ECO-KVS: Energy-aware Compaction Offloading Mechanism for LSM-tree based Key-Value Stores in Edge Federation

Jeeseob Kim[1]*, Hongsu Byun[1]*, Seungjae Lee[1], Myoungjoon Kim[1], Youngjae Kim[1], Zaipeng Xie[2], Sungyong Park[1]†
[1]*Sogang University*, Seoul, Republic of Korea, [2]*Hohai University*, Nanjing, China
{jeeseob, byhs, ctaaone, audwns0820, youkim, parksy}@sogang.ac.kr, {zaipengxie}@hhu.edu.cn

***Abstract*—In recent years, the rise in energy consumption across infrastructure has highlighted the need for more energy-efficient technologies. This is particularly critical in edge computing environments, where resources and power are limited. Consequently, there is increasing interest in improving the energy efficiency of resource-intensive tasks on edge servers. Edge servers commonly use Log-Structured Merge-tree-based Key-Value Store (LSM-KVS), to manage continuous data streams from edge devices. A key operation in LSM-KVS, known as compaction, merges key-value pairs in a CPU-intensive and energy-demanding process. Additionally, delays during compaction can cause write stalls, blocking I/O operations and degrading performance. This creates a significant challenge in balancing energy consumption and system performance. To address these challenges, we propose ECO-KVS, a solution that improves both energy efficiency and performance in LSM-KVS by offloading compaction tasks across edge servers in an edge federation. ECO-KVS leverages a real-time learning model to predict compaction time and energy consumption, reducing write stalls and enhancing overall energy efficiency. Implemented on RocksDB, ECO-KVS achieves up to 21% higher throughput compared to the baseline RocksDB and improves the performance-to-energy efficiency ratio by up to 18% compared to EdgePilot, a state-of-the-art solution for edge environments.**

***Index Terms*—Edge Computing, Edge Federation, Energy-efficiency, Key-Value Store, Log-Structured Merge-Tree.**

I. Introduction

The rapid growth of IT infrastructure in recent years has resulted in a significant rise in energy consumption. According to the International Energy Agency (IEA), data centers worldwide consumed approximately 460 Terawatt-Hours (TWh) of electricity in 2022, representing around 2 percent of global electricity demand [1]. This consumption is projected to increase even more dramatically, surpassing 1,000 TWh by 2026, highlighting the urgent need for energy-efficient technologies.

In this context, the importance of Green IT [2] is becoming more prominent. Green IT refers to technologies and strategies aimed at enhancing the energy efficiency of IT systems while minimizing their environmental impact, playing a crucial role in building a sustainable IT ecosystem.

Edge computing [3] is becoming a crucial technology in advancing Green IT. By processing data closer to the user, edge computing reduces energy consumption associated with network transmission. This not only enhances energy efficiency but also provides low-latency performance compared to cloud computing. As a result, edge computing is increasingly being applied in various fields, including Internet of Things (IoT) devices, autonomous vehicles, and smart factories [4].

Building upon the advantages of edge computing, edge federation [5] extends its capabilities by enabling cooperative task and resource sharing among edge servers. Edge federation addresses the limitations of individual servers by distributing workloads and scaling resources dynamically, making it particularly effective for large-scale data processing in edge environments. By combining the benefits of edge computing and federation, edge federation not only improves performance and scalability but also supports energy efficiency in distributed systems.

Meanwhile, various software applications are running on edge servers, with databases being crucial for the rapid storage and processing of constantly generated data [6]. One such technology, the Log-Structured Merge-tree-based Key-Value Store (LSM-KVS), is widely adopted. LSM-KVS achieves high write performance through an append-only method and ensures efficient read performance via compaction, which removes and organizes duplicate key-value pairs. However, delayed compaction can negatively impact read performance and cause write stalls, blocking I/O operations. Consequently, enhancing compaction in LSM-KVS continues to be a critical focus of ongoing research.

Compaction is a CPU-intensive operation that involves merge sorting, and various approaches have been proposed to improve it, often by using accelerators like FPGAs [7], [8] or disaggregating resources [9], [10]. However, these methods are challenging to implement in edge servers, where hardware limitations hinder improvements in energy efficiency. To address this, the state-of-the-art solution EdgePilot [11] was introduced, which enhances LSM-KVS performance by offloading compaction tasks to other edge servers. While EdgePilot boosts LSM-KVS performance and system efficiency by accelerating compaction, as discussed in Section IV, it suffers from energy consumption amplification. This issue arises from *compaction offloading over-provisioning*, where excessive resources are allocated to prevent write stalls, ul-

*These authors are first co-authors and have contributed equally.
†S. Park is the corresponding author.

timately increasing energy consumption.

The problem of compaction offloading over-provisioning underscores the trade-off between performance and energy efficiency in edge federation environments. Although using powerful server for compaction offloading minimizes the risk of write stalls, it leads to unnecessary energy waste when compaction demands are over-provisioned. Addressing this issue requires a more adaptive and energy-aware compaction strategy. By reducing over-provisioning, it is possible to mitigate energy consumption amplification while preserving the performance benefits of compaction offloading.

In this paper, we introduce ECO-KVS to enhance both performance and energy efficiency of LSM-KVS in edge federations. To the best of our knowledge, this is the first work to simultaneously address both performance and energy efficiency challenges of LSM-KVS in edge environments. We analyze the energy amplification caused by compaction offloading over-provisioning in edge settings and, based on this analysis, propose an energy-aware compaction offloading mechanism. This mechanism aims to prevent write stalls and reduce energy consumption by leveraging real-time, model-based predictions of compaction time and energy usage. We believe ECO-KVS can significantly improve energy efficiency and sustainability in edge environments, contributing to the progress of Green IT.

The contributions of our work are as follows.

- We propose ECO-KVS, an energy-aware compaction offloading mechanism designed to enhance energy efficiency in edge federations.
- We mitigate energy amplification caused by compaction offloading over-provisioning in ECO-KVS by predicting compaction processing time and energy usage through a real-time learning model, effectively alleviating write stalls.
- We implemented ECO-KVS on RocksDB v8.3.2, a widely used LSM-KVS. In representative experiments, ECO-KVS demonstrated a 21% increase in throughput compared to RocksDB and an 18% improvement in the performance-to-energy efficiency ratio over EdgePilot, the state-of-the-art solution in edge federation.

II. Background

In this section, we discuss the edge computing and the edge federation. Subsequently, we describe the Log-Structured Merge-tree, a representative data structure used in NoSQL databases employed within edge computing.

A. Edge Environments

Edge Computing. As depicted in the edge computing of Figure 1 (pink area), edge computing [3] is a technology that processes data at edge servers that are physically close to the edge devices where the data is generated. In contrast to centralized cloud computing, edge computing reduces bandwidth usage and latency, thereby being utilized in various applications that require real-time data processing, ranging from the Internet of Things (IoT), mobility, to machine learning. As described in Figure 1 (gray area), which illustrates the database and

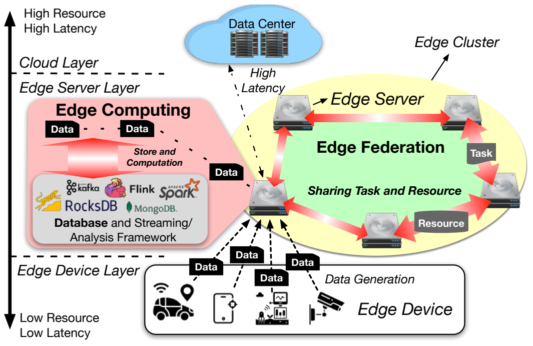


Fig. 1: Overview of edge computing and federation environments.

framework of edge computing, data streaming and analytical frameworks are employed to effectively process continuously generated data. To ensure low latency, databases are hosted on local storage within edge servers rather than in the cloud [6].

Limitations of Edge Computing and Edge Federation. The edge devices or edge servers that compose an edge environment have limited resources and lower computing power compared to the cloud infrastructures. Due to these constraints, edge computing encounters limitations when performing large-scale data processing or complex computations. To overcome the limitations of edge computing, an edge federation has emerged, which shares resources and distributes tasks among edge servers, as illustrated in the edge federation (green area) of Figure 1 [5]. Edge federation improves the scalability of edge computing and the overall performance of the system.

Importance of Energy Management in Edge Server. The limited resources of edge servers include not only lower computational power but also available energy. Therefore, it is essential to manage the constrained power efficiently on edge servers [12]. However, in an edge federation, task offloading and resource sharing to improve overall system performance increases the resource utilization of edge servers, which increases energy consumption [13]. An increase in energy consumption consequently leads to stability issues such as rising operational costs, server overheating, and reduced lifespan, thereby undermining the sustainability of edge federation systems [14]. Therefore, power consumption management in edge computing and federations is essential for long-term operational stability [15].

B. Log-Structured Merge-Tree

Component and Structure. The Log-Structured Merge-tree (LSM-tree) [16] is a data structure used as the engine for key-value stores, which are representative NoSQL databases such as RocksDB and LevelDB. Figure 2 illustrates the structure of an LSM-tree. The LSM-tree consists of two main components: the Memory Component and the Storage Component. Write requests are stored in the MemTable of the memory component in an append-only manner, which provides fast write performance. MemTable transitions into an Immutable MemTable once its size reaches a predefined threshold, than flushed to persistent storage as a Sorted String Table (SST)

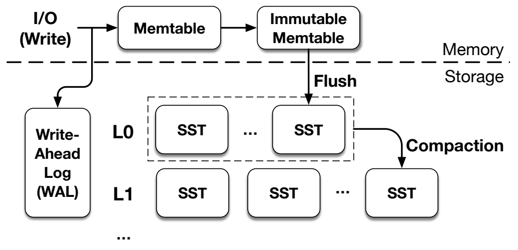


Fig. 2: An architecture of LSM tree.

file. At this point, the SST file is stored in level 0 ($L0$). When the size of $L0$ reaches a threshold, compaction is triggered. This process removes duplicate keys from multiple SST files, performs merge sort on them, and generates new SST files at the next level ($L1$).

Compaction and Write Stall. The LSM-tree ensures read performance by sorting the appended key-value pairs through compaction. However, since the key ranges of $L0$ SST files overlap with one another, if compaction is delayed, the number of SST files increases, leading to a degradation of read performance. If the input ratio is extremely high, causing SST files generated by flush operations to continue increasing despite ongoing compaction, the size of $L0$ may eventually reach the Write Stall Threshold (WST). At this point, a write stall occurs, blocking I/O operations until delayed compaction is completed to prevent further degradation of read performance. Write stalls degrade latency and throughput, making them a critical challenge for LSM-KVS. The causes of write stalls can be categorized into three types, including the $L0$ write stall, which occurs when $L0$ reaches the WST. In this paper, we focus exclusively on $L0$ write stalls, identified as the most critical cause [17], and the term write stall specifically refers to $L0$ write stalls throughout this study.

III. Related Works

In this section, before introducing our approach that addresses both performance and energy efficiency improvements for LSM-KVS in edge environments, we describe related works. These include (i) energy optimization in edge environments, (ii) compaction enhancements in LSM-KVS, and (iii) state-of-the-art approaches for optimizing LSM-KVS performance in edge environments.

A. Energy Optimization in Edge Environment

Energy optimization in edge servers is crucial in edge computing as it addresses high power consumption and overheating issues, thereby reducing operational costs and improving system reliability. Furthermore, with the growing emphasis on Green IT aimed at fostering a sustainable IT environment, energy optimization has become a key challenge in achieving both technological advancement and environmental conservation [2]. Research of energy optimization in edge computing can be categorized into hardware improvements and software optimizations [12]. Hardware improvements primarily focus on reducing overheating and decreasing energy consumption, while software optimization emphasizes enhancing the efficiency of systems and applications running on edge servers to minimize energy usage. For example, improving workload scheduling algorithms can optimize server resource utilization, thereby reducing unnecessary power consumption [18]. Additionally, efficient resource management and data processing optimization of databases and streaming systems ensure high performance while maintaining low energy consumption [19]. Research on energy optimization through software-based approaches supports Green IT goals while contributing to the sustainability and operational efficiency of edge computing.

B. Compaction Optimization in LSM-KVS

To improve compaction performance, two representative approaches include: (i) offloading compaction tasks to accelerators such as FPGAs and (ii) disaggregating computing resources used for compaction. (i) In the accelerator-based offloading approach, Sun et al. integrated FPGA compression offloading through software-hardware codesign [7]. Lim et al. proposed to utilize in-storage processing (ISP) in the compression process to reduce the amount of data movement required for FPGA acceleration [8]. (ii) As a resource disaggregation method, Nova-LSM [9] uses Remote Direct Memory Access (RDMA) to separate the storage and processing components in the cloud environment. Nova-LSM offloads compaction tasks to the storage component and dynamically adjusts the parallelism of compaction operations. ROCKSMASH [10] disaggregated metadata from local storage to cloud storage, performing compaction tasks in the cloud to reduce the compaction overhead on local storage servers. The aforementioned studies focus on reducing compaction execution time using accelerators or mitigating write stalls caused by compaction tasks via resource tiering. However, in edge computing environments, deploying accelerators such as FPGAs on heterogeneous and low-performance servers is challenging. Furthermore, centralized offloading to specific low-performance servers can degrade overall system performance.

C. State-of-the-art of LSM-KVS for Edge Federation

To enhance the performance of LSM-KVS in edge federations, the state-of-the-art approach EdgePilot [11] has been proposed. EdgePilot improves compaction processing time by offloading compaction tasks across heterogeneous edge servers. To achieve this, it compares the estimated local compaction execution time (T_{local}) on the server running the LSM-KVS with the time required to offload and execute compaction on another edge server (T_{offload}) and if T_{offload} is shorter than T_{local}, offloading is performed. In this process, among the edge servers comprising the edge federation, the target server with the fastest T_{offload} is identified, considering the availability of resources.

IV. Preliminary Study and Motivation

This section addresses the issue of increased energy consumption resulting from compaction offloading in edge federations and explores the underlying reasons, thereby explaining

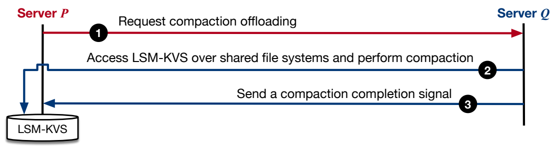

Server *P*
Request compaction offloading
Server *Q*
Access LSM-KVS over shared file systems and perform compaction
Send a compaction completion signal
LSM-KVS

Fig. 3: Flow of compaction offloading between two servers.

the motivation of this study. To clarify the terminology related to compaction offloading, we assume the presence of two servers, P and Q, and categorize compaction as follows.

- **Local Compaction**: Compaction offloading is not performed, each server handles the compaction tasks that occur locally on its own.
- **$P \rightarrow Q$ Compaction Offloading** : Compaction tasks generated on P are offloaded to Q, and Q processes P's compaction. During this time, Q also can perform its local compaction tasks.

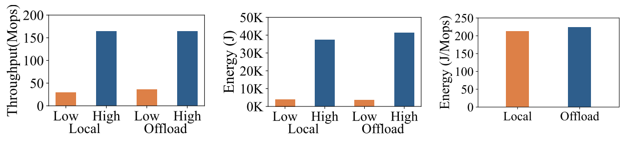
Figure 3 shows the flow of performing compaction offloading from P to Q. ❶ Request compaction offloading from P to Q. ❷ Q accesses the LSM-KVS of P through the shared file system and performs the requested compaction offloading from P. ❸ When the compaction is complete, it sends a completion signal to P and terminates.

Trade-off between Performance and Energy Consumption. First, we conducted a comparative analysis of the relationship between performance and energy consumption during compaction offloading. To compare energy usage, we measured energy consumption based on CPU utilization. Since compaction is a CPU-intensive operation, the energy difference resulting from compaction offloading primarily depends on CPU utilization, while energy differences caused by network and memory usage are negligible [20].

Figure 4 shows the throughput and energy results of compaction offloading when running LSM-KVS on two servers with different computing power (Low and High). We used the FillRandom workload provided by RocksDB's db_Bench. The experimental settings are detailed in Section VII-A, and the server specifications are presented in Table I. On the x-axis, *Local* indicates that both servers perform local compaction independently, while *Offload* indicates compaction offloading from the Low server to the High server.

First, Figure 4(a) shows the throughput results. When compaction offloading was performed, the throughput of the Low server improved by 20%, as compaction offloading alleviated write stalls. In contrast, the High server showed almost no change, indicating that handling additional compaction tasks from the Low server did not degrade its performance. Figure 4(b) presents the energy measurement results. While the energy consumption of the Low server remained nearly unchanged, that of the High server increased by 12% due to the additional energy required for compaction offloading. We refer to this phenomenon as *Energy Consumption Amplification* caused by compaction offloading.

Metric Definition: Energy / Operation. We have shown that while compaction offloading can enhance the performance of LSM-KVS, it introduces the issue of energy consumption amplification. Therefore, it is essential to consider what is critical for evaluating energy efficiency in LSM-KVS. We propose a metric called *Energy per Operation* (EpO). It represents the energy consumed per operation processed by LSM-KVS, where a lower value indicates better energy efficiency.



(a) Throughput (b) Energy (c) Energy/Ops

Fig. 4: Performance and energy consumption trade-off analysis of compaction offloading (a) Throughput, (b) Energy consumption, and (c) Energy/Ops. On the x-axis, Local indicates the Low and High servers perform local compaction, while Offload indicates Low $\rightarrow$ High compaction offloading, with the High server performing its local compaction.

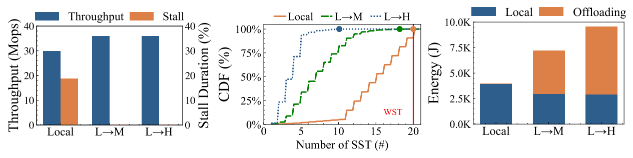
Figure 4(c) shows the EpO results for compaction offloading. The *Offload* has a 5% higher EpO than *Local*. While compaction offloading improves throughput, it results in a decrease in energy efficiency. Therefore, to achieve better energy efficiency relative to throughput, it is necessary to improve the EpO metric.

Compaction Offloading Over-Provisioning. Excessive energy consumption caused by over-provisioning in infrastructure poses significant constraints on operations [21]. This issue is no exception in edge environments, and we address over-provisioning in the context of compaction offloading.

Offloading compaction tasks to a server with higher computing power accelerates compaction and alleviates write stalls, which can improve throughput. Theoretically, according to Amdahl's law, compaction acceleration improves as the computing power of the offloading server increases, up to the point where acceleration limits are reached. However, as shown in Figures 4(b) and (c), higher computing power leads to energy consumption amplification, creating a trade-off between performance and energy efficiency.

To reduce write stalls, as mentioned in Section II-B, it is sufficient to ensure that the size of $L0$ does not reach the write stall threshold. Based on this, we hypothesized that there exists an optimal server with computing power that minimizes energy amplification while effectively reducing write stalls. Therefore, we analyzed the correlation between write stalls and the computing power of the offloading server during compaction offloading.

Figure 5 presents the results for the Low server when compaction tasks are offloaded to Medium and High servers, denoted as Low$\rightarrow$Medium and Low$\rightarrow$High. The experimental setup is identical to that in Figure 4. For convenience of explanation, we use 'L', 'M', and 'H' to represent Low, Medium, and High servers, when using the $\rightarrow$ notation. Figure 5(a) illustrates the throughput and write stall duration. In both L$\rightarrow$M and L$\rightarrow$H scenarios, the write stall on the Low server was completely eliminated, leading to identical improvements in throughput. This indicates that offloading compaction to a



(a) Throughput & Stall (b) Level 0 CDF (c) Energy

Fig. 5: Analysis of performance and energy consumption based on the compaction offloading server: (a) Throughput, (b) Write stall duration, and (c) Energy consumption. On the x-axis of (a) and (c), 'Local' indicates local compaction on the Low server, while 'L→M' and 'L→H' represent compaction offloading from Low to Medium and Low to High servers. In (b), circular markers indicate the point where the CDF reaches 100%.

Medium server is sufficient to avoid write stalls on the Low server.

To analyze the occurrence of write stalls in more detail, Figure 5(b) presents the CDF results for Level 0 of the Low server based on the compaction offloading server. While L→M shows an average of approximately 3 more files than L→H, both remain entirely below the WST (20). Figure 5(c) presents the energy consumption results, showing that energy amplification in L→H increased by 32% compared to L→M.

As a result, when compaction is offloaded from the Low server, both Medium and High servers effectively eliminate write stalls. However, due to the greater energy amplification observed in High server, Medium server is more suitable. Offloading to a High server results in performance-based *over-provisioning* for compaction offloading.

Motivation. Theoretically, for the same amount of work, higher CPU clock speeds result in higher voltage consumption [22], leading to increased energy consumption according to $E = \frac{1}{2}CV^2$ [23], where C is capacitance and V is CPU voltage. In other words, offloading compaction to a server with higher computing power to accelerate compaction inevitably leads to energy consumption amplification. However, we confirmed that avoiding compaction offloading over-provisioning can reduce this amplification. Therefore, when performing compaction offloading, the key to minimizing energy consumption amplification lies in offloading to a server that can mitigate write stalls while consuming less energy.

V. Problem Definition

In this section, we define the problem definition of mitigating write stalls and minimizing energy consumption amplification in LSM-KVS compaction offloading within edge federations, based on preliminary experimental results and motivation. The problem is defined under the following conditions: (i) To reduce the total energy consumption amplification caused by compaction offloading across all edge servers, it is essential to minimize the number of compaction offloading on each edge server. (ii) Compaction offloading should target edge servers where compaction can be accelerated to prevent write stalls on source servers, while avoiding over-provisioning of compaction tasks on the target servers.

To achieve this, we first identify the compaction offloading servers capable of avoiding write stalls. Since a write stall occurs when the L0 size reaches the write stall threshold (WST) before compaction is complete, it is dependent on the L0 size and compaction processing time. Therefore, we set the L0 size caused by compaction on each server.

Let N be the total set of edge server nodes, and let $T_i^{\text{cpt.-local}}$ be the time to perform local compaction when compaction occurs on a particular edge server node N_i.

Compaction Triggering Write Stall. The size of L0 after $T_i^{\text{cpt.-local}}$ (S_i^{L0}) is the sum of the current (S_i^{curr}) and the input size (S_i^{in}) during $T_i^{\text{cpt.-local}}$. We set S_i^{in} as a function of $T_i^{\text{cpt.}}$ because it is workload-dependent. S_i is defined as follows:

$$S_i^{\text{L0}}(T_i^{\text{cpt.-local}}) = S_i^{\text{curr}} + S_i^{\text{in}}(T_i^{\text{cpt.-local}}) \tag{1}$$

If $S_i(T_i^{\text{cpt.local}})$ reaches the write stall threshold size (S_i^{WST}) of N_i, a write stall will occur. The conditions for a write stall to occur are as follows.

$$S_i^{\text{L0}}(T_i^{\text{cpt.-local}}) = S_i^{\text{curr}} + S_i^{\text{in}}(T_i^{\text{cpt.-local}}) \geq S_i^{\text{WST}} \tag{2}$$

Offloading Candidate Server. To identify a compaction offloading server that can prevent write stalls, we assume that all servers are capable of compaction offloading and define $T^{\text{cpt.-offload}}$ as the set of compaction offloading times. Since $T^{\text{cpt.-offload}}$ depends on the computing power and resources of each server, it can be expressed as a function of N. Thus, $f_{\text{offload-time}} : N \rightarrow T^{\text{cpt.-offload}}$. At this point, the subset of servers, $N_{\text{no-stall}}$, that satisfy the conditions to prevent write stalls is defined as follows.

$$N_{\text{no-stall}} = \Big\{ N_i \in N, \;\Big|\; T_j \in T^{\text{cpt.-offload}}, \; i \neq j, \\ S_i^{\text{curr}} + S_i^{\text{in}}(T_j) < S_i^{\text{wst}} \Big\} \tag{3}$$

Minimize Energy Consumption Amplification. Since the energy consumption of compaction offloading is also dependent on each server, let the set of compaction offloading energy consumptions be denoted as $E^{\text{cpt.-offload}}$. Similarly, $E^{\text{cpt.-offload}}$ can be expressed as a one-to-one mapping function $f_{\text{offload-energy}} : N \rightarrow E^{\text{cpt.-offload}}$ with respect to N. The compaction offloading target server, N_{target}, which consumes the least energy, is defined as follows.

$$N_{\text{target}} = \underset{N_i}{\operatorname{argmin}}\, f_{\text{offload-energy}}(N_i) \tag{4}$$

For compactions occurring on each server (N_i), compaction offloading to N_{target} satisfying the Equation 4 can minimize the energy consumption amplification and reduce the energy consumption of all servers.

VI. Design of ECO-KVS

A. Design Goals

The design goals of ECO-KVS are as follows, to reduce write stalls and minimize energy consumption amplification through an energy-aware compaction offloading mechanism.

Prevention of Compaction Offloading Over-Provisioning. The ultimate goal of compaction offloading is to mitigate write stalls by accelerating compaction. However, compaction offloading involves energy consumption amplification, and compaction offloading over-provisioning increases energy consumption amplification. Therefore, avoiding compaction offloading over-provisioning reduces the energy consumption amplification, which in turn reduces the overall energy consumption of the edge cluster.

Prediction of Write Stall. Compaction offloading over-provisioning occurs when a target server for offloading is selected from the set $N_{\text{no-stall}}$, which satisfies Equation 3, but the selected server has excessively high computing power. In other words, to prevent over-provisioning, a server that can accelerate compaction such that S^{L0} does not reach S^{WST} after the compaction offloading is completed should be chosen. To achieve this, it is essential to predict when a write stall will occur based on the time point at which compaction is triggered. This prediction allows for the identification of $T^{\text{cpt.-offload}}$ that satisfies the conditions of $N_{\text{no-stall}}$. Therefore, by predicting the timing of write stall occurrences, compaction offloading can be directed to a server with appropriate computing power, thereby avoiding compaction offloading over-provisioning.

Prediction of Compaction Time and Energy Consumption. The acceleration of compaction through compaction offloading is fundamentally based on the assumption that it can reduce compaction processing time. In other words, to determine whether compaction time is reduced and by how much through compaction offloading, it is necessary to know the compaction time. Additionally, to assess the increase in energy consumption caused by compaction offloading, it is essential to predict the energy consumed during compaction. To address this, ECO-KVS introduces a model that learns in real-time based on data from each server, enabling the prediction of compaction processing time and energy consumption.

B. ECO-KVS *Architecture*

Figure 6 illustrates the architecture of ECO-KVS. ECO-KVS is designed with a *Local System* and a *Global System* to globally manage compactions occurring locally on each edge server. The Local System operates independently on every edge server, while the Global System runs on the master server, which acts as the coordinator among all edge servers. The master server operates both the Global System and the Local System. The Local System and Global System follow a Producer-Consumer Model, functioning as the Producer and Consumer, respectively.

1) Local System: The core function of the Local System is to identify whether Equation 2 is satisfied and to determine whether to perform compaction offloading. To achieve this, the Local System is composed of three modules: *Monitor*, *Local Predictor*, and *Offloader*, as shown in the gray area on the right side of Figure 6. As explained in Section II-B, L0 write stalls are influenced solely by the size of L0. Therefore, the Local System considers only L0 compaction.

Local Predictor. The Local System needs to determine $T^{\text{cpt.-local}}$, the compaction processing time represented by Equation 1, whenever a compaction occurs[1]. To achieve this, the Local System introduces a Local Predictor, which predicts the compaction processing time based on the compaction size included in the compaction log generated by the Local System. The compaction model used in the Local Predictor is based on our evaluation results and employs linear regression to express the processing time as $y = ax + b$ with respect to the compaction size. Here, a and b depend on the server's computing power and resources. Detailed reasoning and results supporting this model configuration are provided in Section VII. Thus, the Local System uses the Local Predictor to estimate $T^{\text{cpt.-local}}$ based on the compaction size.

Monitor. Additionally, the Local System must ultimately determine whether compaction offloading is necessary by identifying whether Equation 2 is satisfied after the $T^{\text{cpt.-local}}$ predicted by the Local Predictor. To achieve this, it is essential to calculate $S^{\text{in}}(T^{\text{cpt.-local}})$, which depends on S^{in}. The Local System introduces a monitor that uses an exponential moving average to analyze the trend of input ratios based on historical input ratio values and determine S^{in}. Therefore, the Local System identifies whether Equation 2 is satisfied by combining the $T^{\text{cpt.-local}}$ predicted by the Local Predictor and the S^{in} determined by the monitor.

Offloader. When it is determined by the Local Predictor and Monitor that compaction offloading is required, the Offloader is responsible for requesting and executing compaction offloading to the Global System. When requesting compaction offloading to the Global System, the Offloader passes as parameters the size of the compaction and $T^{\text{cpt.-offload}}$, that time to avoid write stall using the Equation 2. Then, if the Global System is given a suitable server that can avoid a write stall, it will run compaction offloading to that server.

2) Global System: The Global System of ECO-KVS, shown in the gray area on the left in Figure 6, is responsible for finding target servers that minimize energy amplification while avoiding write stalls for compaction offloading received from the Local System. The Global System consists of three modules: *Scheduler*, *Global Predictor*, and *Tracker*.

Scheduler. First, the Scheduler manages the compaction offloading requests received from the Local System as a request queue. Since it can receive requests from multiple Local Systems at the same time, it serializes them in the form of a message queue and processes them in order. The Scheduler manages the request queue to avoid concurrency problems, but note that the time it takes for the Global System to find a target server for compaction offloading is negligible, so the request queue does not fill up to the size that processing is delayed.

Tracker. The Tracker in the Global System performs the following two functions to identify an appropriate target server for the Local System: (i) It collects compaction data logs from the Local System and forwards the data, including the processing time and energy consumption for each server based on compaction size, to the Global Predictor. The Global Predictor uses the data provided by the Tracker to train the compaction model and energy model. (ii) It monitors CPU and network usage, which can affect compaction processing

[1]Hereafter, the index notation i is omitted.

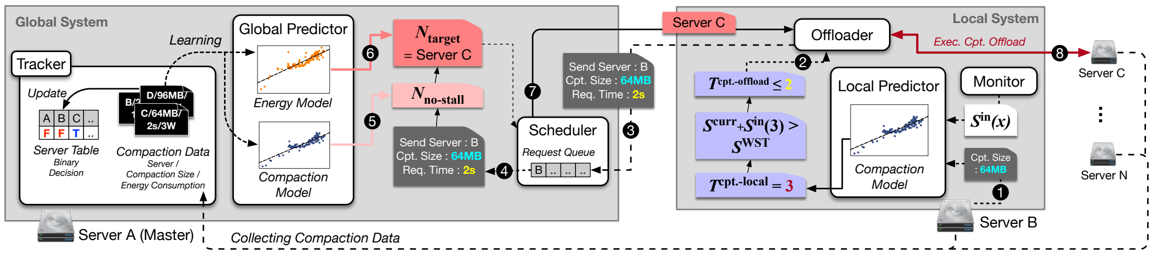


Fig. 6: An architecture of ECO-KVS. ECO-KVS consists of a *Global System* and a *Local System*.

time in the Local System, to manage a binary *Server Table* that identifies whether compaction offloading is feasible. Only servers with a *True* value in the Server Table can be selected as target servers for compaction offloading.

Global Predictor. The Global System uses the Global Predictor to find $N_{\text{no-stall}}$ and N_{target}. The Global Predictor uses the data from the Tracker to train the Compaction model and the Energy model. Like the Local Model, it uses a linear regression model and the detailed results are discussed in Section VII.

C. System Scenario

The operation scenario of ECO-KVS is explained using Figure 6. Consider a situation where Server A is the master among the edge servers and Server B requests compaction offloading. ❶ For the compaction size (64MB) that occurred on Server B, use the Local Predictor to predict the compaction processing time $T^{\text{cpt.-local}} = 3$. Also, use Monitor to calculate S_{in} based on an exponential moving average. If $S^{\text{curr}} + S^{\text{in}}(T^{\text{cpt.-local}}) \geq S^{\text{WST}}$, a write stall occurs, so compaction offloading is required to avoid write stalls. The minimum time $T^{\text{cpt.-offload}} = 2$ that a write stall can be avoided is given by the Equation 2. ❷ The $T^{\text{cpt.-offload}}$ is sent to the Offloader, and ❸ the Offloader sends a signal to the Global System, including the compaction size and $T^{\text{cpt.-offload}}$. ❹ The Global System's Scheduler manages the requests from the Local System in a request queue, and invokes the requests dequeued when it is time to process them. ❺ For the invoked request, the Global Predictor is used to identify $N_{\text{no-stall}}$, a set of servers capable of avoiding write stalls. ❻ Among $N_{\text{no-stall}}$, the target server N_{target} = Server C is selected to minimize energy consumption amplification. ❼ The Scheduler then sends N_{target} to the Offloader in the Local System. ❽ Finally, the Offloader performs compaction offloading to the target server. Following this ECO-KVS design scenario, each Local System can minimize energy amplification while avoiding write stalls. Consequently, the overall LSM-KVS performance of the edge servers is improved, and energy consumption amplification is reduced.

D. Implementation

ECO-KVS is implemented based on RocksDB v8.3.2. Compaction offloading was implemented using *StartV2()* and *WaitForCompleteV2()* provided for RocksDB's Remote Compaction. Global and Local Systems are embedded in RocksDB instances. On the Local System, Monitor uses *GetProperty()* API of RocksDB to detect the MemTable size changes every time window and calculate the input ratio based on the exponential moving average. The time window is set to 1 seconds. The Tracker in the Global System collects *ROCKS_LOG_INFO()*, a compaction log generated in the Local System upon the completion of a compaction. The *ROCKS_LOG_INFO()* includes the compaction size and processing time.

VII. Evaluation

A. Experimental Setup

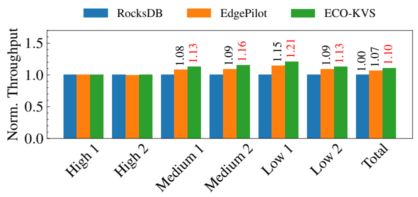
Setup. A heterogeneous server environment was configured for the experiments, with servers classified into three tiers (High, Medium, Low) based on their specifications. To simulate performance levels similar to edge servers, the number of cores was limited. Detailed experimental specifications of the servers are provided in Table I. Each server is connected via a 10 Gbps network, enabling access to the storage of other servers over the network. Also, perf was used to measure the energy consumption of the CPU.

We evaluated the performance of three systems, one of which is ECO-KVS.

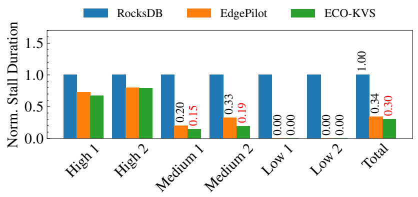
- **RocksDB:** As the baseline, we used the widely adopted LSM-KVS, RocksDB [24], version 8.3.2. RocksDB performs only local compaction without compaction offloading.
- **EdgePilot [11]:** A state-of-the-art system designed to enhance LSM-KVS performance in edge federations through compaction offloading. It is implemented using RocksDB version 8.3.2.
- **ECO-KVS:** Our proposed system, implemented using RocksDB version 8.3.2 to ensure a fair comparison with the baseline and EdgePilot.

TABLE I: Server specifications. The number of cores refers to the number of active cores.

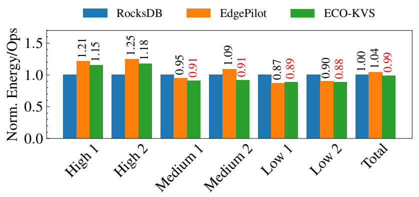
Server	CPU	Memory	Storage
High-1	Ryzen 9 7950X @ 4.5GHz Enable 8 Cores	32 GB (DDR 5)	Samsung 970 EVO 1 TB
High-2	Ryzen 9 7950X @ 4.5GHz Enable 8 Cores	32 GB (DDR 5)	Samsung 970 EVO 1 TB
Medium-1	Ryzen 5 2600 @ 3.4 GHz, Enable 6 Cores	16 GB (DDR4)	Samsung 970 EVO 512 GB
Medium-2	Ryzen 5 2600 @ 3.4 GHz Enable 6 Cores	16 GB (DDR4)	Samsung 970 EVO 512 GB
Low-1	Ryzen 7 1700 @ 1.5 GHz (downclocked) Enable 4 Cores	8 GB (DDR4)	Samsung 970 EVO 250 GB
Low-2	i7-6700 @ 1.5 GHz (downclocked) Enable 4 Cores	8 GB (DDR4)	Samsung PM981 512 GB



(a) Throughput

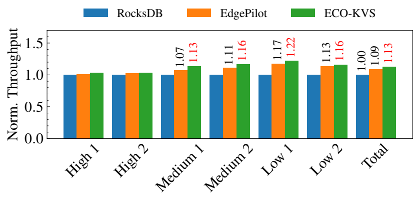


(b) Stall Duration

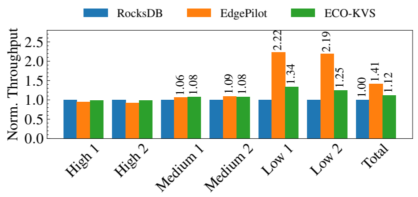


(c) Energy/Ops

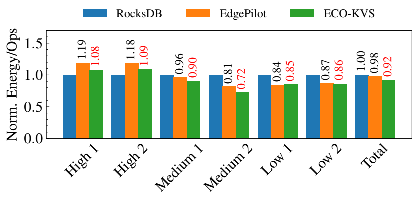
Fig. 7: FillRandom evaluation results: (a) Throughput, (b) Stall Duration, and (c) Energy per Operation.



(a) Throughput (Write)



(b) Throughput (Read)



(c) Energy/Ops

Fig. 8: ReadWhileWriting evaluation results: (a) Write Throughput, (b) Read Throughput, and (c) Energy per Operation.

Workload. For the evaluation, we executed the following three workloads from db_bench of RocksDB for 600 seconds each. The key size was 16B, and the value size was 1000B.

- FillRandom : A 100% write workload.
- ReadWhileWriting : A workload consisting of 100% writes and 100% reads, with writes and reads performed simultaneously by separate threads.
- MixGraph [25] : A workload designed to reflect real-world characteristics of key-value stores, consisting of Put (25%), Get (50%), and Seek (25%) operations. The value size follows a Generalized Pareto distribution with a mean of 1000B.

B. Analysis of Performance and Energy Efficiency

FillRandom. Figure 7 shows the results of evaluating each system with the FillRandom workload, normalized to the baseline (RocksDB). Figure 7(a) shows the throughput, where ECO-KVS achieved the highest throughput on both Medium and Low servers. Specifically, on the Low-1, ECO-KVS achieved a 21% improvement compared to RocksDB and a 6% improvement compared to EdgePilot. The total normalized throughput value of ECO-KVS showed a 10% increase compared to RocksDB and a 3% increase compared to EdgePilot. ECO-KVS demonstrated higher performance compared to EdgePilot, as it prevents over-provisioning of computing resources used for compaction. This approach ensures a more balanced distribution of compaction tasks and reduces the number of offloading operations, which reduces compaction offloading overhead and results in higher throughput compared to EdgePilot.

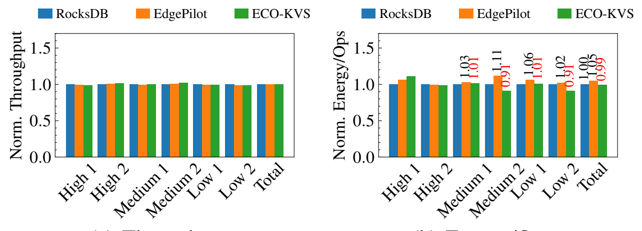
Figure 7(b) shows the write stall duration normalized to RocksDB. Across all servers, ECO-KVS consistently exhibits the lowest stall duration. Notably, on the Low servers, no write stalls occurred, and on the Medium-1, stall duration was reduced by up to 85%.

Figure 7(c) presents the EpO for each system. For High servers, both ECO-KVS and EdgePilot show higher EpO compared to RocksDB, as they only perform additional compaction without offloading. However, for Medium and Low servers, the EpO significantly decreased. ECO-KVS significantly reduced EpO on Medium and Low servers, especially on Medium-2 by up to 18% compared to EdgePilot. On High servers, both ECO-KVS and EdgePilot increased EpO due to additional computation offloading, but in total, ECO-KVS was 5% lower than EdgePilot and lower than RocksDB. EdgePilot over-provisions compaction, increasing compaction offloading to high and medium servers with high computing power, which increases EpO.

ReadWhileWriting. Figure 8 shows the performance of each system under the ReadWhileWriting workload. Figure 8(a) shows the write throughput of each system. The throughput trends for each server are similar to those observed in the FillRandom workload for the same reasons.

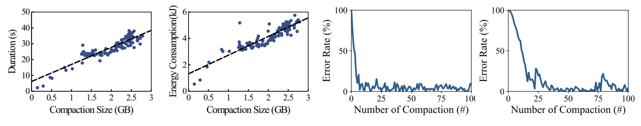
Figure 8(b) shows the read throughput of each system. Both ECO-KVS and EdgePilot outperformed RocksDB on all servers, especially on Low-1 showing a 34% and 122% improvement compared to RocksDB, respectively. EdgePilot has a higher read throughput than ECO-KVS because ECO-KVS minimizes compaction over-provisioning to a level that avoids write stalls, while EdgePilot only aims to maximize compaction acceleration. However, this difference is most pronounced on Low servers. Given that LSM-KVS is primarily used in workloads with heavy write operations, the increase in write performance and reduction in energy consumption offered by ECO-KVS are of substantial value, even when accounting for the slight decrease in read performance.

Figure 8(c) illustrates the normalized EpO for each system under the ReadWhileWriting workload. Compared to RocksDB, ECO-KVS achieved a reduction in EpO on Medium and Low servers, with a maximum decrease of 28% on Medium servers. Compared to EdgePilot, ECO-KVS also



(a) Throughput (b) Energy/Ops

Fig. 9: MixGraph evaluation results: (a) Throughput and (b) Energy per Operation.



(a) (b) (c) (d)

Fig. 10: Distribution by compaction size for (a) execution time and (b) energy consumption. Tracking the error rate changes in linear regression model for (c) execution time and (d) energy consumption.

reduced energy consumption on High and Medium servers, with a maximum reduction of 11% on High servers. The total normalized EpO of ECO-KVS decreased by 8% and 6% compared to RocksDB and EdgePilot.

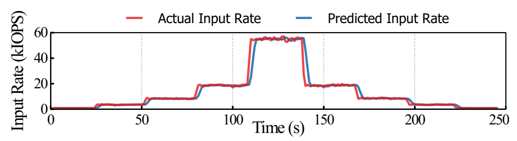
MixGraph. Figure 9 presents the evaluation results for the MixGraph workload. Figure 9(a) shows the throughput of each system. Since MixGraph is not a workload where write stalls occur frequently, the performance differences among the systems are minimal. Figure 9(b) presents the EpO for each system. Compared to RocksDB, ECO-KVS reduced energy consumption by up to 9% on Medium and Low servers. Compared to EdgePilot, ECO-KVS achieved up to an 20% reduction in energy consumption on Medium and Low servers. In total, ECO-KVS reduced total energy consumption by 6% compared to EdgePilot. In a workload where write stalls are infrequent, EdgePilot showed unnecessary energy consumption amplification without performance improvement, while ECO-KVS effectively avoided such energy consumption amplification.

C. Analysis of Prediction Model

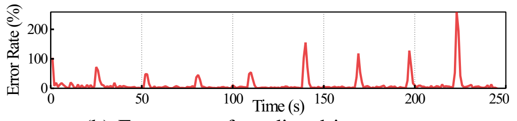
Lightweight Prediction Models. The architecture of ECO-KVS includes two prediction modules: the Local Predictor and the Global Predictor. The Local Predictor uses a compaction execution time prediction model to determine whether to perform compaction offloading. The Global Predictor utilizes both a compaction execution time prediction model and a compaction energy consumption prediction model to select the target server for compaction offloading.

Before selecting the prediction models for compaction execution time and energy consumption, we prioritized models with minimal execution time. This is because the prediction models are part of the critical path in the compaction offloading process of ECO-KVS. Figure 10(a) and (b) illustrate the changes in compaction time and energy consumption, as a function of compaction size during L0 compaction. From Figure 10, we confirmed that both compaction execution time and energy consumption increase proportionally with compaction size. Based on this observation, we adopted linear regression as the prediction method, as it is a simple model with the lowest energy consumption and latency during prediction.

Accuracy of Prediction Model. Figure 10(c) and (d) illustrates the change in error rate for each model as the number of training iterations increases. Figure 10(c) shows the error rate of the compaction execution time prediction model. The model was after 20 iterations, achieving an average error rate of 3.9%. Figure 10(d) presents the error rate of the compaction energy consumption prediction model, which exhibited an average error rate of 4.2% after the model was adapted. With only a small amount of compaction data, the prediction model stabilized, enabling highly accurate predictions of compaction time. As a result, the time and cost required to apply ECO-KVS to newly added servers in the edge federation are reduced, contributing to the high scalability and low architectural complexity of ECO-KVS.



(a) Actual input rate and predicted input rate



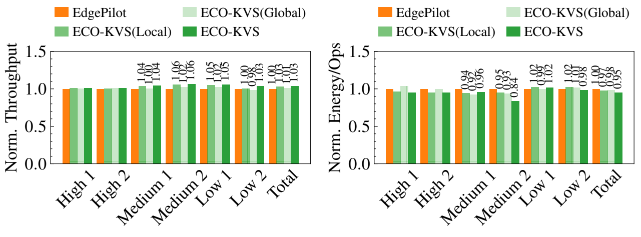
(b) Error rate of predicted input rate

Fig. 11: Tracking the input rate predict model: (a) Actual input rate and predicted input rate (b) Error rate of the predicted input rate.

D. Analysis of Input Rate Prediction

The Monitor in the Local System must predict the remaining time until a write stall occurs to support the Local Predictor's assessment of write stall occurrence. To achieve this, the Monitor employs a moving average, a method commonly used for forecasting irregular real-time data, to predict the input rate generated by edge devices.

Figure 11(a) shows the actual input rate and the input rate predicted using the moving average method. Additionally, Figure 11(b) presents the error rate between the actual values and the predicted values. In this evaluation, we set the window size to 3 seconds and the interval to 1 second for calculating the moving average. These values are configurations and can be adjusted based on the characteristics of the workload. When the actual input rate changes, the predicted input rate converges to the actual input rate within a short period of 3-5 seconds. During this time, the error rate spikes significantly but quickly decreases, stabilizing at an average error rate of approximately 3.6%. This scenario is a synthetic workload and the input rate prediction shows high robustness to rapid input ratio changes, so it should perform well in real-world workloads.



(a) Throughput (b) Energy/Ops

Fig. 12: Effectiveness of Global System and Local System in ECO-KVS: (a) Throughput and (b) Energy per Operation.

E. Analysis of the Impact of ECO-KVS Features

Figure 12 demonstrates the effectiveness of the approaches proposed in ECO-KVS. ECO-KVS(Local) applies only the Local System of ECO-KVS, while ECO-KVS(Global) applies only the Global System of ECO-KVS. The experiment was conducted using the FillRandom workload.

Figure 12(a) illustrates the differences in throughput among the systems. Compared to EdgePilot, ECO-KVS(Local), ECO-KVS(Global), and ECO-KVS all maintained differences within the margin of error for High servers. For Medium and Low servers, both ECO-KVS(Local) and ECO-KVS demonstrated a 5% performance improvement. The selective compaction offloading of the Local System reduces the load on high-performance servers handling offloaded compactions, significantly contributing to reduced write delays and increased throughput. Figure 12(b) shows the EpO. While the differences across individual servers vary slightly depending on the compaction offloading mechanism, in terms of total energy consumption across the cluster, ECO-KVS(Local) achieved a 3% reduction, ECO-KVS(Global) achieved a 2% reduction, and ECO-KVS achieved a 5% reduction compared to EdgePilot. Each system applied in ECO-KVS demonstrated a significant effect on reducing energy consumption.

VIII. Conclusion and Future Work

We proposed ECO-KVS, an energy-aware compaction offloading mechanism for LSM-KVS in edge federations. ECO-KVS optimizes compaction offloading by mitigating energy consumption through selective decisions at the Local and Global System. The Local System offloads compaction when write stalls are anticipated, while the Global System selects servers that minimize energy consumption and avoid write stalls. ECO-KVS improves throughput by up to 21% compared to RocksDB and enhances EpO by up to 18% over EdgePilot with minimal overhead.

Our approach, although designed for edge federations, has the potential to extend beyond edge environments. By leveraging its adaptive and energy-aware compaction strategies, ECO-KVS can be applied to diverse infrastructures such as data centers and cloud where LSM-KVS is widely utilized. This opens opportunities for broader performance improvements and energy optimization in various systems.

Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (RS-2024-00453929) (RS-2024-00416666).

References

[1] International Energy Agency (IEA), "Electricity 2024." https://www.iea.org/reports/electricity-2024, 2024.
[2] M. Vitali and B. Pernici, "A survey on energy efficiency in information systems," *International Journal of Cooperative Information Systems*, vol. 23, no. 03, p. 1450001, 2014.
[3] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.
[4] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Generation Computer Systems*, vol. 97, pp. 219–235, 2019.
[5] X. Cao, G. Tang, D. Guo, Y. Li, and W. Zhang, "Edge federation: Towards an integrated service provisioning model," *IEEE/ACM Trans. Netw.*, vol. 28, p. 1116–1129, jun 2020.
[6] Y. Yang, Q. Cao, and H. Jiang, "Edgedb: An efficient time-series database for edge computing," *IEEE Access*, vol. 7, pp. 142295–142307, 2019.
[7] X. Sun, J. Yu, Z. Zhou, and C. J. Xue, "Fpga-based compaction engine for accelerating lsm-tree key-value stores," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pp. 1261–1272, 2020.
[8] M. Lim, J. Jung, and D. Shin, "Lsm-tree compaction acceleration using in-storage processing," in *2021 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pp. 1–3, 2021.
[9] H. Huang and S. Ghandeharizadeh, "Nova-lsm: A distributed, component-based lsm-tree key-value store," in *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, p. 749–763, Association for Computing Machinery, 2021.
[10] P. Xu, N. Zhao, J. Wan, W. Liu, S. Chen, Y. Zhou, H. Albahar, H. Liu, L. Tang, and C. Xie, "Building a fast and efficient lsm-tree store by integrating local storage with cloud storage," in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 125–134, 2021.
[11] J. Kim, H. Yoo, S. Lee, H. Byun, and S. Park, "Coordinating compaction between lsm-tree based key-value stores for edge federation," in *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*, pp. 419–429, 2024.
[12] C. Jiang, T. Fan, H. Gao, W. Shi, L. Liu, C. Cérin, and J. Wan, "Energy aware edge computing: A survey," *Computer Communications*, vol. 151, pp. 556–580, 2020.
[13] M. V. Barbera, S. Kosta, A. Mei, and J. Stefa, "To offload or not to offload? the bandwidth and energy costs of mobile cloud computing," in *2013 Proceedings Ieee Infocom*, pp. 1285–1293, IEEE, 2013.
[14] K. Haghshenas, B. Setz, Y. Blosch, and M. Aiello, "Enough hot air: the role of immersion cooling," *Energy Informatics*, vol. 6, no. 1, p. 14, 2023.
[15] W. E. Gnibga, A. Blavette, and A.-C. Orgerie, "Latency, energy and carbon aware collaborative resource allocation with consolidation and qos degradation strategies in edge computing," in *2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 2630–2639, IEEE, 2023.
[16] P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil, "The Log-Structured Merge-Tree (LSM-tree)," *Acta Informatica*, vol. 33, no. 4, pp. 351–385, 1996.
[17] J. Yu, S. H. Noh, Y. ri Choi, and C. J. Xue, "ADOC: Automatically harmonizing dataflow between components in Log-Structured Key-Value stores for improved performance," in *21st USENIX Conference on File and Storage Technologies (FAST 23)*, (Santa Clara, CA), pp. 65–80, USENIX Association, Feb. 2023.
[18] L. Gu, J. Cai, D. Zeng, Y. Zhang, H. Jin, and W. Dai, "Energy efficient task allocation and energy scheduling in green energy powered edge computing," *Future Generation Computer Systems*, vol. 95, pp. 89–99, 2019.
[19] M. D. de Assuncao, A. da Silva Veith, and R. Buyya, "Distributed data stream processing and edge computing: A survey on resource elasticity and future directions," *Journal of Network and Computer Applications*, vol. 103, pp. 1–17, 2018.
[20] G. Liao, X. Zhu, S. Larsen, L. Bhuyan, and R. Huggahalli, "Understanding power efficiency of tcp/ip packet processing over 10gbe," in *2010 18th IEEE Symposium on High Performance Interconnects*, pp. 32–39, 2010.
[21] V. Villebonnet, G. Da Costa, L. Lefèvre, J.-M. Pierson, and P. Stolf, "Energy aware dynamic provisioning for heterogeneous data centers," in *2016 28th international symposium on computer architecture and high performance computing (SBAC-PAD)*, pp. 206–213, IEEE, 2016.
[22] I. Takouna, W. Dawoud, and C. Meinel, "Accurate mutlicore processor power models for power-aware resource management," in *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, pp. 419–426, IEEE, 2011.
[23] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power cmos digital design," *IEICE Transactions on Electronics*, vol. 75, no. 4, pp. 371–382, 1992.
[24] Facebook, "Rocksdb: A persistent key-value store for fast storage environment." https://rocksdb.org, 2012.
[25] Z. Cao, S. Dong, S. Vemuri, and D. H. Du, "Characterizing, modeling, and benchmarking rocksdb key-value workloads at facebook," in *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pp. 209–223, 2020.