PANDA
Parallel, Networking and Distributed Applications

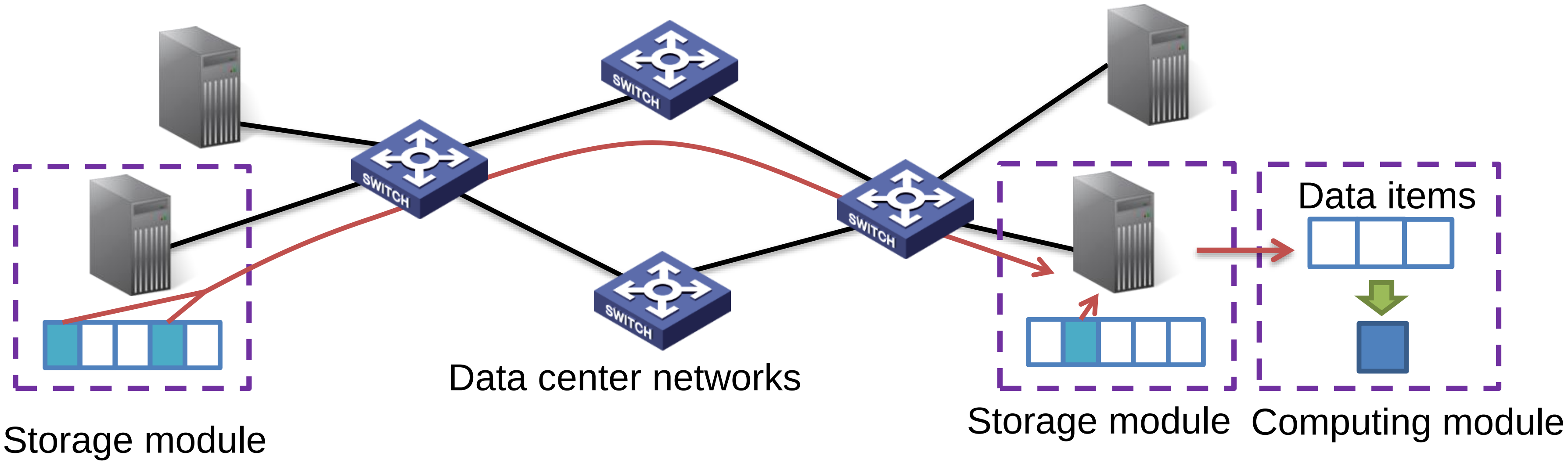
Learning-based Adaptive Data Placement for Low Latency in Data Center Networks

Kaiyang Liu, Jingrong Wang, Zhuofan Liao,

Boyang Yu, Jianping Pan

Data Analytics Services

- Data-intensive applications
  - Data items need to be moved **frequently** between storage nodes
  - This introduces **increased** and **changing** data access latency



- Data placement problem arises

Data Placement

- Data storage
  - Storage locations affect the finish time of the distributed computation tasks
  - Main bottleneck: data movement latency [1]
  - Amazon and Google reported that a slight increase in delay will lead to observable fewer user accesses
- Data movement latency
  - Network latency = Processing + Queuing + Transmission + Propagation latency

  - Different factors contribute to the latency

[1] M.Y. Eltabakh, et al., "CoHadoop: Flexible data placement and its exploitation in Hadoop," in Proc. VLDB Endow., 2011.

Related Work

- **Existing research efforts** [2]—[5]
  - Analyzing the factors that may affect the network latency
  - Hand-crafted design of optimization models
- **Limitations**: not flexible enough to deal with a dynamic environment
  - Different latency factors could be time-variant
  - Many uncertainties
    - Unreliable network links, variable user request patterns, and evolving system configurations

[2] Y. Xiang, et al., "Joint latency and cost optimization for erasure-coded data center storage," IEEE/ACM Trans. Netw., 2016.
[3] X. Ren, et al., "Datum: Managing data purchasing and data placement in a geo-distributed data market," IEEE/ACM Trans. Netw., 2018.
[4] B. Yu, et al., "A framework of hypergraph-based data placement among geo-distributed datacenters," IEEE Trans. Serv. Comput., 2017.
[5] Y. Hu, et al., "Latency reduction and load balancing in coded storage systems," in Proc. of ACM SoCC, 2017.

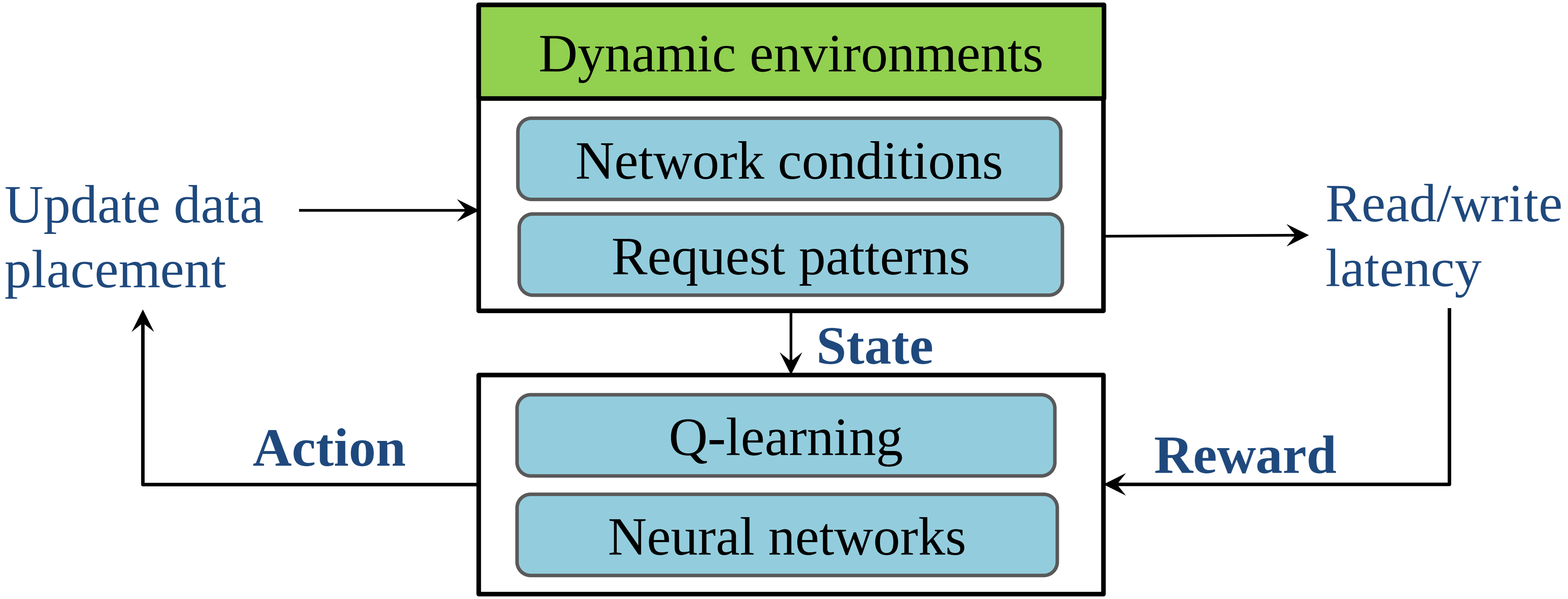
Key Questions

- **Data placement problem**: how to choose the storage locations of data items for low latency?

- **Challenges**
  - **Adaptability:** online schemes to deal with the network uncertainties
  - **Easy Implementation**
    - Low overhead
    - No need to modify the existing storage architecture

DataBot: A Learning-based Solution

- Design overview



Q: States($\mathcal{S}$) × Action($\mathcal{A}$) = Reward($\mathcal{R}$)

Q-learning based Data Placement

- Data placement can be treated as a finite Markov Decision Process (**FMDP**)
  - The number of storage nodes is finite
  - Each action of data placement is independent
  - The performance of placement only depends on the current states and decisions
- The model-free *Q*-learning can find an optimal action selection policy for any given FMDP [6]

[6] L.P. Kaelbling, et al., “Reinforcement learning: A survey,” J. Artif. Intell. Res., 1996.

PANDA
Parallel, Networking and Distributed Applications

Our Contributions

- Solutions to address the limitations of conventional Q-learning

Curse of Dimensionality

- Numerous data items
- Many servers (storage locations)

- Large state-action space
- Slow convergence

+ Q-function design and approximation

Extra Overheads

- Recurrent training and model updating
- Time consuming

- No instantaneous decisions
- Long access latency

+ Efficient asynchronous implementation

Q-Function Design (1)

➢ States ($\mathcal{S}$)

a) Network conditions: $\{\underbrace{L_{ij}^{[\mathrm{R}]}, L_{ij}^{[\mathrm{W}]}}_{\boxed{\text{Average read/write latencies}}}, \underbrace{\forall i, j \in \mathcal{N}}_{\boxed{\text{Set of servers}}}\}$

Exponentially Weighted Moving Average (EWMA) mechanism [7]:

$$L_{ij}^{[\mathrm{R/W}]} = \alpha_l \underbrace{l}_{\boxed{\text{Measured latency for each data movement}}} + (1 - \underbrace{\alpha_l}_{\boxed{\text{Discount factor}}}) L_{ij}^{[\mathrm{R/W}]}$$

Benefits of EWMA: it only needs *O*(1) space for latency estimation

[7] J.S. Hunter, "The exponentially weighted moving average," J. Qual. Technol., 1986.

Q-Function Design (2)

- States ($\mathcal{S}$)
  b) Request Patterns: $\{\underbrace{F_{im}^{[R]}, F_{im}^{[W]}}, \underbrace{\tilde{F}_i^{[R]}, \tilde{F}_i^{[W]}}, \forall i \in \mathcal{N}\}$

Read/write rates to data m from i

Total read/write rates from server i

Discounting Rate Estimator (DRE) method [8]

- Maintains a counter for each item
- Increases with every read/write operation
- Decreases periodically

Benefits of DRE: 1) it reacts quickly to the changes, and 2) only needs $O(1)$ space

[8] M. Alizadeh, et al., "Conga: Distributed congestion-aware load balancing for datacenters," in Proc. of ACM SIGCOMM, 2014.

Q-Function Design (3)

- States ($\mathcal{S}$)
  - c) Source location: 0-1 vector

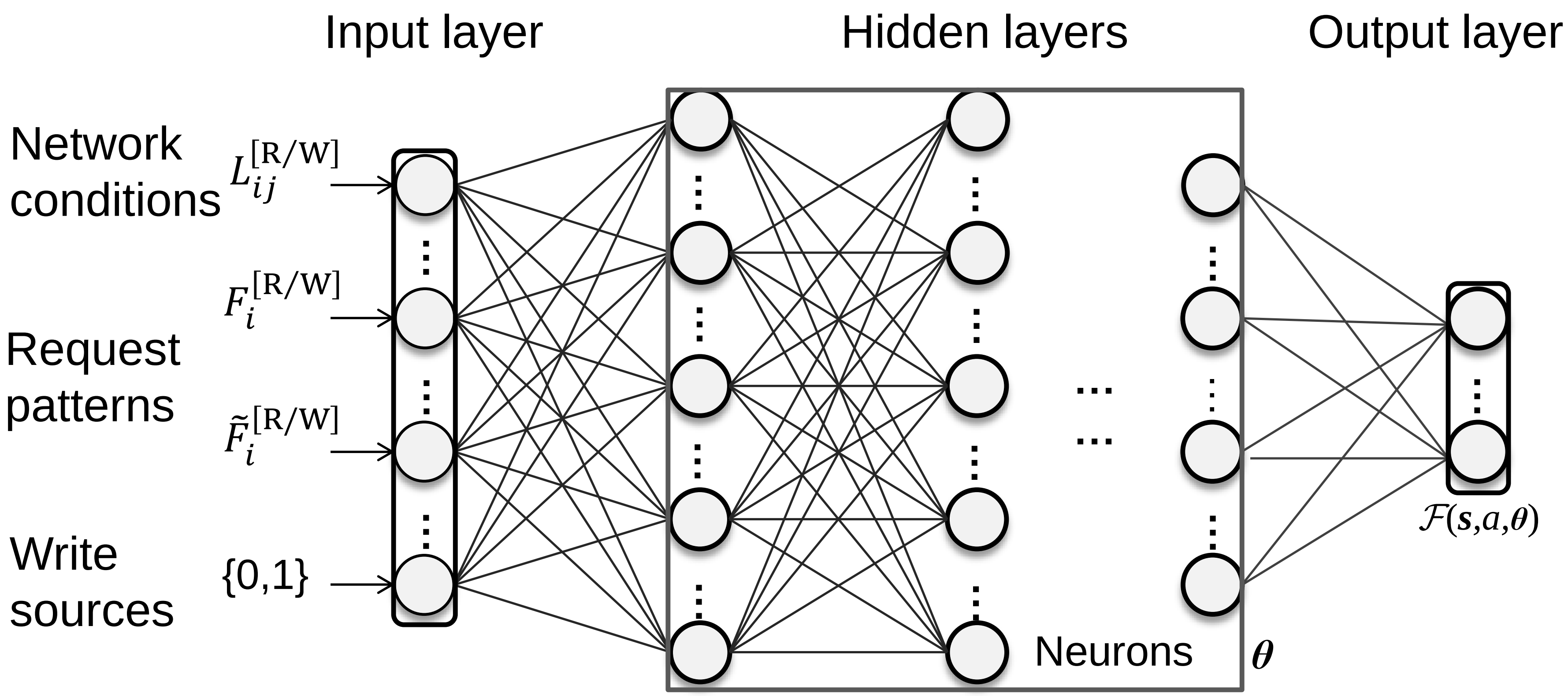
  The size of state $\boldsymbol{s}$ will be: $|\boldsymbol{s}| = 2N^2 + 5N = O(\underline{N}^2) \longrightarrow$ Number of servers

  - The **number of data items** will not affect the deployment complexity
- Actions ($\mathcal{A}$): 0-1 vector (storage locations)
- Rewards ($\mathcal{R}$): $r_t = \underset{\text{Tradeoff parameter}}{\underline{\omega}} \cdot \frac{1}{l^{[\mathrm{W}]}} + (1-\omega) \cdot \frac{1}{|\underset{\text{\# of read operations between two write operations}}{\underline{\mathcal{P}}}|} \cdot \sum_{p \in \mathcal{P}} \frac{1}{l_p^{[\mathrm{R}]}}$

The **measured** read/write latencies are used as the reward

Q-function Approximation

- Neural networks (NN)
  - Learn to output the expected rewards of data placement actions
  - Lower the scale of the state space (**number of servers**)

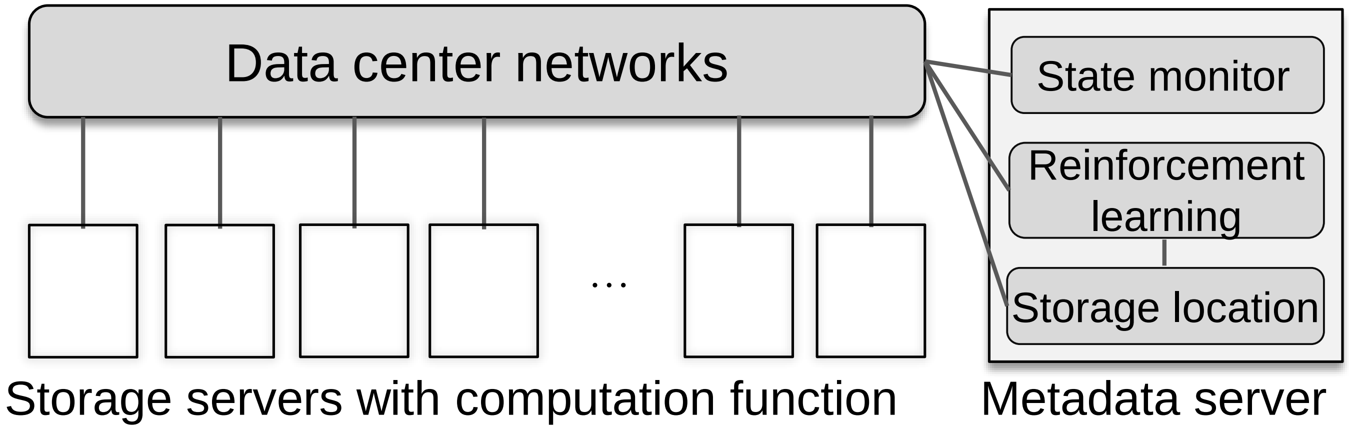


PANDA
Parallel, Networking and Distributed Applications

System Architecture

➢ Data storage architecture

- DataBot is implemented in the metadata server

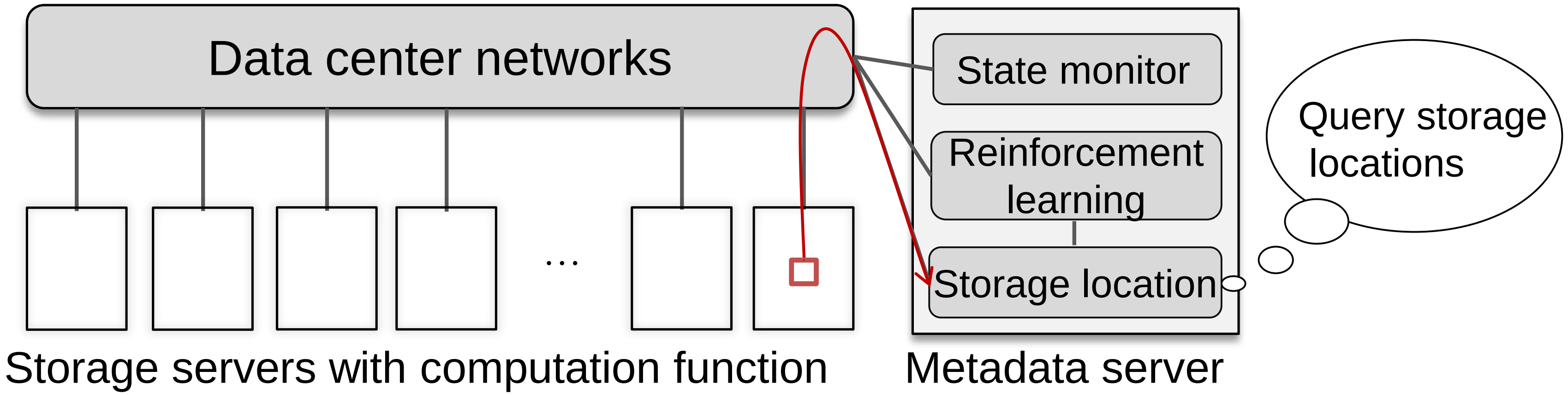


➢ Metadata server

- Manages the storage locations of data items, e.g., using hashtag
- Captures the **logs** of the read/write requests: (**TS**, **R/W**, **Src**, **Dst**, **Lat**)

System Architecture

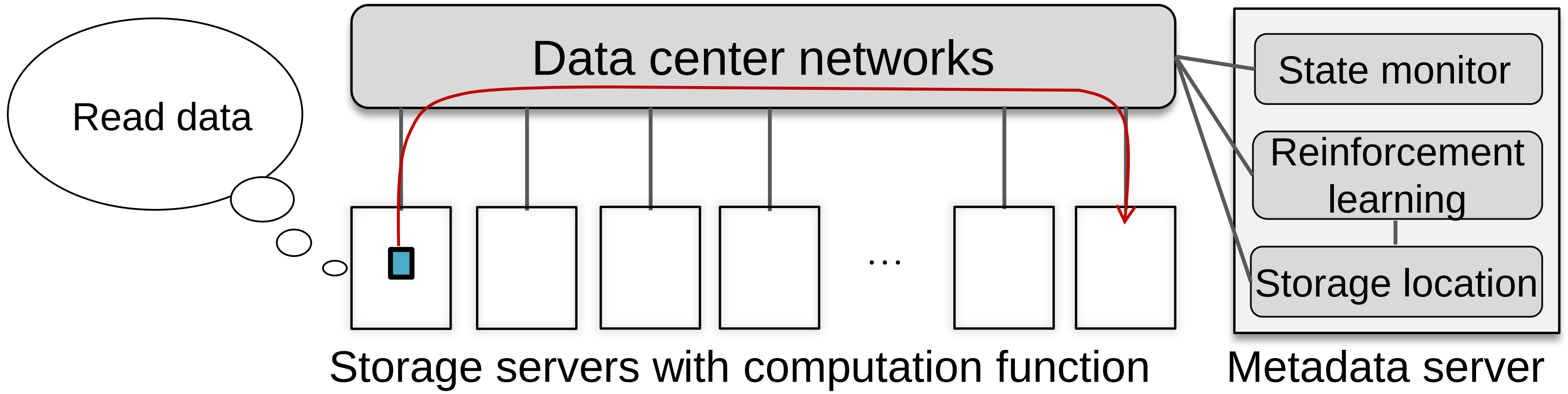
- Data storage architecture
  - DataBot is implemented in the metadata server



- Metadata server
  - Manages the storage locations of data items, e.g., using hashtag
  - Captures the **logs** of the read/write requests: (**TS**, **R/W**, **Src**, **Dst**, **Lat**)

System Architecture

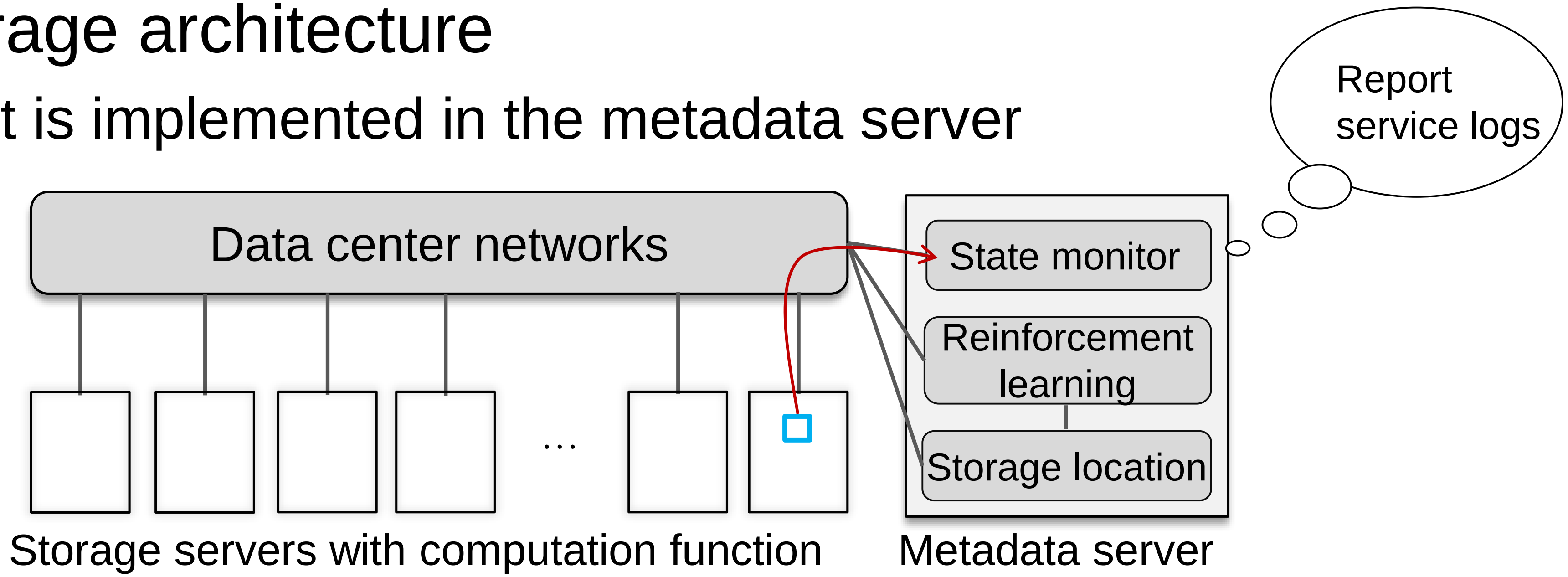
- Data storage architecture
  - DataBot is implemented in the metadata server



- Metadata server
  - Manages the storage locations of data items, e.g., using hashtag
  - Captures the **logs** of the read/write requests: (**TS**, **R/W**, **Src**, **Dst**, **Lat**)

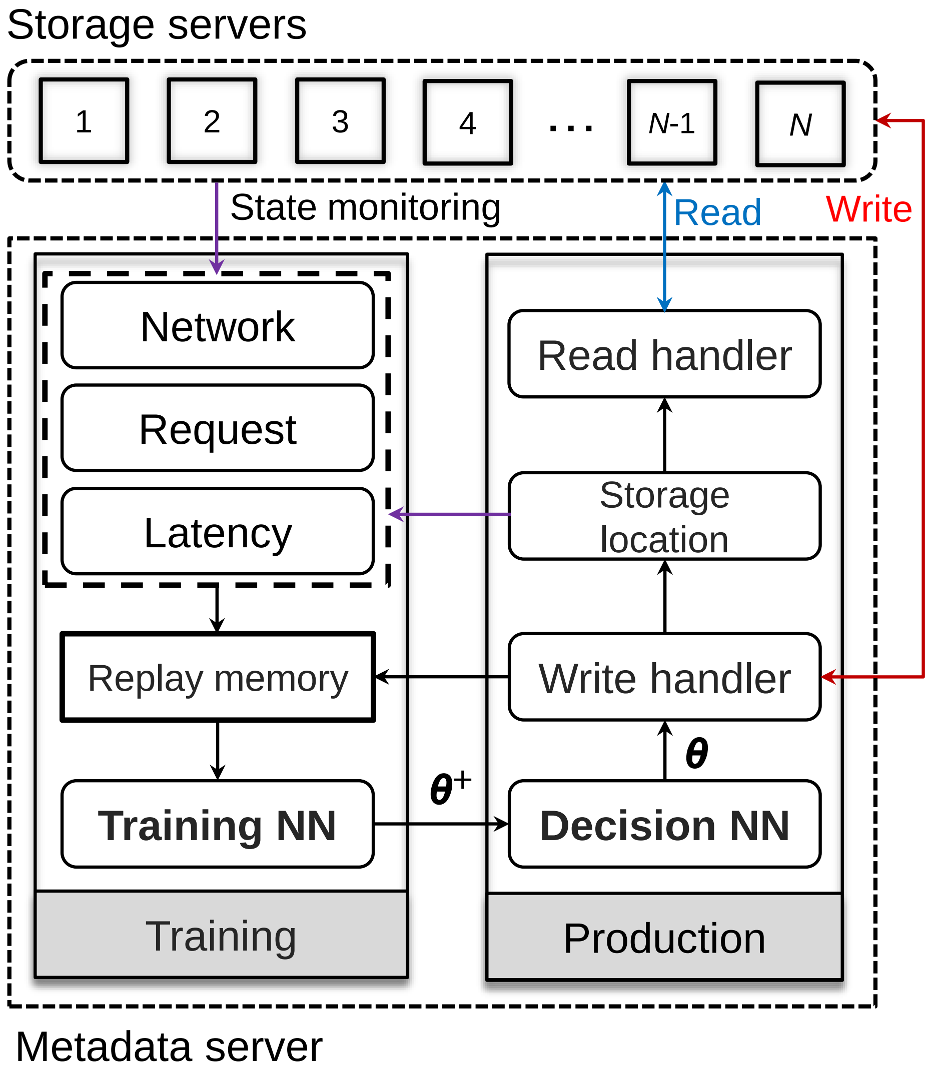
Asynchronous Implementation (1)

- Data storage architecture
  - DataBot is implemented in the metadata server



- Metadata server
  - Manages the storage locations of data items, e.g., using hashtag
  - Captures the **logs** of the read/write requests: (**TS**, **R/W**, **Src**, **Dst**, **Lat**)

Asynchronous Implementation



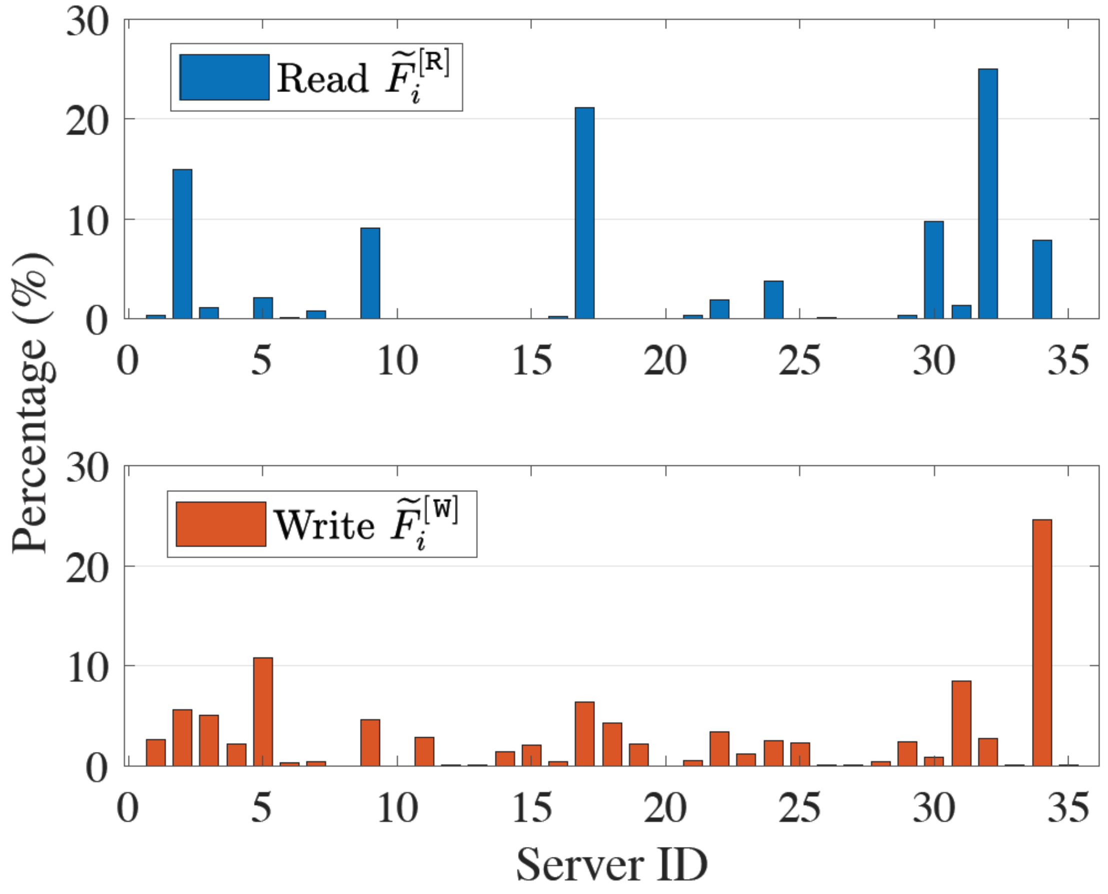
- Production system: decision NN
  - **Input**: state $\boldsymbol{s}_t$; **Output**: expected reward $\mathcal{F}(\boldsymbol{s}_t, a_t, \boldsymbol{\theta})$
  - **ε-greedy method**: with probability ε to select the action a_t that maximizes the output value
  - A tuple $\tau = (\boldsymbol{s}_t, a_t, \boldsymbol{s}_{t+1}, r_t)$ is stored for each request
- Training system: training NN
  - Tuples for a period → Replay memory $\mathcal{R}$ for training
  - Mini-batch stochastic gradient descent (SGD) [9]: training weight vector $\boldsymbol{\theta}^+$
    - **Batch**: all tuples in $\mathcal{R}$ are partitioned into mini-batches
    - **Epoch**: mini-batches are trained with multiple iterations
  - Weight update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^+$

[9] V. Mnih, et al., "Human-level control through deep reinforcement learning," Nature, 2015.

Performance Evaluation

Evaluation Setup: Traces

- **MSR Cambridge Traces** [10]
  - I/O traces of an enterprise data center
  - Hostname, request type (read/write), and timestamp
  - Request distribution is **biased** among 36 storage servers
- **Limitation**: do not specify the detailed data item for each read/write request
- **Assumption**
  - Number of data items: 10,000
  - The request rates of data items follow a Zipf distribution among servers

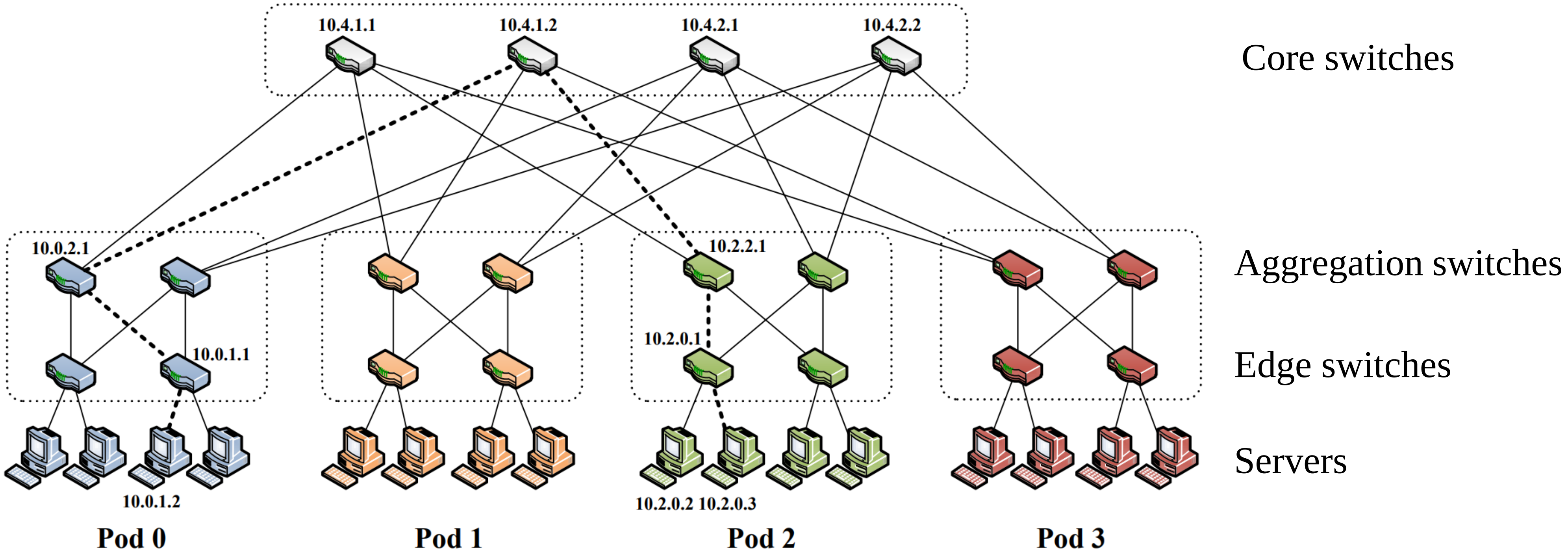


Arrival rates of the read/write requests

[10] D. Narayanan, et al., "Write off-loading: Practical power management for enterprise storage," ACM Trans. Storage, 2008.

Evaluation Setup: Scenarios (1)

- Data center network emulation: **Mininet**
  - Representative network topology: Fat-Tree [11]; Link capacity: 1 Gbps
  - Default data block size: 64 MB [12]



[11] M. Al-Fares, et al., " A scalable, commodity data center network architecture," in Proc. of ACM SIGCOMM, 2008.
[12] "HDFS Architecture Guide." [Online]: https://hadoop.apache.org/

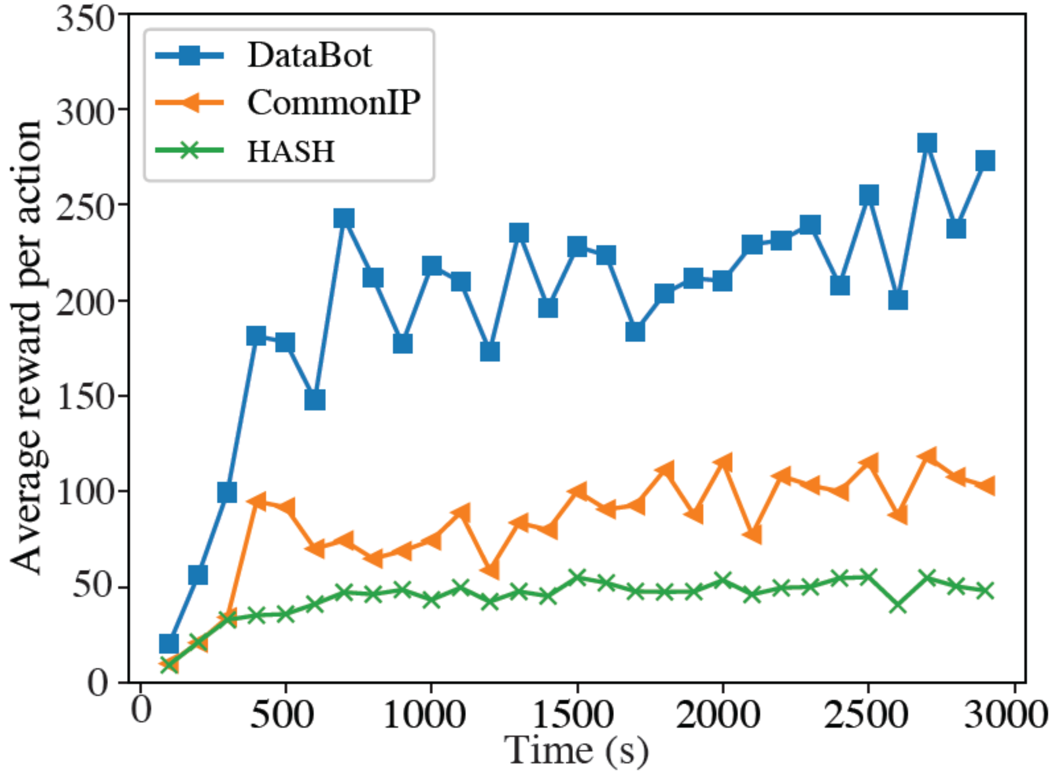
Evaluation Setup: Scenarios (2)

- Client program: Initiates the read/write requests
  - Memcached: the end of data flows for data caching in RAM
- Metadata server program
  - State monitoring
  - Write destination decision
  - NN training: Multilayer perceptron with one kernel
- Performance baselines
  - **HASH** [11] hashes data to servers for load-balancing
  - **CommonIP** [12] places data as close as possible to the IP address that most commonly accesses the data under the constraint of storage capacity

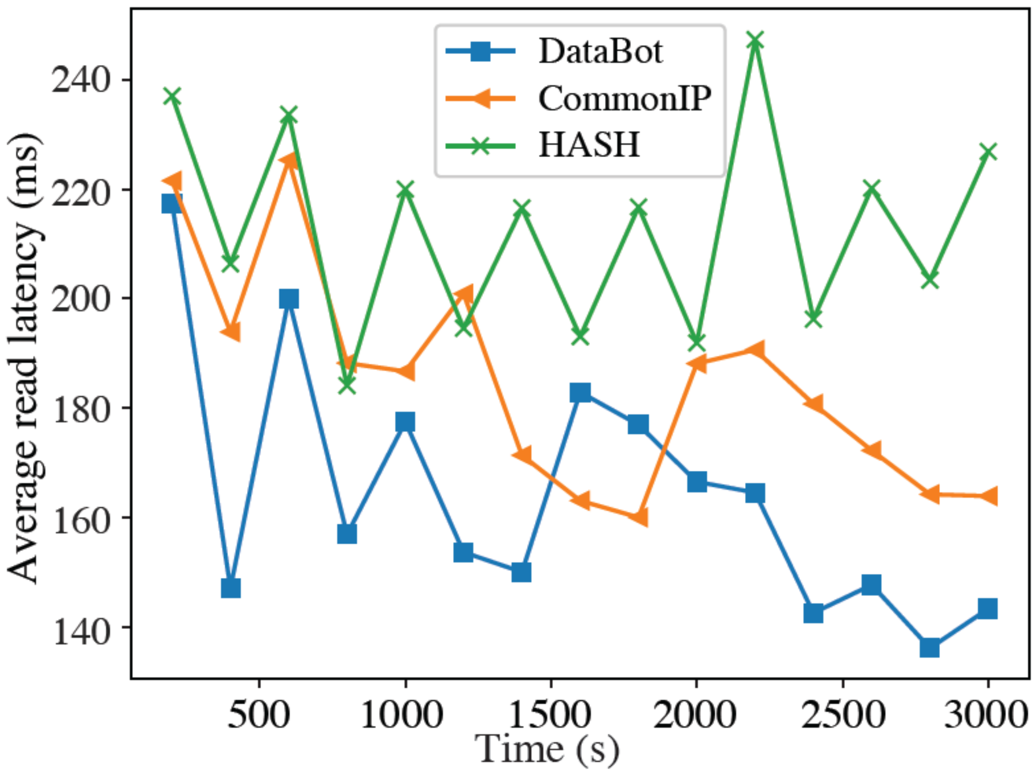
[11] "HDFS Architecture Guide." [Online]: https://hadoop.apache.org/
[12] S. Agarwal, et al., "Volley: Automated data placement for geo-distributed cloud services," in Proc. of USENIX NSDI, 2010.

Results: Read optimized

- Read optimized: write weight ω = 0.2



(a) Average reward per action r_t



(b) Average read latency

Results: Other Factors (1)

➢ Parameter impacts: write weight and number of replicas

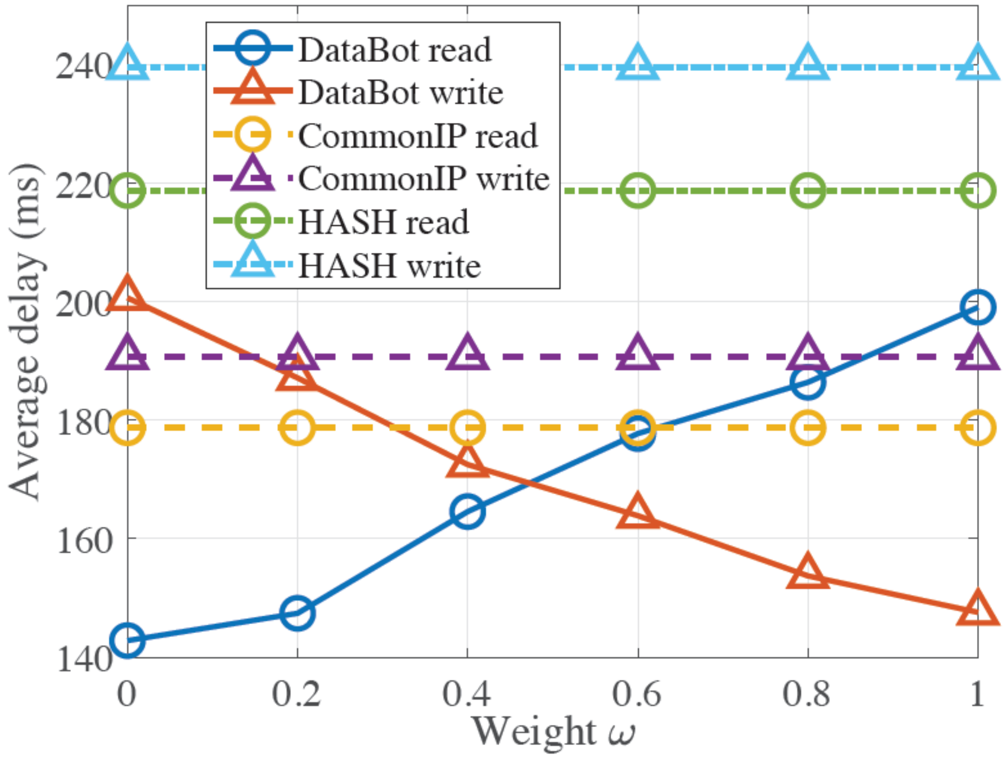


Fig. 7. Impact of weight ω.

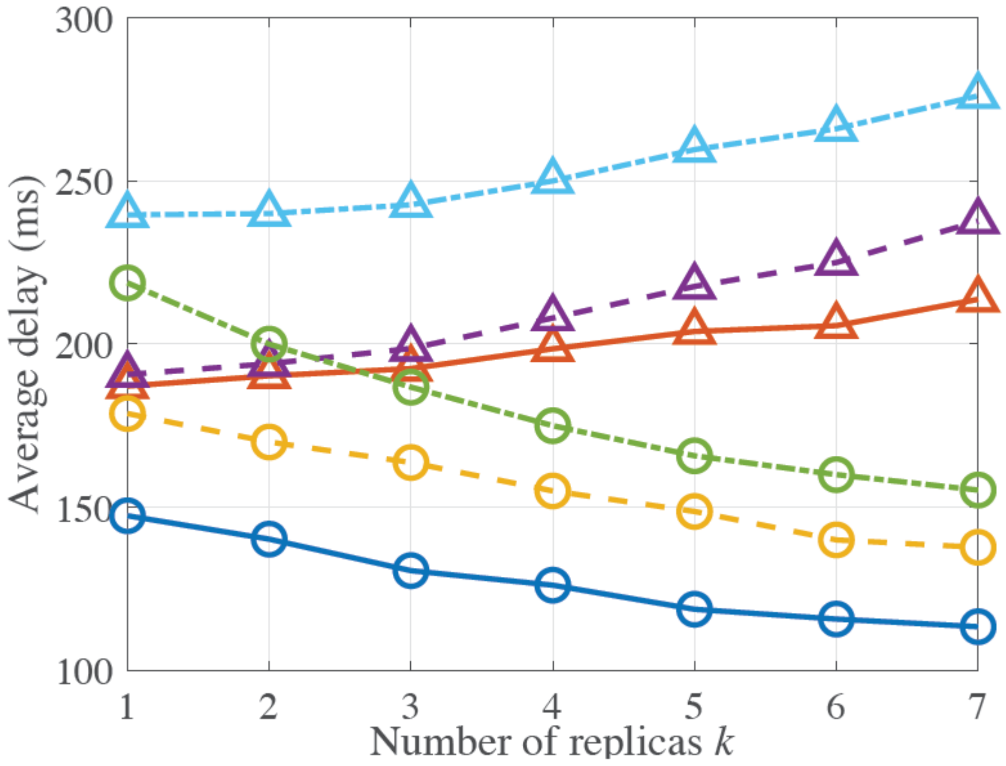


Fig. 8. Impact of replica number k.

Results: Other Factors (2)

- Parameter impacts: number of training epochs and batch size

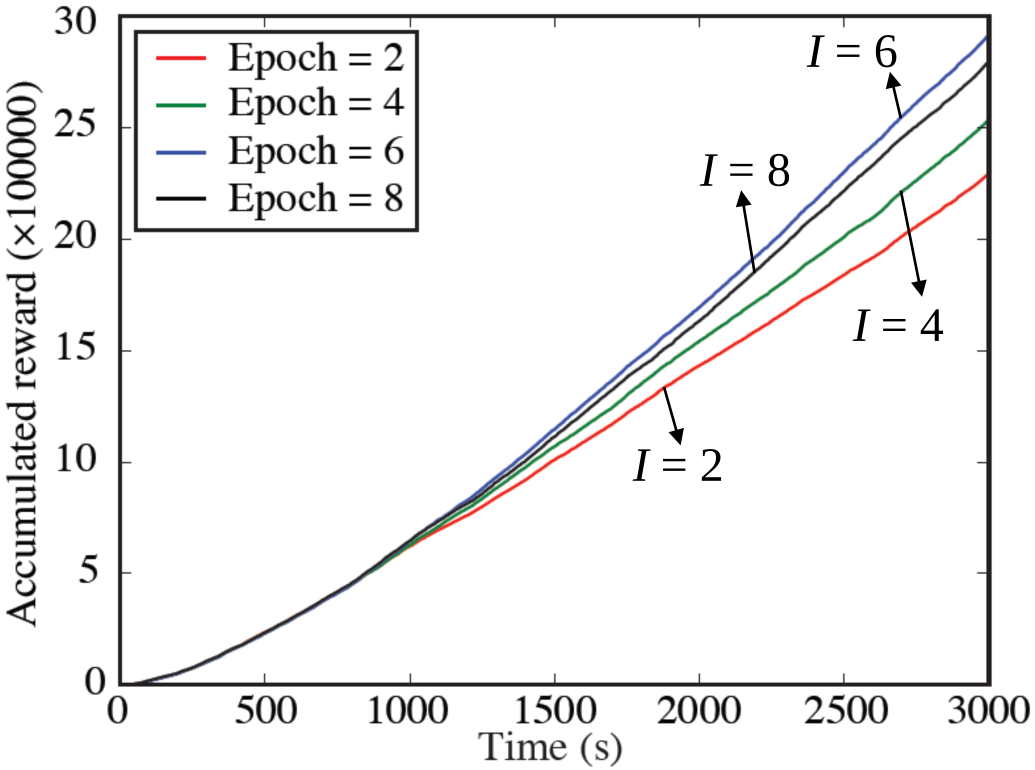


Fig. 9. Impact of epoch number I.

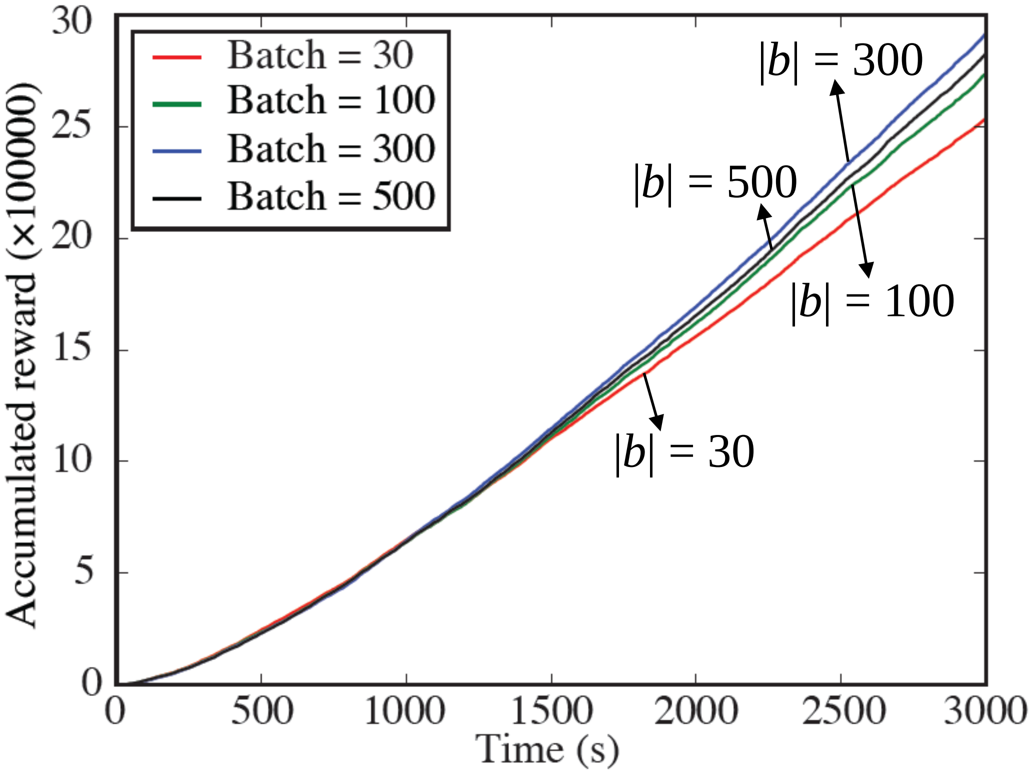


Fig. 10. Impact of batch size $|b|$.

Conclusions

- **DataBot** automatically learns the optimal data placement policies to handle the **system uncertainties**
  - With no future information about the **dynamics**
- Neural networks achieve a quick approximation when combined with Q-learning
- Asynchronous implementation
  - Online decision making and offline training

Thanks

Happy to answer your questions

Backup Slides

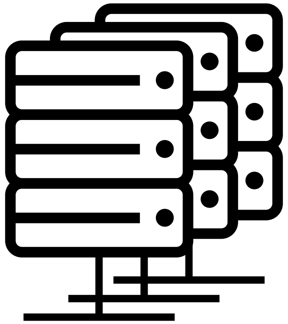
Neural Networks

- Structure of NN
  - Multilayer perceptron (MLP) with one kernel
  - **Input layer**: 1,476 features; **Output layer**: 36 features
  - **Two hidden layers**: 1,000 and 600 features
- Weight vector training
  - Traditional back propagation method
- Implementation based on popular learning frameworks
  - Keras deep learning library [14] (with TensorFlow as backend)

[14] "Keras: Deep learning library" [Online]. Available at: https://keras.io/.

Scalability

- ➢How to improve the scalability when the serves are deployed on a large scale?
  - Hundreds or thousands of servers



- ➢Our solution in the future work

Distributed learning mechanism

- Multiple workers run in parallel to train the partitions of the input dataset
- Works update shared model parameters for training
- The learning process can be sped up with no need of aggregating raw data to a centralized metadata server.