Volume: 11 Issue: VIII Month of publication: Aug 2023

DOI: https://doi.org/10.22214/ijraset.2023.55421

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue VIII Aug 2023- Available at www.ijraset.com

A Learning-based Data Placement Framework for Low Latency in Data Center Networks

Vinay S Navale[1], Chetan Jane[2], Anvita B[3], Kalyan B[4], Satvik A[5]
[1, 3, 4, 5]B.E in Information Science & Engineering, Dept of ISE, Nitte Meenakshi Institute of Technology
[2]B.E in Computer Science & Engineering, Dept of CSE, Bellari Institute of Technology and Management

Abstract: The challenge of achieving low-latency data services has become increasingly crucial for data center applications. In modern distributed storage systems, efficient data placement plays a vital role in minimizing data movement delays, thereby significantly reducing service latency. While previous solutions for data placement have often relied on pre-assumed data request distributions or traced analysis, the dynamic nature of network conditions and evolving user request patterns poses a complex online decision-making challenge. Traditional static model-based approaches are limited in their ability to cope with such dynamic system dynamics. To address these challenges and consider both data movement and analytical latency, we introduce a novel framework named DataBot+. This framework leverages reinforcement learning techniques to autonomously learn optimal data placement policies. DataBot+ employs neural networks that are trained using a modified version of Q-learning. The input to these networks comprises real-time data flow measurements, while the output is a value function that estimates the imminent future latency. To ensure timely decision-making, DataBot+ is designed with a two-fold asynchronous architecture, consisting of production and training components. This separation guarantees that the training process does not introduce additional overhead to the handling of data flows. Our approach is validated through extensive evaluations using real-world traces, which convincingly demonstrate the efficacy of our proposed design.

Keywords: low-latency data services, data center applications, modern distributed storage systems, data placement, data movement delays, service latency reduction, pre-assumed data request distributions, trace analysis, dynamic network conditions, user request patterns, online decision-making, static model-based approaches

I. INTRODUCTION

In the current landscape, we've witnessed a rapid surge in workloads propelled by data-intensive applications, such as web search, social networks, and e-commerce [2]. The central challenge lies in achieving low-latency services for real-time workloads. Prominent cloud service providers like Amazon and Google have underscored the significance of minimal service latency, as even a slight increase can lead to a noticeable reduction in user access and substantial revenue loss [3].

Service latency, as experienced by users, comprises two key components: data movement (both read and write) and analytical latencies. Efficient data analytics necessitates intensive movement of data items across computing or storage nodes, as their storage locations might not align with the execution sites. Notably, the locations where data items reside can significantly impact the completion time of distributed analytics, with movement latency frequently emerging as a principal bottleneck when data frequently traverse storage nodes [4]. Various strategies for data placement have been proposed to identify optimal data storage locations for reducing data movement latency. Prior efforts predominantly involve analyzing diverse factors that could influence data movement latency, utilizing manually designed optimization models [5]–[7], [9],[10]. However, the latency is subject to temporal variations, encompassing network latency, disk latency, and other forms of latency (such as RAM, CPU, etc.) [11]. Consequently, these static methods founded on optimization models lack the adaptability to contend with dynamic systems characterized by numerous uncertainties, such as unstable network connections, evolving user request patterns, and changing system configurations.

Furthermore, the preceding research endeavors exclusively focus on data movement latency when selecting storage locations. Nonetheless, the user-perceived service latency is a collective outcome of data movement and subsequent data analytics. Data analytics entails scrutinizing, refining, transforming, and modeling raw data to unearth valuable insights for informed decision-making. Different data analytics frameworks (like MapReduce [12], Dremel [13], and Spark [14]) have been introduced for processing substantial data volumes. For a specific analytical task at a given scale, the analytical latencies can differ across frameworks—for instance, tens of minutes for MapReduce, several seconds for Dremel, and sub-seconds for Spark. The analytical latency wields influence over the overall service latency, warranting its inclusion in the data placement predicament.

In contrast to prior methodologies, we propose a versatile learning-based framework called DataBot+. This framework autonomously optimizes data placement policies sans the requirement of future dynamic information. The investigated data placement challenge can be conceptualized as a finite Markov Decision Process (FMDP), given that (1) the number of nodes for data storage is finite, (2) each data placement action stands independent, and performance hinges on existing states and placement choices. Consequently, we employ model-free Q-learning, a technique proven to derive the optimal action-selection policy for any given FMDP [15]. In essence, DataBot+ offers a paradigm shift, effectively addressing the complexities of data placement and service latency optimization.

While Q-learning holds promise as a technique, it grapples with the challenge of dimensionality, leading to slow convergence as the state/action space expands. To address this limitation, our learning-based framework incorporates a neural network (NN) that efficiently and accurately approximates optimal outcomes. By inputting the current state information, the NN learns to predict the rewards associated with data placement actions. These rewards, reflecting data read/write and analytical latencies, then train the recurrent model, progressively refining data placement policies.

Given our core objective of prompt decision-making, it is imperative to ensure that training the recurrent NN doesn't introduce extra latency for managing data flows. To achieve this, our design employs a two-pronged asynchronous architecture—comprising the production and training systems—during implementation. This innovative approach entails parallel execution of online decision-making and offline training, departing from the conventional workflow of reinforcement learning (RL). Unlike the traditional RL paradigm, which mandates model updates after each decision, our decoupled setup facilitates seamless and uninterrupted adaptation while avoiding latency-associated setbacks.

II. OBJECTIVES

1) *Dynamic Network Conditions:* Develop adaptive solutions that can dynamically respond to changing network conditions, ensuring consistent low-latency performance.
2) *Real-time Workloads:* Optimize data placement strategies to effectively handle data-intensive applications with real-time workloads.
3) *Uncertainty Management:* Address uncertainties such as unreliable network links, evolving user request patterns, and system configurations to maintain latency reduction even in unpredictable scenarios.
4) *Reduced Latency:* Minimize overall service latency in data center networks to enhance user experience and satisfaction.
5) *Data Movement Efficiency:* Efficiently manage data movement (read and write) across storage nodes to mitigate data movement delays.

III. LIMITATIONS

While a learning-based data placement framework holds promise for enhancing low latency in data center networks, it is important to acknowledge its potential limitations. One significant constraint is the framework's heavy reliance on historical data for effective training. Inaccuracies or incompleteness in the training data can lead to suboptimal decision-making, undermining the framework's overall performance. Additionally, the complexity of neural network models employed within the framework may impede interpretability, making it challenging to comprehend the rationale behind specific decisions and complicating troubleshooting efforts. The implementation of a neural network-based solution introduces computational and memory overhead, potentially impacting the fficiency of the data center network. The resource-intensive nature of neural network training further compounds this challenge, as frequent retraining to adapt to evolving network conditions may disrupt normal operations and strain computational resources. Furthermore, while the framework may excel in certain scenarios, its ability to generalize effectively to unseen or rapidly changing situations remains a potential limitation, potentially resulting in suboptimal decisions during dynamic network conditions.

Balancing between latency reduction and optimizing other performance metrics, such as resource utilization or energy efficiency, poses a complex trade-off. Moreover, the framework's responsiveness to real-time demands might be compromised if the neural network struggles to keep pace with rapidly changing conditions, thereby impacting its ability to consistently achieve low latency. The framework's performance can also be sensitive to specific network architectures and configurations, potentially limiting its effectiveness in unfamiliar setups. Beyond technical considerations, incorporating a learning-based framework raises security concerns. Adversaries could potentially exploit the model's behavior to manipulate data placement decisions, potentially compromising the integrity and security of the data center network. Additionally, ethical considerations come into play, particularly in critical infrastructure settings like data centers, where transparency, accountability, and unintended consequences need to be carefully addressed. In light of these limitations, a comprehensive assessment of the framework's capabilities and shortcomings is crucial for informed decision-making and effective integration within data center environments.

IV. LITERATURE SURVEY

In their 2018 study[1], Liu et al. address the challenge of achieving low latency in data center networks through a learning-based adaptive data placement approach. Their work, presented at the IEEE LCN conference, focuses on optimizing data placement strategies using machine learning techniques. By leveraging adaptive learning and neural network-based models, they aim to dynamically reduce data movement delays and analytical latencies in data-intensive applications. Through their proposed framework, the authors seek to enhance real-time workload processing, considering dynamic network conditions and user request patterns. Their research contributes to the evolving landscape of data center network optimization by introducing a learning-based paradigm for efficient and responsive data placement, as showcased in their comprehensive analysis presented in the conference proceedings.

In their 2018 comprehensive review[2] published in ACM Computing Surveys, Mansouri, Toosi, and Buyya present an extensive investigation into data storage management within cloud environments. Their work contributes to the field by offering a taxonomy and thorough survey of existing research, providing insights into the diverse aspects of cloud data storage. The authors explore various storage models, architectures, techniques, and challenges, aiming to create a holistic understanding of the domain. They discuss key topics such as data consistency, availability, security, and scalability, while also identifying gaps and opportunities for future research directions. By synthesizing a broad range of literature, this survey serves as a valuable resource for researchers and practitioners seeking to navigate the complexities of data storage management in cloud computing.

In [3] their work presented at the ACM SIGCOMM conference in 2015, Pu et al. address the challenge of achieving low-latency geo-distributed data analytics. Through a pioneering approach, the authors delve into optimizing data analytics across geographically dispersed locations. By introducing techniques that intelligently balance data placement, they seek to minimize latencies stemming from data movement and analytical processes. Their study contributes to the advancement of efficient and responsive data analytics in geographically distributed settings, offering insights into critical aspects such as workload scheduling, data partitioning, and network optimizations. The research encapsulated in this conference paper serves as a significant milestone in the pursuit of low-latency data analytics across diverse geographical regions, providing valuable insights into architectural design and operational considerations for geo-distributed data analytics systems.

[4] In their contribution to the VLDB Endowment in 2011, Eltabakh et al. introduce CoHadoop, a framework aimed at enhancing data placement flexibility and its effective utilization within Hadoop ecosystems. By addressing the challenges of data placement and its impact on query performance, the authors propose CoHadoop as a solution that intelligently balances data distribution and co-location in Hadoop clusters. Their work encompasses a novel approach to data partitioning, replication, and query optimization, leading to improved performance in distributed data processing tasks. Through their comprehensive exploration of CoHadoop, the authors advance the understanding of efficient data placement strategies and their implications within Hadoop-based environments, thus contributing significantly to the field of distributed data processing and storage optimization.

[5] In their work published in the IEEE/ACM Transactions on Networking in 2016, Xiang et al. present a comprehensive study focusing on the joint optimization of latency and cost for erasure-coded data center storage. By addressing the interplay between storage performance and economic considerations, the authors propose an innovative approach that integrates erasure coding techniques with storage cost models. Through a meticulous analysis, they offer insights into achieving a balance between reduced latency and cost-effective data center storage solutions. Their research contributes to advancing the understanding of erasure-coded storage strategies and their trade-offs, offering valuable perspectives for designing efficient and economical storage systems in data center environments, as showcased in their in-depth exploration presented in the journal publication.

[6] Published in the IEEE/ACM Transactions on Networking in 2018, Ren et al. introduce Datum, a novel framework that addresses the complexities of data purchasing and data placement in a geo-distributed data market. By presenting an integrated solution, the authors tackle the challenges of data acquisition and optimal placement across distributed locations. Datum encompasses a comprehensive approach to data market dynamics, pricing models, and data transfer strategies, highlighting the intricate interplay between economic and network performance considerations. Through their meticulous analysis, Ren et al. contribute to the understanding of efficient data market operations, offering valuable insights into the design and management of geo-distributed data marketplaces, as presented in their insightful study published in the journal.

In [7] their contribution to the IEEE Transactions on Services Computing in 2017, Yu and Pan introduce a novel hypergraph-based framework for optimizing data placement across geo-distributed data centers. Their work addresses the intricacies of distributing data efficiently across diverse locations by leveraging hypergraph representations. By proposing a comprehensive approach, the authors offer insights into data placement strategies that consider the geographical distribution of data centers, network latency, and resource utilization. Their research advances the understanding of effective data distribution in the context of geo-distributed environments, providing valuable perspectives on hypergraph-based techniques for enhancing data placement efficiency, as elucidated in their study published in the journal.

In their paper [8] presented at the IEEE INFOCOM conference in 2016, Yu and Pan introduce a sketch-based methodology for optimizing data placement in cloud storage across geographically distributed data centers. Their work addresses the challenge of efficient data distribution in the context of geo-distributed environments by leveraging sketching techniques. By proposing a comprehensive framework, the authors contribute insights into data placement strategies that consider network latency, data center characteristics, and workload profiles. Their research enriches the field's understanding of harnessing sketch-based approaches to enhance data placement efficiency within cloud storage systems, as highlighted in their study presented at the conference.

[9] Published in the proceedings of the ACM Symposium on Cloud Computing (SoCC) in 2017, the collaborative work of Hu, Wang, Liu, Niu, and Huang addresses the challenges of latency reduction and load balancing in coded storage systems. By presenting their research in this context, the authors delve into the optimization of data storage and retrieval in distributed systems through coding techniques. Their study offers insights into strategies that simultaneously mitigate latency and balance the computational load across storage nodes. Through a comprehensive framework, they contribute to the field's understanding of efficient data storage and retrieval mechanisms in coded storage systems, as showcased in their research paper presented at the conference.

[10] In their work published in the journal "Computer Networks" in 2016, Fan, Ding, Wang, and Yuan address the vital issue of green and latency-aware data placement within data centers. Their research focuses on optimizing data storage and distribution in data centers with a dual objective of reducing energy consumption and minimizing latency. Through their comprehensive framework, the authors contribute insights into strategies that consider both environmental sustainability and efficient data management. By merging the concerns of energy efficiency and low-latency data access, their study advances the understanding of green data placement approaches in data center environments, as outlined in their research presented in the journal article.

V. DESIGN AND IMPLEMENT

A. System Architecture

Existing research has demonstrated the significance of data placement in enhancing data locality and read/write performance within data-intensive systems. Predicting data requests accurately, Ren et al. [6] formulated data purchasing and placement as an integer linear programming problem, yielding a near-optimal solution that jointly reduces service costs and latency. Jalaparti et al. [16] proposed an offline scheduling approach to concurrently optimize data and task placement based on workload features, thus significantly improving network locality.

Agarwal et al. [19] introduced Volley, an automated data placement scheme driven by trace analysis, which places data items in proximity to end users. Cui et al. [20] constructed a tripartite graph to model data placement, offering a genetic solution to minimize data traffic and latency in cloud environments. Yu et al. [7] addressed data placement across geo-distributed data centers using a hypergraph-based approach, and their subsequent work [8] refined this using a sketch-based method. However, these methods are offline solutions and lack dynamic system consideration.

Steiner et al. [17] presented an online scheme that groups data items used within the same task for placement, reducing inter-rack traffic and task completion time. Chowdhury et al. [18] selected storage locations based on server occupancy links to minimize task completion time. Notably, these solutions overlooked the impact of subsequent read requests on future read-related performance. In contrast, DataBot+ uniquely considers read/write and data analytical latencies, employing reinforcement learning (RL) to adaptively learn an optimal data placement policy without presuming future user requests.

Our work aligns with the integration of RL and neural networks (NN) to address complex online decision-making [21], [22], specifically focusing on the data placement problem in data center networks (DCN). Mao et al. [23] used deep RL for resource management, optimizing an objective function based on rewards. Unlike this, Mirhoseini et al. [24] employed application execution time directly as the RL reward for device placement optimization. Nie et al. [25] employed group-based RL to decrease TCP response latency through real-time flow-level metrics. Chen et al. [26] developed a deep RL system for flow-level traffic optimization.

Our previous work [1] pursued overall service latency reduction in DCNs using end-to-end metrics as rewards. Building on this, DataBot+ extends the framework to account for the impact of data analytics on placement decisions, expanding its scope beyond mere latency optimization as addressed in [1].

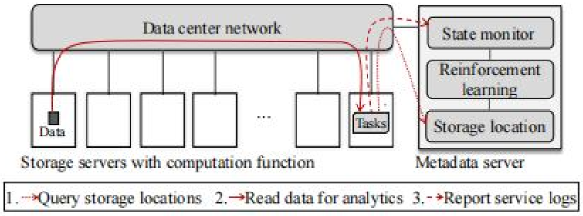

Data center network
State monitor
Reinforcement learning
Storage location
Data
...
Tasks
Storage servers with computation function
Metadata server
1.→Query storage locations 2.→Read data for analytics 3.→Report service logs

Fig. 1. Data storage system: storage servers, data center network, metadata server, and data flows.

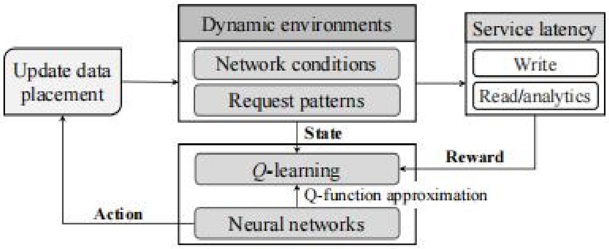


Fig. 2. An overview of the RL-based data placement.

B. Problem Statement

Due to the substantial impact of data movement latency, the selection of storage locations for data items plays a pivotal role in determining the completion time of distributed analytical tasks. To achieve low-latency analytical services within a data center network (DCN), the crux of the data placement problem revolves around the optimal choice of storage locations among available servers when writing or updating data items. The service latency within the DCN encompasses both data movement latency and data analytical latency for computation. The data movement latency is a composite of various components, including transmission, propagation, processing, and queuing latencies inherent in the network. The intricate nature of the entire system, marked by dynamic factors such as evolving network conditions and user request patterns, makes it challenging to devise an exact latency model. In response, a generic solution in the form of reinforcement learning (RL) is embraced to tackle the data placement dilemma. By employing RL, the placement of data items can be envisioned as an agent interacting with its environment, continuously learning from feedback. In contrast to traditional methods that focus on crafting precise mathematical latency models, the adopted RL approach diverges by tailoring itself to the statistical patterns inherent in the ever-changing dynamic environment.

C. Design Overview

The design overview of DataBot+ is depicted in Figure 2, and the fundamental principle of Q-learning-based data placement can be elucidated through the Q-function, which is defined as follows: $Q : \text{State}(S) \times \text{Action}(A) \rightarrow \text{Reward}(R)$. The dynamic characteristics of the storage system (State S) are acquired through extensive data flows, enabling the understanding of the optimal placement of each data item (Action A) to minimize associated service latencies. Subsequently, the observed read/write and analytical latencies are utilized as the Reward R for training the recurrent model. This iterative process allows DataBot+ to progressively enhance its data placement policies over time for long-term optimization.

To elaborate further, when a data item m is scheduled to be written into the storage system at time t, the choice of storage locations is determined based on the present state s and the governing data placement policy π, where $s \in S$. The ensuing action a entails placing data item m in the designated location, denoted as $a \in A$. Until the data item m is subsequently updated at t0, the latencies associated with the most recent write operation at t, as well as the subsequent read and analytical activities transpiring between t and t0, are measured. The collective read/write and analytical latencies, suitably weighted, constitute the immediate reward rt for the undertaken action a. Following the update operation at t0, the system transitions to a new state s0.

Referring to [29], the immediate reward rt at time t inherently influences future instances. The optimal Q-value function $Q*(s, a)$ captures the highest expected long-term reward, formulated as follows: $Q*(s, a) = \max \pi\, E[Xt\geq 0\ \gamma trt|s0 = s, a0 = a, \pi]$, where $\gamma \in (0, 1)$ serves as a discount factor. The attainment of $Q*(s, a)$ hinges on the Bellman equation, articulated as: $Q*(s, a) = Es0\ [r + \gamma \max a0\ Q*(s0 , a0)|s, a]$.

In the context depicted in Figure 2, future rewards are influenced by various dynamic factors in the environment, such as network conditions and request patterns. Traditional RL approaches that rely on temporal differences [30] may not ensure swift convergence to an optimal solution. To address this challenge, a neural network (NN) is introduced to approximate the Q-function with heightened efficiency and precision, contributing to the core efficiency of the DataBot+ framework.

D. *Q-Function Design*

1) *States:* A fundamental characteristic of DataBot+ is its exclusive reliance on end-to-end measurements of data flows, specifically the measured read/write latencies, to inform its data placement decisions. These decisions are guided by five distinct categories of state information encompassed within the state variable (s), and these categories can be derived directly from the service logs**.**
2) *Network Conditions:* This primary category encompasses the network conditions, which exert a significant impact on the central goal of minimizing read/write latencies. It involves the computation of average latencies, denoted as L[R]_{ij} and L[W]_{ij}, for read and write operations between pairs of servers i and j. Notably, this analysis specifically focuses on application-layer data placement, thus excluding link-based metrics such as bit error rates. In the context of this study, the selection pertains to the source and destination of the data flow, as opposed to specifying the precise paths or links. Importantly, it is worth noting that introducing measurements of link-related metrics would introduce additional overhead when compared to the end-to-end approach. Moreover, the end-to-end approach, in contrast to link-related metrics, possesses the flexibility to accommodate a wide range of arbitrary data center network topologies. Despite this, our proposed design harmoniously coexists with underlying link-based or path-based flow scheduling methodologies, offering versatility and compatibility.

3) *Data Analytical Latency:* The second category pertains to the data analytical latency, denoted as:

$$\left\{L^{[\mathrm{A}]}_{j,m}, j \in \mathcal{N}\right\},$$

where L[A]_{j,m} signifies the estimated analytical latency of data item m on the destination server j. This analytical latency is intrinsically influenced by the nature of the analytical task, with computation-intensive tasks typically demanding more analytical time. Additionally, the analytical latency is influenced by factors such as assigned task priority and the server's computational workload level. By leveraging l[A]_m from equation (1), an Exponentially Weighted Moving Average (EWMA) approach is employed to estimate the analytical latency of data m on each server
Importantly, it's notable that the storage location i of data item m does not directly influence the analytical latency on the computing server j. Consequently, optimizing the analytical latency is beyond the scope of this paper. However, the incorporation of analytical latency remains crucial within the data placement problem, as it exerts a notable impact on the overall user-perceived service latency.

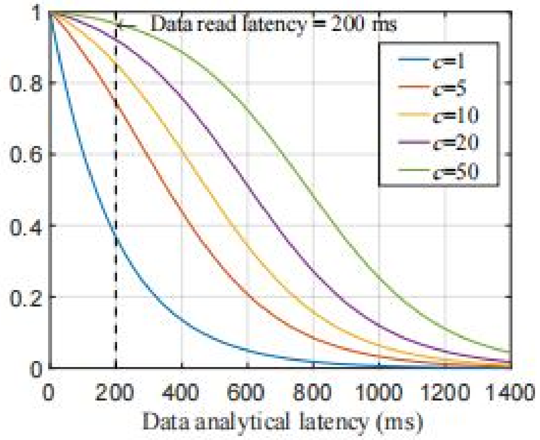


Fig. 3. An example of the weight function $f(l_p^{[\mathrm{R}]}, l_p^{[\mathrm{A}]})$.

4) *Data Analytical Latency:* The second category pertains to the data analytical latency, denoted as:
5) *Request Patterns:* The inclusion of request patterns, encompassing attributes such as the rate of read requests Fi[R] and write requests Fi[W] for all data items from source i, provides insights into how analytical tasks generate workloads within the storage system. These patterns serve as a fundamental driver of network traffic. By leveraging detailed request information for both all data items and specific data instances, our design augments its ability to make informed decisions when selecting optimal storage locations. To construct real-time request pattern information, we employ the Discounting Rate Estimator (DRE) method [32]. In this context, each item in equation (8) maintains an associated counter, which accumulates with every read/write operation executed across server pairs and experiences periodic reduction governed by a ratio factor αr within the interval ![0, 1]. Notably, DRE offers distinct benefits: (1) swift adaptation to changing request patterns, and (2) efficient resource utilization with a mere O(1) space requirement and update time for each counter.
6) *Data Size:* The fourth category encompasses data size χm, a critical attribute that influences the read/write latencies of data item m.
7) *Source Locations:* We introduce a binary vector within the state variable s to signify whether each storage server acts as a source for data write operations. This vector, containing zeros and ones, denotes the source locations of data flows, a crucial factor influencing write operation latencies. In this context, if the number of data replicas is denoted by k_1, the binary vector contains k servers set to one.

Importantly, equation (9) illustrates that the number of data items M does not introduce complexity to the learning-based storage system.

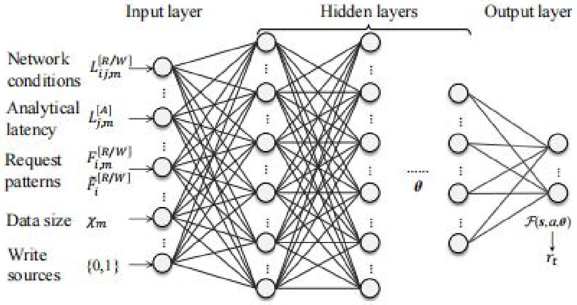


Fig. 5. NN structure for data placement.

E. Asynchronous Implementation

The conventional workflow of reinforcement learning (RL) necessitates model updates following each decision, which can inadvertently introduce latency and prove suboptimal for data center applications. To circumvent this limitation, DataBot+ introduces an innovative approach to enable instant decision-making for query requests regarding write locations. This is achieved through the decoupling of the learning system into two distinct components: the production system and the training system, as visualized in Fig. 6. Operating asynchronously, these components collaborate to ensure that the training process of the neural network (NN) does not impose additional overheads on the handling of data requests within the production system. This design ensures that DataBot+ can swiftly respond to user queries while continuously refining its data placement strategies through learning, thereby enhancing the efficiency and effectiveness of the overall framework.

F. Algorithm

- *Input:* Neural network weight vector θ, current state st, exploration rate ε.
- *Output:* Data placement action a∗t, reward rt, next state st+1.

```
while A request queries the write destination at time t do
    Generate a random number $\eta \in [0, 1]$;
    if $\eta < \varepsilon$ then
        Select action a∗t = arg max
                at∈A
                F(st, at, θ);
    else
        Randomly select action a∗t from A;
    end if

    if Replay memory R is full, with size |R| = R then
        Remove the earliest tuple from R;
    end if
    Store the tuple (st, a∗t, st+1, rt) in replay memory R;
end while.
```

G. *Production System*

The system optimizes storage decisions using a decision-making Neural Network (NN). Given an input state (st) and the current weight vector (θ), the NN outputs a vector F(st, at, θ) representing reward estimates for writing data to various servers. Each element of F(st, at, θ) indicates the expected reward for storing the data item on the corresponding server.

Algorithm 1 provides a pseudocode representation of the production system. Upon receiving a data write/update request at time t, the algorithm employs an ε-greedy approach to select an action. First, a random variable η is generated from a uniform distribution within the range [0, 1]. If η is less than ε, the action a∗t is chosen to maximize the output value of F(st, at, θ), aiming for reduced read/write latency. Conversely, if η ≥ ε, a random action is selected to explore unexplored solution options.

Upon updating storage locations using action at, the system transitions to a new state (st+1). At the end of time interval t, the observed reward (rt) becomes evident.

H. *Training System*

The training system recurrently revisits the stored tuples within the replay memory R to iteratively enhance the weight vector θ+ for guiding forthcoming decisions in the production system. Algorithm 2 provides a structured representation of this training procedure. The weight vector refinement is achieved using the mini-batch stochastic gradient descent (SGD) technique [29], which systematically adjusts the weights to minimize the divergence between the output of the Neural Network (NN) and the desired target value.

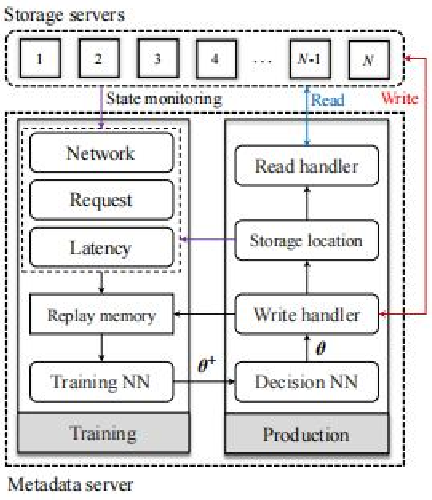


Fig. 6. Production and training system of DataBot+.

I. *The training Algorithm*

- *Input:* Neural network weight vector θ, replay memory R.
- *Output:* Updated weight vector θ+.

```
if Replay memory R is full, with size |R| = R then
    Shuffle all tuples τ in R to generate mini-batches B;
```

```
  for epoch i in I do
    for each tuple $\tau$ in R do
      Calculate target value yt = rt + $\gamma$ * max
                          at+1
                          F(st+1, at+1, $\theta$);
    end for
    for each mini-batch b in B do
      Update $\theta$+ to minimize the objective (14);
    end for
    $\theta$ = $\theta$+;
  end for
end if
```

J. Data Trace Description and Experiment Settings

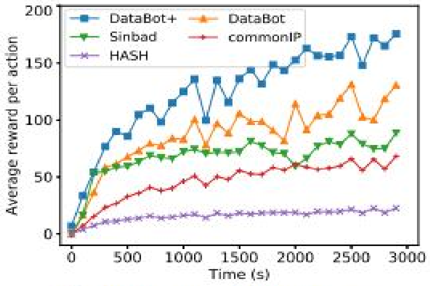
The MSR Cambridge Traces are collected from Microsoft Research Cambridge's enterprise data center, capturing data read and write requests over the span of a week from 36 servers. These traces encompass a range of arrival rates for read and write requests, which are depicted in Figure 7. It's important to note that the distribution of these requests is skewed across the 36 servers due to the characteristics of real-world applications.

For each request, essential information is logged, including the hostname, request type (read or write), the size of the transferred data, and the timestamp. Due to confidentiality considerations, publicly available traces, including the MSR Cambridge Traces, do not provide explicit details about the specific data items associated with each request. It is assumed that the storage system involves a total of M = 10,000 data items.

Similar to prior research studies, the request rates for data items on server i follow a Zipf distribution, with e Fi[R/W] reflecting the normalized probability sum over all data items. Notably, the majority of tasks in the cloud analytical system are short-lived. Empirical studies have highlighted that latency distributions in systems like Amazon EC2 and Microsoft Bing exhibit long tails, indicating a prevalence of extended latencies.

Consequently, the data analytical latencies are simulated based on a power-law distribution with a long tail. These latencies span a wide range, from 50 milliseconds to 300 seconds. This choice of distribution is influenced by the characteristics observed in practical scenarios, corroborated by experimental analysis.

In summary, the MSR Cambridge Traces provide insights into read and write request patterns within a real-world enterprise data center environment. These traces reveal skewed request distributions among servers and incorporate aspects like latency distribution tail behavior to better reflect the complex dynamics of data processing and analysis tasks.



(a) Average reward per action r_t

VI. EXPERIMENTS

To commence the assessment, the read-optimized scenario is evaluated with specific parameter settings: a weight ω of 0.2 and replica count k of 3. The evaluation period spans 3,000 seconds, which has been determined as sufficiently long to achieve a stable performance improvement ratio. The constant c in the reward function is established at 5. Notably, the replay memory retains the latest 2,000 tuples, which are then subjected to training. The training process involves 6 epochs (|I| = 6) and mini-batches of size 300 (|B| = 300). Each round of training consumes an average of 8.498 seconds before transferring the updated weight vector θ+ to the decision Neural Network (NN) within the production system.

It is worth emphasizing that, lacking an asynchronous implementation, NN training would introduce an additional latency of 8.498 seconds to data write requests. This supplementary latency is unsuitable for data center applications. This underscores the advantages brought by the asynchronous implementation.

As depicted in Figure 8(a), the average reward per data placement decision is illustrated. Initially, read/write latencies decrease as Memcached requires warming up with data items. Consequently, the average rewards of Sinbad, commonIP, and HASH show an initial increase for the first few hundred seconds, after which they stabilize. Comparatively, DataBot+ and DataBot learn improved data placement policies through continuous trials and feedback, resulting in a progressive increase in average reward through the learning process. After several training rounds for convergence, the performance improvement ratio of DataBot+ and DataBot stabilizes around the 1,000-second mark.

Figures 8(b) and (c) depict the average read/write latencies measured in intervals of 200 seconds throughout the experimental period. HASH, acting as random storage location selection irrespective of user request patterns or network conditions, incurs the highest read/write latencies of approximately 201.0 ms and 249.9 ms, respectively, during the final 1,000 seconds. commonIP prioritizes data placement on storage servers with the highest data request rates, resulting in average read/write latencies of 175.6 ms and 226.2 ms, respectively, due to queuing delays resulting from skewed data request distributions.

In contrast, Sinbad focuses on low-occupancy link servers, achieving the lowest write latency at 179.6 ms but neglecting read requests with a read latency of 172.4 ms. With an improved placement policy resulting from the learning process, DataBot+ and DataBot exhibit lower read latencies than HASH, commonIP, and Sinbad. In comparison to previous work, DataBot significantly reduces average read/write latencies to 156.8 ms and 211.3 ms, respectively. Conversely, DataBot+ concentrates on reducing the data movement latency of data items with short analytical latencies. Dividing data items into groups based on analytical latencies, DataBot+ prioritizes data items with shorter analytical durations to lower read latencies, and addresses items with longer analytical latencies by minimizing write latencies. This strategy eases overall network congestion and consequently reduces read/write latencies for other data items with short analytical latencies.

Especially noteworthy, as depicted in Figure 9, for data items with analytical latencies ranging from 50 ms to 200 ms, DataBot+ achieves the most substantial reduction in user-experienced latency for data read and analytics, with improvements of up to 23.8%. Furthermore, compared to DataBot, DataBot+ introduces periodic "pseudo" write operations to mitigate hotspots, resulting in average read/write latencies for all data items reduced to 141.5 ms and 193.9 ms, respectively.

Overall, the evaluation underscores the efficacy of the proposed DataBot+ approach in optimizing data placement decisions and improving overall system performance.

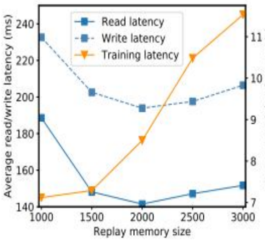


Fig. 12. Impact of replay memory size $|\mathcal{R}|$.

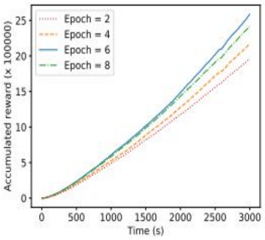


Fig. 13. Impact of epoch number $|\mathcal{I}|$.

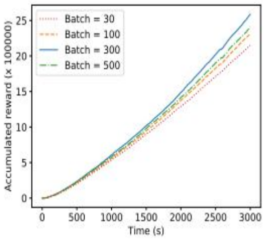


Fig. 14. Impact of batch size $|\mathcal{B}|$.

To comprehensively evaluate the performance of DataBot+, various factors that can impact the data placement process are taken into account. In comparison to the least favorable performance observed with HASH, Figures 10 and 11 depict the percentage reduction in latency for data items with short analytical latency (ranging from 50 ms to 200 ms).

A. *Write Weight ω*

Figures 10(a) and 11(a) portray the influence of write weight ω. In heuristic solutions such as HASH, commonIP, and Sinbad, the average read/write latencies remain relatively stable as ω is not considered. As ω increases from 0 to 1.0, DataBot+ and DataBot progressively prioritize write requests. For DataBot+, the average write latency is reduced by 6.72% (from 202.3 ms to 188.7 ms), while the read latency increases by 37.42% (from 126.4 ms to 173.7 ms). For the read-optimized scenario (ω = 0), compared to HASH, DataBot+, DataBot, Sinbad, and commonIP exhibit reductions of 24.94%, 16.9%, 9.4%, and 8.32% respectively, in user-experienced latency for data read and analytics. For the write-optimized scenario (ω = 1.0), compared to HASH, write latency reductions are achieved for DataBot+ (24.5%), DataBot (23.79%), Sinbad (28.13%), and commonIP (9.48%).

B. Number of Replicas k

Data replication enhances reliability, accessibility, and fault-tolerance. Increasing the replica count from 1 to 7 eases network congestion from read requests, lowering the average read latency of DataBot+ from 151.2 ms to 103.6 ms. Synchronous data writes ensure strong consistency, causing write latency to increase from 195.3 ms to 218.9 ms as data items need to be written into multiple locations. Future work may explore asynchronous writing models to mitigate write latency.

C. Constant c

With an increase in c (as depicted in Fig. 3), the importance of read requests rises in (11), implying a higher read request priority but less emphasis on write requests. As c rises from 1 to 50, the average read latency for DataBot+ decreases from 145.5 ms to 123.5 ms, while the write latency increases from 191.2 ms to 201.9 ms. DataBot, Sinbad, commonIP, and HASH exhibit stable read/write latencies without considering c. DataBot+ achieves increasingly more user-experienced latency reduction for data read and analytics as c increases, although the reduction in write latency diminishes.

D. Replay Memory Size |R|

Figure 12 demonstrates the effect of replay memory size |R|. As |R| increases from 1,000 to 2,000, the learning system approximates the optimal solution more precisely with a larger dataset, resulting in decreased average read/write latencies (25% and 16.6%, respectively). However, training latency increases from 7.117 s to 8.498 s. With |R| further increasing to 3,000, training latency rapidly rises from 8.498 s to 11.533 s. While precision improves, adaptability to network dynamics decreases, leading to increased average read/write latencies (7.2% and 6.4%, respectively).

E. Number of Training Epochs |I|

The number of training epochs |I| influences NN training and placement performance. Figure 13 shows that as |I| increases from 2 to 6, accumulated reward rises due to more training rounds reducing the gap between expected and actual rewards. However, setting |I| to 8 leads to performance degradation from overfitting. Over time, overfit models lose generalization capability for future samples. In our experiments, |I| is set to 6.

F. Batch Size |B|

Batch size affects weight vector θ updates during training. Fig. 14 demonstrates that |B| set to 300 achieves the highest accumulated reward, emphasizing the importance of careful training parameter selection. These findings motivate the exploration of systematic parameter tuning and avoiding overfitting in future work.

In summary, an exhaustive evaluation of DataBot+ explores numerous factors affecting data placement, demonstrating its performance benefits and highlighting the significance of parameter choices for optimal results. This in-depth analysis lays the foundation for further refinement and optimization of the DataBot+ system.

VII. CONCLUSION

This paper presents an innovative learning-based data placement framework, known as DataBot+, designed to address the uncertainties inherent in dynamic systems. This framework offers an automated approach to learning optimal data placement policies. Leveraging a Neural Network (NN), the system estimates near-future latencies by training a weight vector using Q-values. This strategy mitigates the complexity associated with an extensive state space, facilitating quicker convergence towards solutions.

A distinctive aspect of DataBot+ is its strategic handling of data items with short analytical latencies. Given their heightened sensitivity to data movement latency variations, these items are assigned higher priorities. The goal is to maximize the reduction of user-experienced service latency, consequently enhancing overall system efficiency.

Moreover, DataBot+ seamlessly integrates two asynchronous components: online decision-making and offline training. This integration ensures that no additional delays are introduced to manage intensive data flows. Performance evaluations convincingly illustrate reductions in user-experienced service latencies when compared to state-of-the-art solutions.

In anticipation of scalability, future work aims to explore distributed Reinforcement Learning (RL) solutions. These solutions seek to expedite the convergence of the learning process in the data placement problem. An appealing attribute of such an approach is the elimination of the need to aggregate raw data to a centralized metadata server for training. By sidestepping this requirement, the framework becomes even more adaptable and suited for larger-scale applications

VIII. FUTURE WORK

In the realm of future research, several promising avenues present themselves for the advancement of the learning-based data placement framework geared towards minimizing latency in data center networks. Firstly, the exploration of scalable distributed reinforcement learning methods holds great potential to elevate the framework's scalability and speed of convergence. By decentralizing learning processes and eliminating the reliance on centralized metadata servers, this approach could seamlessly integrate the framework into more intricate and expansive data center environments.

Secondly, the adoption of advanced learning paradigms, such as deep reinforcement learning and policy gradient techniques, offers an avenue for refining the framework's decision-making capabilities. These methodologies could unlock nuanced insights and more intricate data placement strategies, potentially leading to even lower latency levels. Furthermore, an adaptable framework capable of dynamically responding to changing workloads and traffic patterns could significantly bolster its utility and effectiveness. Efforts in this direction could empower the framework to maintain optimal data placement policies under varying operational conditions.

Lastly, the integration of hybrid approaches that combine the strengths of learning-based algorithms with established heuristic methods could offer a robust solution. These hybrid strategies might leverage the rapid learning capabilities of the framework while incorporating the stability and reliability of traditional techniques. Extensive benchmarking across diverse scenarios and collaboration with industry stakeholders for real-world validation would provide essential insights into the framework's real-world viability and practical applicability. Collectively, these avenues of future work promise to refine and enhance the framework's capabilities, thereby contributing to the ongoing evolution of efficient low-latency solutions in dynamic data center networks.

REFERENCES

[1] K. Liu, J. Wang, Z. Liao, B. Yu, and J. Pan, "Learning-based adaptive data placement for low latency in data center networks," in Proc. of IEEE LCN, pp. 142–149, 2018.

[2] Y. Mansouri, A.N. Toosi, and R. Buyya, "Data storage management in cloud environments: Taxonomy, survey, and future directions," ACM Comput. Surv., vol. 50, no. 6, pp. 1–51, 2018.

[3] Q. Pu, G. Ananthanarayanan, oP. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics,"in Proc. of ACM SIGCOMM, pp. 421–434, 2015.

[4] M.Y. Eltabakh, Y. Tian, F. ¨Ozcan, R. Gemulla, A. Krettek, and J. McPherson, "CoHadoop: Flexible data placement and its exploitation in Hadoop," in Proc. of VLDB Endow., vol. 4, no. 9, pp. 575–585, 2011.

[5] Y. Xiang, T. Lan, V. Aggarwal, and Y.F.R. Chen, "Joint latency and cost optimization for erasure-coded data center storage," IEEE/ACM Trans. Netw., vol. 24, no. 4, pp. 1063–6692, 2016.

[6] X. Ren, P. London, J. Ziani, and A. Wierman, "Datum: Managing data purchasing and data placement in a geo-distributed data market," IEEE/ACM Trans. Netw., vol. 26, no. 2, pp. 893–905, 2018.

[7] B. Yu and J. Pan, "A framework of hypergraph-based data placement among geo-distributed datacenters," IEEE Trans. Serv. Comput., 2017.

[8] B. Yu and J. Pan, "Sketch-based data placement among geo distributed datacenters for cloud storages," in Proc. of IEEE INFOCOM, pp. 1–9, 2016.

[9] Y. Hu, Y. Wang, B. Liu, D. Niu, and C. Huang, "Latency reduction and load balancing in coded storage systems," in Proc. of ACM SoCC, pp. 365–377, 2017.

[10] Y. Fan, H. Ding, L. Wang, and X. Yuan, "Green latency-aware data placement in data centers," Comput. Netw., vol. 110, pp. 46–57, 2016.

[11] "Latency Definition." [Online]: https://techterms.com/ definition/latency

[12] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," Commun. ACM, vol. 51, no. 1, pp. 107–113, 2008.

[13] S. Melnik, A. Gubarev, J.J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: Interactive analysis of web-scale datasets," Commun. ACM, vol. 54, no. 6, pp. 114–123, 2011.

[14] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in Proc. of USENIX NSDI, pp. 15–28, 2012.

[15] L.P. Kaelbling, M.L. Littman, and A.W. Moore, "Reinforcement learning: A survey," J. Artif. Intell. Res., vol. 4, pp. 237–285, 1996.

[16] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, "Network-aware scheduling for data-parallel jobs: Plan when you can," in Proc. of ACM SIGCOMM, pp. 407–420, 2015.

[17] M. Steiner, B.G. Gaglianello, V. Gurbani, V. Hilt, W.D. Roome, M. Scharf, and T. Voith, "Network-aware service placement in a distributed cloud environment," in Proc. of ACM SIGCOMM, pp. 73–74, 2012.

[18] M. Chowdhury, S. Kandula, and I. Stoica, "Leveraging endpoint flexibility in data-intensive clusters," in Proc. of ACM SIGCOMM, pp. 231–242, 2013.

[19] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated data placement for geo-distributed cloud services," in Proc. of USENIX NSDI, 2010

[20] L. Cui, J. Zhang, L. Yue, Y. Shi, H. Li, and D. Yuan, "A genetic algorithm based data replica placement strategy for scientific applications in clouds," IEEE Trans. Serv. Comput., vol. 11, no. 4, pp. 727–739, 2018.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," in NIPS Deep Learning Workshop, 2013.

[22] I. Bello, H. Pham, Q.V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," arXiv preprint arXiv:1611.09940, 2017.

[23] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in Proc. of ACM HotNets, pp. 50–56, 2016.

[24] A. Mirhoseini, H. Pham, Q.V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, and J. Dean, “Device placement optimization with reinforcement learning,” in Proc. of ICML, pp. 2430–2439, 2017.

[25] X. Nie, Y. Zhao, D. Pei, G. Chen, K. Sui, and J. Zhang, “Reducing web latency through dynamically setting TCP initial window with reinforcement learning,” in Proc. of IEEE/ACM IWQoS, pp. 1–10, 2018.

[26] L. Chen, J. Lingys, K. Chen, and F. Liu, “AuTO: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization,” in Proc. of ACM SIGCOMM, pp. 191–205, 2018.

[27] W. Xia, P. Zhao, Y. Wen, and H. Xie, “A survey on data center networking (DCN): Infrastructure and operations,” IEEE Commun. Surv. Tuts., vol. 19, no. 1, pp. 640–656, 2017.

[28] Q. Xu, R.V. Arumugam, K.L. Yong, and S. Mahadevan, “Efficient and scalable metadata management in EB-scale file systems,” IEEE Trans. Parallel Distrib. Syst., vol. 25, no. 11, pp. 2840–2850, 2014.

[29] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou,H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[30] C. Xu, K. Wang, P. Li, R. Xia, S. Guo, and M. Guo, “Renewable energy-aware big data analytics in geo-distributed data centers with reinforcement learning,” IEEE Trans. Netw. Sci. Eng., 2018.

[31] J.S. Hunter, “The exponentially weighted moving average,” J. Qual. Technol., vol. 18, no. 4, pp. 203–210, 1986.

[32] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, F. Matus, R. Pan, N. Yadav, and G. Varghese, “Conga: Distributed congestion-aware load balancing for datacenters,” in Proc. of ACM SIGCOMM, pp. 503–514, 2014.

[33] “HDFS Architecture Guide.” [Online]: https://hadoop.apache. org/

[34] S. Arora, N. Cohen, N. Golowich, and W. Hu, “A convergence analysis of gradient descent for deep linear neural networks,” in in Proc. of ICLR, 2019.

[35] D. Narayanan, A. Donnelly, and A. Rowstron, “Write off-loading: Practical power management for enterprise storage,” ACM Trans. Storage, vol. 4, no. 3, pp. 1–23, 2008.

[36] P. Delgado, F. Dinu, A. M. Kermarrec, and W. Zwaenepoel, “Hawk: Hybrid datacenter scheduling,” in Proc. of USENIX NSDI, pp. 499– 510, 2015.

[37] Y. Xu, Z. Musgrave, B. Noble, and M. Bailey, “Bobtail: Avoiding long tails in the cloud,” in Proc. of USENIX NSDI, pp. 329–341, 2013.

[38] V. Jalaparti, P. Bodik, S. Kandula, I. Menache, M. Rybalkin, and C. Yan, “Speeding up distributed request-response workflows,” in Proc. of ACM SIGCOMM, pp. 219–230, 2013.

[39] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, “Sparrow: Distributed, low latency scheduling,” in Proc. of ACM SOSP, pp. 69–84, 2013.