# Smart caching a CMS: verso un modello basato su AI

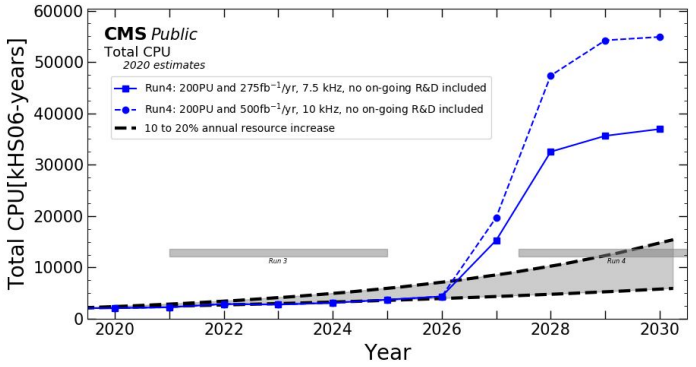
Tommaso Tedeschi[1,2], Mirco Tracolli[1,2,3], Daniele Spiga[2], Loriano Storchi[2], Diego Ciangottini[2]
per la collaborazione CMS

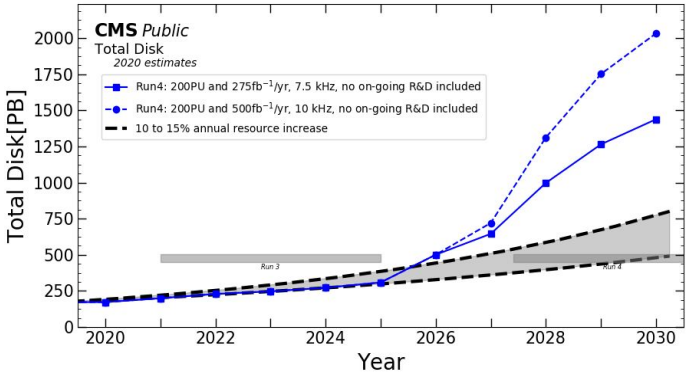
[1] Università degli Studi di Perugia
[2] Istituto Nazionale di Fisica Nucleare - Sezione di Perugia
[3] Università degli Studi di Firenze

# Il futuro di LHC e CMS

Nel prossimo futuro LHC entrerà nella fase di alta luminosità:

- Le richieste dei singoli esperimenti in termini di risorse di calcolo e storage supereranno quelle che possono essere soddisfatte con le tecnologie odierne
- Necessità di un nuovo modello che ottimizzi l'uso dell'hardware e i costi operazionali

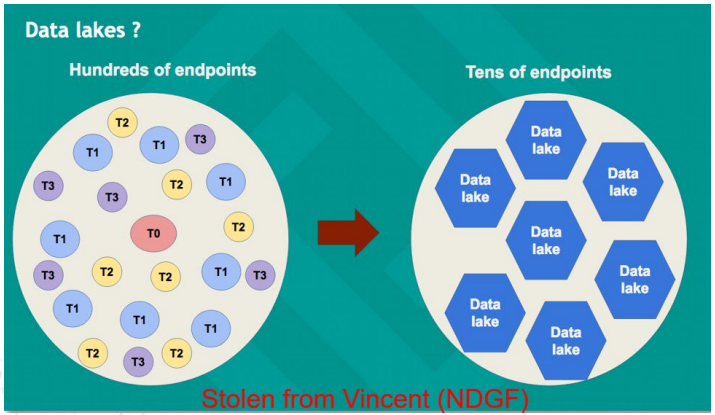


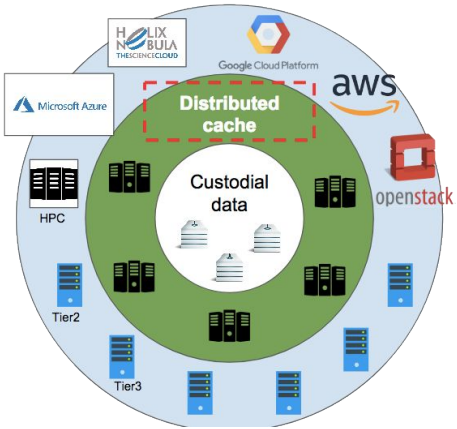



# Data Lake e cache

Soluzione Data Lake: *pochi endpoint di storage con uso intensivo di sistemi di cache per i nodi interni e con una riduzione significativa dei costi di storage.*

- Il sistema di cache deve essere un oggetto trasparente che interagisce con i client ottimizzando il traffico verso i centri di storage (custodial). Una gestione ottimale del sistema permette di: **ridurre** l'uso di banda, spazio-disco e tempi di calcolo; **migliorare** l'esperienza di analisi dati; **contenere** i costi di manutenzione

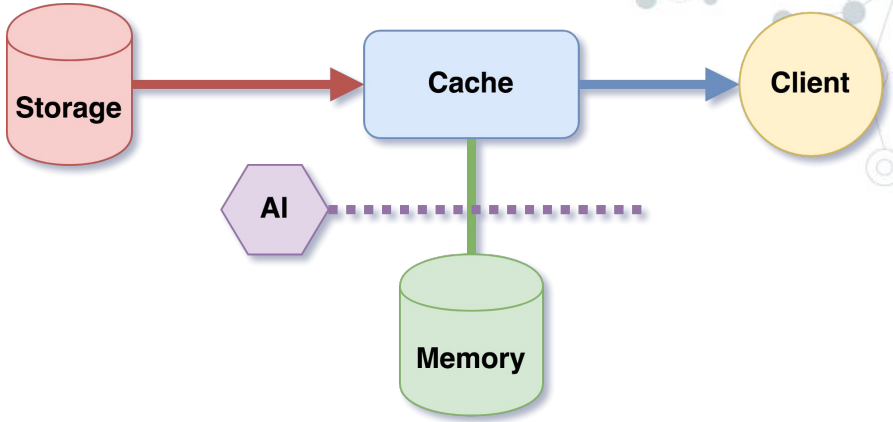


# Approccio AI

Sviluppo di un **sistema di cache intelligente**, utilizzando algoritmi di Intelligenza Artificiale (AI):

- La AI manipola direttamente la memoria della cache, decidendo cosa scrivere o cancellare
- Lo scopo è ottenere un algoritmo utilizzi **meno risorse di storage** degli algoritmi di caching classici (LRU, LFU ecc..) **a parità di efficienza**




L'approccio utilizzato è basato su **Reinforcement Learning (RL)**: settore del Machine Learning che studia **agenti** che apprendono attraverso un processo "**trial and error**" interagendo con un ambiente.

# RL e Q-Learning

*Ad ogni step, l'agente riceve in input dall'ambiente un determinato stato s e sceglie una determinata azione a, ottenendo una ricompensa (punizione) r dall'ambiente stesso, che viene utilizzata per aggiornare l'algoritmo*

Il termine Q-Learning[1] indica una particolare tipologia di algoritmi di RL che hanno lo scopo di **massimizzare la ricompensa totale,** associando a ciascuna coppia *(s,a)* un valore *Q(s,a)*. Dato uno stato *s*, l'azione *a(s)* ottimale è scelta come:

$$a(s) = \arg\max_a Q(s,a)$$

**Q-Table:** I valori *Q(s,a)* sono conservati in una tabella (gli stati *{s}* e le azioni *{a}* possibili sono in numero finito). Ad ogni step:

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

**Deep Q-Network (DQN):** *Q(s,a)* è approssimata da una Deep Neural Network. Ad ogni step:

$$Loss = (r + \gamma \max_{a'} Q(s',a';\theta) - Q(s,a;\theta))^2$$

[1]Sutton, R. & Barto, A. *Reinforcement Learning: An Introduction* (MIT Press, 1998)

# QCache

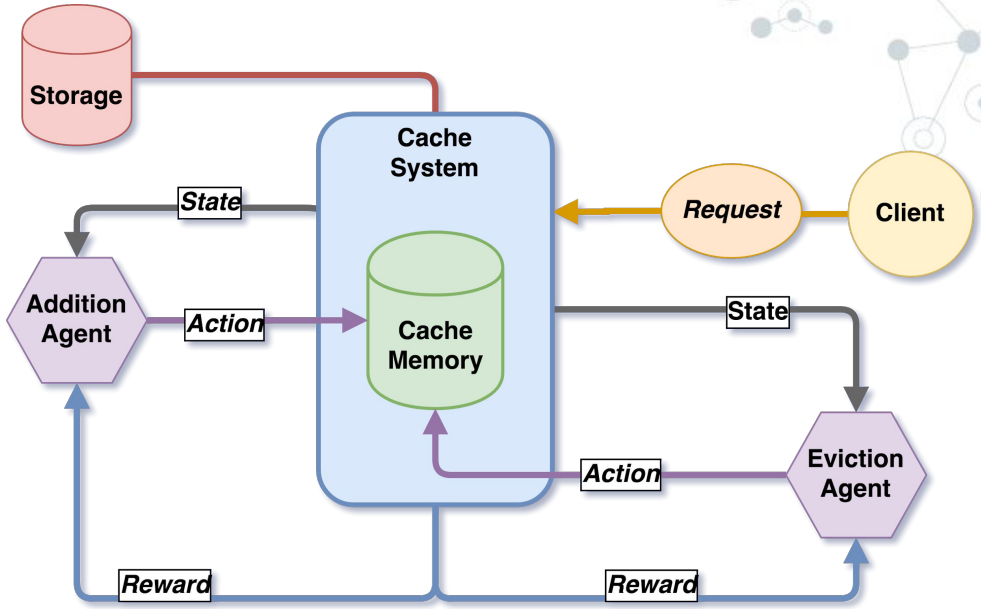
Due agenti Q-Learning (Q-Tables o DQN):

- **Addition agent:** sceglie se salvare in cache o meno un file richiesto
- **Eviction agent:** sceglie se cancellare o meno un file presente in cache

A ciascuna *richiesta* in arrivo e a ciascun *file in cache* corrisponde uno *stato differente*.

Ad ogni richiesta in arrivo che non sia per un file già in cache, l'addition agent sceglie se memorizzare o meno il file in cache.

Ogni *k_eviction* richieste (o se la cache è piena) l'eviction agent viene applicato sui file presenti in cache in quel momento e sceglie per ciascuno di essi se va eliminato o mantenuto in cache.



# Deep QCache

## Deep Q-learning[1]

- ε-greedy policy: ad ogni step vi è una probabilità ε (che decade esponenzialmente nel tempo) che la scelta sia presa a caso, per favorire l'esplorazione di nuove azioni
- ricompensa (reward): calcolata per entrambi gli agenti osservando le *time_span* richieste successive all'azione:
  - Se l'azione è stata *keep*:
    - Se *n > 0* hit* relative a tale file: *reward = n * size*
    - Se no hit: *reward = - size*
  - Se l'azione è stata *don't keep*:
    - Se *n > 0* miss* relative a tale file: *reward = - n * size*
    - Se no miss: *reward = size*
- experience replay: i vettori *(s,a,r,s')* sono salvati in memoria e utilizzati per l'aggiornamento dei pesi della rete in batch

## Parametri DNN (per entrambi gli agenti)

- 2 hidden layer (16, 32 neuroni, attivazione sigmoide, output layer 2 neuroni con attivazione lineare)
- Adam optimizer (*learning rate = 0.001*)
- Huber loss (*delta = 1*)
- Glorot Uniform

*Training implementato con **Keras** (**Tensorflow** back-end) e ambiente cache implementato con classi C++ esportate in Python tramite **Pybind11***

**hit* = file richiesto presente in cache
**miss* = file richiesto non presente in cache

[1]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

# Input e metriche

## VARIABILI IN INPUT

- ***FILE SIZE*: dimensione del file richiesto
- ***FILE FREQUENCY*: numero totale di richieste per quel file dall'inizio della simulazione
- ***FILE RECENCY*: numero di richieste passate dall'ultima richiesta di tale file
- ***FILE DATATYPE*: flag indicante dati o MC
- ***CACHE OCCUPANCY*: riempimento attuale della cache
- ***CACHE HIT RATE*: hit rate giornaliero attuale della cache

## METRICHE (valori giornalieri)

- **HIT RATE**: `#hit / (#hit + #miss)`
- ***READ ON HIT RATIO*:** `read_on_hit_data / read_data`
- ***BAND SATURATION***: `read_on_miss_data / daily_bandwidth`
- ***THROUGHPUT***: `(read_on_hit_data - written_data) / cache_size`
- ***CACHE COST***: `(written_data + deleted_data) / cache_size`
- ***FREE SPACE***: `(cache_size - mean_daily_size) / cache_size`

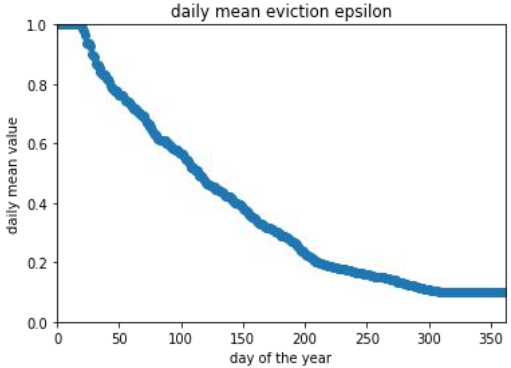
# Dataset e apprendimento

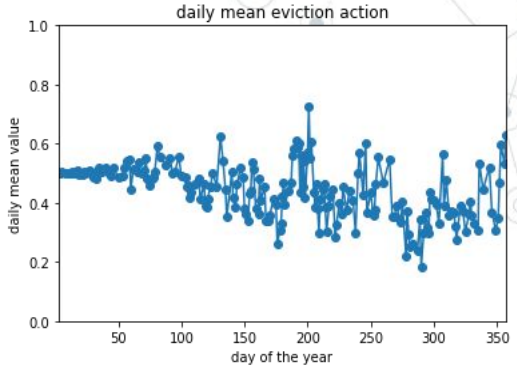
Dataset creato a partire dai dati storici aggregati[1,2] (sulle richieste di analisi dati CRAB dell'anno 2018) usati per predire la popolarità dei dataset, filtrati per la regione **ITALIA**

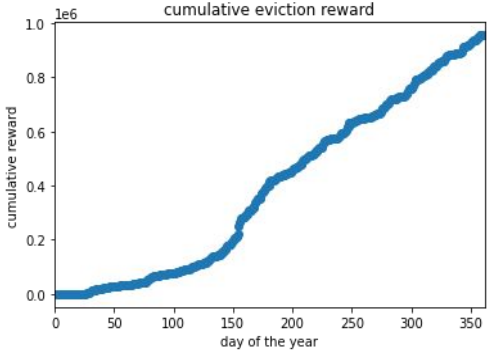
Primi test:

- *Cache size: 100T*
- *time_span addition: 100000*
- *time_span eviction: 200000*
- *k_eviction: 30000*
- *Ad ogni step non vengono considerati reward o stati futuri ma solo quelli attuali*

ESEMPIO: eviction agent




[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

# Conclusioni e passi futuri

*La fase di training sembra produrre degli agenti che si comportano come da specifiche: apprendono come massimizzare il reward ad ogni step.*

Passi futuri:

- Confronto tra i valori delle metriche ottenuti con questo approccio e quelli ottenuti con algoritmi classici di gestione cache (LRU, LFU, ecc..) e con approcci QTable

- Ottimizzazione dei parametri