EdgeKV

Hansen Sheng

April 18, 2024

Abstract

Edge computing has emerged as a promising paradigm to address these challenges by decentralizing computation and storage capabilities closer to the data source. This paper introduces EdgeKV, a novel key-value store system designed specifically for edge computing environments. Leveraging the proximity of edge nodes to data sources, EdgeKV aims to provide efficient and reliable data access services at the network edge. Key features include constant fetching for ultimate consistency, advanced cache eviction strategies, and range prefetching based on key patterns. Experimental evaluation demonstrates EdgeKV's efficacy in meeting the demands of edge computing environments, offering efficient and responsive data access services. This paper contributes to the advancement of storage systems optimized for edge environments, addressing the unique challenges posed by limited computational resources, heterogeneous hardware configurations, and communication over wide-area networks or wireless protocols.

1 Introduction

As the Internet of Things (IoT) continues to grow and permeate various aspects of our lives, the generation of massive volumes of data from smart devices and sensors poses new challenges for processing and storage. Traditionally, cloud data centers have served as the primary infrastructure for storing and processing this data, benefiting from their scalability, security, and reliability. However, as IoT technology becomes increasingly prevalent, there is a growing demand to process data at the network edge, closer to the data source, to achieve lower latency and higher responsiveness.[1][2]

Edge computing has emerged as a paradigm that aims to address these challenges by decentralizing computation and storage capabilities to the network edge. By bringing processing closer to the data source, edge computing mitigates the latency and bandwidth limitations associated with centralized cloud infrastructures, making it particularly advantageous for latency-sensitive applications like autonomous vehicles, industrial IoT, and online multiplayer games.

While the concept of edge computing offers promising solutions, the design and implementation of efficient and scalable storage systems optimized for edge environments present unique challenges. Edge nodes, deployed at the network edge, possess limited computational resources, storage capacity, and power availability. Furthermore, these nodes often exhibit heterogeneous hardware and software configurations

and communicate over Wide-Area Networks or wireless protocols. These characteristics necessitate the development of storage systems tailored specifically to the edge, capable of accommodating these constraints and delivering low-latency, responsive, and highly reliable key-value store services.

In this paper, we introduce EdgeKV, a novel key-value store system designed to meet the demands of edge computing. Leveraging the proximity of edge nodes to data sources, EdgeKV aims to provide efficient and reliable data access at the network edge. Our system incorporates several key features and mechanisms to address the unique challenges faced in edge environments, ensuring optimal performance and consistency. [1]

Our system makes several key contributions to address the challenges associated with edge computing:

1. Constant fetching of new values for ultimate consistency: EdgeKV employs a mechanism to continuously fetch the latest values, ensuring data consistency across edge nodes.

2. Advanced Cache Eviction Strategies: To optimize limited storage resources on edge nodes, EdgeKV incorporates intelligent cache eviction strategies that maximize the utilization of available storage while prioritizing frequently accessed data.

3. Range prefetching based on key: EdgeKV introduces a range prefetching technique that proactively fetches data based on key patterns, reducing latency by minimizing round trips for subsequent requests.

The remainder of this paper is organized as follows: Section 2 provides background of edge computing and key-value stores. Section 3 presents the design principles and architecture of EdgeKV. Section 4 discusses the implementation details. Section 5 presents the evaluation results, and Section 6 discusses future research directions. Finally, Section 7 concludes the paper.

2 Background

2.1 Content Delivery Network

A Content Delivery Network (CDN) represents a geographically dispersed infrastructure comprising proxy servers and associated data centers strategically positioned across various regions. The primary objective is to ensure high availability and performance by spatially distributing services in close proximity to end users. Within this framework, edge nodes play a crucial role by caching static resources such as images, audio files, and videos within local data centers.

Upon receiving a user request for resources, the CDN's scheduling system orchestrates the routing process, directing the request to the most optimal edge node based on predefined policies. This dynamic allocation of resources ensures efficient utilization of network resources and minimizes latency, thereby enhancing the overall user experience. By accessing data from nearby edge nodes, users can significantly reduce

[1] This work has been open-sourced to my repository: https://github.com/shenghansen/EdgeKV.

latency and alleviate congestion within centralized data centers.

In contemporary internet infrastructure, CDNs have emerged as the predominant solution for accelerating user access to website resources. They currently handle a substantial portion of internet traffic, surpassing 70%, and this trend is expected to continue its upward trajectory. The ongoing proliferation of digital content consumption, coupled with the exponential growth of internet-connected devices, underscores the indispensable role of CDNs in facilitating seamless content delivery and optimizing network performance on a global scale.[3]

2.2 Edge Computing

Edge computing represents a paradigm shift in computing architecture, aiming to decentralize computation and storage capabilities by bringing them closer to the data source, typically at the network edge. This decentralized approach offers several benefits over traditional centralized cloud infrastructures, including reduced latency, optimized network bandwidth utilization, and support for real-time decision-making in latency-sensitive applications.

However, the transition to edge computing introduces a host of unique challenges that must be addressed to realize its full potential. One significant challenge lies in the design and implementation of efficient and scalable systems tailored to edge environments. Unlike centralized cloud infrastructures, edge nodes operate under stringent resource constraints, including limited computational power, storage capacity, and availability of power sources. These constraints necessitate the development of resource-efficient algorithms and architectures optimized for edge deployments.

Moreover, the heterogeneous nature of edge nodes further complicates system design and deployment. Edge nodes often comprise diverse hardware and software configurations, ranging from lightweight sensors to powerful edge servers. This heterogeneity poses challenges in ensuring interoperability and compatibility across different edge devices, as well as in developing solutions that can effectively utilize the available resources while accommodating the variability in hardware capabilities.

Additionally, the distributed nature of edge computing introduces complexities in managing and orchestrating edge resources efficiently. Traditional centralized management approaches may not be suitable for edge environments, where nodes are geographically dispersed and interconnected through potentially unreliable networks. As a result, new approaches and tools for resource management, monitoring, and orchestration are needed to ensure the reliable and efficient operation of edge computing systems.

Addressing these challenges requires a multidisciplinary approach that integrates expertise from various domains, including computer science, networking, hardware design, and system optimization. Collaborative efforts among researchers, industry stakeholders, and standardization bodies are essential to develop standardized frameworks, protocols, and best practices for edge computing. By overcoming these challenges, edge computing has the potential to revolutionize the way we deploy and manage computing infrastructure, enabling a wide range of innovative applications

and services at the network edge.[4]

2.3 KV Service

KV, as a global key-value data store, stands at the forefront of modern distributed systems, embodying a foundational pillar in the architecture of contemporary digital infrastructures. Its significance lies in its ability to streamline and optimize data retrieval and storage operations, offering unparalleled efficiency and low latency across vast geographical distances.

At the heart of KV's architecture is the strategic centralization of data within a select number of primary data centers. This centralized repository serves as the authoritative source of truth, housing the entirety of the dataset in a cohesive and organized manner. By consolidating data in this manner, KV simplifies data management and enhances data integrity, ensuring consistency and coherence across distributed environments.

The deployment of KV enables organizations to achieve remarkable performance gains by leveraging edge caching mechanisms to serve frequently accessed data closer to end users. This distributed caching approach significantly reduces access latency, enabling the creation of highly responsive applications and services that rely on real-time data processing.

One of the key strengths of KV lies in its ability to support exceptionally high read volumes while maintaining low latency. This capability is instrumental in the development of dynamic APIs and applications that demand rapid data retrieval and processing. By efficiently managing read requests across distributed edge caches, KV empowers developers to build scalable and responsive systems capable of handling fluctuating workloads with ease.

However, despite its advantages, KV faces challenges in ensuring consistent low latency for all access patterns. While reads are periodically revalidated in the background to maintain data freshness, requests that miss the cache and require retrieval from the centralized backend may experience elevated latencies. This latency spike can be particularly pronounced in scenarios where the centralized backend is located far from the edge nodes or experiences high network congestion.

To mitigate these challenges, KV systems employ various optimization techniques, such as intelligent caching policies, prefetching mechanisms, and distributed load balancing algorithms. These strategies aim to minimize the frequency of cache misses and optimize data retrieval paths to reduce latency and improve overall system performance.

Furthermore, ongoing research efforts focus on enhancing KV's resilience to latency spikes and improving its ability to adapt dynamically to changing network conditions. By combining advanced caching strategies with efficient data synchronization mechanisms, KV continues to evolve as a foundational technology for building high-performance distributed systems capable of meeting the demands of modern applications and services.[5]

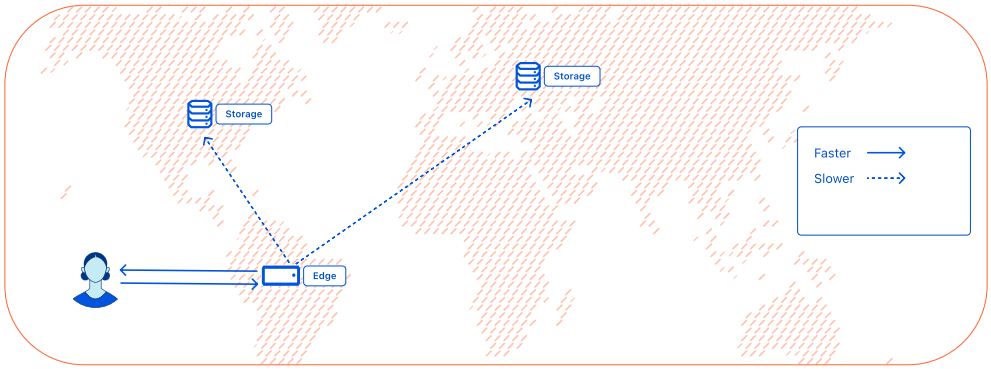


Figure 1: Workflow of the KV service.

3 Design

Figure 1 provides a comprehensive visualization of the intricate workflow of the KV service, elucidating the journey of data from its initial access point to its ultimate delivery to end users.

The process initiates with the first read request originating from a specific location, wherein the requested data lacks a cached value. In such instances, the data retrieval process commences from the nearest regional tier within the KV infrastructure. If the data is not found within the regional tier, the request progresses hierarchically to a central tier, and ultimately to the central store where genuinely cold global reads are fulfilled. Although the initial access may incur a global latency overhead due to the retrieval from the central store, subsequent requests benefit from improved responsiveness, particularly when requests concentrate within a singular region.

A crucial distinction within the KV workflow lies between "hot reads" and "cold reads." A "hot read" signifies that the requested data is readily available within the edge network, having been cached through the CDN. Conversely, a "cold read" denotes the absence of cached data, necessitating users to retrieve the data directly from the storage provider, typically the central store.

As users repetitively access data from identical locations, KV optimizes response times by delivering the cached value without the need for external data retrieval. This caching mechanism ensures the swiftest response times for recurrent requests. Moreover, KV diligently maintains the freshness of cached data by periodically refreshing it from upper tiers and the central data stores in the background. This background refresh process is meticulously orchestrated to ensure the continuous serving of accessed assets from the cache without any interruptions, thereby preserving the responsiveness of the KV service.

KV's design is meticulously tailored to cater to high-read-rate applications, making it particularly adept for scenarios characterized by infrequent writes but rapid and recurrent reads. In such scenarios, infrequently accessed values are fetched from alternate data centers or the central store, while more popular values are strategically cached in the data centers where they are frequently requested. This approach optimizes resource utilization and ensures that KV efficiently cate. [6]

The architectural framework of EdgeKV, as illustrated in Figure 2, delineates a sophisticated network topology wherein data centers serve as the operational hubs for the KV service. This intricate network operates through Remote Procedure Calls (RPC) facilitating seamless communication between the data centers and the Edge infrastructure. At the forefront of this network, the Edge serves as a pivotal intermediary, entrusted with the crucial task of caching data sourced from the data centers.

Ensuring data consistency across this distributed system is paramount. To achieve this, the Edge diligently accepts updates propagated from the data centers, thereby harmonizing data coherence throughout the network. However, the Edge's cache capacity is finite, necessitating the implementation of highly efficient cache eviction strategies. These strategies are meticulously designed to intelligently purge cached data when storage limits are reached, ensuring optimal cache utilization and bolstering cache hit rates.

Furthermore, the utilization of Log-Structured Merge Trees (LSM) underscores the sophistication of EdgeKV's caching mechanism. By harnessing the inherent capabilities of LSM, EdgeKV empowers itself to engage in proactive key-based range prefetching. This strategic prefetching mechanism anticipates data access patterns, preemptively retrieving data segments based on key ranges, thereby enhancing overall data access efficiency and reducing latency in data retrieval operations.

4 Implementation

4.1 Caching strategy

Ensuring efficient cache utilization in Edge environments is imperative, given the inherent constraints of limited memory resources. Advanced cache management policies play a pivotal role not only in maintaining a high cache hit rate but also in ensuring lightweight implementations capable of swiftly responding to access requests.

The FIFO (First-In-First-Out) eviction policy presents itself with several desirable attributes such as simplicity, speed, scalability, and compatibility with flash-based storage systems. However, its primary limitation lies in its low efficiency, often resulting in a high miss ratio, thus necessitating further refinement.

Addressing this challenge, the state-of-the-art S3-FIFO algorithm emerges as a solution tailored to achieving elevated cache hit rates. Its effectiveness stems from the recognition that in skewed workloads, a significant proportion of objects experience only transient access within a short timeframe. This underscores the significance of early eviction strategies, also referred to as quick demotion. The essence of S3-FIFO lies in the integration of a compact FIFO queue that selectively filters out most objects from entering the primary cache, thereby facilitating swift and precise demotion processes.

The architecture of S3-FIFO revolves around three distinct FIFO queues: a small queue (S), a primary queue (M), and a ghost queue (G). Through empirical evaluation across diverse workload scenarios, a cache space allocation of 10% to S has been found

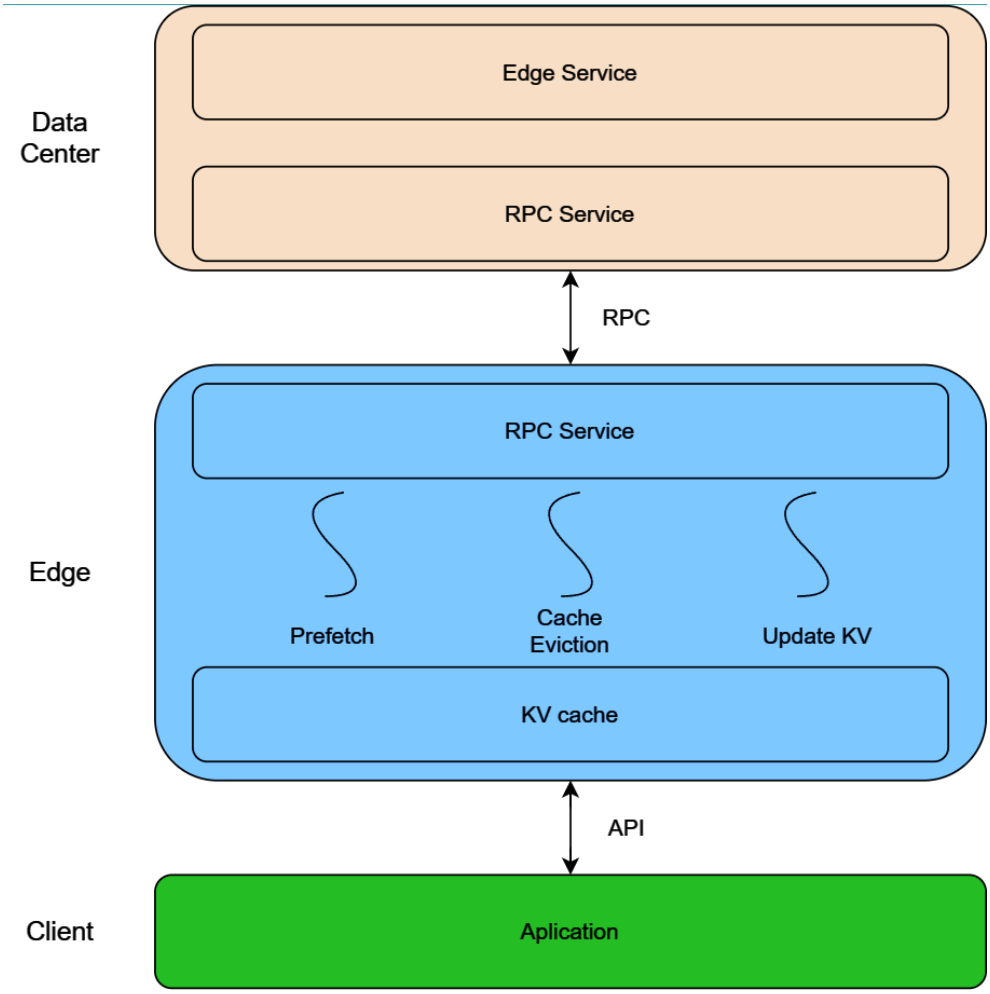


Figure 2: The architectural framework of EdgeKV.

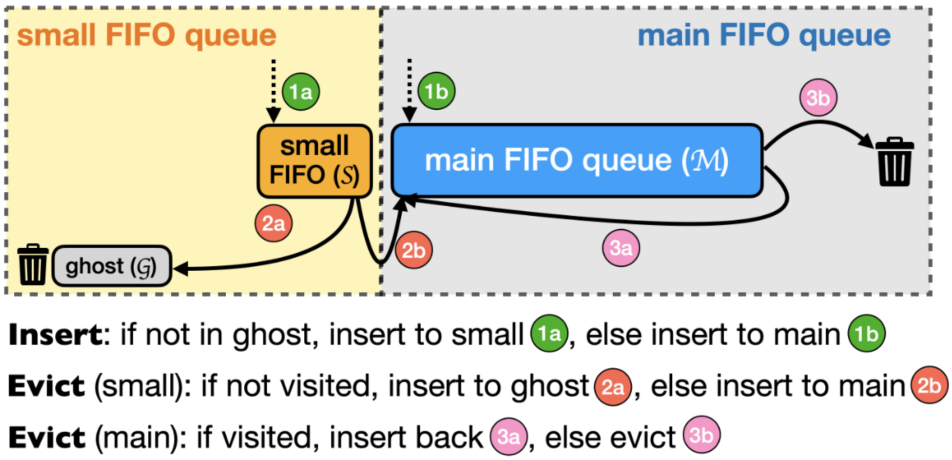


Figure 3: The algorithmic depiction of S3FIFO.

to yield robust performance. Subsequently, M utilizes the remaining 90% of the cache space, while G accommodates an equivalent number of ghost entries as M, thereby facilitating efficient cache read operations. S3-FIFO incorporates a two-bit tracking mechanism per object to monitor its access status, inspired by a capped counter with a frequency limit of three. Cache hits within the S3-FIFO framework increment the counter atomically by one, with popular objects often requiring no further update. Regarding cache writes, newly accessed objects are inserted into S if absent from G; otherwise, they are inserted into M. Upon reaching capacity, S triggers the eviction process, wherein the least-accessed object at the tail is either demoted to M if accessed more than once or to G if not, with its access bits reset during the transition. Upon reaching capacity, G evicts objects in FIFO order. The management of M employs a strategy akin to FIFO-Reinsertion, leveraging a two-bit mechanism to track access information. Objects accessed at least once undergo reinsertion with one bit set to 0, simulating a decrease in frequency by 1. The algorithmic workflow is visually depicted in Figure 3, accompanied by the corresponding pseudo-code provided in Algorithm 1.

To further bolster responsiveness, the integration of Negative Lookups within the Edge environment proves invaluable. This mechanism logs instances where a requested key is found to be nonexistent, thereby enabling swift responses without necessitating queries to the central data center, thus streamlining overall system performance and reducing latency.

4.2 Update KV

Updates in KV adhere to the principle of eventual consistency, necessitating both data centers and edges to ensure the timely propagation of the latest KV values to the edges. KV is engineered to excel in high-read-rate applications and is well-suited for scenarios requiring infrequent writes but rapid and recurrent reads.

Therefore, the primary objective is to ensure that updates eventually reach the edges without compromising edge responsiveness. We adopt a data center-driven up-

Algorithm 1 S3-FIFO algorithm

```
function READ(x)
    if x in S or x in M then                          ▷ Cache Hit
        x.freq ← min(x.freq + 1, 3)                   ▷ Frequency is capped to 3
    else                                              ▷ Cache Miss
        INSERT(x)
        x.freq ← 0
    end if
end function

function INSERT(x)
    while cache is full do
        EVICT
    end while
    if x in G then
        insert x to head of M
    else
        insert x to head of S
    end if
end function

function EVICT
    if S.size ≥ 0.1· cache size then
        EVICTS
    else
        EVICTM
    end if
end function

function EVICTS
    evicted ← false
    while not evicted and S.size > 0 do
        t ← tail of S
        if t.freq > 1 then
            insert t to M
            if M is full then
                EVICTM
            else
                insert t to G
            end if
            evicted ← true
        end if
        remove t from S
    end while
end function

function EVICTM
    evicted ← false
    while not evicted and M.size > 0 do
        t ← tail of M
        if t.freq > 0 then
            insert t to head of M
            t.freq ← t.freq − 1
```

date model, which not only conserves edge resources but also facilitates the attainment of consistency in distributed systems.

When a KV update is initiated by a user, the propagation process commences with the immediate update of the nearest edge node, ensuring proximity to the originating user for enhanced responsiveness. Subsequently, the updated data is transmitted back to the central data center, where a meticulous verification process is undertaken. This verification entails cross-referencing with other edge nodes housing identical data sets to ascertain consistency across the distributed network. Leveraging Remote Procedure Calls (RPC), the data center autonomously orchestrates the dissemination of updates to all relevant edge nodes, thereby fortifying the network-wide coherence of KV values. This proactive approach not only streamlines the update process but also mitigates latency overhead, underscoring KV's commitment to maintaining the integrity and consistency of data across distributed environments.

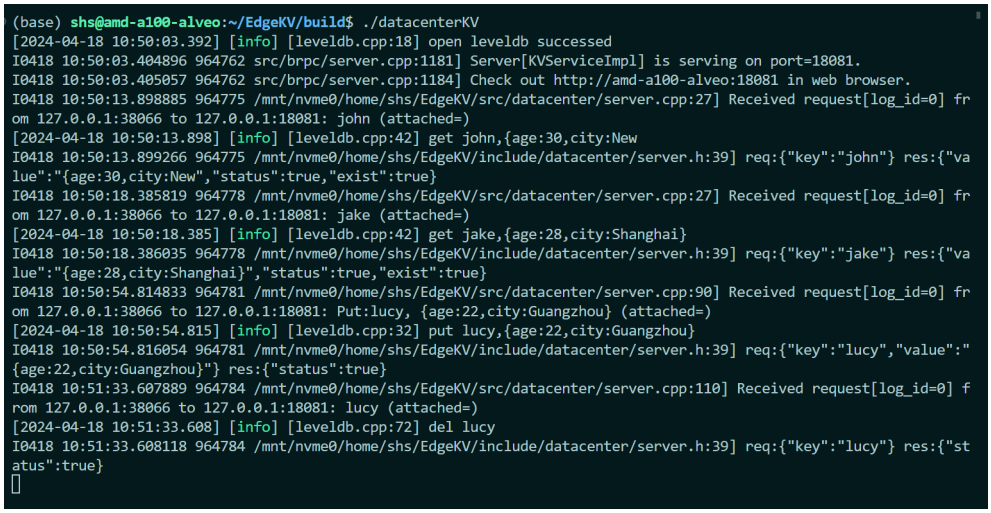
4.3 Prefetch

In the pursuit of enhancing caching performance within the KV framework, prefetching emerges as a pivotal technique. Prefetching strategically anticipates data needs, preemptively retrieving information likely to be accessed in the near future. However, within the context of KV's decentralized architecture, selecting the most suitable data for prefetching presents a formidable challenge.

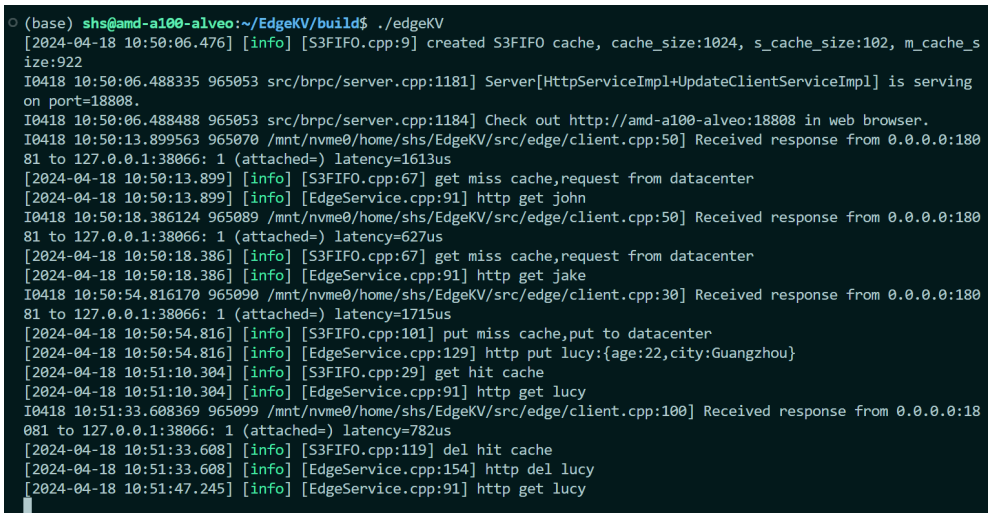
To address this challenge, we advocate for the adoption of a key-based prefetch strategy. This approach capitalizes on the inherent characteristics of KV stores, where adjacent keys often correspond to data items with a high likelihood of being requested by users in tandem. By prioritizing the prefetching of data associated with neighboring keys, KV systems can proactively anticipate user demands and reduce latency by preemptively caching relevant information.

Central to the efficacy of key-based prefetching is the ordered structure of Log-Structured Merge Trees (LSM)[7]. LSM organizes data in a sequential manner, facilitating expedited retrieval of adjacent key-value pairs. This inherent ordering property enables KV systems to efficiently identify and prefetch data segments, thereby optimizing cache utilization and minimizing access latency.

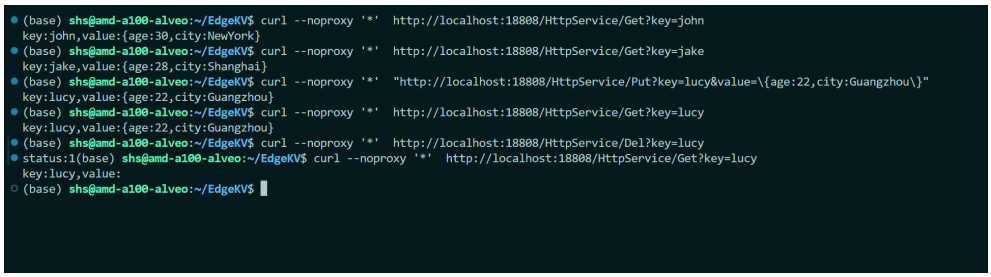
By harnessing the synergy between key-based prefetching and the ordered nature of LSM, KV systems can significantly enhance their caching performance, ultimately leading to improved responsiveness and user experience in distributed environments.



(a) DataCenter



(b) Edge



(c) User

Figure 4: EdgeKV functional test

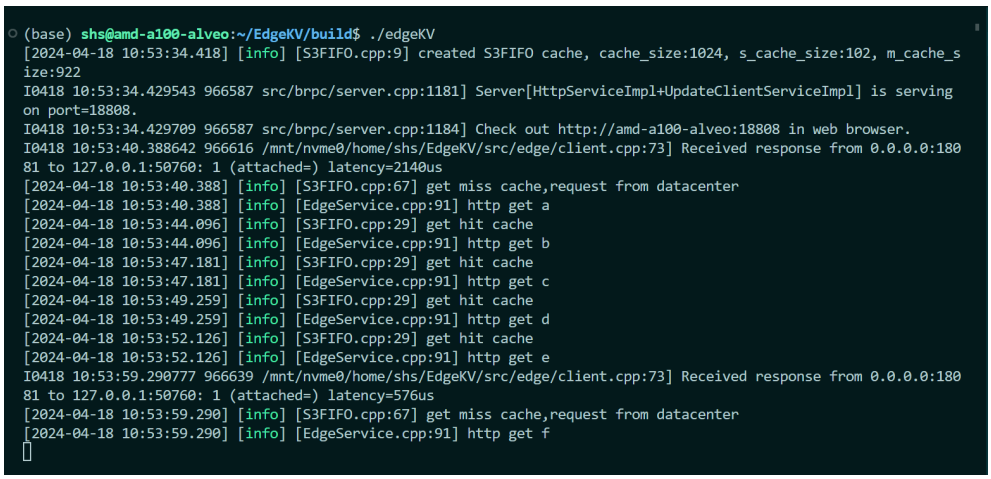
5 Experiment

We leverage two concurrent processes running on a single server to emulate distinct entities: a data center and an edge server, respectively. The server configuration comprises an AMD EPYC 7773X with 128 cores and 1TB of memory.

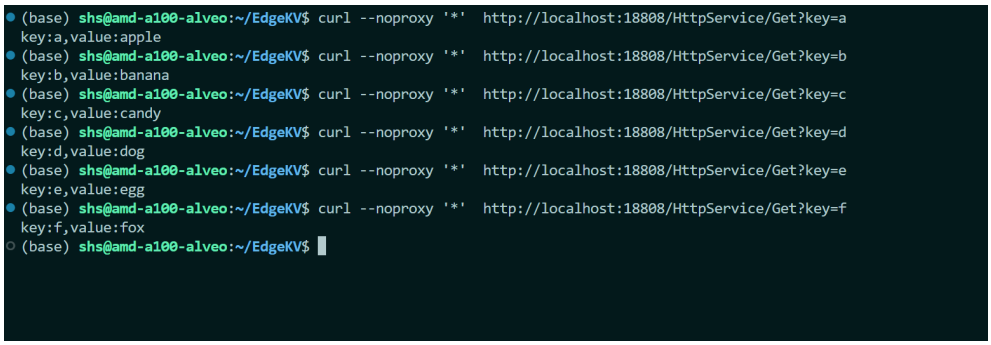
To ascertain the correctness of the system functionality, an initial suite of operations encompassing get, put, and delete is executed. As depicted in Figure 4, the system demonstrates sustained and accurate operation. Subsequently, the efficacy of prefetching is examined. As delineated in Figure 5, the introduction of prefetching notably enhances the cache hit rate when accessing neighboring keys.

Lastly, the performance of S3FIFO is scrutinized through a comparative analysis against LRU and the native FIFO. The findings, delineated in Figures 6 and 7, underscore the superior performance of S3FIFO across diverse test scenarios. Additionally, a conspicuous trend manifests whereby, as the cache size decreases, S3FIFO exhibits a proportionally superior hit rate compared to the other two methods.

(a) DataCenter



(b) Edge



(c) User

Figure 5: Prefetching test

6 Discussion

The system elucidated in this paper has materialized as a reliable and responsive EdgeKV infrastructure. However, inherent deficiencies persist in the domains of security, fault tolerance, and consistency.

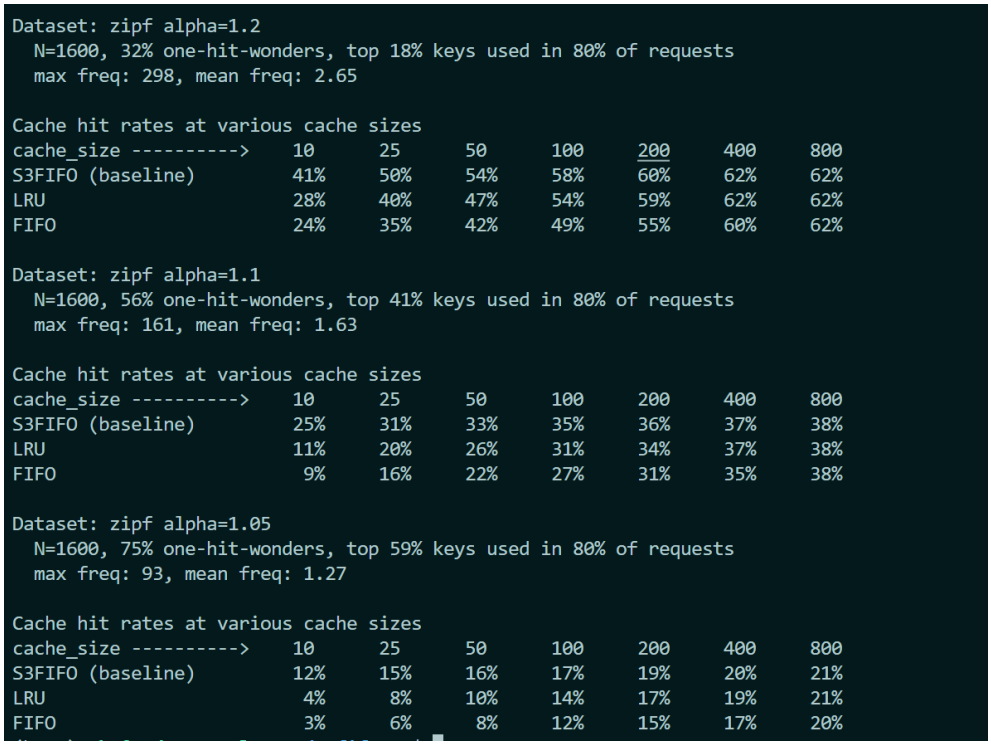
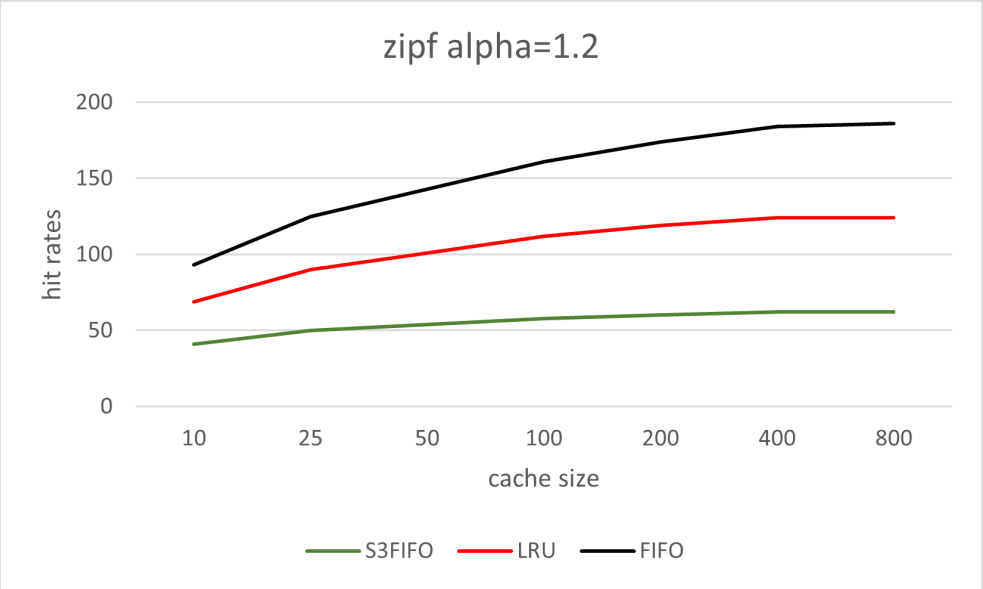


Figure 6: S3FIFO performance test

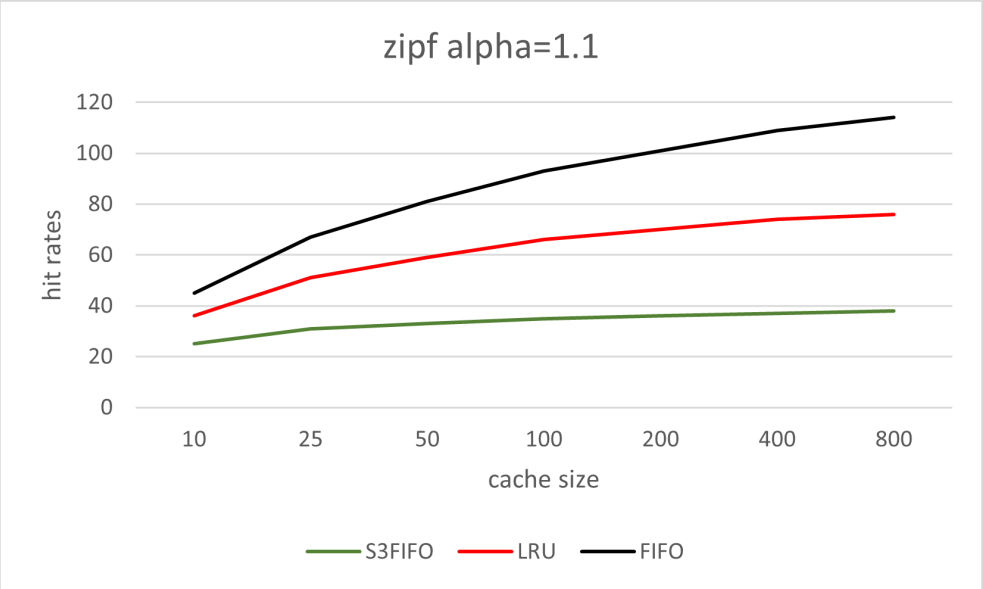
Primarily, with regard to security, the EdgeKV system remains incompletely fortified against data security breaches and privacy infringements. In the milieu of distributed environments, the imperatives of data security loom large, particularly amidst the specter of network assailments and the specter of data integrity compromises. Thus, forthcoming enhancements ought to prioritize the fortification of encryption protocols governing data transmission and storage, thereby assuring the sanctity and confidentiality of user data.

Secondarily, in the realm of fault tolerance, notwithstanding the system's extant measures to mitigate node failures and network partitions, avenues for fortification abound. Especially in the face of systemic upheavals or network vicissitudes, the resilience and availability of the system may be imperiled. Hence, prospective endeavors might delve into the refinement of more robust fault tolerance mechanisms, such as multi-replica data redundancy and dynamic load balancing, to augment the system's fortitude against adversarial contingencies.

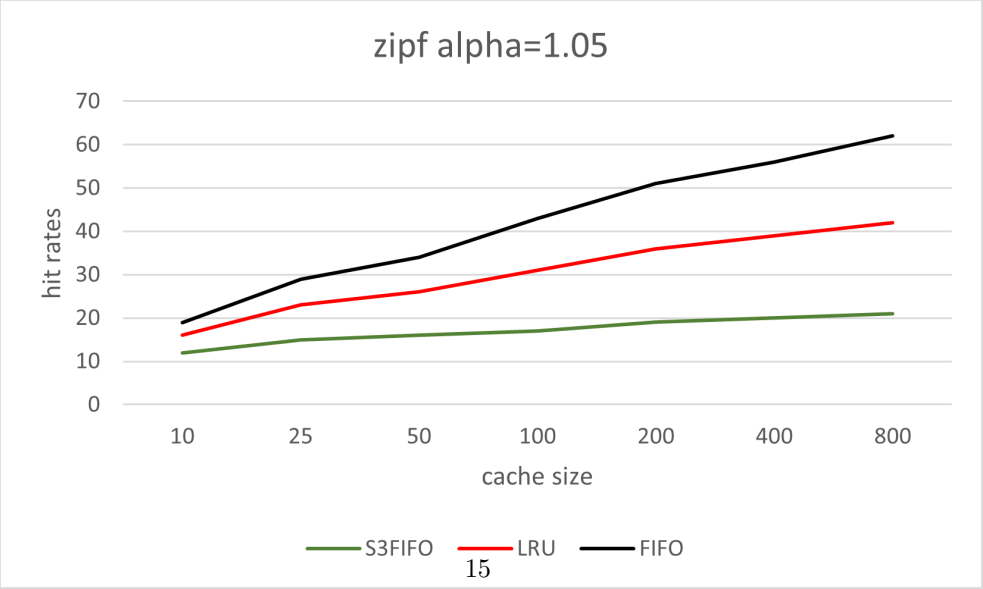
Lastly, concerning consistency, notwithstanding the system's employment of a data center-driven update paradigm and the judicious deployment of Remote Procedure Calls (RPC) for data synchronization, vestiges of inconsistency may persist in exceptional scenarios. Notably, scenarios characterized by intense write and update operations may engender synchronization lags or data incongruities. Ergo, future endeavors could be directed towards the optimization of data synchronization algorithms to uphold stringent consistency standards and fortify the system's reliability across diverse operational exigencies.



(a) zipf alpha=1.2



(b) zipf alpha=1.1



(c) zipf alpha=1.05

Figure 7: S3FIFO performance

7 Conclusion

The paper presents an EdgeKV system designed to fulfill the demands of edge computing. This system delivers efficient and reliable data access services, bolstered by several distinctive features aimed at enhancing performance and ensuring swift responsiveness. Experimental validation underscores the system's capacity to effectively meet operational requirements.

References

[1] Kashif Bilal, Osman Khalid, Aiman Erbad, and Samee U Khan. Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. *Computer Networks*, 130:94–120, 2018.

[2] Yield Enhancement. International roadmap for devices and systems, 2018 edition (2019). *IEEE. White Paper: Proactive Particle Control in Ultrapure Water (UPW) in Silicon Wafer Cleaning Process. Supplemental Materials, YE IRDS*, 2018.

[3] Juncheng Yang, Anirudh Sabnis, Daniel S Berger, KV Rashmi, and Ramesh K Sitaraman. {C2DN}: How to harness erasure codes at the edge for efficient content delivery. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 1159–1177, 2022.

[4] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. An overview on edge computing research. *IEEE access*, 8:85714–85728, 2020.

[5] cloudflare. How kv works, 2024. https://developers.cloudflare.com/kv/reference/how-kv-works/.

[6] Juncheng Yang, Yazhuo Zhang, Ziyue Qiu, Yao Yue, and Rashmi Vinayak. Fifo queues are all you need for cache eviction. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 130–149, New York, NY, USA, 2023. Association for Computing Machinery.

[7] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. The log-structured merge-tree (lsm-tree). *Acta Informatica*, 33:351–385, 1996.