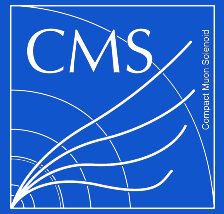


International Symposium on Grids & Clouds 2021

March 23rd, 2021

Reinforcement Learning for Smart Caching at the CMS experiment

Tommaso Tedeschi[1,2] (tommaso.tedeschi@pg.infn.it)

On behalf of the CMS collaboration

Mirco Tracolli[1,2,3], Daniele Spiga[2], Loriano Storchi[2], Diego Ciangottini[2], Marco Baioletti[1], Valentina Poggioni[1]

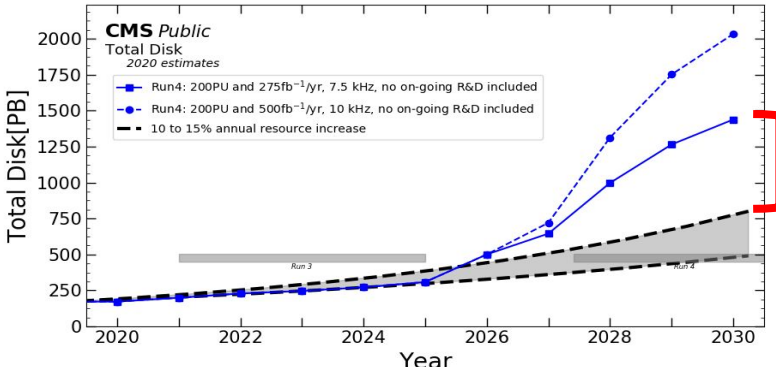
[1] *Università degli Studi di Perugia, Perugia, Italy*
[2] *Istituto Nazionale di Fisica Nucleare - Sezione di Perugia, Perugia, Italy*
[3] *Università degli Studi di Firenze, Florence, Italy*

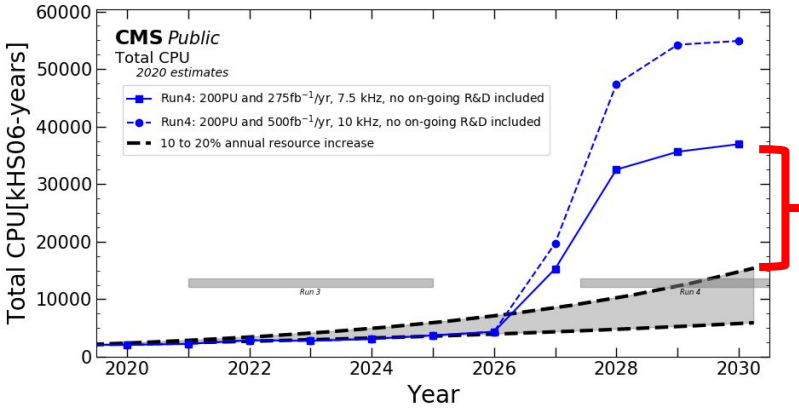
Outline

- Introduction to the context
  - Data Lake and Data caching
- Smart Caching
  - Reinforcement Learning
    - Q-Learning
    - DQN
  - QCACHE
    - SCDL and SCDL2
    - DQN
- Results
  - Evaluation metrics
  - Dataset and simulation
- Conclusions and next steps

HL-LHC challenges

In 2026+ LHC will enter the EB era (**HL-LHC**):
expected storage and computing needs are above the expected technology evolution





Need for a new model to optimize hardware and operational costs

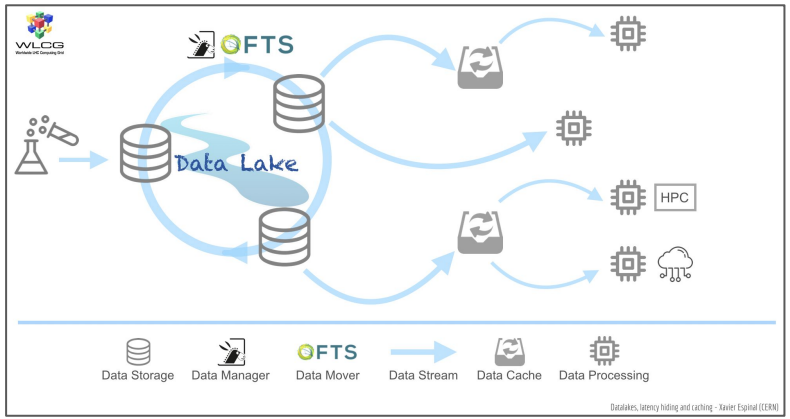
Storage is the most concerning problem due to cost and lack of straightforward solutions. R&D and combination of many different ones:

- Intensive usage and integration of specialized hardware (GPU, FPGA, etc..)
- Reduced data format (NanoAOD at CMS)
- **New distributed model**
- ...

Data Lake @HL-LHC

Data Lake Model

- A small number of Data Lakes across the world with a reduced number of storage endpoints wrt the current WLCG model
- Envisions a mix of distributed caches directly accessed from compute nodes



In a Data Lake like design:

- the **network** is obviously central
- **caches** are crucial in the design of analysis optimized systems

A lot of R&D (and testbedding) has been done on both topics:

- WLCG DOMA
- EU projects like ESCAPE
- US CMS Data lake proposal
- SoCal Cache
- INFN distributed cache
- ...

CMS studies

CMS already tested cache system usage **benefits**. In particular it was shown that the usage of caches allows to:

- **reduce** the overall WAN **traffic**
- make the processing job that requested the data **more efficient** by reducing I/O wait time for remote data
- provide **read ahead** capability
- **save disk space**

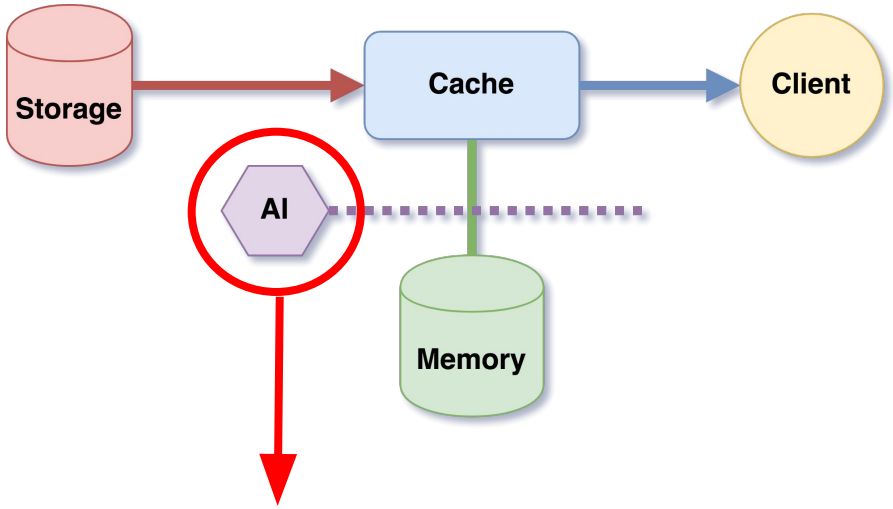
We also foresee a local cache as a key component of the future **Analysis Facility** we are prototyping at INFN

Now the main goal is to enhance cache efficiency with a smart caching algorithm

Smart Caching at CMS

The aim of the project is to **develop a smart cache system using AI algorithms in the CMS experiment context**:

- AI directly manipulates cache memory, deciding what to write or delete
- The goal is to obtain an algorithm that uses fewer storage resources wrt classic cache algorithms (LRU, LFU, etc..) while maintaining similar performance



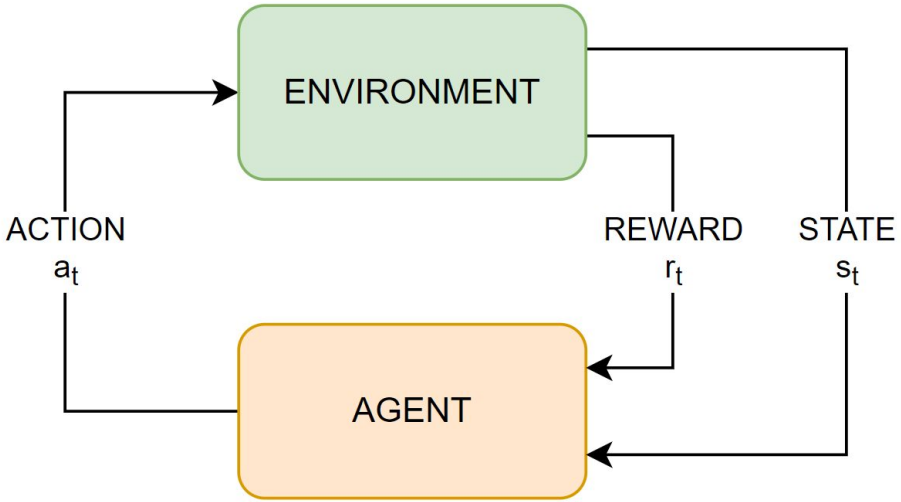
QCACHE:
a Reinforcement Learning-based framework

Reinforcement Learning - 1

Reinforcement Learning (RL) is the branch of Machine Learning that studies agents that learn through an iterative "trial and error" process interacting with an environment.

At each step, a RL agent receives a **state** (or a partial observation) from the environment as input and chooses a certain **action**, getting a **reward** (or punishment) from the environment, which is used to update the agent itself.

The ultimate goal is to maximize the cumulative reward (the so-called **return**)



Reinforcement Learning - 2

A RL agent learns a **policy**, which is the function π that the agent uses to choose which action to take given a certain state or observation. In principle, the policy could be either deterministic or stochastic.

A RL agent seeks to find the **optimal policy π^***, which maximizes the expected return when the agent acts according to it.

Optimal Action-Value Function $Q^*(s,a)$: gives the expected return if, starting from *s*, you take an arbitrary action *a* and from then on you act according to the optimal policy. It obeys the **Bellman Equation**:

$$Q^*(s,a) = \mathop{\mathbb{E}}_{s' \sim P(\cdot|s,a)} [r(s,a) + \gamma \max_{a'} Q^*(s',a')]$$

Q-learning

Q-learning is a family of RL methods that try to learn $Q^*(s,a)$ function.

An **ε-greedy Q-learning** agent tries to learn $Q^*(s,a)$ function acting ε-greedily, i.e. selecting a random action a (exploration) with probability ε (that decays over time), otherwise selecting action a (exploitation) according to

$$a(s) = \arg\max_{a'} Q(s, a')$$

Q values are saved in a tabular form for each (s,a) pair, and an update is performed at each step, according to

$$Q^{new}(s,a) \longleftarrow Q(s,a) + \alpha(r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t))$$

DQN

The Deep Learning extension of RL is called **Deep Reinforcement Learning (Deep RL)**, where RL functions are approximated by Deep Neural Networks (DNNs).

The most classic and fundamental Deep RL extension of Q-learning is **DQN algorithm**:

- *Q(s,a)* is approximated by a DNN: the objective function is still based on the Bellman equation, and an **experience replay** buffer and a **target network** are used for a stable training

Algorithm 1: deep Q-learning with experience replay.

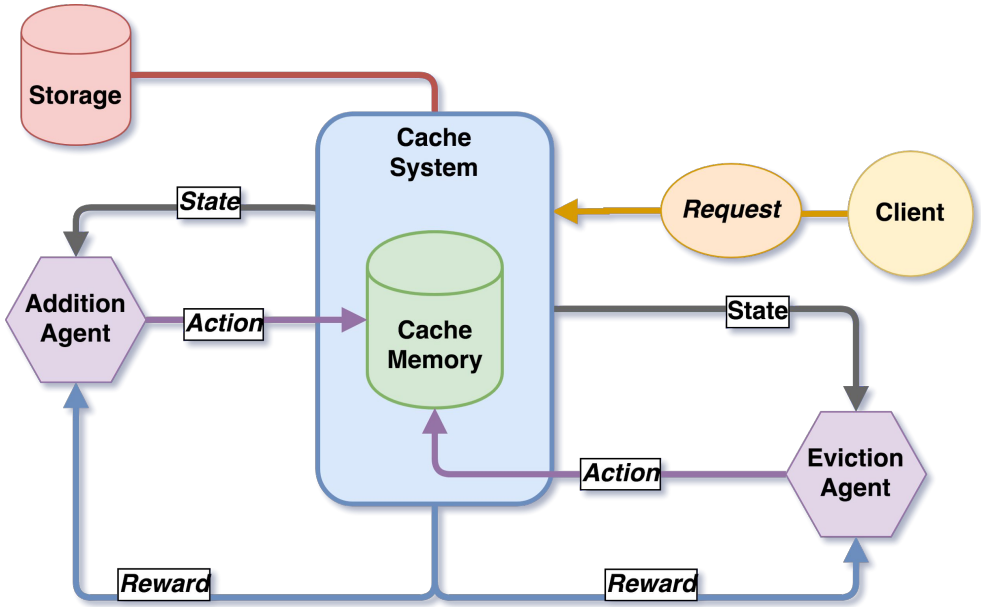
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

From Mnih, V., Kavukcuoglu, K., Silver, D. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015). https://doi.org/10.1038/nature14236

QCACHE

Two Q-learning or DQN agents:

- **Addition agent:** chooses whether to write or not a requested file
- **Eviction agent:** chooses whether to remove or not a cached file



QCACHE - SCDL and SCDL2

Simple Q-learning agents (Q-table)

*hit = requested file is in cache
**miss = requested file is not in cache

- **SCDL**[1]:
  - **Contextual Bandit** approach
  - One agent: **addition agent + LRU**
  - Positive or negative reward proportional to the **size of the requested file** according to the increase or decrease of the data read from the cache memory
- **SCDL2**[2]:
  - SCDL extension to a **full Reinforcement Learning** approach
  - One agent: **addition agent + LRU**
  - **Unitary** rewards according to the historical number of hit or miss of the file, also summing additional malus or bonus if a specific situation is verified

For both

- Agent inputs (**discretized**):
  - file statistics + cache occupancy percentage + cache hit rate
- Use of high and low **watermarks**

[1]M. Tracolli, M. Baioletti, V. Poggioni, D. Spiga, Caching suggestions using reinforcement learning, in International Conference on Machine Learning, Optimization, and Data Science, pp. 650–662, Springer, 2020

[2]M. Tracolli, M. Baioletti, V. Poggioni and D. Spiga,Effective big data caching through reinforcement learning, in2020 19th IEEE International Conference on Machine Learning and Applications(ICMLA), pp. 1499–1504, IEEE, 2020.

QCACHE - DQN

DQN agents

*hit = requested file is in cache
**miss = requested file is not in cache

- Each file request or each cached file corresponds to a specific state
- Whenever a file request comes in, addition agent chooses whether that file has to be cached or not
- Every k requests, or if the high watermark is reached, eviction agent is applied on every cached file choosing which has to be evicted or kept
- Agents inputs (**continuous**):
  - file statistics + cache occupancy percentage + cache hit rate

- ε exponential decay
- **reward is based on hit/miss information** - it is computed for both agents observing h_{window} requests after the action:
  - If action was "keep":

$$r = \begin{cases} n_{hit} \cdot size, & \text{if } n_{hit} > 0 \\ -size, & \text{otherwise} \end{cases}$$

  - If action was "don't keep":

$$r = \begin{cases} -n_{miss} \cdot size, & \text{if } n_{miss} > 0 \\ size, & \text{otherwise} \end{cases}$$

Keras for ML training (Tensorflow back-end), C++ classes exported via Pybind11 for cache environment

Evaluation metrics

In order to evaluate and compare the performance of our solution to those of the classic caching algorithms we defined three metrics.

- To evaluate the quality of a cache in fulfilling the client requests: $$throughput = \frac{readOnHitData}{cacheSize}$$
- To evaluate the operational cost of the cache: $$cost = \frac{writtenData + deletedData}{cacheSize}$$
- To evaluate the trade-off between the aforementioned metrics: $$score = throughput - cost$$

Dataset and simulation

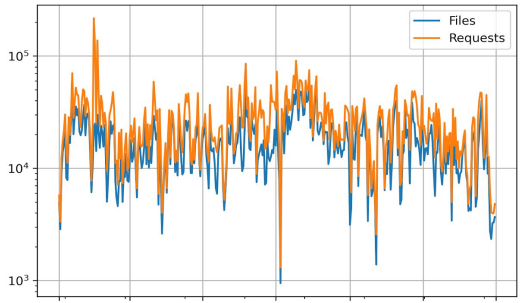
Dataset obtained from historical monitoring data about CMS experiment analysis jobs in **2018**[1,2] (data popularity) filtered for the **Italian region**: these data are used to **simulate daily file requests** flow for all 2018.

Different cache sizes are simulated:

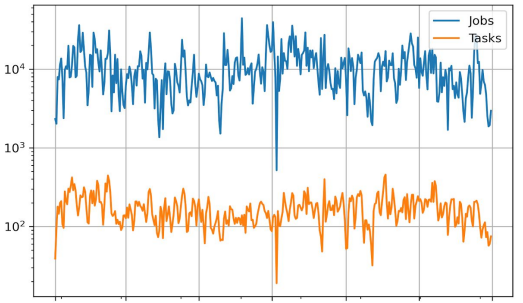
- **100TB**
- **200TB**
- **500TB**

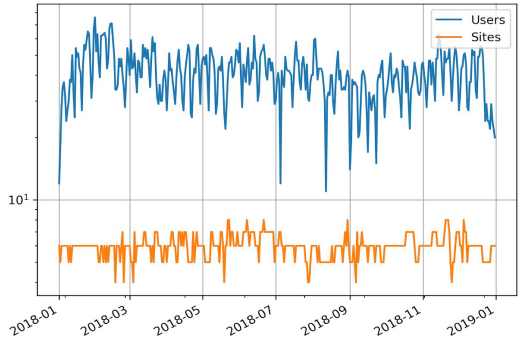
Bandwidth limit (10 Gb/s) is also considered

Daily num. of files and requests

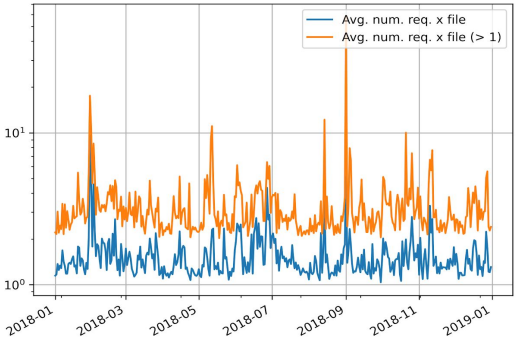


Daily num. of jobs and tasks





Daily num. of users and sites



Daily average num. of requests per file

[1]V. Kuznetsov, T. Li, L. Giommi, D. Bonacorsi, and T. Wildish. Predicting dataset popularity for the CMS experiment.arXiv preprint arXiv:1602.07226, 2016

[2]M. Meoni, R. Perego, and N. Tonellotto. Dataset popularity prediction for caching of CMS big data. Journal of Grid Computing, 16(2):211–228, 2018

DQN QCACHE implementation

DNN parameters

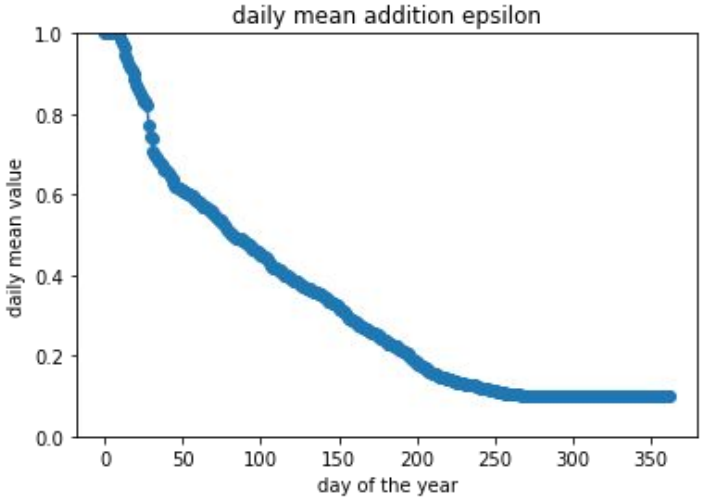
Parameter	Value
Hidden layers	2 (16 and 32 neurons, sigmoid activation)
Output layer	2 neurons, linear activation
Optimizer	Adam (lr = 0.001)
Loss function	Huber ($\delta = 1$)
Weights initialization	Glorot Uniform
Batch size	32

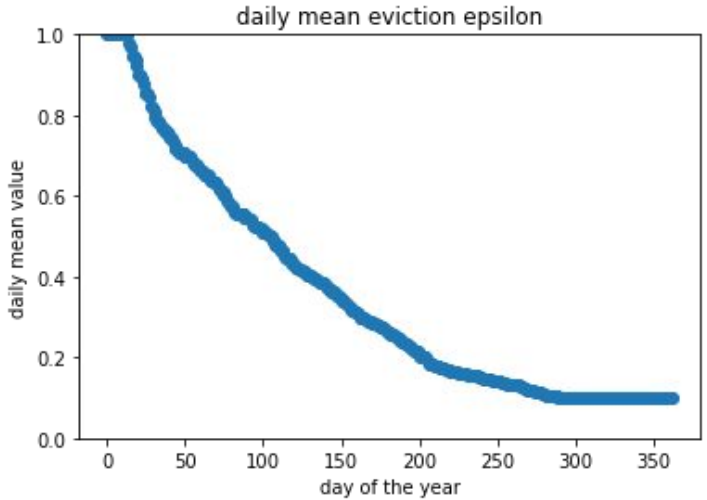
RL parameters

Parameter	Value
Replay memory	1M
γ	0.5
Addition h_{window}	100000
Addition ε decay rate	0.0000003
Addition ε starting value	1
Addition ε lower limit	0.1
Addition target model update	10000
Eviction h_{window}	200000
Eviction ε decay rate	0.0000005
Eviction ε starting value	1
Eviction ε lower limit	0.1
Eviction target model update	10000
Eviction k	50000

Training

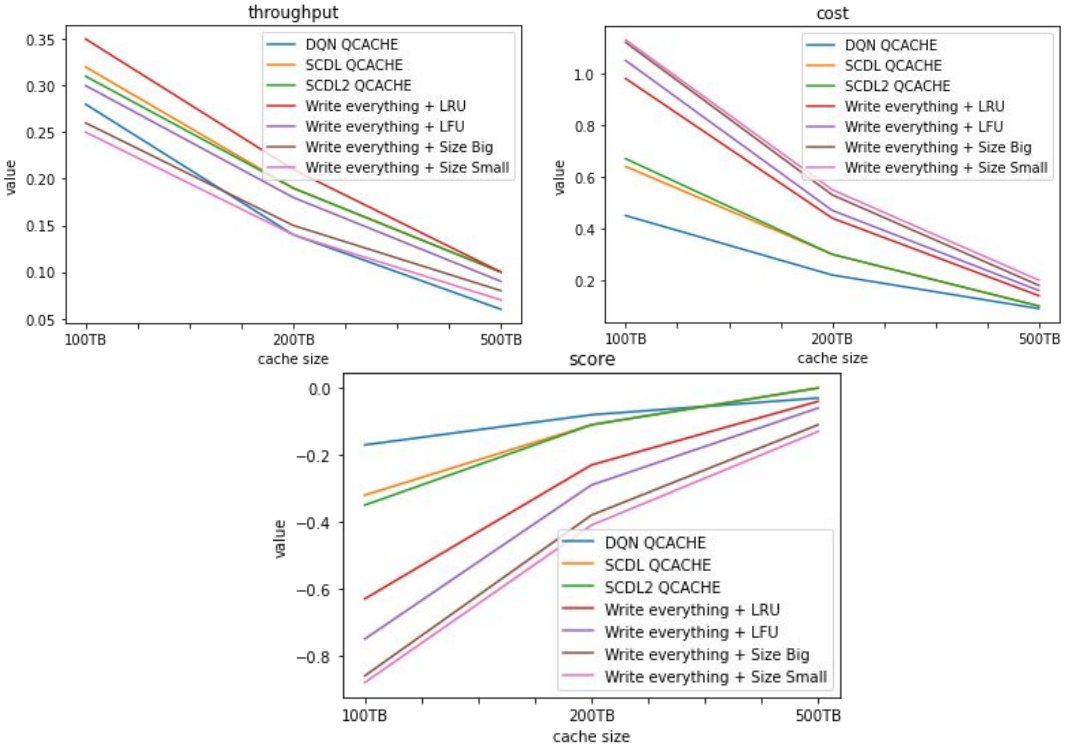
As an example, ε decay throughout the year for both DQN agents in 100TB simulation:





Results

Daily values averaged across the year



100 TB - 10 Gb/s			
Algorithm	Score	Throughput	Cost
DQN QCACHE	**-0.17**	**0.28**	**0.45**
SCDL QCACHE	-0.32	0.32	0.64
SCDL2 QCACHE	-0.35	0.31	0.67
Write everything + LRU	-0.63	0.35	0.98
Write everything + LFU	-0.75	0.30	1.05
Write everything + Size Big	-0.86	0.26	1.12
Write everything + Size Small	-0.88	0.25	1.13

200 TB - 10 Gb/s			
Algorithm	Score	Throughput	Cost
DQN QCACHE	**-0.08**	**0.14**	**0.22**
SCDL QCACHE	-0.11	0.19	0.30
SCDL2 QCACHE	-0.11	0.19	0.30
Write everything + LRU	-0.23	0.21	0.44
Write everything + LFU	-0.29	0.18	0.47
Write everything + Size Big	-0.38	0.15	0.53
Write everything + Size Small	-0.41	0.14	0.55

500 TB - 10 Gb/s			
Algorithm	Score	Throughput	Cost
SCDL2 QCACHE	0.00	0.10	0.10
SCDL QCACHE	0.00	0.10	0.10
DQN QCACHE	**-0.03**	**0.06**	**0.09**
Write everything + LRU	-0.04	0.10	0.14
Write everything + LFU	-0.06	0.09	0.16
Write everything + Size Big	-0.11	0.08	0.18
Write everything + Size Small	-0.13	0.07	0.20

Conclusions and next steps

DQN has a better score on small-sized caches:

- Maybe, the 500TB cache could be above the actually useful size threshold: probably a whole file set, considered as all accessed files during a specific period, fits in it.
  - As a result, the smartness-related advantages are quite low with the biggest size used in the simulations.
- DQN parameters have been optimized for 100TB cache

These results show that a smart caching approach is able to **reduce cost** while keeping **throughput at a reasonable value** at simulation level (where delete and write are two "zero-time" operations) → We expect that a **lower cost has a big impact in a real environment** where fewer file deletion and writing operations imply better performances in terms of files serving.

Next steps:

- Optimizing parameters for various cache sizes
- Enhancing reward-giving process
- Inserting domain features
- Testing on US data
- **Working on testbed setup to test in a real environment**