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

- 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

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

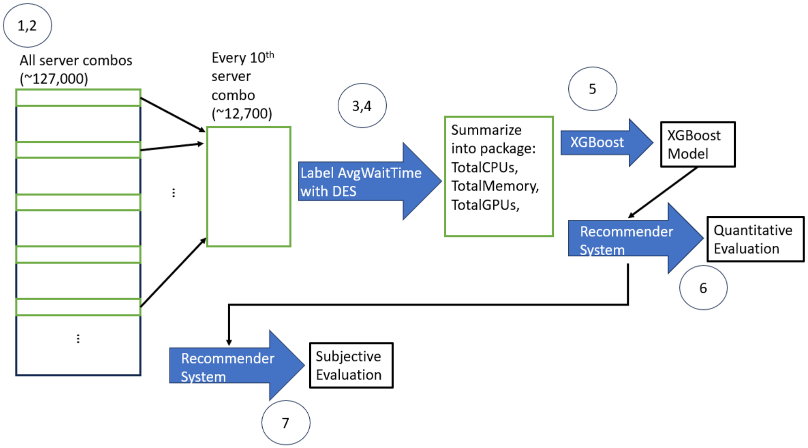
Contributions of this work

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 `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



Computing Server Combinations

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

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

- 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 appox. 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

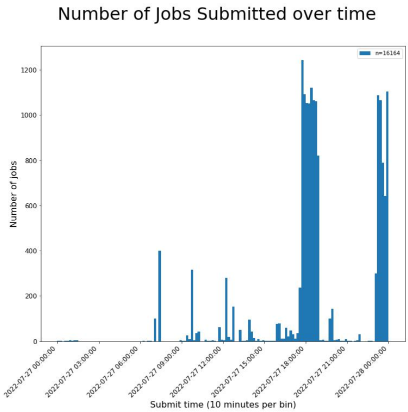
Server package example:

ComputeNode1, \$6,960	BigMemNode1, \$11,112	GPUNode1, \$14,730	...	Package Cost	Money Left
141	0	1	...	\$996,090	\$3,910
139	1	1	...	\$993,282	\$6,718
138	2	1	...	\$997,434	\$2,566
⋮	⋮	⋮	⋱	⋮	⋮
0	1	67	...	\$998,022	\$1,978

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:

$$\text{New duration} = \frac{\text{logged duration} * \text{logged processor performance}}{\text{new processor performance}}$$



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

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

- Recommendor 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

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

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

$$\text{F1@k} = \frac{(2 * \text{precision@k} * \text{recall@k})}{(\text{precision@k} + \text{recall@k})}$$

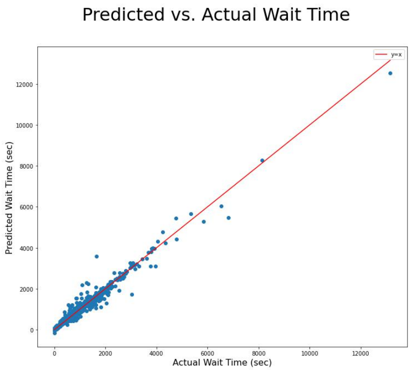
Feature Correlation

- Pearson Correlation coefficients closer to 1 or -1 indicate stronger correlation between variables
- `TotalCPUs` was strongly correlated to `AvgWaitTime` with a correlation coefficent
  - 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

	TotalMem	*TotalCPUs*	*TotalGPUs*	*AvgWaitTime*
TotalMem	1.00	0.14	-0.54	-0.23
TotalCPUs	0.14	1.00	-0.42	-0.70
TotalGPUs	-0.54	-0.42	1.00	0.44
AvgWaitTime	-0.23	-0.70	0.44	1.00

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

k value	*Precision@k*	*Recall@k*	*F1@k*
10	1.00	0.02	0.03
50	0.92	0.07	0.13
100	0.81	0.13	0.22
500	0.74	0.59	0.66
632	0.72	0.72	0.72
1000	0.58	0.91	0.71

Recommender System Performance (Subjective)

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

Sum of servers in top 50 recommendations:

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

- 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 recommendor 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

[1] R. T. Evans *et al.*, "Optimizing gpu-enhanced hpc system and cloud procurements for scientific workloads," in *International Conference on High Performance Computing*, pp. 313–331, Springer, 2021.

[2] C. Kutzner *et al.*, "More bang for your buck: Improved use of gpu nodes for gromacs 2018," *Journal of computational chemistry*, vol. 40, no. 27, pp. 2418–2431, 2019.

[3] D. G. Feitelson, D. Tsafrir, and D. Krakov, "Experience with using the parallel workloads archive," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2967–2982, 2014.

[4] "Second quarter 2023 spec cpu2017 results," 2023. https://www.spec.org/cpu2017/results/res2023q2, Accessed on June 14, 2023.

[5] S. Sharkawi *et al.*, "Performance projection of hpc applications using spec cfp2006 benchmarks," in *2009 IEEE International Symposium on Parallel & Distributed Processing*, pp. 1–12, IEEE, 2009.

[6] Y. Wang, V. Lee, G.-Y. Wei, and D. Brooks, "Predicting new workload or cpu performance by analyzing public datasets," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 15, no. 4, pp. 1–21, 2019.

[7] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), pp. 785–794, ACM, 2016.

See paper for exhaustive list of references

Questions

Questions?