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

Yan li, Kenneth Chang, Oceane Bel, Ethan L. Miller, Darrel D. E. Long

Performance Tuning

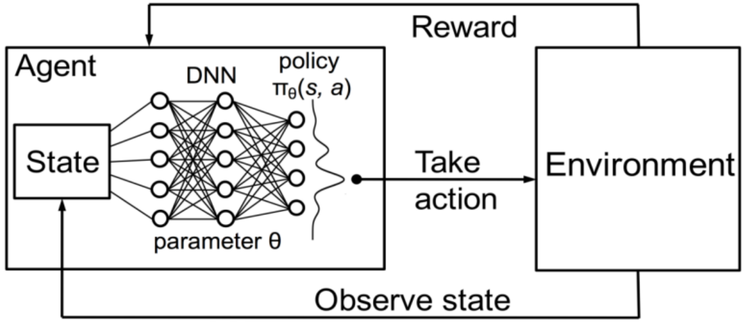
- Tuning system’s parameters for high performance
- Can be very challenging
    - Correlation between several variables in a system
    - Delay between action and resulting change in performance
    - Huge search space
    - Requires extensive knowledge and experience
- Static parameter values for dynamic workloads
- Congestion Curse-Exceeding certain load limit will negatively affect the performance of several components
- Automated Performance Tuning is required!!

Automated Parameter Tuning

- Challenges
  - Systems are extremely complex.
  - Workloads are dynamic and they also affect each other
  - Responsiveness
  - Scalability
  - Has to be tuned for multiple objective functions.
- Dynamic parameter tuning-Partially Observable Markov Decision Process
- Hard Problem
  - Varying delays between action and result
  - Change in performance could be a result of sequence of modifications
- Credit Assignment Problem

CAPES

- Computer Automated Performance Enhancement System
- Unsupervised Problem
  - Parameters can change based on several factors not just workload. So labelled data is impractical
- Model-less Deep Reinforcement Learning
  - A game to find parameter values that maximize/minimize some function(may be throughput or latency)
  - Use of deep learning techniques with reinforcement learning.

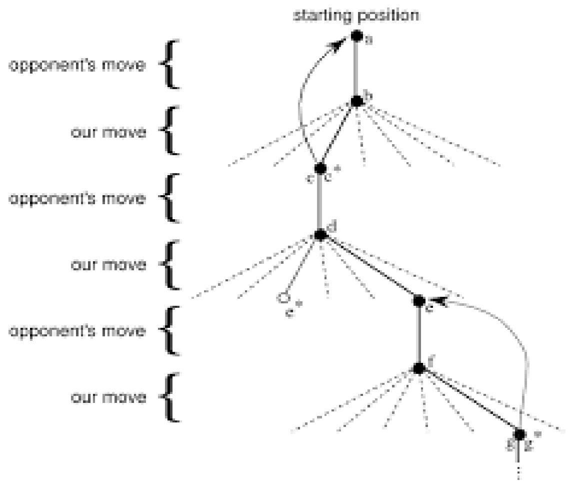


Q-value

Return: $R_t = \sum_{i=t}^{n} r_i$ $\quad R_t = \sum_{i=t}^{n} \gamma^{i-t} r_i$

Q-value: $Q(s_t, a_t) = \max_{\pi} R_{t+1}$

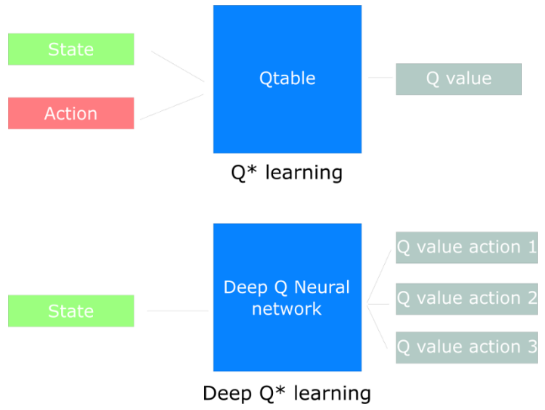
Policy: $\pi(s) = \max_{a} Q(s, a)$



Bellman Equation:

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

Deep-Q-Learning



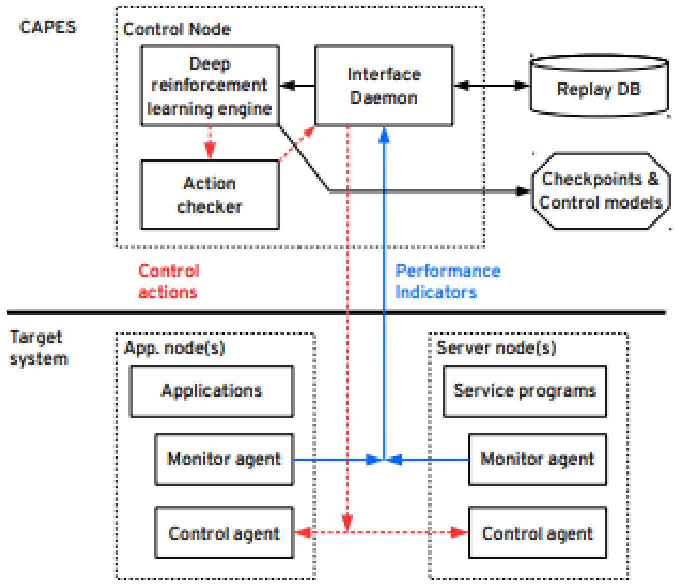
- Need to learn Q-function
  - Core of Q-learning
- Q-network
  - A deep neural network to approximate the Q-function
  - Output of Q-network will be a Q-value for a given state and action
  - Weights of the network to reduce the MSE for samples

$$L_i(\theta_i) = \mathbf{E}_D[(r_i + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)^2]$$

- Since we don't have the actual Q-value of all possible actions we try to approximate and over time we update the weights to predict reasonable predictions.

Architecture

- Monitoring Agent
  - Gather Information about current state of the network and rewards(objective function)
  - Communicate with Interface daemon
- Replay Database
  - Stores received information and performed actions
  - Experience DB
- DRL Engine
  - Reads the data from replay DB and sends back an acti
- Control Agents
  - Performs the received action on the nodes.
- Interface Daemon
  - Communicates between CAPES and target system
- Action Checker
  - Checks if the action is valid

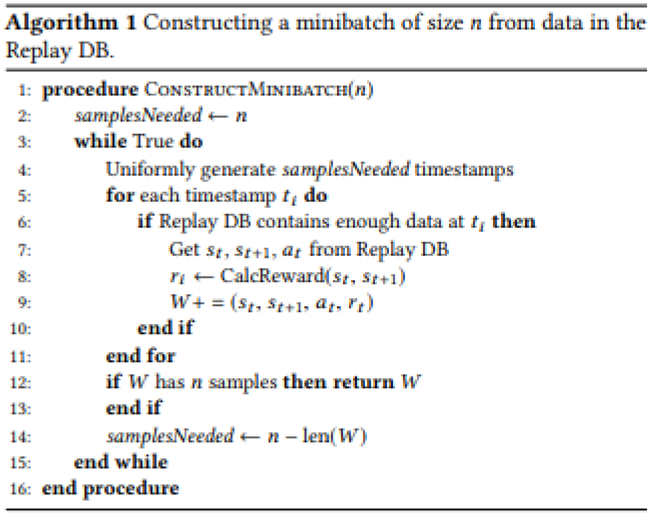


Algorithm

- Data is collected at certain frequency(1 sec)
    - Sampling Tick
    - Sends only when its different from previous tick
- Observation matrix to capture the trend

$$s_t = \begin{vmatrix} d_{1,t-S+1} & d_{2,t-S+1} & \cdots & d_{N,t-S+1} \\ d_{1,t-S+2} & d_{2,t-S+2} & \cdots & d_{N,t-S+2} \\ \cdots & \cdots & \cdots & \cdots \\ d_{1,t} & d_{2,t} & \cdots & d_{N,t} \end{vmatrix}$$

d=objective ,i=node, j=time,N=total nodes,S=sampling ticks



Algorithm 1 Constructing a minibatch of size n from data in the Replay DB.

```
1: procedure ConstructMinibatch($n$)
2:     $samplesNeeded \leftarrow n$
3:     while True do
4:         Uniformly generate $samplesNeeded$ timestamps
5:         for each timestamp $t_i$ do
6:             if Replay DB contains enough data at $t_i$ then
7:                 Get $s_t$, $s_{t+1}$, $a_t$ from Replay DB
8:                 $r_t \leftarrow$ CalcReward($s_t$, $s_{t+1}$)
9:                 $W+ = (s_t, s_{t+1}, a_t, r_t)$
10:            end if
11:        end for
12:        if $W$ has $n$ samples then return $W$
13:        end if
14:        $samplesNeeded \leftarrow n - \text{len}(W)$
15:    end while
16: end procedure
```

Batches of these observations are send to DRL engine

Reduce the data movement overhead

Neural Network Training

- It is proven that a NN with 1 hidden layer can approximate any mathematical function
- 2 hidden layer network
  - Adam optimizer is used
  - Tanh activation is used
- Output layer consists of same number of nodes as the number of actions each denoting a action.
- Each training step needs the state transition information which is checked in Replay DB before training.

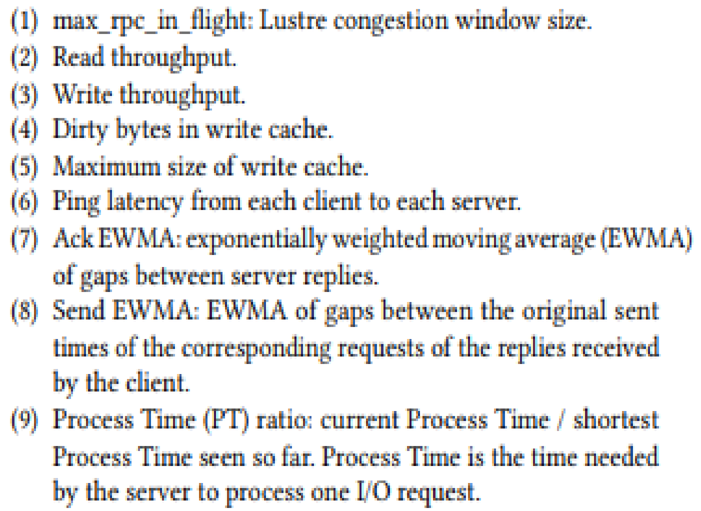
$$w_t = (s_t, s_{t+1}, a_t, r_t)$$

Performance Indicators and Rewards

- Performance Indicators-Feature extraction problem
  - Can be relaxed as DNN are known for feature extraction
  - Date and time can be included as separate features if workloads seem to be cyclic
  - Raw and secondary system status can be used
- Rewards
  - Immediate rewards are taken after an action is performed
  - Reward is objective function like latency or throughput
  - No need to worry about delay in change of the performed action
- Actions
  - Increase or decrease the value of parameter by a step size-can be varied based on system
  - Null action is also included if no action is required
  - This makes total number of actions 2 x tunable_parameter +1

Implementation

- Lustre file system-high performance distributed file system
- 1 Object Storage client/client and 4 servers and implemented using 5 clients.
- All nodes have the same system configuration
  - 113MB/s read ,106 MB/s write
  - Default stripe count of 4 with 1MB stripe size
  - 1:1 network to storage bandwidth ratio -HPC
- CAPES runs on different dedicated node
- Only 2 parameters are tuned
  - Max_rpc_in_flight:congestion window size
  - I/O rate limit:outgoing I/O requests allowed



(1) max_rpc_in_flight: Lustre congestion window size.
(2) Read throughput.
(3) Write throughput.
(4) Dirty bytes in write cache.
(5) Maximum size of write cache.
(6) Ping latency from each client to each server.
(7) Ack EWMA: exponentially weighted moving average (EWMA) of gaps between server replies.
(8) Send EWMA: EWMA of gaps between the original sent times of the corresponding requests of the replies received by the client.
(9) Process Time (PT) ratio: current Process Time / shortest Process Time seen so far. Process Time is the time needed by the server to process one I/O request.

Evaluation

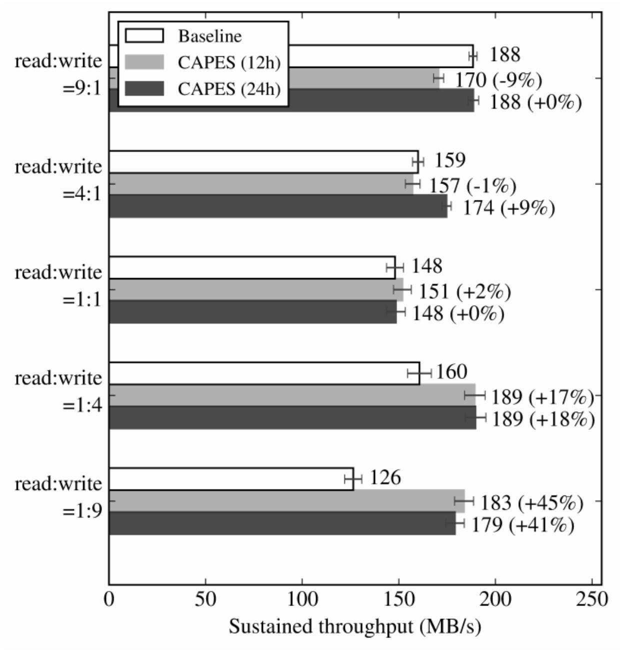


Figure 2: Overview of random read write workloads evaluated with CAPES. Throughput before, after 12 hours training, and after 24 hours training are shown. Baseline uses default Lustre settings. Error bars show 95% confidence intervals.

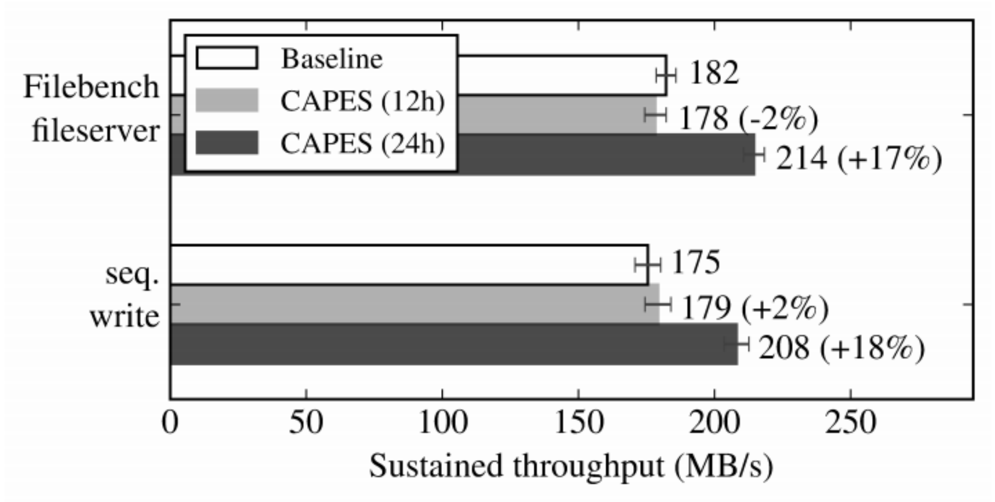
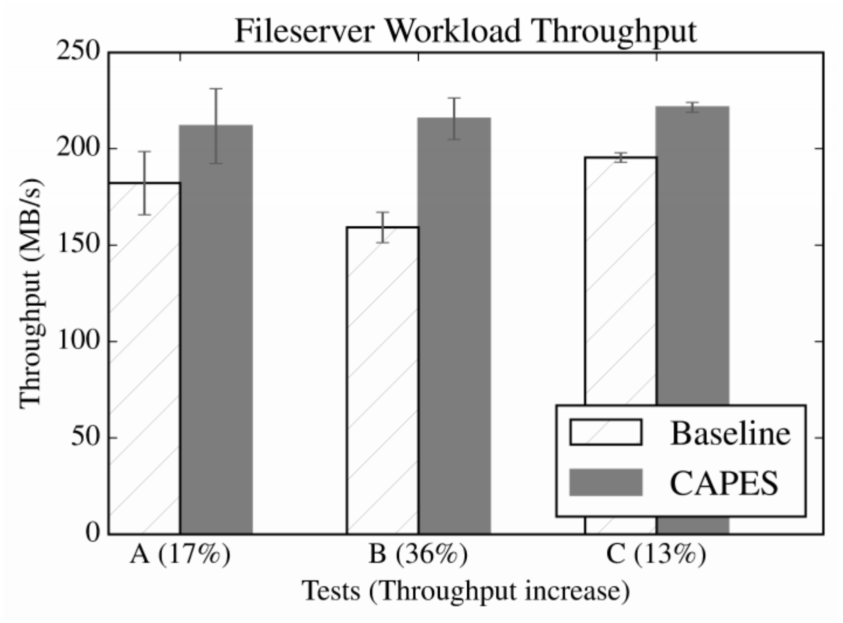


Figure 3: Overview of Filebench file server and sequential write workload evaluated with CAPES. Throughput before and after CAPES tuning are shown. Baseline uses default Lustre settings. Error bars show 95% confidence intervals.

Training Evaluation



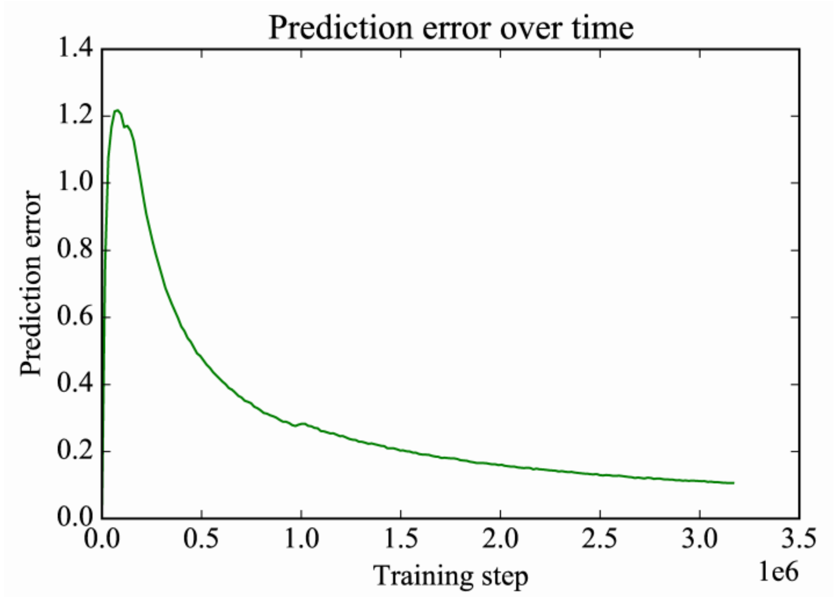


Figure 4: Fileserver workload throughput with and without CAPES tuning. Baseline uses default Lustre settings. Error bars show the confidence interval at 95% confidence level.

Training impact on performance

Random action during start of training

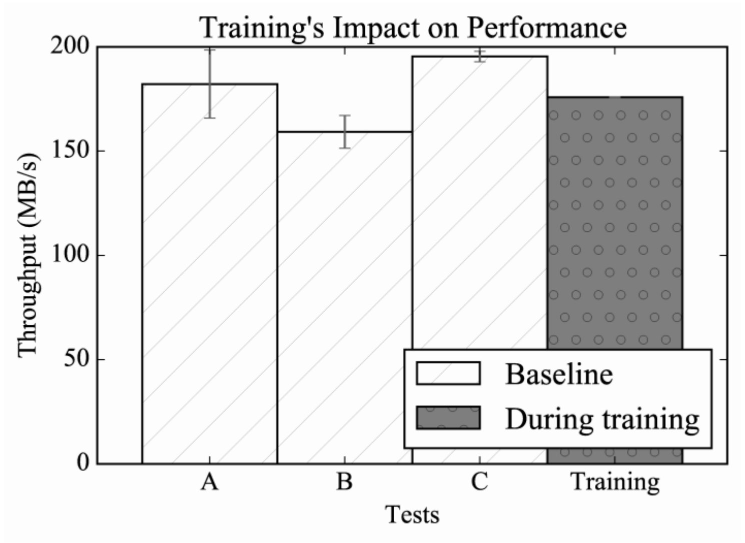


Figure 6: Baseline throughputs and training session overall throughput. Error bars show the confidence interval at 95% confidence level.

Thoughts:

- It would be better if CAPES/other technique on top of capes can even select/give more importance to different tunable parameters based on requests.
- There is still a possibility for improvement by using other RL methods like Actor-critic where multiple agents are trained for the same problem-each will have different experience .
- Increment or decrement of parameter by a fixed step size doesn’t seem logical.It can also be scaled based on the workload.