CAPES: Unsupervised Storage Performance Tuning Using Neural Network-Based Deep Reinforcement Learning

Yan Li[1], Kenneth Chang[1], Oceane Bel[1], Ethan Miller[1,2], Darrell Long[1]

[1] University of California, Santa Cruz

[2] Pure Storage

What is parameter tuning?

Find the parameter values to achieve optimal performance for a certain workload running on a certain device.

Sample parameters:

- I/O queue depth
- RPC rate limit
- worker thread count
- Buffer sizes

Parameter tuning doesn’t change:

- Hardware
- System design
- Source code
- Application
- Settings that destroy data

The Problem

Parameter tuning is important:

- Parameter tuning can greatly affect a system's performance.

Parameter tuning is challenging and costly:

- Every system, every workload is different.
  - Hardware/software bugs and quirks.
  - Device aging.
  - Slow shifting workloads.
- Need to hire domain experts.
- Finding the optimal setting is a lengthy trial-and-error process.
- Few can afford 24x7 parameter tuning.

Automatic parameter tuning is hard

Model-based methods are usually impractical:

- Different models are required for different hardware/software.
- Nobody has resource to maintain these models.

Fundamental challenges:

- Correlating parameter changes with performance change is hard.
- Huge parameter spaces to scan.

An ideal automatic parameter tuning system

Goal:

- Customizable optimization goal.
- Online training.

Features:

- Tune a wide range of parameters.
- Model-less.
- Requires no prior knowledge of system or workload.
- Work on many kinds of systems.
- Short training time.
- Stable.
- Works 24x7.

Who can benefit from automatic parameter tuning

Large Installations:

- Public cloud providers.
- Supercomputers.
- Services for a large enterprise.

Small Installations:

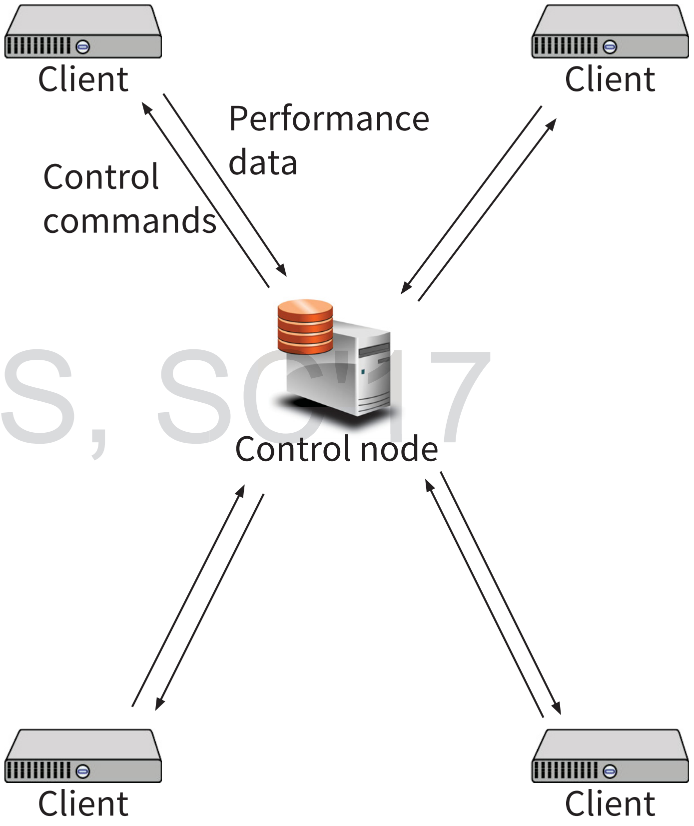
- Private on-site clouds.
- Prototype systems.
- Evaluating emerging technologies.

Usually these systems are poorly tuned because small installations have no expertise or resource to tune at all.

CAPES high-level architecture

CAPES: Computer Automated Performance Enhancement System

- Control node collects performance data and tweaks parameter values.
- Requires (small size) communications between client and control node.



Constructing it as a Reinforcement Learning problem

Reinforcement learning is about:
How an agent behaves in an environment to maximize reward.

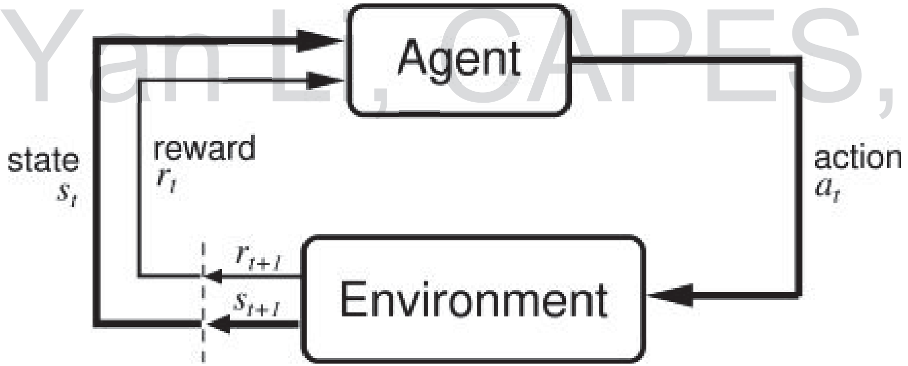
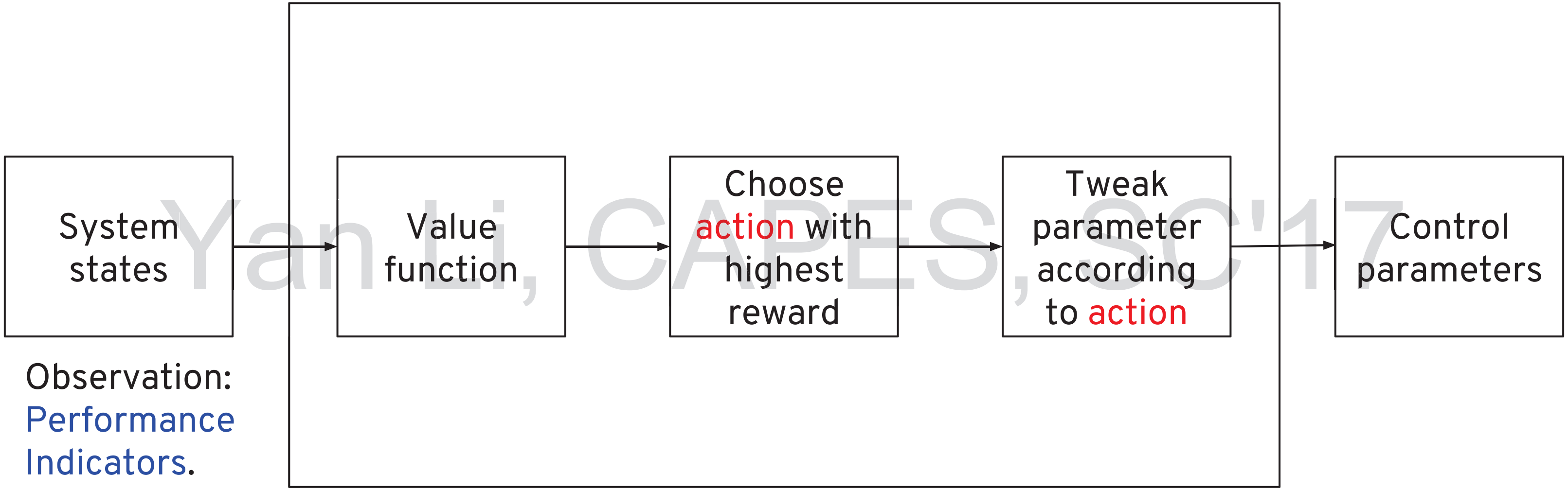


Image: https://webdocs.cs.ualberta.ca/~sutton/book/ebook/node28.html

Applying reinforcement learning to parameter tuning

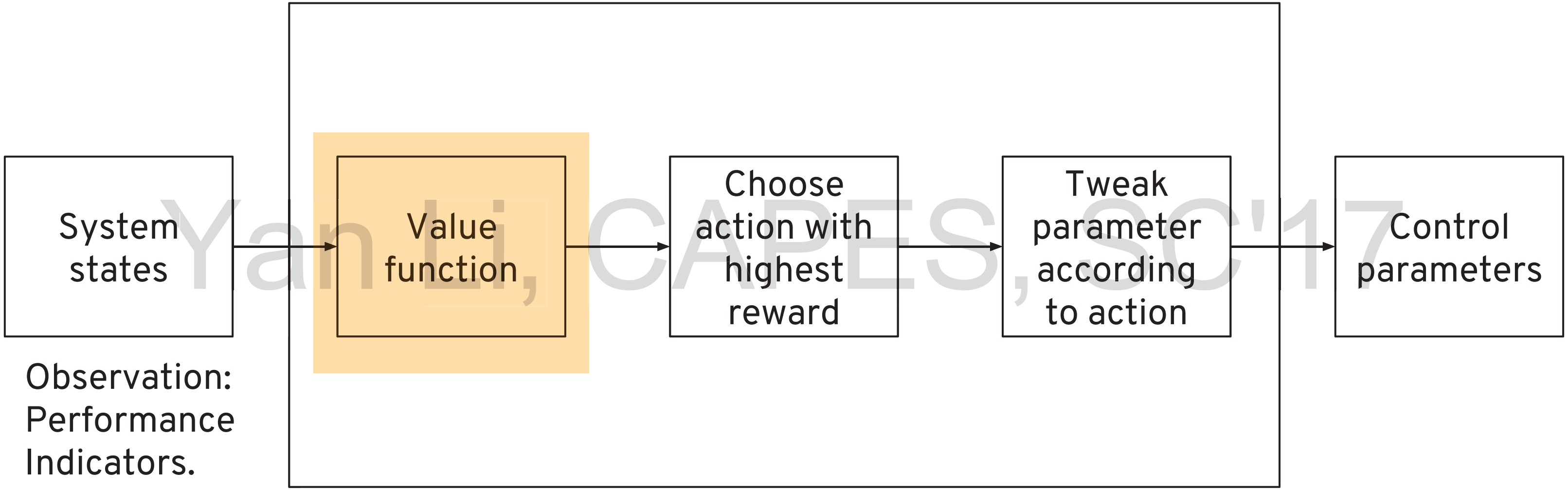
Applying reinforcement learning to parameter tuning



- Observation: Performance Indicators.
- Reward: Performance.

Reinforcement learning controller

Finding the value function is critical



- Observation: Performance Indicators.
- Reward: Performance.

Reinforcement learning controller

Challenges of reinforcement learning

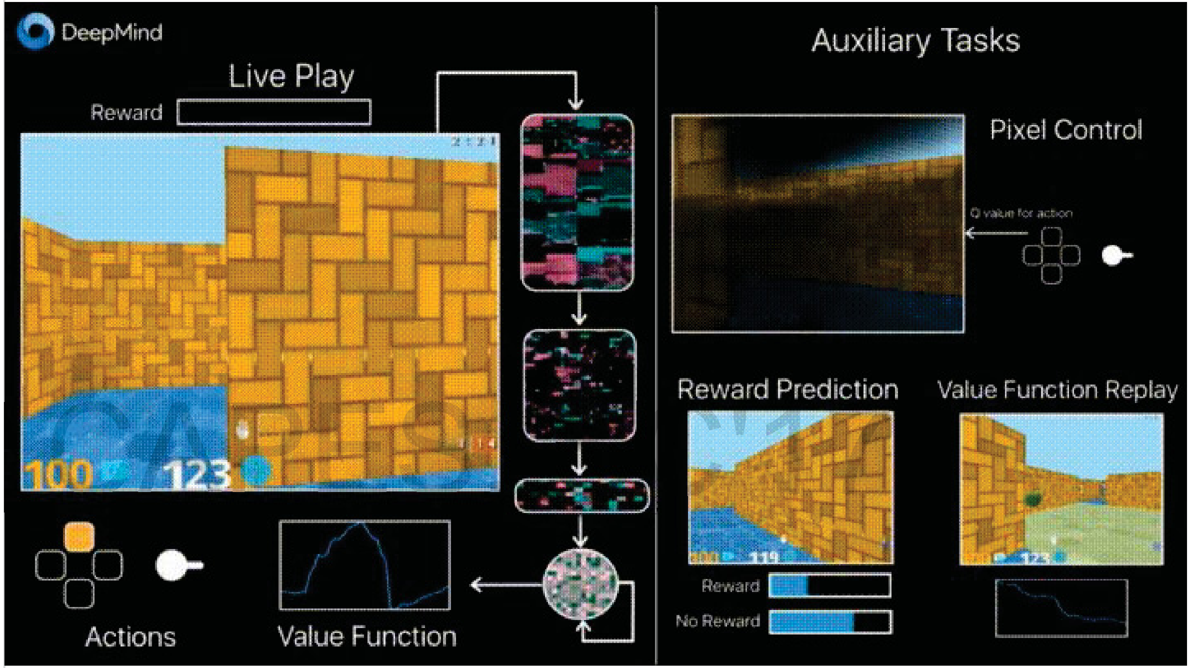
1. Long and non-uniform delay between action and reward.
2. Need huge amount of data for training.
3. Unpredictable performance during training.

Challenges for using neural networks as the value function:

1. Instability.
2. Slow to converge.
3. Overfitting.

Deep Reinforcement Learning outperforms human in many games

Google DeepMind, "Human-level control through deep reinforcement learning", *Nature* 518, 529–533 (26 February 2015)



https://deepmind.com/blog/reinforcement-learning-unsupervised-auxiliary-tasks/

Deep Q-Learning (DQL)

Deep Reinforcement Learning using Q-function

- Q-function: the maximum discounted future reward when performs perfectly.

  $$Q(s_t, a_t) = \sum_{i=t}^{n} \gamma^{i-t} r_t$$

  (s_t is system state at time t, a_t is action at time t, r_t is reward at time t, γ is reward discount.)

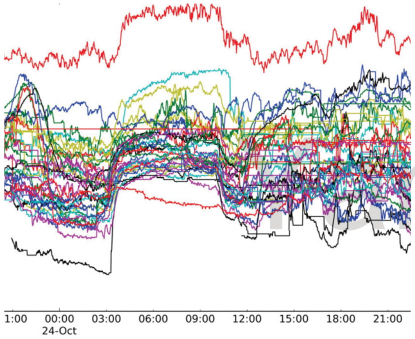
- Q can be solved iteratively (Bellman's equation)

  $$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$$

Deep Q-Learning (DQL)

- Works with multi-dimensional nonlinear systems.
- Can take noisy raw data as input.
- Can handle long, non-uniform delays between action and reward.
- Doesn't require a predefined model (model-free).
- Training is online, unsupervised, and off-policy. Off-policy training is based on using minibatch.

Applying reinforcement learning to parameter tuning



Unfiltered raw performance indicators
(system state)

→ Deep neural network →

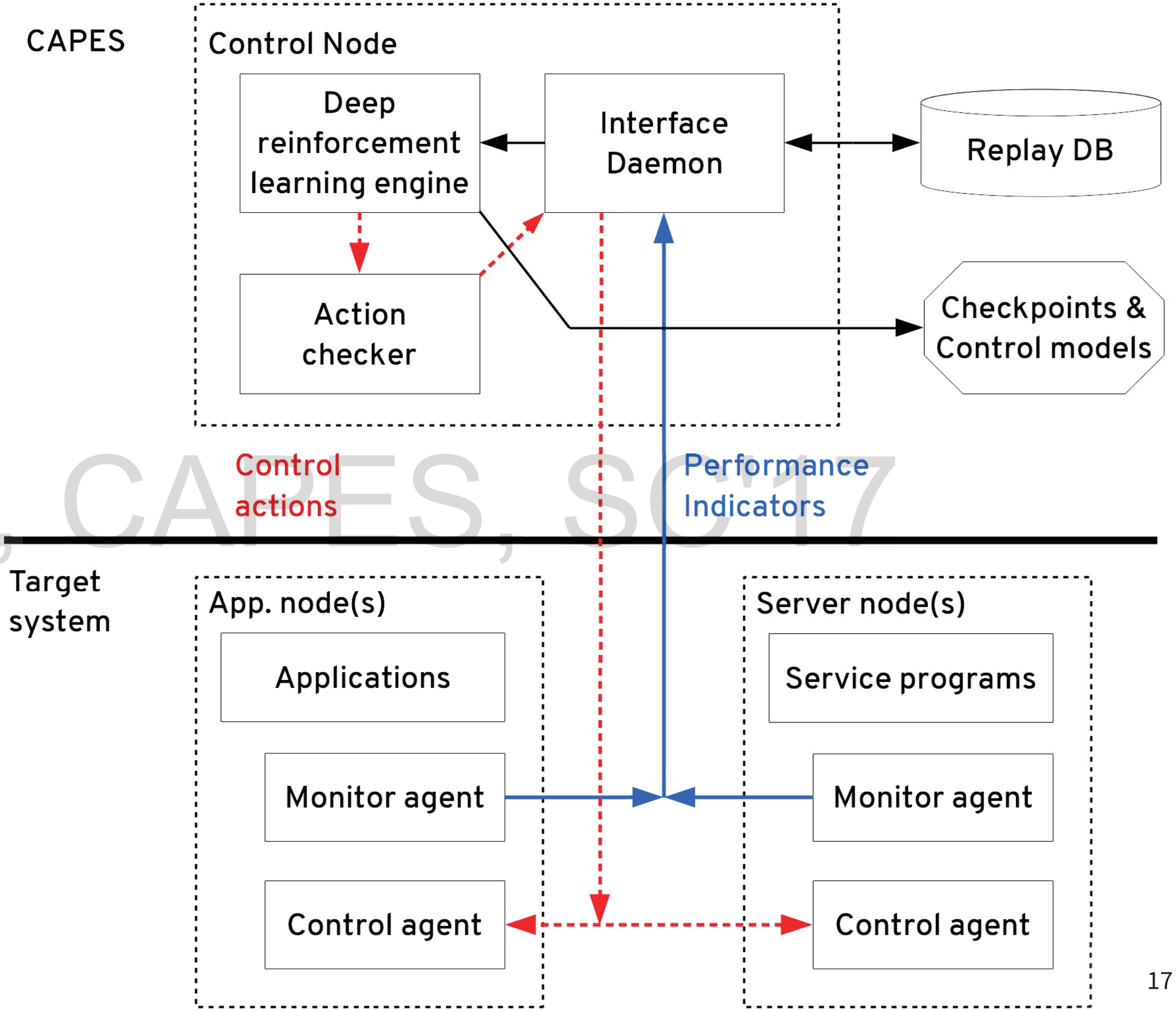
Candidate action	Predicted reward
Action 0: Do nothing	0.382
Action 1: Increase Parameter A	0.741
Action 2: Decrease Parameter A	0.127
Action 3: Increase Parameter B	0.547
Action 4: Decrease Parameter B	0.123
Action 5: Increase Parameter C	0.372
Action 6: Decrease Parameter C	0.457

Action table
(possible actions)

https://s3.amazonaws.com/stratany2012/Temperature.png

CAPES Prototype for Lustre

Computer Automated Performance Enhancement System



Performance Indicators used in CAPES/Lustre Prototype

Performance indicators	Definition
write throughput	the write throughput of the past second
read throughput	the read throughput of the past second
ack_ewma	exponentially weighted moving average (EWMA) of gaps between RPC acks
send_ewma	EWMA of gaps between sender timestamp embedded in RPC acks

Performance indicators	Definition
pt	the time needed for server to finish reading/writing 1 MB data request
pt_ratio	current pt / min(pt) seen so far
dirty bytes in write cache	the dirty bytes on the client write cache

Parameters to be tuned in CAPES/Lustre Prototype

1. Client I/O Rate limit
2. Client I/O queue depth limit
   (congestion window size)

Also a bag of tricks

- Use two networks for training:
  One fast moving, the other slow moving. More stable and faster to converge.
- Mini-batch training:
  Each training step uses a 32-sample minibatch randomly sampled from historical training data. Reduces overfitting and faster to converge.
- Action checker:
  Check candidate action against preset rules to prevent bad parameter values. Avoids bad performance during training.

Evaluation of CAPES on Lustre

Test setup

- Lustre 2.9
- 4 servers and 5 clients
- 1 GB ethernet

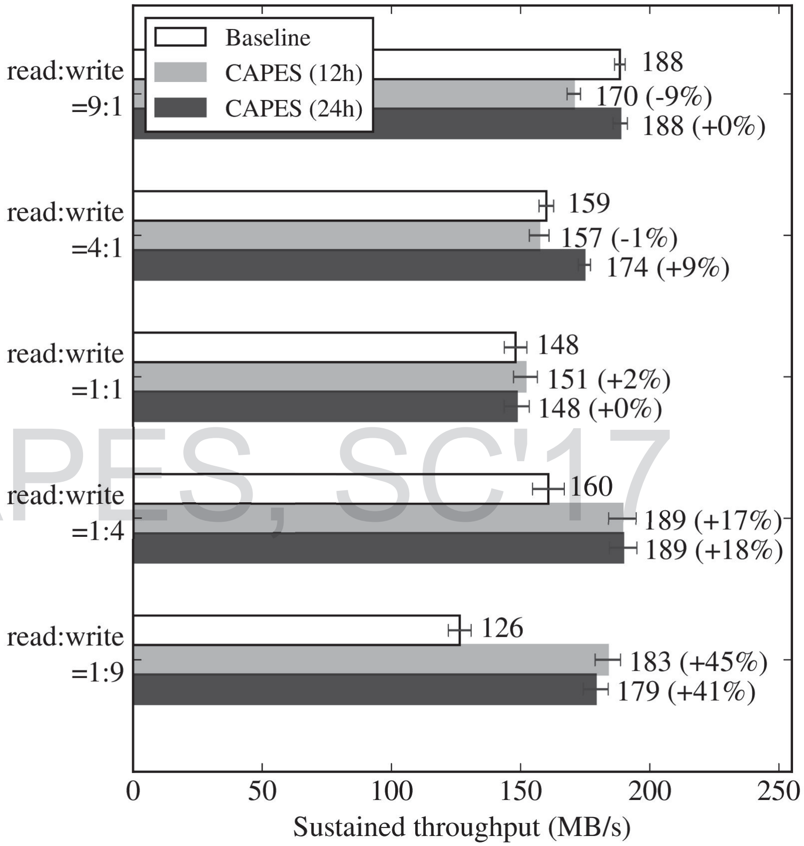
CAPES Control Node

- Xeon E5-2637
- 128 GB RAM
- nVIDIA GTX 1080 GPU
- TensorFlow

Random read/write workload

Four threads on each client. Continual 1 MB random read/write.

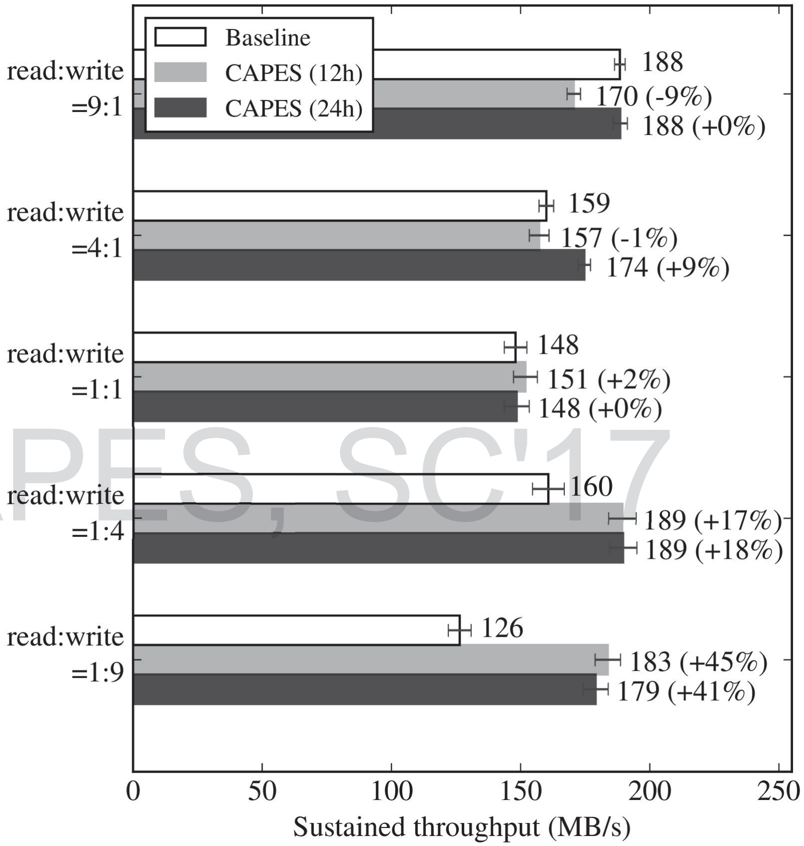
All evaluation workloads generate enough I/O to saturate the servers.



Error bars show the confidence interval at 95% confidence level.

Not effective for read heavy workloads.

Effective for write heavy workloads.



Error bars show the confidence interval at 95% confidence level.

Filebench fileserver workload

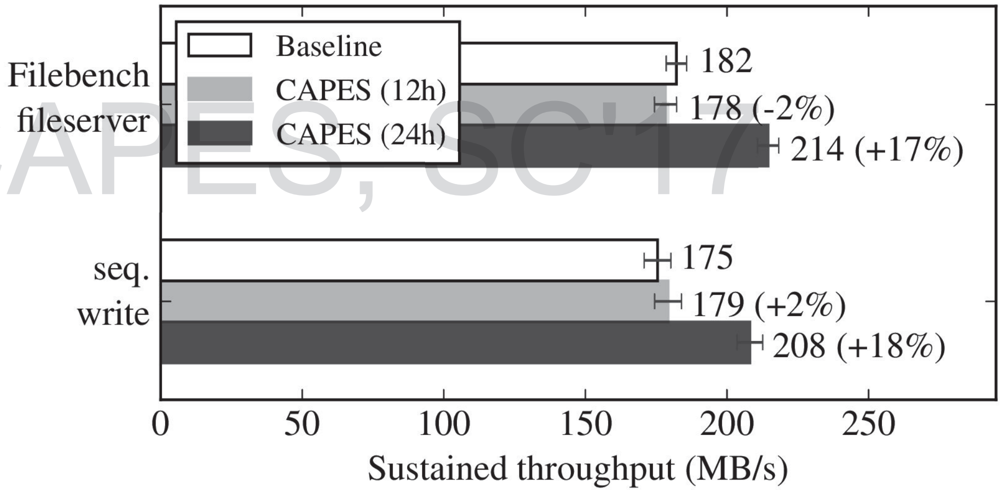
Workload includes read, write, and metadata operations:

1. Create a file and write the file to 100 MB.
2. Open another file and append random sized data (mean at 100 MB).
3. Open a randomly picked file and read 100 MB.
4. Delete a random file.
5. Stat a random file.

All evaluation workloads generate enough I/O to saturate the servers.

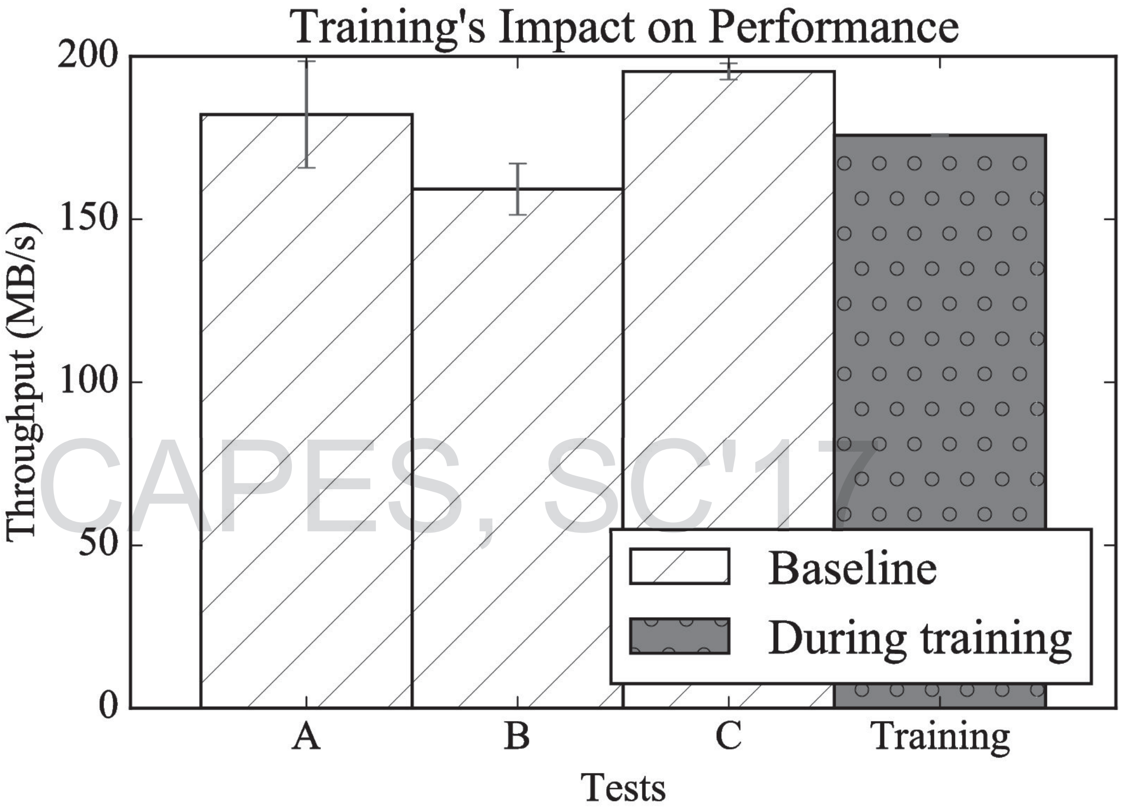
Sequential write workload

Five threads on each client. Continual 1 MB sequential write.



Error bars show the confidence interval at 95% confidence level.

Training has little impact on system performance



Fileserver workload throughput with and without CAPES training.
Error bars show the confidence interval at 95% confidence level.

Conclusion

CAPES ...

- Worked well for a complex system like Lustre.
- Doesn't require human supervision.
- Can be turned on 24x7 to handle changing workloads.
- Caused little impact during training.
- Doesn't require a special training step.
- Worked best when changing parameters has a great impact on performance.

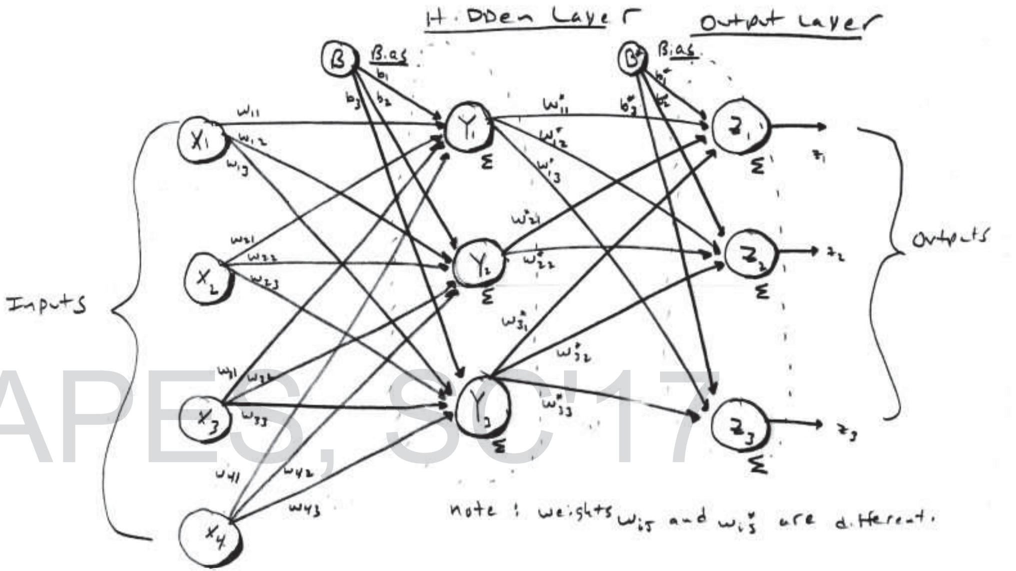
Future work

- Looking for collaborators: https://github.com/tuneupai/capes-oss
- Evaluation on larger systems.
- Evaluation on other storage systems, like Ceph, OpenStack, Apache Cassandra, etc.
- Tuning more parameters.
- Fine tuning the training algorithm.

Any comments or ideas, please let us know!

Yan Li yanli@tuneup.ai

https://github.com/tuneupai/capes-oss



https://www.gamedev.net/uploads/monthly_06_2011/ccs-8549-0-74375900-1307091491.jpg

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 (http://www.ssrc.ucsc.edu/sponsorlist.html).