CAPES: Unsupervised Storage Performance Tuning Using Neural Network-Based Deep Reinforcement Learning (Poster Extended Summary)

Yan Li
University of California, Santa Cruz
yanli@ucsc.edu

Kenneth Chang
University of California, Santa Cruz
kchang44@ucsc.edu

Oceane Bel
University of California, Santa Cruz
obel@ucsc.edu

Ethan L. Miller
University of California, Santa Cruz
Pure Storage
elm@ucsc.edu

Darrell D. E. Long
University of California, Santa Cruz
darrell@ucsc.edu

ABSTRACT

Parameter tuning is important for storage performance optimization. Current practice is to hire domain experts and do numerous manual tweak-benchmark cycles to find the optimal parameter values. The whole process can take a lot of time and is so costly that few organizations can afford to hire experts to monitor the system performance 24x7. The result is that many deployed systems are poorly tuned. We developed CAPES to address this issue. It is a model-less deep reinforcement learning-based unsupervised parameter tuning system driven by a deep neural network (DNN). It is designed to find optimal values for storage systems that have tunable parameters, including simple client-server systems and large data centers. CAPES takes periodic measurements of the target computer system's state, and trains a DNN using Q-learning, which calculates new parameter values for the target system. CAPES is minimally intrusive, and can be deployed into a production system to do continuous tuning. Evaluation of a prototype on a Lustre file system demonstrates an increase in I/O throughput up to 45% at saturation point.

CCS CONCEPTS

• **Information systems** → **Storage management**; *Distributed storage*; • **Computing methodologies** → Neural networks;

KEYWORDS

performance tuning, deep learning, q-learning

1 INTRODUCTION

Parameter tuning is a common task, in which human experts analyze a system's historical and real-time performance indicators, and tweak values of parameters in order to increase certain performance metrics for running certain workloads. It is challenging for a variety of reasons: First, the effect of adjusting a system's parameters can be influenced by factors such as system hardware, software versions, and running workloads. This correlation of variables makes predicting the effects of changes difficult at best, at worst humans may not have a clue what might happen if something were to change in a complex system. This issue arises from the fact that computers are nonlinear systems and there is no known method to quantify and model a complex system to the level of precision required by performance tuning; in practice, all performance tuning has to be done on the actual system. Second, the delay between an action and the resulting change in performance makes it even harder to correlate the relationship between system input and output. Third, the available parameter space is huge, often including thousands of parameters and each parameter can take a wide range of values. Humans can only propose and evaluate a few commonly accepted parameter values derived from past experience, and they tend to reuse those same values across many systems for simplicity, leaving a larger, more diverse, parameter space unexplored. Fourth, assigning human experts to monitor dynamic workloads 24x7 is simply too costly in practice.

In machine learning, we can approach this problem as a game where the goal is to find an appropriate setting that will render the system more efficient. By observing some indicators in the system, the player can maximize (or minimize) a cumulative reward such as operations per second, data transfer throughput, or energy use. Recent advancements in machine learning showed that Deep Reinforcement Learning (DRL) can perform unsupervised learning well on diverse data featuring long delays between action and reward [4]. Many techniques are being used in tandem to make this possible, such as Deep Q-learning and experience replay [3].

We developed CAPES (Computer Automated Performance Enhancement System) and demonstrated that it can increase the throughput of the Lustre high performance storage system [5] up to 45% under heavy workloads. The training is online and requires around 12 to 24 hours, which can be done during the system's daily operation. This is a significant increase in efficiency and reduction in cost, especially for large and expensive system installations.

In comparison to earlier automatic tuning systems, CAPES has several advantages:

- It requires no prior knowledge of the target system.
- It requires little change to the target system and little downtime for setting up.
- It can run continuously to adapt to dynamically changing workloads.
- It can dynamically chooses optimal values for parameters that used to be set statically.
- It can combine training experiences from different sessions to accelerate the training process for new models.

2 DESIGN AND ALGORITHM

The CAPES Design figure in the poster shows one possible way to use CAPES with a target system. Each node has a Monitoring Agent and a Control Agent running on them. The Monitoring Agents gather Performance Indicators and Rewards from the target system's nodes and send them to the Interface Daemon. Performance Indicators are system measurements that are related to the system's operating status. Rewards vary based on current tuning efforts, and reflect the successfulness of the current tuning. The Interface Daemon relays the incoming Performance Indicators into the Replay Database (Replay DB). The DRL Engine reads the Performance Indicators from the Replay DB to do training. At a fixed interval, the DRL Engine sends back an Action via the Interface Daemon, which will broadcast the action to the action's targeted Control Agents. These actions are also stored within the Replay DB as part of Experience Replay. Finally, Control Agents change the parameter values according to received actions.

Performance Indicators are inputs to the DNN, which analyzes them to understand how the system is running. We should include system states that are related to the metric we wish to tune. This is a feature selection problem, which was deemed to be one of the most important steps for successfully applying almost any machine learning algorithm. However, advances in DNN has rendered this step less important because DNN is good at selecting useful data among noisy, raw inputs. Therefore, we can be quite liberal on choosing performance indicators; all measurements that are likely related to the performance of the system should be included. Both raw and secondary system statuses, derived from raw system status, can be included.

Reward guides the direction of the tuning process. After performing each action on the target system, CAPES measures an immediate reward. We do not need to consider the delay between an action and a reward since the Q-function will ultimately converge to the optimal oracle-like function after iterative trainings, according to Bellman's proof [6].

Training steps are to minimize the prediction error for the training data. Prediction error is the different between the DNN's predicted performance after observing the system's status and the actual system performance one second later. The Q function is parameterized using a neural network that maps an observation to an array of Q-values of each action [4]. We use the Adam optimizer [1] for training and use the target network method to prevent overfitting.

Actions dictate what a target system's parameters should be, and CAPES can tune many parameters at the same time. At a fixed rate (every action tick), CAPES decides on an action that either increases or decreases one parameter by a step size. The valid range and tuning step size are customizable for each target system parameter.

3 IMPLEMENTATION AND EVALUATION

We chose the Lustre file system as the target system for testing whether CAPES can improve the throughput of the workload during peak times and to understand its effectiveness on a variety of workloads. The Prototype and Evaluation box on the poster lists the setup of the evaluation system and results. It can be seen that CAPES works best with workloads that are dominated by writes; it increased the performance of the workload with $1:9$ read to write ratio by 45%. CAPES did not show obvious effect on read-heavy workloads. This is related to what parameters we chose to tune: tuning the number of allowed outstanding I/O requests (congestion window size) of Lustre does have a bigger impact on write than read. The evaluation used storage servers that have hard disk drives as the underlying storage device, which need to spend a large portion of the I/O time doing seek when handling random reads, and having more read requests in the queue does little to mitigate the long seek time. In contrast, outstanding random write requests can be merged and handled more efficiently if there are more of them in the I/O queue, thus tuning the number of allowed outstanding write requests has a bigger impact on the efficiency of the merge, and in turn the performance.

We also observed that 12 hours training is not enough to find the optimal tuning policy for optimizing the Filebench file server workload. The file server workload is especially challenging for Q-learning because, unlike other random read/write workloads, it involves a wide range of read, write, and metadata operations. It required about 24 hours of training to converge to a good policy that can lead to a 17% increase in throughput.

For overfitting analysis and a thorough discussion of the algorithm and design, please read our Supercomputer '17 technical paper [2].

To further promote research on this topic, we will open source CAPES and our modified Lustre system at https://github.com/mlogic/capes-oss and https://github.com/mlogic/ascar-lustre-2.9-client.

ACKNOWLEDGMENTS

This research was supported in part by the National Science Foundation under awards IIP-1266400, CCF-1219163, CNS-1018928, CNS-1528179, by the Department of Energy under award DE-FC02-10ER26017/DESC0005417, by a Symantec Graduate Fellowship, by a grant from Intel Corporation, and by industrial members of the Center for Research in Storage Systems.

REFERENCES

[1] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. (2015). arXiv:cs.LG/1412.6980

[2] Yan Li, Kenneth Chang, Oceane Bel, Ethan L. Miller, and Darrell D. E. Long. 2017. CAPES: Unsupervised Storage Performance Tuning Using Neural Network-Based Deep Reinforcement Learning. In *Proceedings of the 2017 International Conference for High Performance Computing, Networking, Storage and Analysis (SC17)*. ACM, Denvor, CO, USA. DOI: https://doi.org/10.1145/3126908.3126951

[3] Long-Ji Lin. 1993. *Reinforcement learning for robots using neural networks.* Technical Report. DTIC Document.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (26 02 2015), 529–533. http://dx.doi.org/10.1038/nature14236

[5] Open Scalable File Systems, Inc. 2014. The Lustre® file system. http://www.opensfs.org/. (2014).

[6] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA.