@k</th>
<th>F1@k</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.00</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td>50</td>
<td>0.92</td>
<td>0.07</td>
<td>0.13</td>
</tr>
<tr>
<td>100</td>
<td>0.81</td>
<td>0.13</td>
<td>0.22</td>
</tr>
<tr>
<td>500</td>
<td>0.74</td>
<td>0.59</td>
<td>0.66</td>
</tr>
<tr>
<td>632</td>
<td>0.72</td>
<td>0.72</td>
<td>0.72</td>
</tr>
<tr>
<td>1000</td>
<td>0.58</td>
<td>0.91</td>
<td>0.71</td>
</tr>
</tbody>
</table>
Recommender System Performance (Subjective)

- XGBoost regression model used to predict performance of un-simulated server combinations

Sum of servers in top 50 recommendations:

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Node Description</th>
<th>Sum of Servers Across Top 50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute Nodes</td>
<td>Cheapest w/ 256Gb</td>
<td>232</td>
</tr>
<tr>
<td></td>
<td>Cheapest w/ 512Gb</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>Expensive w/ 256Gb</td>
<td>3,467</td>
</tr>
<tr>
<td></td>
<td>Expensive w/ 512Gb</td>
<td>0</td>
</tr>
<tr>
<td>Big Memory Nodes</td>
<td>Cheapest w/ 1024Gb</td>
<td>232</td>
</tr>
<tr>
<td></td>
<td>Expensive w/ 1024Gb</td>
<td>111</td>
</tr>
<tr>
<td>GPU Nodes</td>
<td>2 GPUs in one server</td>
<td>732</td>
</tr>
<tr>
<td></td>
<td>4 GPUs in one server</td>
<td>267</td>
</tr>
</tbody>
</table>

- Compute node: More cores with less memory
- Big memory node: Cheaper processor with fewer cores
- GPU node: Fewer GPUs per node
- Budget breakdown: 58% on compute nodes, 8% on big memory nodes, 34% on GPU nodes
Conclusions

▶ Technique is **NOT** intended to replace human-in-the-loop decision makers, but act as another informative tool

▶ $k=50$ is thought to be a reasonable “human parsable” amount of data vs. 127,000 possible server combinations

▶ 92% precision@50 for this application is thought to be excellent

▶ Precision@50 shows recommender system is viable for assisting with narrowing down alternatives to a reasonable set of alternatives for evaluation by experts

▶ Development of regression model saved time/compute resources vs. simulating all possible combinations
Use of this data and model

▶ Model and data set are made freely available under the GPLv3 license
▶ Code used for this research is also available
▶ https://github.com/shutchison/Optimal-Hardware-Procurement-for-a-HPC
▶ If it will be of some value to you, please use it!
Bibliography


See paper for exhaustive list of references
Questions

Questions?