Reducing Cross-Cloud/Region Costs with the Auto-Configuring MACARON Cache

Hojin Park[1] Ziyue Qiu[1,2] Gregory R. Ganger[1] George Amvrosiadis[1]
[1]Carnegie Mellon University [2]Uber

Abstract

An increasing demand for cross-cloud and cross-region data access is bringing forth challenges related to high data transfer costs and latency. In response, we introduce Macaron, an auto-configuring cache system designed to minimize cost for remote data access. A key insight behind Macaron is that cloud cache size is tied to cost, not hardware limits, shifting the way we think about cache design and eviction policies. Macaron dynamically configures cache size and utilizes a mix of cloud storage types, in order to adapt to workload changes and reduce cloud costs. We demonstrate that Macaron can reduce cross-cloud workload costs by 65% and cross-region costs by 67%, mainly by reducing outgoing data transfer and by leveraging object storage alongside DRAM to reduce cache capacity cost.

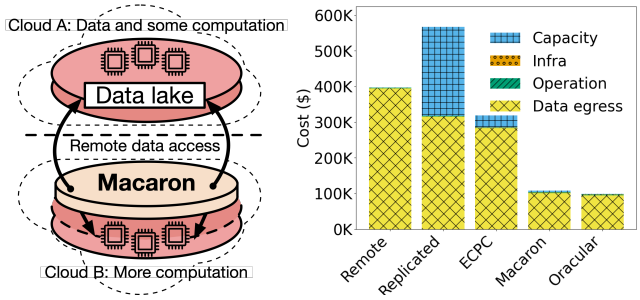
1 Introduction

Demand for multi-cloud is surging [1–3], driven by factors including: disparities among cloud provider features [4, 5], the desire to avoid vendor lock-in [6–8], and evolving organizational structures, such as consolidations [9]. Similarly, multi-region solutions (within a cloud provider) are becoming more popular due to data sovereignty requirements [10–12], service latency reduction [13, 14], and availability.

Despite many efforts to optimize resource use within a cloud or region [15–23], achieving cost-efficiency across clouds and regions remains a prominent challenge [24–28] hindering the adoption of multi-cloud/region strategies. Current cross-cloud/region data access solutions often lead to substantial increases in overall costs. Organizations typically resort to direct data access across clouds or regions [29–31], which results in prohibitively high data egress cost for frequently accessed data and high data access latency. While many organizations address the latency issue through data replication [32–36], the costs associated with maintaining replicas and synchronization data egress remain substantial.

SOSP '24, November 4–6, 2024, Austin, TX, USA

ACM ISBN 979-8-4007-1251-7/24/11
https://doi.org/10.1145/3694715.3695972



(a) Macaron in multi-cloud **(b)** Costs for each approach

Figure 1. Multi-cloud/region workloads are challenging to run cost-efficiently. Early industry adopters rely on accessing all data remotely or replicating it locally, with some using an existing in-memory caching cloud service (ECPC). Macaron significantly reduces costs compared to these methods and achieves costs comparable to offline optimal (Oracular). The experiment details are provided in §7.2.

Public cloud providers and third parties offer caching services that can be used to keep only hot data locally [37–42], but those rely on manual configuration of the cache size.

We analyzed object storage traces from Uber, VMware, and the IBM cloud [43], and derived three insights for cache design. First, we find that manual selection of cache capacity and storage type can lead to cost-inefficient decisions, whereas strategies that support re-configuration of the cache size and type can optimize cost and performance. Second, the high data egress costs of workloads skewed toward lower accesses per object highlight the need for large cache capacities that are feasible only when cheap cloud storage types are used. Third, diverse and dynamic access patterns within and across traces make it essential to monitor workloads and automatically reconfigure the cache when needed.

In response to these findings we introduce **Macaron**, an auto-configuring cache system that monitors the workload, dynamically adjusts cache capacity, and utilizes different cloud storage types to minimize data access costs and latency from remote data lakes (Fig. 1a). A key insight behind Macaron is that cache sizes in the cloud are constrained by cost rather than hardware constraints, causing us to rethink cache design and eviction policy. Macaron analyzes the need to add new objects to the cache and adjust cache capacity accordingly, while exploiting cheap storage types for caching,

rather than solely relying on a cache replacement policy [44–54] or efficient sharding of limited storage hardware among applications [55–63].

Macaron leverages two storage types for caching. Object storage capacity cost is considerably lower than the egress cost incurred by cache misses, so Macaron uses it to minimize the overall cost of accessing remote data. A distributed DRAM component is elastically adjusted to ensure acceptable latency. The capacity of each tier is periodically assessed and adjusted based on data access patterns, using a version of the miniature simulation technique [64], modified to obtain byte miss curves and average latency curves. We also explored a variant of Macaron that adopts a cache policy using time-to-live (TTL) for eviction, adjusting TTL instead of capacity. Our results confirmed that both approaches achieve similar cost savings. A serverless implementation allows Macaron to achieve performance suitable for online miniature simulation of very large cache capacities.

Fig. 1b shows that Macaron significantly improves upon the cost of existing approaches. Each bar represents the total cost of running the 19 cloud-based object storage workloads analyzed in this paper (see §3.2) across clouds. Macaron achieves 73% cost reduction compared to accessing all data remotely by avoiding egress costs, and a 81% cost reduction compared to replicating all data locally by reducing both capacity and synchronization-driven egress costs. Macaron also achieves 66% cost reduction compared to elastic cloud provider caching (ECPC) services, which incur expensive DRAM storage costs even when tuned intelligently. An oracular approach with perfect knowledge of future accesses only improves cost savings by an additional 9%, compared to Macaron, without latency reduction.

We have evaluated Macaron with traces from IBM, Uber, and VMware, and our results show the importance of adapting the cache size and configuration to workload changes and that substantial cost savings can be achieved by combining object storage and an elastic DRAM cache cluster.

Contributions. (1) We collected cloud storage workload traces from Uber and VMware, and publicly released Uber trace [65]. (2) We analyze real-world cloud storage workloads and derive design objectives for effectively caching these workloads. (3) We describe the Macaron cache that is adaptively auto-configured to minimize costs without compromising latency by leveraging object storage as a cache storage type. (4) We experimentally demonstrate Macaron's ability to achieve 65% reduction on average in remote data access costs compared to existing solutions. (5) Lastly, we release the Macaron prototype [66] and simulator [67] code.

2 Motivation and Challenges

Cross-region data access within a cloud provider is gaining prevalence for several reasons. First, computation and data could end up separated due to resource unavailability within a region [68, 69], e.g., due to high demand for GPUs. Second, new services are usually not available in all regions simultaneously, requiring applications to span regions in order to adopt new technologies [70, 71]. Finally, data sharing via cross-region data access is essential for international business teams [72] and applications can be distributed across multiple regions to provide lower latency to end-users [73].

Operation	AWS	Azure	GCP
Egress to Internet (per GB)	9¢	8.7¢	11¢
Egress btw. regions (per GB)	2¢	2¢	2¢
Object storage (per GB-mo.)	2.3¢	2.1¢	2.3¢
DRAM (per GB-mo.)	700-1200¢		
Object GET (per 1k requests)	0.04¢	0.05¢	0.04¢
Object PUT (per 1k requests)	0.5¢	0.65¢	0.5¢

Table 1. Cloud storage pricing[1] of three public cloud providers is similar, with egress cost dwarfing other costs.

Recent research [74, 75] has shown that distributing a data pipeline across multiple clouds saves costs due to price differences between providers, as demand for multi-cloud solutions is rising. A recent survey [76] found that 55% of multi-cloud users already deploy a single workload across multiple clouds, which often requires cross-cloud data transfers. In conversations with a major trading company we found that during workload bursts, they utilize multiple cloud providers to ensure trading performance scales with demand even when resource availability becomes an issue for one cloud provider [77]. Other companies have to split data to abide by data residency rules preventing data movement [78].

While multi-region/cloud strategies have clear advantages, we identify two important challenges: high data egress cost, and increased data access latency.

Challenge 1: Prohibitive data egress cost. While transferring data into public clouds is often free, moving data out incurs substantial charges based on the volume of data being transferred (Table 1). For instance, one of the IBM traces we have analyzed accesses 694TB in a week, resulting in cross-cloud data transfer costs of $64K/week or $3.3M/year. The same workload would incur $14K/week or $0.73M/year if data was transferred across regions of the same cloud.

Despite being controlled by public cloud providers, data egress costs have remained stable over time – GCP, AWS, and Azure have maintained their egress costs unchanged for the past 6 to 10 years. Moreover, egress costs have been consistently identified as a barrier to cross-region/cloud adoption in many surveys [76, 79]. *Reducing the data egress cost is crucial for embracing the multi-region/cloud era.*

Challenge 2: High access latency. In latency-sensitive workloads, like real-time analytics or streaming services, encountering consistently high latency is unacceptable [80, 81].

[1] Prices from N. Virginia region, <10 TB egress to the Internet, inter-region transfers within N. America, and <50 TB storage capacity.

Trace	Operation %		Skewness	Total	Data accessed		Remarks
	Put	Get	(Zipf factor)	data size	Put	Get	
IBM 9	N/A	100	0.22	6 TB	0	34 TB	Short lifetime: last access - first access < 10min
IBM 12	1	99	0.97	5 TB	4 TB	603 TB	High data access repetitiveness
IBM 18	2	98	0.64	4 TB	231 GB	14 TB	High request rate, small object sizes
IBM 55	55	45	0.42	13 TB	12 TB	10 TB	Strong diurnal access pattern
IBM 83	40	60	0.72	64 TB	37 TB	94 TB	Low compulsory miss ratio (=0.12)
IBM 96	58	42	0.20	78 TB	46 TB	36 TB	High compulsory miss ratio (=0.87)
Uber	N/A	100	0.52	324 TB	0	941 TB	Stable data access pattern
VMware	N/A	100	0.47	215 GB	0	71 TB	Small total data size, high request rate for testing

Table 2. We collected and analyzed new traces from Uber and VMware to understand how to efficiently cache cloud storage workloads. The IBM traces represent diverse access patters among the busiest cloud object storage traces from IBM's repository.

Even in less latency-sensitive workloads, high data access latency can increase costs by increasing runtimes and causing workloads to use compute resources for longer periods [24].

Cross-region data access causes higher latency compared to single-region access. We measured object retrieval times at a public cloud and found that retrieving a 1KB object from local object storage in one U.S. region showed significantly lower latency, taking 10s of milliseconds, than fetching the same object from another U.S. region or Europe, which took 100s of milliseconds. Real-world workloads we evaluated experienced $2-5\times$ higher average latency with cross-region data access. *High data access latency to remote data should be mitigated for both performance and cost.*

3 Macaron Design Drivers

In this section, we analyze existing approaches for cross-cloud/region data access, and real-world cloud object storage workloads from three large companies. We derive three design objectives that have inspired the design of Macaron.

3.1 Limitations of Current Approaches

The simplest approach to accessing data across clouds or regions is remote access, requiring no additional synchronization efforts. However, in scenarios with repetitive data access patterns, as observed in various storage workloads [54, 82–84], egress costs are repeatedly incurred for the same objects, alongside latency issues.

One effective approach to reduce latency is to replicate all data and access them from local object storage, which eliminates recurring data egress costs as well. However, it does not solve the cost problem entirely, as the transfer cost to synchronize *dark data* [85–87] (i.e., data that is written once but never accessed) inflates the egress cost. Recent surveys [88–91] have indicated that the percentage of dark data can range from 40% to as high as 95% across different organizations. Even worse, maintaining a large capacity of replicated data lake is also expensive.

We view the above two patterns as endpoints of a spectrum, with caching solutions providing a middle ground. While using existing cloud caching services appears straightforward, no service currently offers cost optimization solutions for addressing high egress costs associated with remote data access. Users must manually configure cache settings, such as cache capacity and storage type, a task that can be challenging even for experienced cache experts. Moreover, most existing services prioritize using DRAM or block storage for caching, primarily focusing on single-region data access performance, but our evaluation confirms that such approaches remain costly due to expensive capacity expenses.

Design objective 1: *We need caching strategies that bridge the gap between all-remote data access and full data replication. These strategies should support auto-configuration of the cache to optimize both cost and performance.*

3.2 Cloud Object Storage Workload Characteristics

To better understand how cloud object storage workloads should be cached, we analyzed IBM cloud object storage traces [43], along with traces we collected from Uber and VMware. These traces are collected from systems operating within a single region. The Uber and VMware traces represent workload accesses that are not expected to change substantially when moved to an architecture that spans regions or clouds. Additionally, we evaluated diverse workload patterns using IBM traces to broaden our evaluation, extrapolating these workloads to the cross-region and cross-cloud scenarios described in §2. More details about how we collected traces are in Appendix A.1.

IBM traces. These are anonymized object access logs from IBM cloud's object storage over a 7-day period. We identified 15 traces [2] with the most traffic, together making up 95% of all data accessed across all 98 IBM traces. While we have studied all 15 traces, for brevity we present detailed results for 6 traces that are representative of all unique workload characteristics that appear in the trace collection (Table 2).

Uber trace. We collected object access logs generated from Uber's Presto production deployment, primarily used for processing and analyzing large-scale real-time event data

[2] IBM traces with IDs 4, 9, 11, 12, 18, 27, 34, 45, 55, 58, 66, 75, 80, 83, 96.

streams, and querying data streamed through Apache Kafka to provide real-time data insights [92]. Our logs span three Presto engines and over 18 days. Over 70% of the accesses are generated by periodic jobs.

VMware trace. We collected AWS S3 requests generated by AWS Athena queries from VMware's test infrastructure, spanning an 8-day period. These analytics queries, comprising a mix of ad-hoc and scheduled jobs, analyze security data and resemble queries in the production system but are smaller in scale as part of testing before deployment in production [93]. This workload exhibits a high data request rate despite its relatively small dataset size.

Our analysis of these workloads helped us derive two design objectives for caching cloud object storage workloads.

Large objects and higher spread of accesses. Data accesses often follow the Zipf distribution [83, 84, 94–96], and we have confirmed that to be a good fit for our cloud object storage traces as well. Zipf's exponent α [97] represents the skewness in data access frequency per object. Higher α values indicate fewer objects receiving most accesses, so smaller cache capacities suffice for these workloads.

We find that cloud object storage workloads generally have lower α than those of KV-store or block I/O traces. Over 78% of IBM workloads and both Uber and VMware workloads have $\alpha < 0.6$, while more than half of Twitter KV-store traces [84] have $\alpha > 1.1$. Thus, while previously studied workloads achieve acceptable cache miss ratios with small cache sizes relative to the overall dataset, cloud object storage workloads need to cache a significant portion of data to mitigate bytes missed, which directly affects egress costs.

Many objects in cloud object storage workloads are large. For example, the IBM traces' median object is 10-100KB, while for Twitter it is 20-30B, orders of magnitude smaller. Given that data access frequency being skewed towards few accesses per object, this suggests that reducing egress costs through caching requires a large cache capacity.

Design objective 2: *Cloud object storage workloads are skewed towards low accesses per object, so to reduce high data egress cost we need large cache capacities, which are feasible by leveraging cheap storage types.*

Diverse and dynamic data access patterns. Cloud object storage serves as backend storage for a variety of workloads including machine learning [98–101], ETL [102, 103], SQL queries [104–106], and file serving [107–109], resulting in diverse data access patterns. Understanding the unique characteristics of each trace in Table 2 is vital for establishing a cost-efficient cache configuration. For instance, while IBM 96 is larger than IBM 83, the difference in data access skewness necessitates larger cache capacities for IBM 83, which are cost-efficient only using cheaper cloud storage. IBM 9, despite having low data access skewness, does not benefit from large cache capacity due to its short-lived objects, necessitating a different approach for cost-efficiency.

Despite thorough observation and understanding of workload characteristics, a cost-efficient cache configuration varies over time. For traces with more dynamicity, like IBM 80, dynamically adjust cache size results in 85% cost reduction compared to a statically configured cache. But even for traces with stable, periodic data access patterns, like Uber, a dynamically configured cache can result in a 15% cost reduction.

Design objective 3: *Cache re-configuration based on workload monitoring is necessary to accommodate diverse and evolving data access patterns.*

4 Macaron Design

Macaron is an auto-configured cache system designed to minimize the cost of remotely accessing data stored across clouds or regions, while ensuring acceptable latency. Macaron intelligently configures the cache storage type and adaptively adjusts cache size by periodically analyzing data access patterns. We elaborate on the key characteristics guiding Macaron's design (§4.1) addressing the design objectives from Section 3, then describe Macaron's architecture (§4.2).

4.1 Macaron Design Characteristics

Adopting object storage for cache storage type. While object storage is typically used for data lake storage [101, 110, 111], Macaron uses it as a second-level cache storage type. Cloud-based object storage workloads, as detailed in §3.2, are often cache-unfriendly [112–114], necessitating a large cache capacity to mitigate costly egress expenses. By leveraging the remarkably low object storage capacity cost (300× cheaper than DRAM), Macaron can provision extensive cache capacity cost-effectively, and still reduce overall costs through lower egress transfer costs.

With DRAM-based caches, however, neither opting for a large cache size nor settling for a smaller one presents a satisfactory solution to minimize costs; the former incurs prohibitively high capacity costs, while the latter results in a significant total miss penalty (i.e., egress costs). Therefore, as a first-level cache, Macaron uses the smallest DRAM cache cluster size that meets performance requirements.

While flash caching is often used as an inexpensive storage type [115–117], we leave exploring other options for future work, as object storage remains significantly cheaper and its inherent elasticity aligns better with adaptive cache reconfiguration. Otherwise, Macaron would need to manage a virtual storage cluster for flash caching. By default, Macaron uses standard object storage types like S3 Standard or Azure Blob Storage Hot Tier for caching, with support for other types by adjusting the cost policy.

Adaptive cache reconfiguration. Macaron periodically analyzes data access patterns and adjusts capacity of each

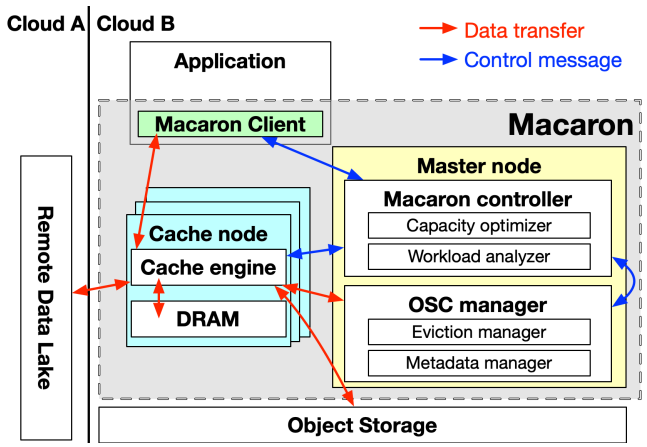


Figure 2. Macaron Overview: an Object Storage Cache (OSC) manages data egress costs, Cache nodes leverage DRAM to improve latency, and the Macaron controller is responsible for cache auto-configuration.

cache level, assuming patterns will repeat in the future, similar to prior work [55, 118, 119]. Macaron determines a cost-efficient object storage cache capacity that minimizes overall remote data access costs and a DRAM cluster size that meets acceptable average latency. Our evaluation confirms that frequent optimization (every 15 minutes) leads to more cost-efficient solutions (§7.3), enabled by cloud resource elasticity and Macaron's rapid workload analysis. To achieve the latter, we extend miniature simulation techniques [64] and implement them in serverless functions. Achieving highly accurate workload prediction [120, 121] falls beyond the scope of our work, yet our evaluation demonstrates Macaron's robustness in handling real-world workloads, even when unobserved workload patterns emerge (§7.2).

4.2 Macaron Architecture

Macaron consists of four components: a Macaron client, a cache engine, an object storage cache manager, and the Macaron controller (Fig. 2). Macaron uses two-level caching. As the first-level cache, a cache engine and DRAM cache scale together across cache nodes that make up a cache cluster. An object storage cache (OSC), the second-level cache of Macaron, is controlled by the OSC manager. Macaron uses inclusive caching, where data in the cache cluster is also stored in the OSC.

The **Macaron client** is the primary interface for applications, facilitating a connection to Macaron and the transmission of data requests, such as put, get, and delete operations to a remote data lake. It employs consistent hashing for message routing to the cache cluster that is auto-scaled. The Macaron client maintains up-to-date cache cluster information by communicating with the Macaron controller to determine which node to access.

Upon receiving requests from the Macaron client, the **Cache Engine cluster** interacts with both cache layers and

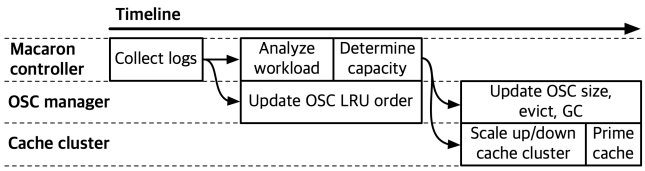


Figure 3. End-to-end pipelining of cache capacity optimization and reconfiguration. Macaron executes processes in parallel wherever feasible for fast reconfiguration.

the remote data lake. Macaron uses a write-through and inclusive policy by default, so the cache engines are responsible for cache promotion.When Macaron targets solely minimizing the cost, the cache cluster is not deployed, and the cache engine is co-located in the same node as the Macaron client.

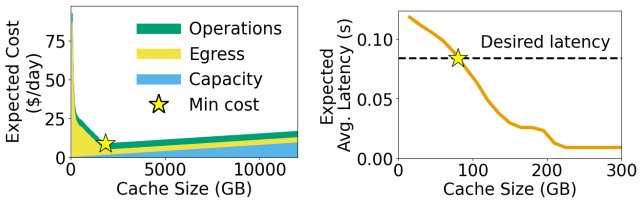
While the DRAM cache is self-managed, Macaron deploys an **OSC manager** to manage metadata and cache eviction from the OSC. To reduce operational costs of OSC, Macaron uses object packing that combines small objects into packing blocks when writing cache items into the OSC. OSC manager's metadata manager provides the mapping of cache objects to the corresponding packing block. Macaron lazily evicts cache items from the OSC using the Eviction Manager. We use the LRU eviction policy for both the OSC and DRAM cache, but alternative policies can be easily incorporated.

Finally, the **Macaron controller** is responsible for adaptive cache management. Its optimizer determines the sizes of both the OSC and the cache cluster to minimize cost while striving to enhance performance based on past data access patterns. Then it scales both cache layers accordingly. The Macaron controller gains insights into data access patterns through the Workload Analyzer that periodically collects and analyzes data access logs.

Supported operations. The Put operation synchronously writes data to the packing block being constructed, the DRAM cache (if present), and the remote data lake, before returning to the client. The packing block is asynchronously flushed to the OSC in the background. This ensures data durability for the remote data lake. The Get operation attempts to retrieve data from the DRAM cache, the OSC, and the remote data lake in sequence, returning it immediately upon success. The Delete operation removes data from Macaron's DRAM cache, OSC, and the remote data lake before returning.

4.3 Consistency model

Macaron is designed to guarantee the same consistency model as using the remote data lake alone. To do so, Macaron assumes data is immutable, a paradigm prevalent in data lakes such as Meta and Alibaba data warehouses [122, 123], AWS S3 lakeFS [124], and in cloud database formats like Apache Parquet and Apache Iceberg. With its write-through policy, Macaron ensures consistency equivalent to using the remote data lake alone. Applications requiring mutable data must handle it, potentially by using TTL for each item or



(a) Expected cost curve **(b)** Expected avg. latency curve

Figure 4. Example curves (Trace 55) utilized to optimize OSC and the cache cluster capacities.

functions like S3 Object Lambda to detect and invalidate stale data.

5 Cache Auto-configuration Pipeline

We introduce the workflow used by Macaron to adaptively optimize the capacity of each cache level by analyzing data access patterns for both cost and performance.

Workflow. Macaron triggers the reconfiguration process at fixed intervals, set to 15 minutes by default. When reconfiguration is triggered (Fig. 3): the Macaron controller first collects data access logs from cache engines, then the Workload Analyzer (§5.2) generates key metrics on the recent data access pattern (e.g., miss ratio curve, byte miss curve), and the OSC manager updates the LRU cache item list. Based on these metrics, the Macaron controller determines the most cost-efficient capacity and reconfigures the OSC and cache cluster accordingly (§5.1).

5.1 Capacity optimizer

OSC capacity. Macaron determines the OSC capacity that minimizes the overall cost of accessing remote data across clouds or regions. The capacity optimizer generates an *expected cost curve* based on past data access patterns, predicting the expected cost for different OSC capacities during the next time window and selects the size that minimizes the expected cost (Fig. 4a). The expected cost for a cache capacity (C) is computed as:

$$\begin{aligned} TotalCost(C) &= OSCCapacityCost(C + GarbageSize) \\ &\quad + EgressCost(C) + OpCost(C) \\ EgressCost(C) &= EgressPrice \times ByteMissCurve(C) \\ OpCost(C) &= PutPrice \times \\ &\quad \left(\frac{\#Writes + \#Reads \times MissRatioCurve(C)}{\#Objects\ per\ Packing\ Block} \right) \end{aligned}$$

In summary, the expected total cost consists of the object storage capacity cost, egress cost, and operation cost of object storage. Capacity cost is based on OSC size and garbage size, a side effect of object packing, tracked by the OSC manager, which monitors total data stored in OSC and its determined capacity. Egress cost is proportional to the bytes missed from the OSC. Operational costs for storing cache items to OSC are proportional to the number of Put operations and cache admissions, divided by the number of packed objects, since admitted objects are written to the OSC as a block. The costs unaffected by changing the OSC capacity are omitted in this formula, including the VM cost for Macaron controller, data transfer costs incurred by write operations (due to the write-through policy). As capacity increases, capacity cost increases while operation cost and egress cost decrease, attributed to the reduction in miss ratio and byte miss in OSC.

Cache cluster capacity. Macaron aims to configure the minimal cache cluster capacity needed to achieve better average latency than the replication approach. Macaron utilizes the average latency curve (Fig. 4b) to select the minimum cache cluster capacity that meets the desired latency threshold. However, in traces with high cold miss ratios, achieving this objective may not always be feasible. In such cases, the Macaron controller uses a maximum curvature method [125] to identify the knee-point. It connects the latency-cache size curve's two endpoints and locates the farthest point between the curve and this line, beyond which further expansion of the cluster size yields no latency improvement.

TTL cache for OSC. Given that there is no capacity limit in object storage, implementing a TTL cache on the object storage is another viable option. To assess whether Macaron's techniques can be applied to adaptive TTL-based object storage caching, we implemented Macaron-TTL, a variant of Macaron that uses a TTL cache and automatically determines a TTL that minimizes the total cost of accessing remote data. This variant employs the same Workload Analyzer to generate necessary metrics for computing the expected cost for TTL instead of cache size, considering a similar trade-offs: a TTL that is too short increases cache misses and egress costs, while one that is too long raises storage expenses. Our evaluation, as shown in §7.8, confirms that cost savings achieved by Macaron-TTL are nearly identical to those from optimizing OSC capacity. Further details of Macaron-TTL algorithm are available in Appendix B.

5.2 Workload Analyzer

Macaron uses a short optimization window, set to 15 minutes, to leverage the cost benefits of frequent reconfigurations, which requires fast yet accurate workload analysis. The Workload Analyzer adopts and extends the miniature simulation technique [64] to derive three key metrics representing the recent access pattern: miss ratio curve (MRC), byte miss curve (BMC), and average latency curve (ALC). Then, using accumulated metrics from historical access patterns, Macaron generates aggregated metrics for capacity optimization. For Macaron-TTL, the same curves are used but with TTL on the X-axis instead of cache size.

Miniature Simulation. Waldspurger et al. [64] introduced a technique for generating MRCs that emulates caches

of any specified size by proportionally scaling down the actual cache size and using spatial sampling to sample data accesses. Among MRC generation studies [95, 126–130], we chose this method for its efficiency in generating MRCs online and its adaptability in computing additional metrics utilized by Macaron, such as missed bytes or average latency.

We follow the original miniature simulation approach to obtain the MRC, and monitor cache miss bytes from the mini-caches, dividing them by the sampling ratio to approximate miss bytes of the original cache sizes, thereby generating the BMC. This process deviates only slightly from a full simulation, with a mean absolute error of 0.0023 for the MRC and a mean average percentage error of 0.015 for the BMC, evaluating all 19 traces.

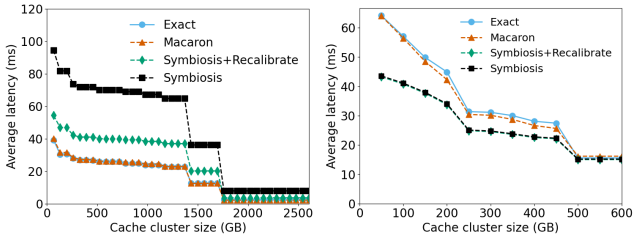
Symbiosis [55] uses miniature simulation to generate a MRC and produces an ALC based on the MRC to auto-tune application and kernel cache sizes. It uses access latency measured at the beginning and hit ratios measured at runtime of each cache layer to calculate average latency. However, Macaron considers two more factors, workload change and false positive hits, improving accuracy further.

First, Symbiosis assumes cache and disk latency do not change, but we observe that the access latency distribution to object storage varies over time since it depends on the object size distribution, which varies over time. Second, when uncached data are consecutively accessed within a very short time (before a remote access completes), Macaron's cache engine delays subsequent accesses until the first request completes to reduce redundant egress costs, causing them to experience remote access latency. However, in simulation, subsequent accesses after the first one are classified as hits, underestimating latency by using the cache cluster latency and generating an ALC with lower latency values.

To resolve these issues, Macaron directly computes average latency for each access during miniature simulation and aggregates them afterward, and we added the request delay in the simulation used by Macaron. Moreover, we ran two-level mini-caches that depict the cache cluster and OSC, using the same sampling and scaling logic. The cache cluster capacity serves as the independent variable for ALC, while OSC's cache capacity works as input to ALC optimization, since it is decided by the Macaron controller.

Fig. 5a illustrates the accuracy of Macaron factoring in workload changes compared to Symbiosis that uses the unchanged measured latencies for 7 days. In this case, the workload changes from accessing large objects to small objects, thus Symbiosis yields inaccurately higher latency (black). Symbiosis behaves much better once we force it to recalibrate every 15 minutes, but still worse than Macaron. Fig. 5b demonstrates the effect of false positive hits, where Symbiosis reports inaccurately lower average latency.

Metric Aggregation. After analyzing the local data access pattern, the Workload Analyzer stores the results for future



(a) ALC of IBM 55 **(b)** ALC of IBM 12

Figure 5. Macaron's ALC achieves high accuracy by computing the latency at runtime and incorporating proper request delaying, closely matching the exact average latency.

reference. It aggregates those saved local metrics according to the optimization goal.

To optimize costs, retaining historical data access patterns is crucial due to the high data egress cost, which aligns with the cost incurred for storing the same object in object storage over extended periods, specifically 116 days for cross-cloud and 26 days for cross-region accesses. Therefore, conservatively including old data access patterns is a key for cost-efficiency. To achieve this goal and adapt to workload changes, we use an exponential decay mechanism by multiplying metrics by a decay factor $\gamma^{elapsed\_time}$, which diminishes the influence of older metrics. This is simple yet yields effective results (§7.3). Additionally, we also multiply weights proportional to the number of requests per reconfiguration window.This prevents metrics derived from a small number of requests from misrepresenting the overall data access pattern.

To optimize performance, however, only the latest access pattern is important as Macaron needs to quickly scale-in the cache cluster when the large cache cluster capacity is ineffective. Thus, Macaron uses the latest ALC to configure the cache cluster capacity.

5.3 Policy during observation period

Macaron starts to trigger optimization after the cache is warmed up and stable data access patterns are observed, using the first day as the observation period. During the observation period, Macaron can either cache all accessed data or none at all. We find that storing all accessed data led to a significant reduction in the cost of remote data access, averaging at 37% compared to not storing any data. The main reason for this is twofold: (1) object storage cache is cheap, so storing all data for 24 hours does not incur significant overhead, and (2) on the first day, if no data is cached, the egress cost for repetitively accessed data is very high. Comparing to the use of optimal cache capacity during the observation period did not yield significant cost savings either.

5.4 Offline optimal algorithm

To assess Macaron's cost-efficiency, we compare it with the optimal solution, `Oracular`, which has complete knowledge

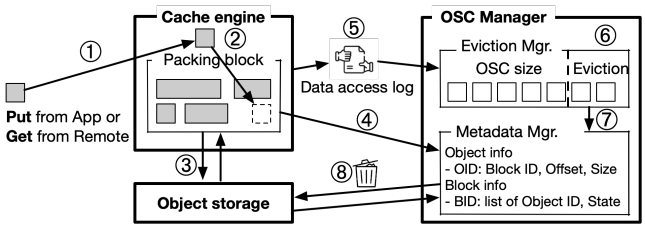


Figure 6. Overview of how OSC manager manages object packing, lazy eviction, and garbage collection.

of trace requests. While the Belady algorithm [131] is known to be the optimal eviction algorithm, OSC differs in two main ways: (1) its elastic nature, eliminating the need for forced evictions, and (2) its focus on overall cost rather than just miss ratio. Given these differences, `Oracular` determines for each cache item access whether the cost to store data in the OSC until the next access is less than the data transfer cost. If higher, the item might be evicted or not stored.

For our comparison, we assume zero operation cost for `Oracular`, suggesting optimal packing and minimal operation costs. Also, we take the end of the trace length as the end of the workload for `Oracular`, but real-world workloads continue beyond trace lengths. Hence, `Oracular` stands as an idealized benchmark that Macaron aims to approach, even if it may not be reachable.

6 Implementation

We implemented the Macaron prototype in 8k LoC of C++, and we explain the important implementation details that influence performance and cost.

6.1 Object storage cache implementation

Object packing. Object storage write operations are 12.5-13× more expensive than reads. However, Macaron has to perform frequent writes for cache admissions, driven by the low skewness of object storage traces. Macaron mitigates this issue through object packing [21], bundling small objects into larger blocks before writing them to object storage. Macaron's Cache Engine (Fig. 6 ①-④) combines objects that need to be written in OSC into blocks, then full blocks are written to OSC, and OSC manager metadata is updated to map blocks to objects. The Cache Engine uses byte-range fetches to retrieve objects from blocks.

By default, Macaron sets a packing threshold of up to 40 objects and a block size of 16MB. The workloads we studied break data in up to 4MB objects for caching, for which object packing can achieve 4× operation cost reduction, while smaller objects see reductions up to 40×. Larger block sizes reduce request costs but increase memory consumption on the cache nodes, which store data at block granularity.

Lazy eviction and garbage collection. Traditional caches evict items when reaching physical capacity. Macaron exploits object storage elasticity, delaying evictions and batch processing them to reduce operational costs. When eviction is triggered (Fig. 6 ⑤-⑧), the OSC manager leverages access logs to update its state of OSC objects to `Evicted`. Lazy evictions are followed by garbage collection for blocks with over 50% `Evicted` or `Deleted` objects, where a block is read and a new block is written out to OSC containing only active objects. Note that Macaron does not traverse all blocks for garbage collection. Instead, when a Delete or Evict occurs, it computes the percentage of valid items in each affected block. If the percentage falls below a threshold, the block ID is added to the *GCList* for tracking. Only the blocks in this list are targeted during garbage collection.

Lazy evictions resolve performance issues related to updating cache replacement policy metadata [43, 132, 133], by removing it from the critical path of requests.

6.2 Cache cluster implementation

DRAM cache priming. The speed at which Macaron can warm up new cache capacity following an auto-scaling event is crucial for efficiently utilizing new cache nodes.

We observe that many object storage workloads, especially IBM traces, have lower object request rates than the other key-value or block I/O workloads. For example, while average data request rates of Twitter [84] and Enterprise VDI storage traces [134] are 7k and 33k RPS, IBM traces do not exceed 344 RPS. This disparity is likely due to the high latency and monetary costs associated with retrieving data from cloud object storage, prompting users to maximize the use of fetched data. Hence, new cache capacity will get populated slowly, reducing our ability to mitigate latency spikes through existing methods like a Gradual algorithm [135].

To address this, Macaron incorporates cache priming for newly launched cache nodes. During this process, the OSC manager scans the LRU order of cache items and preloads data into new cache nodes until they are full.

6.3 Macaron controller implementation

Miniature simulation. Using spatial sampling at a ratio of 5%, we ran at most 200 mini-caches (ghost caches), with the largest covering the total data size of each workload[3]. Macaron deploys each mini-cache as a serverless function, running 200 simulations in parallel for rapid analysis (§7.7). To avoid replaying all accumulated traces in each optimization, Macaron stores the states of each mini-cache in Amazon EFS after simulation, loads the states back for subsequent simulation, and updates them during simulation execution. Metrics like miss ratio, bytes missed, and average latency, generated by each mini-cache during each optimization window, are also stored in EFS for use by the Macaron controller's capacity decision. As detailed in §5.2, Macaron runs two types of miniature simulation: one to generate MRC and BMC in a single run, and another to produce the ALC.

[3]We used uniform intervals between mini-cache sizes, with the smallest mini-cache size to cover at least 50GB cache simulation.

The pay-as-you-go pricing model offers cost and performance benefits for running miniature simulations on serverless functions instead of the master node. With serverless functions, costs are incurred only during active periods of execution. Serverless functions run 31 seconds (average across traces) for each 15 minutes optimization window. This makes them more cost-efficient than dedicated instances, due to the memory usage required by miniature simulation, even with sampling. Also, running 200 simulations quickly within a short optimization window requires high parallelism, and provisioning a large dedicated instance that is used only for short intervals would incur higher costs. We evaluate the cost and performance overheads in §7.7.

Scaling caches. To scale the OSC, the Macaron controller leverages the elasticity of object storage by storing more cache items to the OSC or deleting objects as needed. For scaling the cache cluster, the Macaron controller tracks the number of cache nodes, deploying new ones or terminating existing ones when the cluster size changes. It then communicates with Macaron clients to update their cache cluster information for consistent hashing-based message routing. For improved load balancing, availability, and scalability, advanced shard managers like Google's Slicer [136] or Meta's Shard Manager [137] could be employed.

7 Evaluation

We evaluate Macaron using real-world object storage workloads to assess the following aspects: its cost-efficiency compared to existing approaches (§7.2), its adaptability to workload changes (§7.3), cost-breakdown of Macaron's optimization techniques (§7.4), its ability to utilize the cache cluster cost-efficiently to achieve the desired performance (§7.5), its robustness under varying cloud conditions through sensitivity analysis (§7.6), simulation accuracy and prototype reconfiguration overhead (§7.7), and its TTL-based variant's cost-efficiency (§7.8).

7.1 Experimental setup

Traces. We evaluate Macaron with 15 IBM traces, 3 Uber traces, and 1 VMware trace. For brevity, we provide detailed analysis of results obtained from 6 IBM, 1 Uber, and 1 VMware trace (Table 2). We use the first day of each trace as the observation period, with optimizations triggered every 15 minutes after the first day. Each evaluation reports the remote data access cost and latency for the remaining days.

For the IBM traces, as in the original paper [43], large objects are divided into 4MB blocks, with each smaller object treated as a separate cache item. We use the same policy for the VMware traces, while for Uber we use 1MB blocks, which is their default policy.

Configurations. Unless stated otherwise, experiments assume workloads running on a different cloud provider than the one hosting the remote data lake, with accesses incurring cross-cloud egress charges. Workloads are located in the N. Virginia region, while the remote data lake is in N. California, but we did perform a sensitivy analysis considering different configurations (§7.6). We use AWS's pricing model, but note that cloud providers have similar pricing models.

Baselines and costs. We compare Macaron to three baselines mirroring approaches used today (§3.1): accessing all data from a `Remote` cloud, having all data `Replicated` locally, and using existing in-memory caching solutions (`ECPC`) like AWS ElastiCache. Since `ECPC` products rely on users to provide scaling policies, we use Macaron's optimizer to efficiently auto-scale the cache. Finally, we evaluate Macaron against the `Oracular` caching solution (§5.4).

At a high-level, `Remote` incurs egress and operation costs for all data accesses, while `Replicated` is plagued by synchronization costs. Egress and capacity costs for `Replicated` are computed based on the rate of increase in total data size, using a 90-day data retention period and 70% dark data, but different portions of dark data are explored too (§7.6). We exclude operation costs for `Oracular`'s object storage cache (§6). All others incur infrastructure costs for OSC manager and/or Macaron controller, with Macaron and `ECPC` incurring additional serverless function expenses.

Macaron Simulator. Replaying all traces once in a real cloud would cost over $1.5 million (Fig. 1b). Thus, we developed a simulator that allows us to assess Macaron across various configurations and constraints. The simulator models key components, replicating their functionalities (§4) and interactions. The simulator manages message exchanges between components, including data and control requests for reconfiguration and eviction, ensuring messages are generated in the same way as by our prototype implementation.

To simulate message latencies accurately, we measured data access latencies on AWS for various object sizes accessed from a remote data lake, OSC, and the cache cluster, and fit a Gamma distribution to the collected data. Cloud resource types and cache software we used for evaluation are detailed in Appendix A.2 and detailed latency generator validation is in Appendix A.5 and §7.7. We have open-sourced the simulator code [67].

7.2 Cost-efficiency Analysis

Observation 1: *For individual traces, Macaron reduces cross-cloud data access costs by 65% and 75% on average compared to* `Remote` *and* `Replicated`*, respectively.*

Macaron aims to minimize costs when accessing cross-cloud/region data. We compare remote data access costs of Macaron with those of our baselines. Fig.7 shows results for two representative IBM traces for brevity, with a discussion of the overall results provided below. See Appendix A.3 for the individual results of all traces and additional case studies.

Overall results. Across 19 traces, Macaron achieves a cost reduction of 2.4% to 99.3% (avg. 65%) compared to `Remote`,

and 24.9% to 82.9% (avg. 75%) compared to Replicated in cross-cloud scenarios (Fig.7b). In cross-region scenarios, for the 16 traces that have lower than 20% compulsory miss ratios, Macaron provides cost reductions of 28.2% to 98.5% (avg. 67.4%) and 18.1% to 91.1% (avg. 78.4%) compared to Remote and Replicated, respectively (Fig.7a). In cross-region scenarios, for the IBM 27, 66, and 96 traces that have high compulsory miss ratios (57%, 79%, 87%), Macaron incurs 24%, 5.8%, and 1.5% higher costs compared to Remote, respectively, because the savings achieved are less than the cost of running a single VM to operate Macaron controller and OSC manager. Since cross-cloud egress cost (9¢/GB) is higher than cross-region (2¢/GB), Macaron selects a larger cache capacity for cross-cloud scenarios, allowing for further reduction of egress costs.

Comparison with ECPC. When caching in object storage, the Macaron optimizer exploits its low capacity cost and mitigates egress costs by provisioning high capacities. When caching in DRAM, however, cache capacity costs increase rapidly, leading to a much smaller capacity point for cost optimization. ECPC, using DRAM, incurs higher capacity costs even with small capacities, along with increased egress costs compared to Macaron. As a result, across 19 traces, Macaron reduces overall costs by 3.5–89.1% (avg. 46%) compared to ECPC for cross-cloud data accesses.

Remote vs. Replicated. Fig. 7 shows that neither Remote nor Replicated consistently outperforms the other. For cross-cloud scenarios, 6 traces are cheaper to manage with Remote, the remaining 13 show cost savings with Replicated, and Macaron outperforms both across all traces.

Comparison with Oracular. Oracular leverages future knowledge for optimal caching decisions, while Macaron relies on past access patterns to predict the future. Still, this results in Oracular accessing cross-cloud data at 0.4%-18.3% lower cost (avg. 6.8%) than Macaron across all traces.

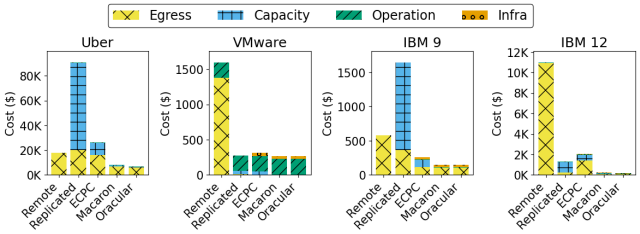
In IBM 12, Oracular's cost is 18% lower than that of Macaron due to misprediction of workload behavior for cross-cloud scenario. For days 1-5, the workload consistently accesses half of the previous day's data, adding an equal amount of new data. On day 6, it accesses data from day 2, causing unexpected cache evictions and remote re-fetching accesses, incurring egress costs.

Despite uncertain workload behaviors, Macaron remains more cost-efficient than the baselines. Monitoring longer-term data access patterns might reduce this uncertainty.

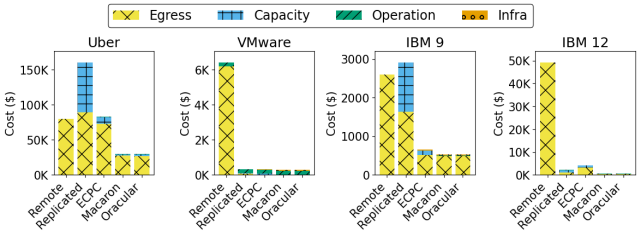
Observation 2: *Macaron's cost reduction stems from aggressively reducing data egress costs by exploiting cheap storage to build large caches, while finding the cache size that minimizes capacity costs.*

We provide a detailed case study for each workload shown in Figure 7b. Additional case studies are in Appendix A.3.

The VMware workload involves running numerous tests periodically on the test dataset, leading to a high frequency



(a) Cross-region data access



(b) Cross-cloud data access

Figure 7. Detailed analysis on four workloads representing diverse data access patterns.

of repetitive accesses. Thus, there is a 96% cost reduction compared to Remote, and 25% reduction compared to Replicated.

The Uber workload has the largest data size among 19 traces, and Macaron achieves 81% cost reduction compared to Replicated. Macaron finds a cache capacity of 180TB (56% of total data) yields benefit by avoiding egress transfers.

The IBM 9 workload exhibits a periodic burst for 15 minutes every hour, during which new data retrieval is followed by repeated access. The Workload Analyzer identifies this pattern, provisioning only 1% of the total data size to cache all the data accessed during each burst. This results in 79% cost reduction compared to Remote, managing repetitive accesses from OSC, and 82% reduction compared to Replicated due to the small cache capacity used.

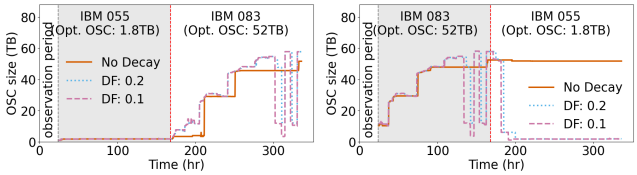
For the IBM 12 workload, Macaron achieves 98.9% reduction in data egress costs compared to Remote, due to strong cache locality. Over 50% of objects are accessed more than 100 times, making caching highly effective. Using Replicated is still expensive due to the 101× higher storage cost compared to caching only hot data identified by Macaron.

7.3 Impact of Adaptivity Mechanisms

Observation 3: *Macaron's adaptive reconfiguration reduces costs by 12% compared to static configurations. When workloads change, Macaron decays its knowledge leading to an additional 5% cost reduction compared to no decaying.*

Macaron optimizes configurations every 15 minutes to match the latest data access patterns. Here, we quantify the benefits of this frequent reconfiguration, and Macaron's mechanism for decaying older learned access patterns.

Reconfiguration window. We evaluate the benefit of Macaron's frequent reconfiguration, by comparing it to a static configuration that is fixed to the optimal capacity obtained from the first day of the trace. For cross-cloud scenarios,



(a) No decaying leading to high egress cost.

(b) No decaying leading to high capacity cost.

Figure 8. Testing Macaron's adaptivity to a workload change (vertical red line), with and without knowledge decaying.

Macaron achieves cost reduction 0-85% (avg. 12%) across 19 traces, while for cross-region, it is 0-78% (avg. 8%). When Macaron's reconfiguration window is reduced from 24 hours to 15 minutes, cost is reduced by 0-41% (avg. 4%) for cross-cloud and 0-25% (avg. 3%) for cross-region scenarios.

Exponential decay. By default, Macaron uses a decay factor of 0.2 (i.e., $\gamma^{1day} = 0.2$) to phase out learned patterns. We assess Macaron's cost-efficiency under varying decay factors – 1.0 (`NoDecay`), 0.2 (`Default`), 0.1 (`SmallDecay`) – using 15 IBM traces. Specifically, we evaluate its performance (1) within a single trace, and (2) when concatenating two different traces to simulate abrupt changes in data access patterns often observed in real-world workloads [138–141].

For a single trace, the IBM and VMware workloads span one week, and 18 days for Uber, and exhibit fairly consistent data access patterns. This benefits knowledge accumulated over time. Specifically, our traces show insignificant differences of ±1% with and without knowledge decaying.

To assess Macaron's adaptivity during workload changes we created 30 new concatenated workloads by combining the 6 selected IBM traces and assessed evaluated costs during the second trace's execution to see how Macaron adapts to changes in data access patterns. For 25 concatenated workloads, `Default` and `SmallDecay` reduce costs by 0-30% (avg. 5.2%) and 0-34% (avg.6.1%), respectively, compared to `NoDecay`, which continues to rely on past traces and hampers quick adaptation. For example, combining IBM 55 and IBM 83 in Fig 8 results in `NoDecay` facing high egress costs when executing IBM 83 after IBM 55 due to slow scaling out, while changing the order leads to expensive capacity costs because of slow scaling in. However, `Default` and `SmallDecay` can adapt rapidly to such changes.

We observed that for five concatenated workloads[4], `NoDecay` incurred lower overall costs. This was due to the fortuitous alignment of the workloads' unpredictable access patterns with `NoDecay`'s lack of adaptability, preventing it from slowly reducing capacity and retaining unnecessary cache items.

7.4 Effects of Macaron Optimizations

Macaron's cost savings are primarily attributed to two key optimizations: (1) determining the cost-efficient OSC size and (2) packing small objects when caching them in the OSC. Next, we assess the effectiveness of these optimizations.

Observation 4: *While the cost-efficient cache size varies significantly for each workload, Macaron identifies efficient setups by analyzing each access pattern. Making a less optimal choice can lead to increased costs.*

OSC size optimization. Figure 9 depicts the OSC capacity changes over the last six days (after observation period) across 15 IBM traces, with the total data size. The ratio of OSC capacity to data size varies across workloads, ranging 1-98%, highlighting that there is no single ratio ensuring a cost-efficient cache size, and the need for a tool like Macaron.

For IBM 18, Macaron aligns cache capacity with the total data size, suggesting a very large cache can mitigate overall costs by minimizing data egress fees. Unlike conventional caching studies, which favor compact, frequently-accessed data caches, the prohibitively expensive egress costs in cross-cloud, cross-region settings advocate for larger caches.

We found that all traces except one among the 19 evaluated workloads adjusted the cost-efficient OSC capacity at least once, with a standard deviation in the changing OSC size to total data size ratio ranging 0-0.28. The average standard deviation value is 0.1, suggesting that the ratio changed 10% per day, emphasizing the importance of Macaron's adaptivity. This capacity ratio typically increases from day 1 to 7.

Fig. 10 demonstrates that erroneous OSC capacity allocations can notably affect cost. Using the same 14% cost-efficient capacity ratio from IBM 55 on IBM 83 causes a 1.5× uptick in expected cost relative to Macaron's selection.

Observation 5: *Object packing, especially for workloads with small objects and high request rates, can yield significant savings, with up to a 36% cost reduction.*

Object packing. In our evaluation, IBM 18 and IBM 45 realized cost reductions of 36% and 5%, respectively, due to object packing. Traces with smaller objects and higher request rates, like these two, tend to benefit the most. Similar to Amdahl's law, the higher the contribution of operational costs to the total costs, the greater the potential savings, as object packing impacts only operational costs. Though operational costs average 4% of total costs in cross-cloud scenarios due to high egress expenses (resulting in 3% cost savings), factoring in cross-region egress prices increases operational costs to 8% and savings to 7%.

7.5 Macaron with low latency

Observation 6: *Macaron achieves 61% lower latency and 64% cost savings than* `Replicated` *with its dynamic cache cluster.*

We evaluate whether Macaron can combine performance and cost-efficiency, without compromising on performance. By dynamically adjusting its cache cluster, Macaron cuts remote data access latency by 61% compared to `Replicated`,

[4]These concatenated traces are 9⟶18, 18⟶12, 55⟶12, 55⟶18, and 96⟶12

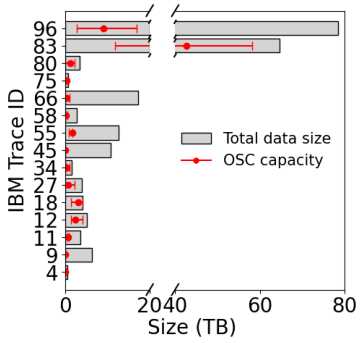


Figure 9. Macaron's average OSC capacity (red dot) is 0.8-75.1% of the accessed data size. Error bars show the range over 6 days.

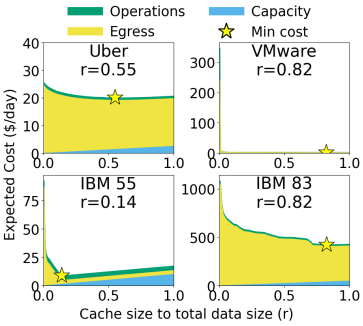


Figure 10. Expected cost curves generated by Macaron controller. Sub-optimal OSC size choices can result in higher expenses.

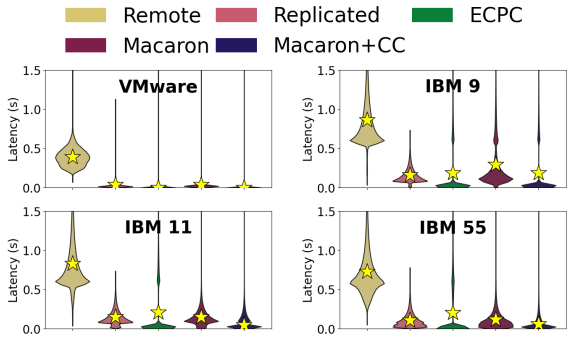


Figure 11. Violin plots of latency for each method. By dynamically adjusting cache cluster (Macaron+CC), Macaron reduces latency by 62% and cost by 58% compared to `Replicated`. The star marks the average.

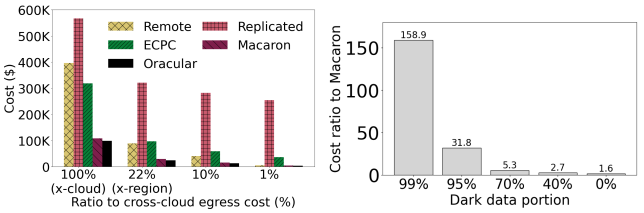
and still saves 64% costs across 10 traces that showed lower average latency than `Replicated`. This is achieved by serving hot data from cache cluster and smartly scaling the cluster during inactive periods or when smaller capacities suffice. Macaron without cache cluster exhibits, on average, 10% higher latency compared to Replicated. This increase is because Macaron 's latency is lower bounded by the latency of object storage, which is the same as that experienced by `Replicated`. However, using 30% more costs for the cache cluster (still significantly cheaper than `Replicated`), Macaron substantially improves latency.

For 6 remaining IBM traces and Uber traces with high compulsory miss ratios, only full replication achieves low latency, albeit at the previously discussed high cost in §7.2. Our rough estimates indicate that if the compulsory miss ratio surpasses 10-20% (depending on workload object size), achieving lower average latency than local object storage becomes challenging, even with all other requests retrieve data from the cache cluster, which matches with our results.

Fig. 11 shows the violin graphs that illustrate the latency distributions of four traces, all showing similar characteristics. Interestingly, despite `ECPC` being DRAM-centric solution, it often shows higher latency than Macaron with a cache cluster (Macaron+CC) or even without a cache cluster (IBM 11 and 55 in Fig. 11), as not enough DRAM cache server is allocated by prioritizing cost-efficiency, thus resulting in high latency. Specifically, in Fig. 11, Macaron+CC demonstrates cost savings of 0.3%, 9%, 37%, and 16% for the VMware, IBM 9, 11, and 55 traces, respectively, while also reducing latency by 3%, 1%, 76%, and 70% compared to `ECPC`.

The plot also shows Macaron's tail latency without a cache cluster resembling `Remote`, while its low latency mirrors `Replicated`. With a cache cluster, Macaron's low latency distribution matches cache cluster latency, substantially lowering average latency.

Our percentile latency analysis further validates these observations. For example, in IBM 9 with a 21% compulsory



(a) Cost comparison across pricing models.

(b) Effect of dark data on cost of `Replicated` relative to Macaron.

Figure 12. Analysis of the efficiency of Macaron under varying pricing models and with changing dark data portions.

miss ratio, Macaron's p90 and p99 latencies using DRAM cache are from remote data accesses, but are 27% and 15% lower than `Remote`'s p90 and p99, indicating the impact of serving many requests from OSC. In IBM 55, where the compulsory miss ratio is below 0.1%, Macaron's p90 and p99 latencies from OSC using DRAM cache are even 15% and 6% lower than `Replicated`'s p90 and p99, showcasing the efficiency of serving from the cache cluster.

7.6 Sensitivity analysis

We assess Macaron under varying experimental settings, specifically latency, egress cost, and dark data portions.

Different egress costs. We tested Macaron with three alternative egress cost models to ensure its effectiveness across different pricing: 22% (cross-region egress cost), 10% (0.9¢/GB), and 1% (0.09¢/GB) of the standard cross-cloud rate of 9¢/GB. Macaron consistently surpasses the baselines across different pricing models, achieving substantial cost reductions as depicted in Fig. 12a, even when egress costs are as low as 1% of the cross-cloud rate.

Different latency. We evaluate Macaron's cost-efficiency with varying latency distributions by switching the inter-region setting from US N. Virginia and US N. California to US N. Virginia and Europe Frankfurt. In this new scenario, higher inter-continent latency makes it harder to achieve

local object storage access latency with Macaron. Consequently, one less trace outperforming Replicated for both cost and latency compared to the intra-continent scenario, achieving a 71% lower average cost while paying 62% less.

Different dark data percentage. We evaluated Macaron's efficiency against Replicated across varying dark data portions, previously set at 70%. Fig. 12b shows that with a 0% dark data portion, which means the working set size of the trace is equal to the entire data size, Macaron is 37.5% cheaper than Replicated. At 99% dark data, Replicated is 158.9× costlier than Macaron.

7.7 Simulation accuracy & Reconfiguration overhead

Observation 7: *Simulator closely mimics Macaron with minimal gaps in cost and latency, up to 0.17% and 7.6%. Reconfiguration time comprises less than 9% of the total runtime, while the cost overhead remains low at 0.6% of the total cost.*

We evaluate Macaron simulator's accuracy by comparing its cost and latency results with those obtained from running our prototype implementation on AWS. Due to budget constraints, we selected three IBM traces: IBM 9, 55, and 58, representing read-only, read/write mixed, and read/write/delete mixed scenarios, respectively. Overall, the cost gap between the simulator and prototype was minimal, ranging from 0.08% to 0.17%. Similarly, the average latency gap was 4-7.6%. Additionally, we validated the number of Get operations hit at each cache level match. More detailed results are provided in Appendix A.4.

Next, we evaluate reconfiguration overhead. Across the three traces we evaluated, end-to-end reconfiguration took 6 to 418 seconds (avg. 71*sec*). When there is no change in cache cluster configuration, it takes only 7 seconds on average, but if there is a change, especially when scaling out, it takes 274 seconds on average. Since reconfiguration occurs only when optimization results in configuration changes, the total reconfiguration time across three traces amounts to 1.6 hours, representing just 9% of the total runtime of 18 hours. During reconfiguration, requests continue to be served without any downtime, and the cost of the resources required to carry it out are factored into Macaron's savings.

Further breakdown reveals that the largest portion of reconfiguration time is spent on miniature simulation and cache cluster reconfiguration. Miniature simulation time is proportional to the request count in the optimization window, taking 0.3–44 seconds (avg. 31*sec* across all optimization windows of 19 traces). Cache cluster reconfiguration, including VM initialization and cache priming, took 132–387 seconds (avg. 256*sec*). Finally, the cost overhead of running miniature simulation on AWS Lambda is negligible, accounting for only 0.003–4% (avg. 0.6%) of the total cost of running each trace end-to-end across 19 traces.

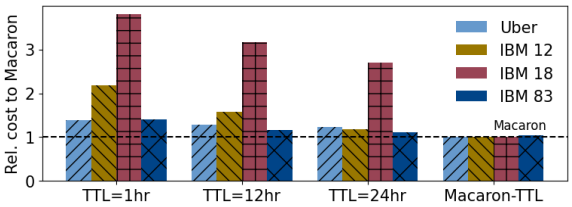


Figure 13. Comparison of Macaron against static TTL caches and Macaron-TTL shows that dynamic cache adjustments of Macaron's variants result in cost savings compared to static TTL caches.

7.8 Size-based and TTL-based Macaron variants

Observation 8: *Macaron and Macaron-TTL demonstrate similar cost-efficiency, successfully identifying cost-efficient cache sizes and TTLs.*

We evaluated the effects of optimizing TTL (Macaron-TTL) rather than cache size (Macaron) in cross-cloud scenarios. Across 18 traces, Macaron-TTL's cost ranges from -0.8% to 3.3% compared to Macaron, indicating similar performance. In IBM 80, Macaron-TTL was 17% more expensive due to its TTL=24hr selected based on the past data access pattern, which led to all data being forcibly evicted during a two-day no-access period, whereas Macaron saves more in egress costs by avoiding evictions.

We assessed Macaron-TTL 's ability to identify the most cost-efficient TTL for each trace. Through exhaustive search, we tested various static TTL caches throughout each traces[5], and pinpointed those that minimized costs. In the IBM traces, we observed that the optimal TTLs varied widely, ranging from 1 to 168 hours, with an average of 72 hours and a standard deviation of 56 hours. Despite this variability, Macaron-TTL accurately identified the optimal TTLs for 16 traces. For IBM 34, 45, and 58, although Macaron-TTL selected TTLs of 144, 132, and 84 hours – differing from the optimal TTLs of 72, 108, and 24 hours – the cost gaps between the static TTL policies using TTLs chosen by Macaron-TTL and the optimal TTLs were negligible, all under 0.7%. This is because OSC capacity costs are significantly lower than egress costs, so the additional capacity cost has minimal impact.

Fig. 13 illustrates a cost comparison between Macaron and Macaron-TTL against static TTL policies. Across 19 evaluated traces, Macaron achieved average cost reductions of 22%, 13%, and 9%, with maximum reductions of up to 74%, 69%, and 63% compared to static TTL policies set at 1, 12, and 24 hours, respectively. This highlights the importance of dynamic cache adjustments for enhancing cost-efficiency.

8 Related work

Cache auto-configuration. Prior works have focused on enhancing performance by reallocating a fixed-capacity shared

[5]For this exhaustive search, TTL intervals of 1 hour, 6 hours, and then every 12 hours up to the full trace length were used (e.g., 1, 6, 12, 24, 36, 48, …).

memory between application runtime needs and caching. Robinhood [142] redistributes cache resources across backend services to reduce latency variability and tail latency. LAMA [143] adjusts Memcached's memory partitioning to improve miss ratios and response times. Memshare [144] reallocates memory among applications to maximize hit rates using idle CPU and memory bandwidth. Sundarrajan et al. [145] enhance CDN cache provisioning using footprint descriptors. D3N [146] adjusts cache sizes to improve big-data job performance and reduce network traffic. While they focus on dynamic reallocation of fixed memory, Macaron assumes total cache capacity is elastic at a cost, and utilizes object storage to reduce that cost. While some studies [135, 147–149] explore dynamic cache provisioning, they mainly focus on using DRAM or Flash as cache storage medium, and do not account for expensive data transfer costs as part of the miss penalty, which plays a significant role for Macaron.

Optimizing cloud resource costs. With the growing trend of migrating workloads to public clouds, prior work [15–18] has explored methods for cost-efficiently provisioning cloud storage for various applications. InfiniCache [19] has focuses on cost-effectively utilizing serverless functions for caching data, and there are studies [20, 21] optimizing the use of object storage for file systems and backup solutions, and our packing strategy is based on them. Others [22, 23] have leveraged spot instances] to reduce VM costs by utilizing the affordability of ephemeral VMs, and some of their ideas could be applied to Macaron when dynamically scaling DRAM cache servers. Skyplane [24] aims to optimize egress network resources in public clouds. However, they primarily focus on methods for cost-efficient one-time transfers of bulk data. Macaron's emphasizes optimizing data access costs for long-running workloads with repetitive data access patterns.

Multi-cloud data management. Previous works [25–28, 150–153] explored optimizing data placement across clouds or regions for fault tolerance, latency, and cost-efficiency, allowing free data migration or replication across regions. In contrast, Macaron addresses scenarios where data placement is already determined and data cannot be freely migrated due to cost, proximity to users or sources of data generation. In such cases, we need to run applications across regions or clouds as use cases described in §2. In these contexts, an auto-configuring caching solution is more suitable than data placement optimization. We believe both approaches are orthogonal and complementary.

Cache replacement policy. Prior work [44–54] focuses on cache replacement policies to enhance miss ratio or latency. However, in the unique context of multi-cloud, multi-region environments, where high egress costs serve as a significant miss penalty and cache capacity cost is very cheap and elastic, even if replacement policy is not optimal, Macaron can extend cache capacity accordingly with minimal costs, making determining the cost-efficient cache capacity more crucial than refining cache replacement policies. Our evaluation, comparing Macaron to `Oracular`, supports this notion. While `Oracular` represents an optimal solution for both cache replacement and capacity determination, our results demonstrate that solely achieving the right cache capacity without exploring replacement policies can yield results close to those of `Oracular`.

Existing approaches. Major cloud providers like AWS, Azure, and GCP already support cross-region data replication [32–34], and companies such as Snowflake [35] and Juicedata [36] have recently introduced cross-cloud data replication capabilities [154]. There are existing caching services [37–39] provided by cloud providers that are built on distributed in-memory stores [155, 156]. The third-party services like Alluxio [40], MinIO [41], and Avere [42] support cloud-native cache solutions using memory or flash devices. However, their primary goal is achieving high performance for accessing local data and are not optimized for cross-cloud or cross-region data access costs. Our `ECPC` baseline mimics these approaches but is enhanced by intelligent auto-scaling, yet Macaron remains more cost-efficient.

9 Conclusion

Macaron addresses the high data transfer and latency costs associated with cross-cloud or cross-region data access. It dynamically auto-configures the size and storage types of cache space for remote data, reducing cross-cloud dollar cost by 65% for a collection of real workload traces.

Acknowledgments

We thank the anonymous reviewers and our shepherd, Atul Adya, for their help improving the presentation of this paper. We thank Uber and VMware Research for sharing their insights and data. We thank the members and companies of the PDL Consortium (Amazon, Google, Honda, IBM, Intel, Jane Street, Meta, Microsoft, Oracle, Pure Storage, Salesforce, Samsung, Two Sigma, Western Digital) and VMware for their interest, insights, feedback, and support.

This work was supported in part by U.S. Army Research Office and the U.S. Army Futures Command under Contract No. W911NF-20-D-0002. This work was supported by the AWS Cloud Credit for Research program. Hojin Park is supported in part by Korea Foundation for Advanced Studies.

George Amvrosiadis holds concurrent appointments at Carnegie Mellon University (CMU) and Amazon Web Services – this paper describes work performed at CMU and is not associated with Amazon. The content of the paper does not necessarily reflect the position or the policy of the government and no official endorsement should be inferred.

References

[1] 98% of Enterprises Using Public Cloud Have Adopted a Multicloud Infrastructure Provider Strategy. https://www.oracle.com/emea/news/announcement/98-percent-enterprises-adopted-multicloud-strategy-2023-02-09/.

[2] The Latest Cloud Computing Statistics (updated March 2024) | AAG IT Support. https://aag-it.com/the-latest-cloud-computing-statistics/. Section: Business.

[3] Cody Slingerland. 101 Shocking Cloud Computing Statistics. https://www.cloudzero.com/blog/cloud-computing-statistics/, December 2023.

[4] Understanding Hybrid Cloud vs. Multicloud: Key Differences Explained - Crayon. https://www.crayon.com/us/resources/blogs/multicloud-a-guide-to-multicloud-strategy-management-and-multicloud-architecture/.

[5] The benefits and challenges of multi-cloud management. https://www.liquidweb.com/blog/multi-cloud-benefits-challenges/, February 2024.

[6] Justice Opara-Martins, Reza Sahandi, and Feng Tian. Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. *Journal of Cloud Computing*, 5(1):4, December 2016.

[7] Gabriel Costa Silva, Louis M. Rose, and Radu Calinescu. A Systematic Review of Cloud Lock-In Solutions. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, pages 363–368, Bristol, United Kingdom, December 2013. IEEE.

[8] Ataollah Fatahi Baarzi, George Kesidis, Carlee Joe-Wong, and Mohammad Shahrad. On Merits and Viability of Multi-Cloud Serverless. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 600–608, Seattle WA USA, November 2021. ACM.

[9] Lauren van der Vaart. How Salesforce® Optimizes Multi-Cloud Costs With CloudHealth® by VMware®. https://blogs.vmware.com/cloud/2021/07/19/salesforce-optimizes-multi-cloud-costs/, July 2021.

[10] Sandipan Sarkar. Living in a data sovereign world. https://www.ibm.com/blog/living-in-a-data-sovereign-world/, October 2023.

[11] Oracle Cloud®for Sovereignty: Customized, Flexible Solutions. https://www.oracle.com/cloud/sovereign-cloud/.

[12] Data residency - Microsoft® Cloud for Sovereignty. https://learn.microsoft.com/en-us/industry/sovereignty/data-residency, December 2023.

[13] How to Accelerate Performance and Availability of Multi-region Applications with Amazon S3® Multi-Region Access Points | AWS News Blog. https://aws.amazon.com/blogs/aws/s3-multi-region-access-points-accelerate-performance-availability/, September 2021. Section: Amazon Simple Storage Service (S3).

[14] CockroachDB® : The multi-region database. https://dantheengineer.com/cockroachdb-multi-region/, February 2023.

[15] Ashraf Mahgoub, Alexander Michaelson Medoff, Rakesh Kumar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. OPTIMUSCLOUD: Heterogeneous configuration optimization for distributed databases in the cloud. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 189–203. USENIX Association, July 2020.

[16] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. Selecta: Heterogeneous cloud storage configuration for data analytics. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 759–773, Boston, MA, July 2018. USENIX Association.

[17] Ana Klimovic, Yawen Wang, Patrick Stuedi, Animesh Trivedi, Jonas Pfefferle, and Christos Kozyrakis. Pocket: Elastic ephemeral storage for serverless analytics. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 427–444, Carlsbad, CA, October 2018. USENIX Association.

[18] Hojin Park, Gregory R. Ganger, and George Amvrosiadis. Mimir: Finding Cost-efficient Storage Configurations in the Public Cloud. In *Proceedings of the 16th ACM International Conference on Systems and Storage*, SYSTOR '23, pages 22–34, New York, NY, USA, June 2023. Association for Computing Machinery.

[19] Ao Wang, Jingyuan Zhang, Xiaolong Ma, Ali Anwar, Lukas Rupprecht, Dimitrios Skourtis, Vasily Tarasov, Feng Yan, and Yue Cheng. InfiniCache: Exploiting ephemeral serverless functions to build a Cost-Effective memory cache. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 267–281, Santa Clara, CA, February 2020. USENIX Association.

[20] Iwona Kotlarska, Andrzej Jackowski, Krzysztof Lichota, Michal Welnicki, Cezary Dubnicki, and Konrad Iwanicki. InftyDedup: Scalable and Cost-Effective cloud tiering with deduplication. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*, pages 33–48, Santa Clara, CA, February 2023. USENIX Association.

[21] Saurabh Kadekodi, Bin Fan, Adit Madan, Garth A. Gibson, and Gregory R. Ganger. A case for packing and indexing in cloud file systems. In *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*, Boston, MA, July 2018. USENIX Association.

[22] Ataollah Fatahi Baarzi, Timothy Zhu, and Bhuvan Urgaonkar. BurScale: Using Burstable Instances for Cost-Effective Autoscaling in the Public Cloud. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '19, pages 126–138, New York, NY, USA, November 2019. Association for Computing Machinery.

[23] Cheng Wang, Bhuvan Urgaonkar, Aayush Gupta, George Kesidis, and Qianlin Liang. Exploiting Spot and Burstable Instances for Improving the Cost-efficacy of In-Memory Caches on the Public Cloud. In *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, pages 620–634, New York, NY, USA, April 2017. Association for Computing Machinery.

[24] Paras Jain, Sam Kumar, Sarah Wooders, Shishir G. Patil, Joseph E. Gonzalez, and Ion Stoica. Skyplane: Optimizing transfer cost and throughput using Cloud-Aware overlays. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 1375–1389, Boston, MA, April 2023. USENIX Association.

[25] Maomeng Su, Lei Zhang, Yongwei Wu, Kang Chen, and Keqin Li. Systematic Data Placement Optimization in Multi-Cloud Storage for Complex Requirements. *IEEE Transactions on Computers*, 65(6):1964–1977, June 2016.

[26] Pengwei Wang, Caihui Zhao, Yi Wei, Dong Wang, and Zhaohui Zhang. An Adaptive Data Placement Architecture in Multicloud Environments. *Scientific Programming*, 2020:e1704258, June 2020.

[27] Quanlu Zhang, Shenglong Li, Zhenhua Li, Yuanjian Xing, Zhi Yang, and Yafei Dai. CHARM: A Cost-Efficient Multi-Cloud Data Hosting Scheme with High Availability. *IEEE Transactions on Cloud Computing*, 3(3):372–386, July 2015.

[28] Zhe Wu, Michael Butkiewicz, Dorian Perkins, Ethan Katz-Bassett, and Harsha V. Madhyastha. SPANStore: cost-effective geo-replicated storage spanning multiple cloud services. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, pages 292–308, New York, NY, USA, November 2013. Association for Computing Machinery.

[29] Accessing tables across Regions - AWS Lake Formation. https://docs.aws.amazon.com/lake-formation/latest/dg/data-access-across-region.html.

[30] Querying across regions - Amazon Athena®. https://docs.aws.amazon.com/athena/latest/ug/querying-across-regions.html.

[31] Load data with cross-cloud operations | BigQuery®. https://cloud.google.com/bigquery/docs/load-data-using-cross-cloud-transfer.

[32] Cross-region replication in Azure®. https://learn.microsoft.com/en-us/azure/reliability/cross-region-replication-azure, April 2023.

[33] Bhagat Nimesh. Ensure business continuity across regions with replicas. https://cloud.google.com/blog/products/databases/introducing-cross-region-replica-for-cloud-sql.

[34] Replicating objects - Amazon Simple Storage Service. https://docs.aws.amazon.com/AmazonS3/latest/userguide/replication.html.

[35] The Data Cloud | Snowflake®. https://www.snowflake.com/en/.

[36] JuiceFS® - Open Source Distributed POSIX File System for Cloud. https://juicefs.com/en/.

[37] Redis®and Memcached-Compatible Cache – Amazon ElastiCache – Amazon Web Services. https://aws.amazon.com/elasticache/.

[38] Azure Cache for Redis | Microsoft Azure. https://azure.microsoft.com/en-us/products/cache.

[39] Memorystore: in-memory Redis compatible data store. https://cloud.google.com/memorystore.

[40] Alluxio®- Data Orchestration for the Cloud. https://www.alluxio.io/.

[41] MinIO Inc. MinIO® | MinIO Enterprise Object Store Cache feature. https://min.io.

[42] Avere vFXT - Scalable File System for Hybrid HPC | Microsoft Azure. https://azure.microsoft.com/en-us/products/storage/avere-vfxt.

[43] Ohad Eytan, Danny Harnik, Effi Ofer, Roy Friedman, and Ronen Kat. It's time to revisit LRU vs. FIFO. In *12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 20)*. USENIX Association, July 2020.

[44] Sorav Bansal and Dharmendra S. Modha. CAR: Clock with adaptive replacement. In *3rd USENIX Conference on File and Storage Technologies (FAST 04)*, San Francisco, CA, March 2004. USENIX Association.

[45] Pei Cao and Sandy Irani. Cost-Aware WWW proxy caching algorithms. In *USENIX Symposium on Internet Technologies and Systems (USITS 97)*, Monterey, CA, December 1997. USENIX Association.

[46] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, S.H. Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE Transactions on Computers*, 50(12):1352–1361, December 2001. Conference Name: IEEE Transactions on Computers.

[47] Song Jiang, Feng Chen, and Xiaodong Zhang. CLOCK-Pro: An effective improvement of the CLOCK replacement. In *2005 USENIX Annual Technical Conference (USENIX ATC 05)*, Anaheim, CA, April 2005. USENIX Association.

[48] Song Jiang and Xiaodong Zhang. LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. *ACM SIGMETRICS Performance Evaluation Review*, 30(1):31–42, June 2002.

[49] Theodore Johnson and Dennis Shasha. 2q: A low overhead high performance buffer management replacement algorithm. In *Very Large Data Bases Conference*, 1994.

[50] Nimrod Megiddo and Dharmendra S. Modha. ARC: A Self-Tuning, low overhead replacement cache. In *2nd USENIX Conference on File and Storage Technologies (FAST 03)*, San Francisco, CA, March 2003. USENIX Association.

[51] Elizabeth J. O'Neil, Patrick E. O'Neil, and Gerhard Weikum. The lru-k page replacement algorithm for database disk buffering. SIGMOD '93, page 297–306, New York, NY, USA, 1993. Association for Computing Machinery.

[52] Yannis Smaragdakis, Scott Kaplan, and Paul Wilson. EELRU: simple and effective adaptive page replacement. *ACM SIGMETRICS Performance Evaluation Review*, 27(1):122–133, May 1999.

[53] Chen Zhong, Xingsheng Zhao, and Song Jiang. LIRS2: an improved LIRS replacement algorithm. In *Proceedings of the 14th ACM International Conference on Systems and Storage*, SYSTOR '21, New York, NY, USA, 2021. Association for Computing Machinery.

[54] 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.

[55] Yifan Dai, Jing Liu, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. Symbiosis: The art of application and kernel cache cooperation. In *22nd USENIX Conference on File and Storage Technologies (FAST 24)*, pages 51–69, Santa Clara, CA, February 2024. USENIX Association.

[56] Nosayba El-Sayed, Anurag Mukkara, Po-An Tsai, Harshad Kasture, Xiaosong Ma, and Daniel Sanchez. Kpart: A hybrid cache partitioning-sharing technique for commodity multicores. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 104–117, 2018.

[57] Xiameng Hu, Xiaolin Wang, Yechen Li, Lan Zhou, Yingwei Luo, Chen Ding, Song Jiang, and Zhenlin Wang. LAMA: Optimized locality-aware memory allocation for key-value cache. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 57–69, Santa Clara, CA, July 2015. USENIX Association.

[58] Ricardo Koller, Ali José Mashtizadeh, and Raju Rangaswami. Centaur: Host-side ssd caching for storage performance control. In *2015 IEEE International Conference on Autonomic Computing*, pages 51–60, 2015.

[59] Jaewon Kwak, Eunji Hwang, Tae-Kyung Yoo, Beomseok Nam, and Young-Ri Choi. In-memory caching orchestration for hadoop. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 94–97, 2016.

[60] Fei Meng, Li Zhou, Xiaosong Ma, Sandeep Uttamchandani, and Deng Liu. vcacheshare: automated server flash cache space management in a virtualization environment. USENIX ATC'14, page 133–144, USA, 2014. USENIX Association.

[61] G. Suh, Larry Rudolph, and Sahana Devadas. Dynamic partitioning of shared cache memory. *The Journal of Supercomputing*, 28:7–26, 04 2004.

[62] Cheng Pan, Yingwei Luo, Xiaolin Wang, and Zhenlin Wang. predis: Penalty and locality aware memory allocation in redis. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '19, page 193–205, New York, NY, USA, 2019. Association for Computing Machinery.

[63] Trausti Saemundsson, Hjortur Bjornsson, Gregory Chockler, and Ymir Vigfusson. Dynamic performance profiling of cloud caches. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, page 1–14, New York, NY, USA, 2014. Association for Computing Machinery.

[64] Carl Waldspurger, Trausti Saemundsson, Irfan Ahmad, and Nohhyun Park. Cache modeling and optimization using miniature simulations. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 487–498, Santa Clara, CA, July 2017. USENIX Association.

[65] Uber object access trace. https://github.com/uber-research/presto-HDFS-read-data/.

[66] Macaron prototype code. https://github.com/hojinp/macaron/.

[67] Macaron simulator code. https://github.com/hojinp/macaron_simulator/.

[68] Ask HN: Azure has run out of compute – anyone else affected? | Hacker News. https://news.ycombinator.com/item?id=33743567.

[69] Diana Goovaerts. The one where we hate on egress fees even more. https://www.silverliningsinfo.com/ai/one-where-we-hate-egress-fees-even-more, January 2024.

[70] GPU regions and zones availability | Compute Engine Documentation. https://cloud.google.com/compute/docs/gpus/gpu-regions-zones.

[71] Azure Products by Region | Microsoft Azure. https://azure.microsoft.com/en-us/explore/global-infrastructure/products-by-region/.

[72] Snowflake Staff. Pfizer® uses Snowgrid® for cross-region collaboration and business continuity. https://www.snowflake.com/blog/pfizer-uses-snowgrid-for-cross-region-collaboration/, April 2023.

[73] Denis Magda. Multi-Region Applications With Kong Mesh and YugabyteDB®. https://medium.com/@magda7817/multi-region-applications-with-kong-mesh-and-yugabytedb-3f472b4dc88f, September 2023.

[74] K. Indira and M. K. Kavithadevi. Efficient Machine Learning Model for Movie Recommender Systems Using Multi-Cloud Environment. *Mobile Networks and Applications*, 24(6):1872–1882, December 2019.

[75] Pasquale Salza, Erik Hemberg, Filomena Ferrucci, and Una-May O'Reilly. Towards evolutionary machine learning comparison, competition, and collaboration with a multi-cloud platform. In *Proceedings*

of the Genetic and Evolutionary Computation Conference Companion, GECCO '17, pages 1263–1270, New York, NY, USA, July 2017. Association for Computing Machinery.

[76] Charlie Custer. The State of Multi-Cloud 2024. 2024.

[77] Laurent Gil. GPU Shortage Mitigation: How to Harness the Cloud Automation Advantage. https://cast.ai/blog/gpu-shortage-mitigation-how-to-harness-the-cloud-automation-advantage/, June 2023.

[78] Do You Know What Multicloud Is? https://www.oracle.com/cloud/multicloud/what-is-multicloud/.

[79] Andrew Smith, Natalya Yezhkova, and Ashish Nadkarni. Future-proofing storage: Modernizing Infrastructure for Data Growth Across Hybrid, Edge, and Cloud Ecosystems. *IDC*, March 2021.

[80] Robert B. Miller. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, AFIPS '68 (Fall, part I), pages 267–277, New York, NY, USA, December 1968. Association for Computing Machinery.

[81] Ankit Singla, Balakrishnan Chandrasekaran, P. Brighten Godfrey, and Bruce Maggs. The Internet at the Speed of Light. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, HotNets-XIII, pages 1–7, New York, NY, USA, October 2014. Association for Computing Machinery.

[82] Ziyue Qiu, Juncheng Yang, Juncheng Zhang, Cheng Li, Xiaosong Ma, Qi Chen, Mao Yang, and Yinlong Xu. Frozenhot cache: Rethinking cache management for modern hardware. In *Proceedings of the Eighteenth European Conference on Computer Systems*, EuroSys '23, page 557–573, New York, NY, USA, 2023. Association for Computing Machinery.

[83] Zhichao Cao, Siying Dong, Sagar Vemuri, and David H.C. Du. Characterizing, modeling, and benchmarking RocksDB Key-Value workloads at facebook. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 209–223, Santa Clara, CA, February 2020. USENIX Association.

[84] Juncheng Yang, Yao Yue, and K. V. Rashmi. A large scale analysis of hundreds of in-memory cache clusters at twitter. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 191–208. USENIX Association, November 2020.

[85] Ce Zhang, Jaeho Shin, Christopher Ré, Michael Cafarella, and Feng Niu. Extracting Databases from Dark Data with DeepDive. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 847–859, New York, NY, USA, June 2016. Association for Computing Machinery.

[86] P. Bryan Heidorn. Shedding Light on the Dark Data in the Long Tail of Science. *Library Trends*, 57(2):280–299, 2008.

[87] David J. Hand. Dark Data: Why What You Don't Know Matters. In *Dark Data*. Princeton University Press, February 2020.

[88] Jonathan Johnson. What Is Dark Data? The Basics & The Challenges. https://www.bmc.com/blogs/dark-data/.

[89] Businesses must eliminate the unnecessary energy costs of data processing, August 2022. https://venturebeat.com/data-infrastructure/businesses-must-eliminate-the-unnecessary-energy-costs-of-data-processing/.

[90] Unlocking the hidden value of dark data. https://www.cio.com/article/404526/unlocking-the-hidden-value-of-dark-data.html.

[91] Anurag Gupta, Deepak Agarwal, Derek Tan, Jakub Kulesza, Rahul Pathak, Stefano Stefani, and Vidhya Srinivasan. Amazon Redshift and the Case for Simpler Data Warehouses. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1917–1923, Melbourne Victoria Australia, May 2015. ACM.

[92] Yang Yang. Presto® on Apache Kafka® At Uber Scale. https://www.uber.com/en-AU/blog/presto-on-apache-kafka-at-uber-scale/, April 2022.

[93] Sabina Hristova. Join Us in Transforming Cybersecurity. https://blogs.vmware.com/bulgaria/2021/06/01/join-us-in-transforming-cybersecurity/, June 2021.

[94] L. Breslau, Pei Cao, Li Fan, G. Phillips, and S. Shenker. Web caching and Zipf-like distributions: evidence and implications. In *IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)*, volume 1, pages 126–134 vol.1, March 1999. ISSN: 0743-166X.

[95] Damiano Carra and Giovanni Neglia. Efficient miss ratio curve computation for heterogeneous content popularity. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 741–751. USENIX Association, July 2020.

[96] Jiqiang Chen, Liang Chen, Sheng Wang, Guoyun Zhu, Yuanyuan Sun, Huan Liu, and Feifei Li. HotRing: A Hotspot-Aware In-Memory Key-Value store. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 239–252, Santa Clara, CA, February 2020. USENIX Association.

[97] George Kingsley Zipf. Relative Frequency as a Determinant of Phonetic Change. *Harvard Studies in Classical Philology*, 40:1–95, 1929.

[98] Or Ozeri, Effi Ofer, and Ronen Kat. Object Storage for Deep Learning Frameworks. In *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning*, DIDL '18, pages 21–24, New York, NY, USA, December 2018. Association for Computing Machinery.

[99] Using Amazon S3 with Amazon ML - Amazon Machine Learning. https://docs.aws.amazon.com/machine-learning/latest/dg/using-amazon-s3-with-amazon-ml.html.

[100] Z. Daher and Hassan Hajjdiab. Cloud storage comparative analysis amazon simple storage vs. microsoft azure blob storage. *International Journal of Machine Learning and Computing*, 8:85–89, 02 2018.

[101] Matei A. Zaharia, Ali Ghodsi, Reynold Xin, and Michael Armbrust. Lakehouse: A new generation of open platforms that unify data warehousing and advanced analytics. In *Conference on Innovative Data Systems Research*, 2021.

[102] Michael Armbrust, Tathagata Das, Liwen Sun, Burak Yavuz, Shixiong Zhu, Mukul Murthy, Joseph Torres, Herman van Hovell, Adrian Ionescu, Alicja Łuszczak, Michał undefinedwitakowski, Michał Szafrański, Xiao Li, Takuya Ueshin, Mostafa Mokhtar, Peter Boncz, Ali Ghodsi, Sameer Paranjpye, Pieter Senster, Reynold Xin, and Matei Zaharia. Delta lake: high-performance acid table storage over cloud object stores. *Proc. VLDB Endow.*, 13(12):3411–3424, aug 2020.

[103] Eftim Zdravevski, Petre Lameski, Ace Dimitrievski, Marek Grzegorowski, and Cas Apanowicz. Cluster-size optimization within a cloud-based etl framework for big data. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 3754–3763, 2019.

[104] Mengchu Cai, Martin Grund, Anurag Gupta, Fabian Nagel, Ippokratis Pandis, Yannis Papakonstantinou, and Michalis Petropoulos. Integrated querying of sql database data and s3 data in amazon redshift. *IEEE Data Eng. Bull.*, 41:82–90, 2018.

[105] Interactive SQL - Amazon Athena - AWS. https://aws.amazon.com/athena/.

[106] Matthew Perron, Raul Castro Fernandez, David DeWitt, and Samuel Madden. Starling: A scalable query engine on cloud functions. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 131–141, New York, NY, USA, 2020. Association for Computing Machinery.

[107] Mahmoud Ismail, Salman Niazi, Gautier Berthou, Mikael Ronström, Seif Haridi, and Jim Dowling. Hopsfs-s3: Extending object stores with posix-like semantics and more (industry track). In *Proceedings of the 21st International Middleware Conference Industrial Track*, Middleware '20, page 23–30, New York, NY, USA, 2020. Association for Computing Machinery.

[108] Kunal Lillaney, Vasily Tarasov, David Pease, and Randal Burns. The case for dual-access file systems over object storage. In *11th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 19)*, Renton, WA, July 2019. USENIX Association.

[109] Alysson Bessani, Ricardo Mendes, Tiago Oliveira, Nuno Neves, Miguel Correia, Marcelo Pasin, and Paulo Verissimo. SCFS: A shared cloud-backed file system. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pages 169–180, Philadelphia, PA, June 2014. USENIX Association.

[110] Haoqiong Bian and Anastasia Ailamaki. Pixels: An Efficient Column Store for Cloud Data Lakes. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 3078–3090, May 2022. ISSN: 2375-026X.

[111] Anna Levin, Shelly Garion, Elliot K. Kolodner, Dean H. Lorenz, Katherine Barabash, Mike Kugler, and Niall McShane. AIOps for a Cloud Object Storage Service. In *2019 IEEE International Congress on Big Data (BigDataCongress)*, pages 165–169, July 2019. ISSN: 2642-7273.

[112] Hyojun Kim, Sangeetha Seshadri, Clement L. Dickey, and Lawrence Chiu. Evaluating Phase Change Memory for Enterprise Storage Systems: A Study of Caching and Tiering Approaches. *ACM Transactions on Storage*, 10(4):15:1–15:21, October 2014.

[113] Luis Pedrosa, Rishabh Iyer, Arseniy Zaostrovnykh, Jonas Fietz, and Katerina Argyraki. Automated synthesis of adversarial workloads for network functions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 372–385, New York, NY, USA, August 2018. Association for Computing Machinery.

[114] Leul Belayneh, Haojie Ye, Kuan-Yu Chen, David Blaauw, Trevor Mudge, Ronald Dreslinski, and Nishil Talati. Locality-Aware Optimizations for Improving Remote Memory Latency in Multi-GPU Systems. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, PACT '22, pages 304–316, New York, NY, USA, January 2023. Association for Computing Machinery.

[115] Sara McAllister, Benjamin Berg, Julian Tutuncu-Macias, Juncheng Yang, Sathya Gunasekar, Jimmy Lu, Daniel S. Berger, Nathan Beckmann, and Gregory R. Ganger. Kangaroo: Caching Billions of Tiny Objects on Flash. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, pages 243–262, New York, NY, USA, October 2021. Association for Computing Machinery.

[116] Yongseok Oh, Eunjae Lee, Choulseung Hyun, Jongmoo Choi, Donghee Lee, and Sam H. Noh. Enabling Cost-Effective Flash based Caching with an Array of Commodity SSDs. In *Proceedings of the 16th Annual Middleware Conference*, Middleware '15, pages 63–74, New York, NY, USA, November 2015. Association for Computing Machinery.

[117] Tzu-Wei Yang, Seth Pollen, Mustafa Uysal, Arif Merchant, and Homer Wolfmeister. CacheSack: Admission optimization for google datacenter flash caches. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 1021–1036, Carlsbad, CA, July 2022. USENIX Association.

[118] Rong Gu, Simian Li, Haipeng Dai, Hancheng Wang, Yili Luo, Bin Fan, Ran Ben Basat, Ke Wang, Zhenyu Song, Shouwei Chen, Beinan Wang, Yihua Huang, and Guihai Chen. Adaptive online cache capacity optimization via lightweight working set size estimation at scale. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 467–484, Boston, MA, July 2023. USENIX Association.

[119] Ashraf Mahgoub, Paul Wood, Alexander Medoff, Subrata Mitra, Folker Meyer, Somali Chaterji, and Saurabh Bagchi. SOPHIA: Online reconfiguration of clustered NoSQL databases for Time-Varying workloads. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 223–240, Renton, WA, July 2019. USENIX Association.

[120] Daniel Lin-Kit Wong, Hao Wu, Carson Molder, Sathya Gunasekar, Jimmy Lu, Snehal Khandkar, Abhinav Sharma, Daniel S. Berger, Nathan Beckmann, and Gregory R. Ganger. Baleen: ML admission & prefetching for flash caches. In *22nd USENIX Conference on File and Storage Technologies (FAST 24)*, pages 347–371, Santa Clara, CA, February 2024. USENIX Association.

[121] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 153–167, New York, NY, USA, 2017. Association for Computing Machinery.

[122] Arnab Choudhury, Yang Wang, Tuomas Pelkonen, Kutta Srinivasan, Abha Jain, Shenghao Lin, Delia David, Siavash Soleimanifard, Michael Chen, Abhishek Yadav, et al. MAST: Global scheduling of ML training across Geo-Distributed datacenters at hyperscale. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 563–580, 2024.

[123] Yuzhen Huang, Yingjie Shi, Zheng Zhong, Yihui Feng, James Cheng, Jiwei Li, Haochuan Fan, Chao Li, Tao Guan, and Jingren Zhou. Yugong: Geo-distributed data and job placement at scale. *Proceedings of the VLDB Endowment*, 12(12):2155–2169, 2019.

[124] lakefs and amazon s3 express one zone: Highly performant data version control for ml/ai. https://aws.amazon.com/blogs/storage/lakefs-and-amazon-s3-express-one-zone-highly-performant-data-version-control-for-ml-ai/.

[125] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a "Kneedle" in a Haystack: Detecting Knee Points in System Behavior. In *2011 31st International Conference on Distributed Computing Systems Workshops*, pages 166–171, June 2011. ISSN: 2332-5666.

[126] Frank Olken. Efficient methods for calculating the success function of fixed space replacement policies. *Perform. Evaluation*, 3(2):153–154, 1983.

[127] Qingpeng Niu, James Dinan, Qingda Lu, and P. Sadayappan. Parda: A fast parallel reuse distance analysis algorithm. In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium*, IPDPS '12, page 1284–1294, USA, 2012. IEEE Computer Society.

[128] Carl A. Waldspurger, Nohhyun Park, Alexander Garthwaite, and Irfan Ahmad. Efficient MRC construction with SHARDS. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pages 95–110, Santa Clara, CA, February 2015. USENIX Association.

[129] Xiameng Hu, Xiaolin Wang, Lan Zhou, Yingwei Luo, Chen Ding, and Zhenlin Wang. Kinetic modeling of data eviction in cache. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, pages 351–364, Denver, CO, June 2016. USENIX Association.

[130] Jake Wires, Stephen Ingram, Zachary Drudi, Nicholas JA Harvey, and Andrew Warfield. Characterizing storage workloads with counter stacks. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 335–349, 2014.

[131] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5(2):78–101, 1966.

[132] Zhenyu Song, Daniel S. Berger, Kai Li, and Wyatt Lloyd. Learning relaxed belady for content distribution network caching. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 529–544, Santa Clara, CA, February 2020. USENIX Association.

[133] Juncheng Yang, Ziming Mao, Yao Yue, and K. V. Rashmi. GL-Cache: Group-level learning for efficient and high-performance caching. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*, pages 115–134, Santa Clara, CA, February 2023. USENIX Association.

[134] Chunghan Lee, Tatsuo Kumano, Tatsuma Matsuki, Hiroshi Endo, Naoto Fukumoto, and Mariko Sugawara. Understanding storage traffic characteristics on enterprise virtual desktop infrastructure. In *Proceedings of the 10th ACM International Systems and Storage Conference*, SYSTOR '17, pages 1–11, New York, NY, USA, May 2017. Association for Computing Machinery.

[135] Timothy Zhu, Anshul Gandhi, Mor Harchol-Balter, and Michael A. Kozuch. Saving cash by using less cache. In *4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 12)*, Boston, MA, June

2012. USENIX Association.

[136] Atul Adya, Daniel Myers, Jon Howell, Jeremy Elson, Colin Meek, Vishesh Khemani, Stefan Fulger, Pan Gu, Lakshminath Bhuvanagiri, Jason Hunter, et al. Slicer:Auto-Sharding for Datacenter Applications. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 739–753, 2016.

[137] Sangmin Lee, Zhenhua Guo, Omer Sunercan, Jun Ying, Thawan Kooburat, Suryadeep Biswal, Jun Chen, Kun Huang, Yatpang Cheung, Yiding Zhou, et al. Shard manager: A generic shard management framework for geo-distributed applications. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, pages 553–569, 2021.

[138] Mikhail Genkin and Frank Dehne. Autonomic workload change classification and prediction for big data workloads. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 2835–2844, 2019.

[139] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, and Alfons Kemper. Workload analysis and demand prediction of enterprise data center applications. In *2007 IEEE 10th International Symposium on Workload Characterization*, pages 171–180, 2007.

[140] Jialin Li, Jacob Nelson, Ellis Michael, Xin Jin, and Dan R. K. Ports. Pegasus: Tolerating skewed workloads in distributed storage with In-Network coherence directories. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 387–406. USENIX Association, November 2020.

[141] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 121–136, Shanghai China, October 2017. ACM.

[142] Daniel S. Berger, Benjamin Berg, Timothy Zhu, Siddhartha Sen, and Mor Harchol-Balter. RobinHood: Tail latency aware caching – dynamic reallocation from Cache-Rich to Cache-Poor. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 195–212, Carlsbad, CA, October 2018. USENIX Association.

[143] Xiameng Hu, Xiaolin Wang, Yechen Li, Lan Zhou, Yingwei Luo, Chen Ding, Song Jiang, and Zhenlin Wang. LAMA: Optimized locality-aware memory allocation for key-value cache. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 57–69, Santa Clara, CA, July 2015. USENIX Association.

[144] Asaf Cidon, Daniel Rushton, Stephen M. Rumble, and Ryan Stutsman. Memshare: a dynamic multi-tenant key-value cache. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 321–334, Santa Clara, CA, July 2017. USENIX Association.

[145] Aditya Sundarrajan, Mingdong Feng, Mangesh Kasbekar, and Ramesh K Sitaraman. Footprint descriptors: Theory and practice of cache provisioning in a global cdn. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, pages 55–67, 2017.

[146] Emine Ugur Kaynar, Mania Abdi, Mohammad Hossein Hajkazemi, Ata Turk, Raja R. Sambasivan, David Cohen, Larry Rudolph, Peter Desnoyers, and Orran Krieger. D3N: A multi-layer cache for the rest of us. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 327–338, December 2019.

[147] Farhana Kabir and David Chiu. Reconciling Cost and Performance Objectives for Elastic Web Caches. In *2012 International Conference on Cloud and Service Computing*, pages 88–95, November 2012.

[148] Trausti Saemundsson, Hjortur Bjornsson, Gregory Chockler, and Ymir Vigfusson. Dynamic Performance Profiling of Cloud Caches. In *Proceedings of the ACM Symposium on Cloud Computing*, SOCC '14, pages 1–14, New York, NY, USA, November 2014. Association for Computing Machinery.

[149] Jinho Hwang and Timothy Wood. Adaptive Performance-Aware distributed memory caching. In *10th International Conference on Autonomic Computing (ICAC 13)*, pages 33–43, San Jose, CA, June 2013. USENIX Association.

[150] Muhammed Uluyol, Anthony Huang, Ayush Goel, Mosharaf Chowdhury, and Harsha V Madhyastha. Near-Optimal latency versus cost tradeoffs in Geo-Distributed storage. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 157–180, 2020.

[151] Sharad Agarwal, John Dunagan, Navendu Jain, Stefan Saroiu, Alec Wolman, and Habinder Bhogan. Volley: Automated data placement for geo-distributed cloud services. In *NSDI*, 2010.

[152] Tiemo Bang, Chris Douglas, Natacha Crooks, and Joseph M. Hellerstein. Skypie: A fast & accurate oracle for object placement. *Proc. ACM Manag. Data*, 2(1), mar 2024.

[153] Zhe Wu, Curtis Yu, and Harsha V. Madhyastha. CosTLO: Cost-Effective redundancy for lower latency variance on cloud storage services. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 543–557, Oakland, CA, May 2015. USENIX Association.

[154] Introduction to replication and failover across multiple accounts | Snowflake Documentation. https://docs.snowflake.com/en/user-guide/account-replication-intro.

[155] memcached - a distributed memory object caching system. https://memcached.org/.

[156] Redis. https://redis.io/.

A Supplementary Evaluation Details (not peer-reviewed)

We provide detailed information on trace collection, the cloud resources used for prototype evaluation, and additional evaluation results.

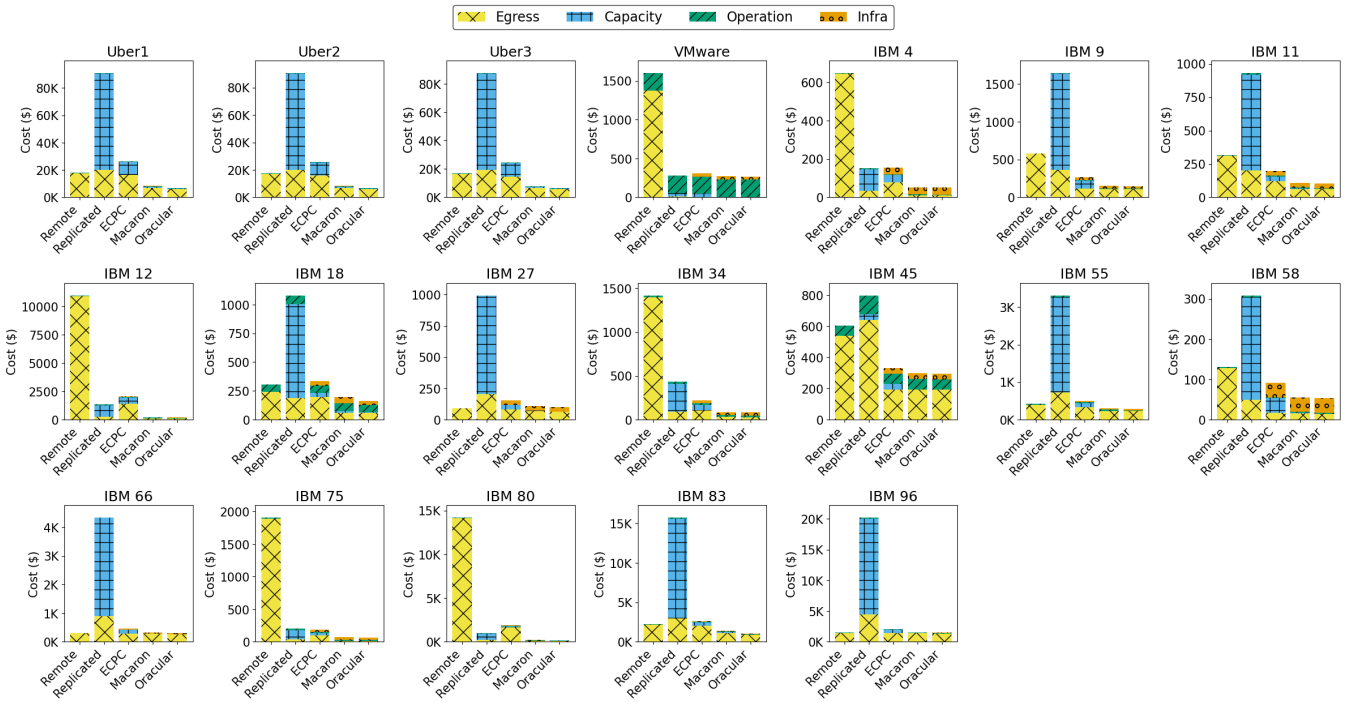
A.1 Trace collection details

Uber trace. We collected object access logs generated from Uber's Presto workload in their production system. To ensure no impact on production, we collected logs with a spatial sampling with a sampling ratio at 1% from three Presto engines over 18 days. To confirm that 1% sampling retains workload characteristics, we collected a two-hour trace without sampling and performed the same sampling method, where we observed small differences of 7%, 2%, and 8% in request count, accessed data size, and object size, respectively. Therefore, in our evaluation, we scaled the sampled traces by 100× and presented detailed results of one of the three traces where we confirmed that they share very similar data access characteristics. Due to the fact that over 70% of the accesses are generated by periodic jobs, the workload exhibits a stable data access pattern during the collection period.

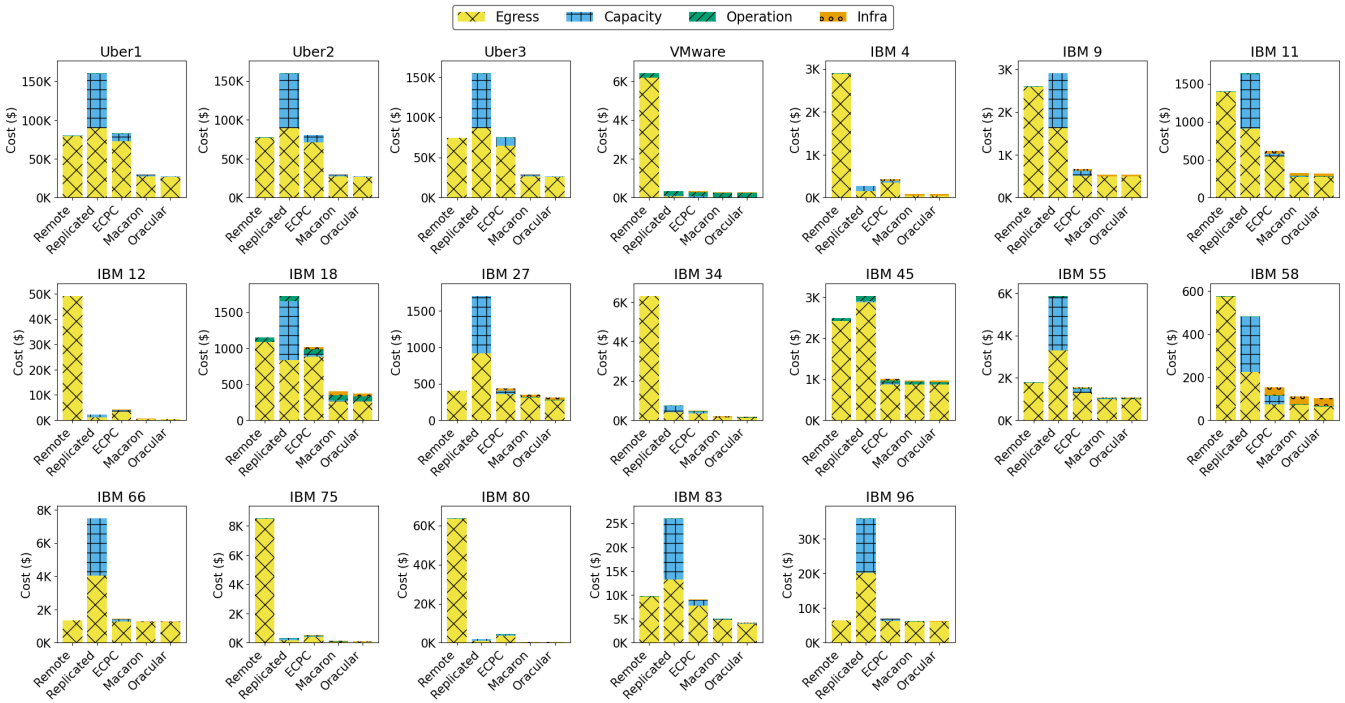
VMware trace. VMware trace involves AWS S3 requests generated by AWS Athena queries from VMware's test infrastructure, spanning an 8-day period. We collected logs by enabling Amazon S3 server access logging service to catch data access requests sent to S3 buckets.

A.2 Cloud resource types used for evaluation

For the cache cluster, we used Redis as a distributed in-memory cache solution as it is well-supported, performant, and effectively handles large objects, unlike Memcached.



(a) Cross-region data access



(b) Cross-cloud data access

Figure 14. Detailed cost analysis for all 19 traces.

For both simulator and prototype evaluations, we used the same VM types for consistency. We used an `r5.xlarge` instance for the master node on AWS, which we confirmed had sufficient resources. Since the Macaron controller's workload analysis, a computation-heavy task, ran on AWS Lambda, the CPU power of the `r5.xlarge` instance was adequate. Cache nodes also used the `r5.xlarge` type, which provides 32 GiB of memory. However, our observations showed that the Redis server typically utilized around 26 GiB, aligning with the `cache.r5.xlarge` specification of ElastiCache. As

Trace	Total costs ($)		Get hits at each level		Avg. lat (s)	
	Sim	Pro	Sim	Pro	Sim	Pro
IBM 9	5.86	5.87	45:35:20	46:34:20	0.24	0.26
IBM 55	11.83	11.82	50:5:45	47:9:44	0.30	0.28
IBM 58	2.61	2.60	40:0:60	39:1:60	0.47	0.49

Table 3. Detailed results of the accuracy evaluation between the simulator and prototype, with simulator results on the left and prototype results on the right. The comparison includes total costs, the number of Get hits at each level (cache cluster:OSC:remote data lake), and the average latency after running the trace.

a result, we assumed 26 GiB of memory for the cache nodes in the simulator as well. Additionally, we used AWS Lambda functions with 8 GiB of memory, which provided sufficient resources to run the miniature simulations quickly and cost-effectively.

A.3 Cost-efficiency analysis across all traces

Fig. 14 presents the cost comparison results of all the traces we evaluated and we explain two more case studies.

In IBM 83 (similar to IBM 55), ranking second in total data size accessed among 15 IBM traces, Macaron achieves 86% cost reduction compared to `Replicated`. Using an average cache capacity of 52 TB (81% of the total data size) and given the low price of object storage, Macaron prioritizes overall cost-efficiency over minimizing capacity costs and incurring higher data egress expenses associated with `Replicated`.

Despite similar total data sizes for IBM 96 and 83, Macaron allocates only 7% of the total data size as cache capacity for IBM 96 due to low data access skewness (Zipfian α=0.2) and a high cold miss ratio. Larger caches don't effectively reduce egress costs for such workloads, yet Macaron remains 1.4% and 81.7% cheaper than baselines for IBM 96.

A.4 Details of simulator accuracy evaluation

We carefully designed experiments to validate whether the simulator and prototype yield consistent results, both in terms of cost and performance, across different scenarios: one where Macaron utilizes a cache cluster for performance, and another where it solely relies on OSC to minimize costs. To reduce cost of running experiments without compromising the validity of results, we implemented the following approach.

For Macaron without cache cluster, we ensured both prototype and simulator execute the same reconfiguration for each optimization window and validated the total costs are the same by executing traces end-to-end. We sampled the requests with spatial sampling at 1% and accelerated execution by 10× to reduce costs and speed up experiments. Reconfiguration occurred every 3 minutes after a 3-hour observation period to make both simulator and prototype to run as many optimizations as possible and show the results are still matching each other. These adjustments do not compromise the validity of cost comparisons.

For Macaron with cache cluster, we focused on comparing average latency results between the prototype and simulator. In this experiment, rather than sampling the trace, we truncated the trace to the first 6 hours and commenced optimization after 30 minutes, with 15-minute optimization windows, maintaining the original request rate to compare latency correctly.

Table 3 shows the detailed data of the results we presented in §7.7.

A.5 Latency generator evaluation

As outlined in §7.1, Macaron simulator's latency generator fits a Gamma distribution to the latency measurements taken from the cloud. This distribution is then used to simulate latencies between each component in the system. Here, we verify the accuracy of the latency generator in reproducing a latency distribution similar to the real measurements.

Fig. 15a illustrates the cloud-measured latency distribution alongside the distribution generated by the simulator's latency generator for each object size, between the cache engine and each data source (cache cluster, OSC, and remote data lake). Across the average latencies for different object sizes, the mean absolute percentage error was low at 2%.

Fig. 15b depicts the end-to-end latency distribution for retrieving data from each source. We further confirmed the simulator's fidelity to the measured latency by showing a mean absolute percentage error of 1.5% between the average latencies.

B Macaron-TTL Details (not peer-reviewed)

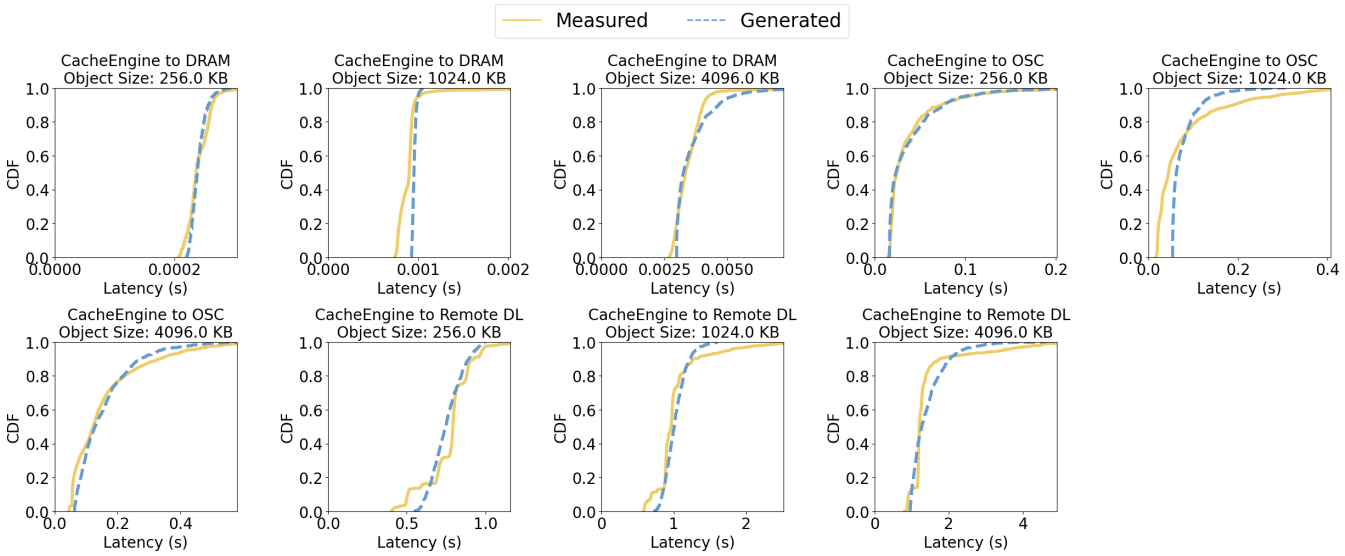
We extended Macaron 's optimization technique to develop Macaron-TTL. While the core optimization workflow of Macaron, described in §5, remains unchanged, the *expected cost curve* explained in §5.1 now uses TTL as its parameter instead of cache size:

$$TotalCost(TTL) = OSCCapacityCost(TTL, GarbageSize) + EgressCost(TTL) + OpCost(TTL)$$

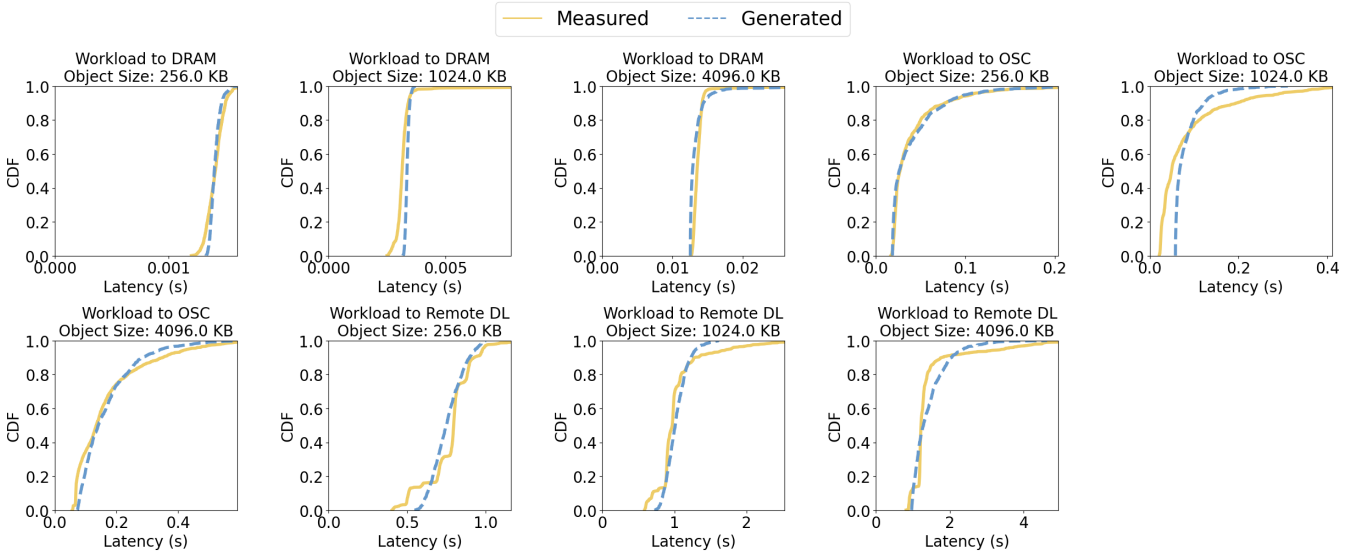
$$EgressCost(C) = EgressPrice * ByteMissCurve(TTL)$$

$$OpCost(C) = PutPrice\times \left(\frac{\#Writes + \#Reads \times MissRatioCurve(TTL)}{\#Objects\ per\ Packing\ Block}\right)$$

To calculate this, we adapted the miniature simulation to use TTL as the X-axis for the miss ratio curves and bytes miss curves. While OSC capacity was straightforward to calculate when using capacity as the parameter, we now



(a) Cache engine to data source latency



(b) End-to-end data access latency

Figure 15. Comparison of latency distributions generated by Macaron simulator's latency generator and measured ones. Each graph shows latency distributions for each object size we measured.

need to compute OSC capacity for each TTL during the simulation, producing an OSC Capacity Curve.

For the miniature simulation, we continue using spatial sampling for data access requests. However, we no longer reduce the cache size for simulating mini-caches since cache size does not affect cache eviction under TTL cache. After the simulation, the measured miss ratios from each mini-cache are used as is. Bytes missed are divided by the sampling ratio, similar to Macaron, and OSC capacity is also divided by the sampling ratio to reconstruct the original workload's MRC, BMC, and OSC Capacity Curve before sampling.