Serverless Replication of Object Storage across Multi-Vendor Clouds and Regions

Junyi Shu
School of Computer Science
Peking University
UCLA

Xiaolong Huang
School of Computer Science
Peking University

Gang Huang
School of Computer Science
Peking University

Hong Mei
School of Computer Science
Peking University

Xuanzhe Liu
School of Computer Science
Peking University

Xin Jin
School of Computer Science
Peking University

Abstract

Cross-cloud data replication is vital for improving reliability and performance. Since cloud providers lack native support, users turn to open-source solutions that rely on VMs. However, these are slow to provision, leading to high replication delays and costs. We propose a serverless approach for data replication using cloud functions, which cut provisioning overhead from tens of seconds to just a few. While functions offer sufficient bandwidth, they suffer from performance asymmetry across clouds and variability among instances. Our system, λReplica, mitigates this uncertainty through proactive planning and adaptive runtime adjustments. Prior to replication, λReplica formulates an SLO-compliant plan. During runtime, it employs decentralized scheduling to manage slow instances and uses changelog propagation and batching to further reduce costs. Implemented on three major clouds, λReplica outperforms existing solutions by reducing replication delay by 61%-99% with cost savings of up to three orders of magnitude. On production traces, it keeps p99.99 replication delay below 10 seconds.

CCS Concepts: • **Information systems** → **Cloud based storage**; • **Networks** → **Cloud computing**.

Keywords: Cloud Computing, Serverless Computing, Data Replication, Object Storage

ACM Reference Format:
Junyi Shu, Xiaolong Huang, Gang Huang, Hong Mei, Xuanzhe Liu, and Xin Jin. 2026. Serverless Replication of Object Storage across Multi-Vendor Clouds and Regions. In *European Conference on Computer Systems (EUROSYS '26), April 27–30, 2026, Edinburgh, Scotland Uk.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3767295.3769367

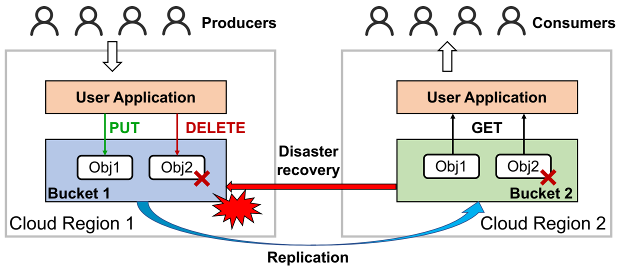


Figure 1. Cross-region bucket replication.

1 Introduction

Data replication across clouds and regions is essential for ensuring data availability and reliability, as well as improving access latency for cloud applications [41, 53, 58, 67, 70, 75, 83, 84]. Although cloud providers strive to isolate the impacts of individual incidents, region-wide outages are not rare for cloud platforms [12, 21, 29]. Sometimes, these accidents can span multiple regions [3] or cause severe data loss [4]. By replicating data across different geographic regions or cloud platforms, organizations can safeguard against such localized failures. Moreover, cross-cloud/region data replication improves data access latency by bringing data closer to end users, which ensures a seamless user experience across diverse locations.

Object storage, such as Amazon S3 [10], Azure Blob Storage [15], and Google Cloud Storage [30], is typically used to store unstructured data which accounts for 80%–90% of enterprise data according to multiple analyst estimates [2]. Today, even applications that manage structured data, such as database engines, start leveraging object storage as their backend storage solution for various reasons. Notable examples include Snowflake and RocksDB [39, 55, 78].

Object storage exhibits two key patterns based on analysis of public traces [33, 57]. First, objects are highly diverse in terms of size ranging from a few bytes to many terabytes while small objects dominate. The replication system should be able to handle small objects with low overheads while avoiding high tail latency caused by large objects. Second, request rates are highly variable, with significant fluctuations in throughput over short periods. The replication system should

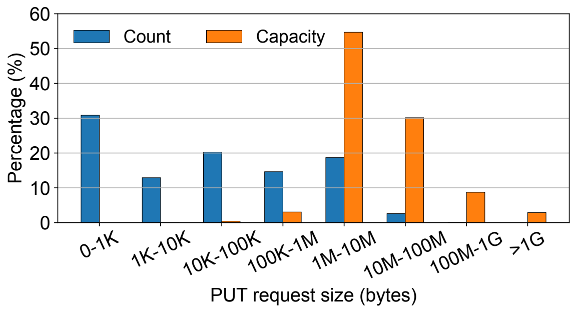


Figure 2. PUT request distribution in the IBM COS traces.

be able to maintain steady and predictable performance without significant resource overprovisioning.

There are many existing solutions for data replication of object storage. Major clouds provide proprietary services that support object replication across regions [35, 37, 38]. While strong consistency can significantly ease the development of some distributed applications, it imposes significant latency and availability trade-offs in wide-area environments. An eventual consistency model is a more practical choice and adopted by major clouds in their cross-region replication solutions. For use cases such as data archival [5], data lake [45], and content delivery [6], a modest replication lag is acceptable in exchange for higher performance and availability of the main application.

Yet these proprietary solutions have a couple of limitations. First, cloud providers lack the incentive to support cross-cloud data replication, as it would make it easier for users to migrate to a competitor. Second, intra-region replication delays within a cloud typically range from tens of seconds for small objects to tens of minutes for larger ones or during periods of high load [34]. Third, enabling eventually consistent replication incurs extra costs, including storage overheads on versioning and extra service fees on fast replication.

Skyplane [60, 81] is an open-source, platform-independent, VM-based solution that enables cross-cloud/region data replication between regions of major cloud providers, without requiring native support from them. Nevertheless, the VM-based approach has high replication delay and cost. Although VMs offer sufficient bandwidth for replication, provisioning a VM can take tens of seconds, making it difficult to handle transient traffic efficiently. Moreover, VMs often incur a substantial minimum billable duration, leading to non-negligible costs for short-duration tasks.

Serverless computing [63, 73] emerges as a promising solution, offering invocation times typically under a second and billing at millisecond-level granularity, making it ideal for short-duration tasks like replicating small objects. Based on our characterization, cloud functions, just like VMs, can provide satisfactory ingress/egress bandwidth, with performance scaling nearly linearly with the number of workers. This ensures that even relatively large objects can be replicated quickly with sufficient parallelism.

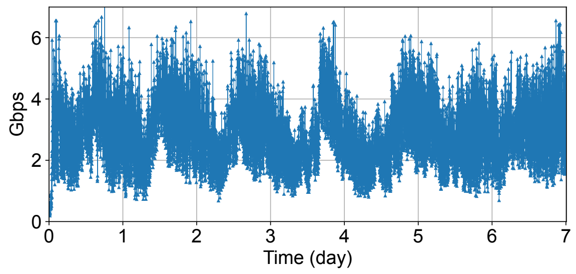


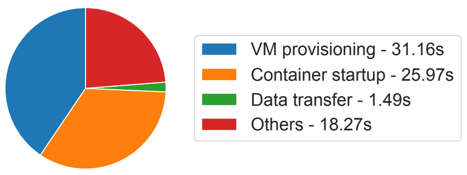
Figure 3. Write throughput in the IBM COS traces.

However, our characterization reveals two key challenges of using cloud functions for cross-cloud/region replication. First, the ingress/egress bandwidth is not simply dependent on the source and destination regions. Unique traits of different cloud platforms and regions must be taken into consideration. Second, the effective bandwidth of a cloud function can vary from instance to instance, and there is no clear pattern that we can leverage to predict and act in advance.

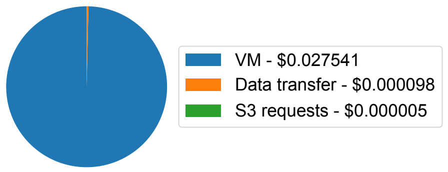
To address the inherent challenges of inter-cloud/region performance asymmetry and inter-instance performance variability of cloud functions, λReplica introduces dynamic strategy optimization and decentralized part-granularity scheduling. λReplica's strategy planner formulates a replication plan aligned with a user-defined SLO by leveraging a distribution-aware performance model. Its decision-making process integrates insights into the characteristics of cloud functions at both the source and destination regions, as well as the dynamics of λReplica's replication workflow at varying parallelization levels. At runtime, rather than dispatching data parts from each replicator instance uniformly, λReplica's replication engine enables replicator instances to autonomously acquire data parts from a shared pool, naturally adapting to performance variations and mitigating the long tail across the hundreds of replicator instances.

To further reduce the replication cost, λReplica avoids unnecessary replication as much as possible. The immutable nature of object storage forces a full replication of an object even when it is created from existing objects or only partially updated. λReplica opportunistically propagates changelogs rather than the objects. Common operations, such as copy and concatenation of existing objects, incur near-zero cost with λReplica. λReplica also aggregates frequent updates on hot objects as long as the user-defined SLO is not violated.

We implement a prototype of λReplica as a set of serverless functions with a shared core library and conduct a comprehensive evaluation on three major clouds (AWS, Azure and Google Cloud Platform). The evaluation results show that λReplica outperforms SkyPlane and the cloud platforms' proprietary solutions by reducing 61%-99% replication delay while achieving a cost saving of up to three orders of magnitude on common object sizes. We also run λReplica against a 60-minute real-world cloud object storage trace [33, 57] and demonstrate that λReplica can stay within a 10-second p99.99



(a) Time



(b) Cost

Figure 4. Breakdown of Skyplane replication time and cost.

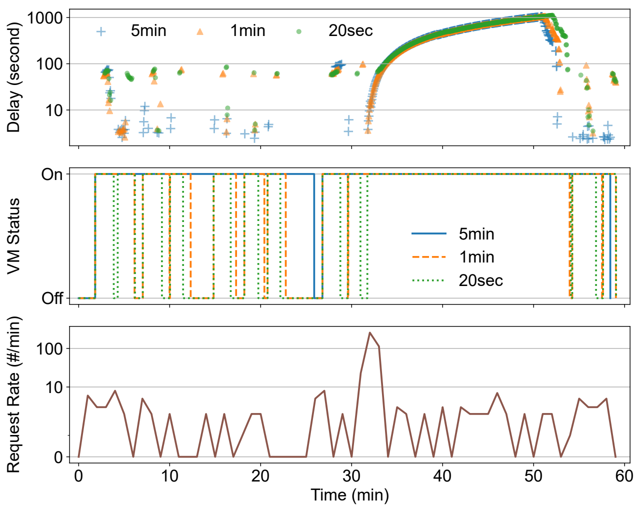


Figure 5. Handling dynamic workloads with Skyplane.

SLO for production workloads. A CLI version of λReplica is available at https://github.com/pkusys/LambdaReplicaCLI and https://doi.org/10.5281/zenodo.17198608.

2 Object Storage

Object storage is a data storage solution that abstracts data as discrete units (i.e., objects). One of the defining characteristics of object storage is its high scalability. It can handle vast amounts of data across distributed systems, making it ideal for cloud environments. All major cloud providers have object storage services today [10, 15, 30], and many cloud applications store their unstructured data in object storage.

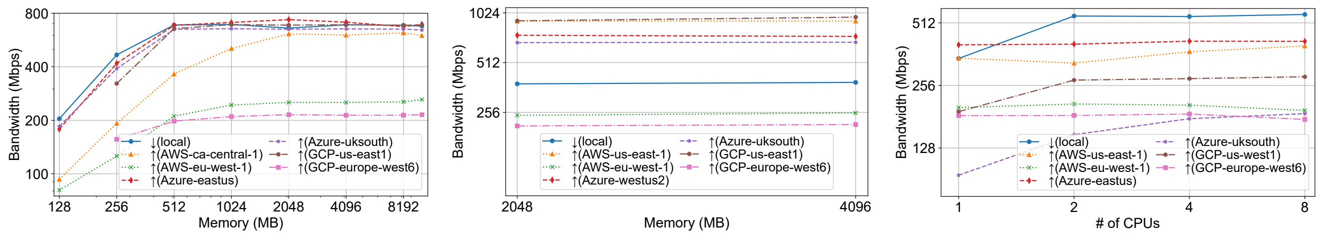
Read and write characteristics. Compared to traditional storage systems, object storage possesses distinct read and write characteristics. First, object storage provides a simple PUT/DELETE write interface. Once an object is created, it cannot be modified in part. When changes are made, a new version must be created to overwrite the original one. Second, the GET interface is more flexible, allowing users to read a continuous segment of an object from a given offset. It avoids unnecessary read of unneeded data and can accelerate read of large objects by parallelizing it. Third, object storage allows a large object to be divided into smaller parts and written in parallel. It improves the efficiency and reliability of uploading a large object by breaking it into manageable chunks.

Usage patterns. To reveal typical usage patterns of cloud object storage, we analyze the IBM Cloud Object Storage (COS) traces [33, 57]. The dataset contains around 1.6 billion requests. Each individual record represents an operation issued in IBM COS during a single week. From the traces, we draw two important conclusions. First, small objects dominate in cloud object storage. In Figure 2, we show the size distribution of all the PUT requests. ~80% of the PUT requests are below 1MB. Although other major clouds have not revealed exact numbers, estimates are in the same range based on public information [1]. Second, request rates are unstable over time. In Figure 3, we show the average write throughput per minute served in the traces. Throughput can change sharply from minute to minute. The variation becomes even more pronounced when analyzed at the per-tenant level.

Cross-cloud/region bucket replication. Cross-region bucket replication [35, 37, 38] is a feature offered by cloud providers that allows automatic asynchronous replication of objects between buckets in different regions. Using cross-region bucket replication can bring a few clear benefits. First, placing data closer can improve access latency for users who are geographically distant from the source bucket. Second, it avoids repetitive cross-region GET requests for frequently accessed items, thereby reducing data egress costs. Finally, it improves data availability and durability in case of regional outages.

However, cross-region bucket replication has its limitations. First, cloud providers have little incentive to support cross-cloud replication, as it could facilitate data migration to competitors. This lack of cross-cloud replication support prevents users from storing data across multiple clouds. Second, replication latency within a single cloud remains high and unpredictable, ranging from tens of seconds to tens of minutes [34]. Third, replication requires versioning to be enabled on both the source and destination buckets, which introduces additional storage costs and increases the complexity of managing the lifecycles of versioned objects.

Skyplane [60, 81] is an open-source replication system that bypasses the restrictions imposed by cloud providers, enabling object replication between major clouds. It further optimizes performance through the use of overlays. However, for each object it transfers, Skyplane creates one or more VMs, deploys containers on them, transfers the object, and then shuts the VMs down. The workflow is both slow and costly for common object sizes. In Figure 4, we characterize the time and cost of Skyplane replicating a 10 MB object from



(a) AWS us-east-1 **(b)** Azure eastus **(c)** GCP us-east1

Figure 6. Download/upload bandwidth vs. configuration (↓ for download and ↑ for upload).

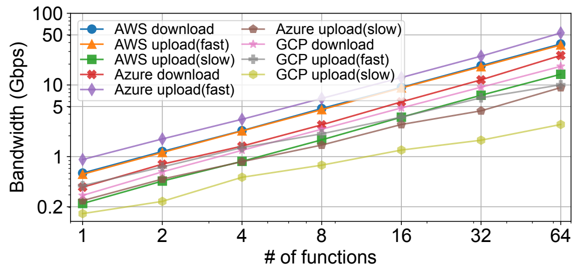


Figure 7. Aggregate bandwidth vs. # of functions.

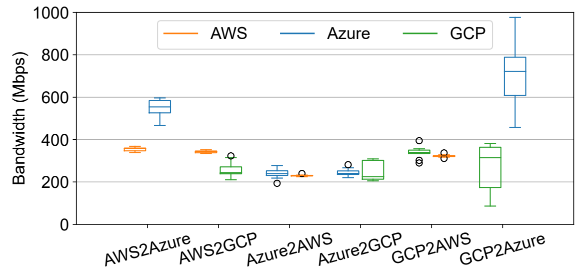


Figure 8. Asymmetric behaviors of different cloud functions.

AWS us-east-1 to us-east-2. Only 2% of the time is spent on data transfer, while over 99% of the cost is spent on the VMs.

The replication delay and cost of Skyplane remain significant even after we optimize the open-source version of Skyplane by keeping VMs alive and amortizing the overhead across multiple transfers. In Figure 5, one of the IBM COS traces from a tenant with moderate usage is replayed. We configure Skyplane to use one VM in each region and automatically shut down the VMs after they are idle for five minutes, one minute, and twenty seconds. The replication delay can reach a few minutes when VM instance provisioning is necessary and when there are transient bursts. And aggressively shutting down VMs in twenty seconds only saves less than 30% VM cost compared to a keep-alive strategy.

3 Charaterization of Serverless Computing

Considering the size distribution and load fluctuation of cloud object storage, serverless computing, such as AWS Lambda [13], Azure Functions [18], and Google Cloud Run Functions [28], becomes a natural replacement for VMs. Serverless computing is a cloud computing paradigm where the lower-layer compute resources (i.e., VMs and containers) are managed by the cloud providers [63, 73]. Instead, cloud functions are exposed to users as the interface.

Using cloud functions for object replication yields a few advantages. Compared to VMs, the start time of cloud functions is highly optimized [46, 48, 56, 69, 72], which reduces the overhead on small object replication. With high elasticity, we can create many parallel cloud functions instantaneously to react to sudden bursts. Furthermore, the millisecond-level billing of cloud functions can bring significant cost savings.

However, there is limited understanding about the performance of different serverless computing platforms on data transfer tasks between cloud regions, although previous work has characterized cross-cloud/region transfer rates of cloud VMs [60], and intra-region performance models are built for cloud functions or workflows of them [61, 62, 68, 74, 86, 88]. Therefore, we first validate the feasibility of serverless cross-cloud/region replication and identify the challenges.

Opportunity #1: satisfactory bandwidths. Unlike VMs which have a set of fixed specifications, the number of vCPUs and total memory size of a cloud function are often configurable in finer granularity. In Figure 6, we show the download and upload rates in cloud regions on the East Coast of the US. On AWS and Azure, only memory is configurable, and CPUs and network bandwidth scale proportionally with memory, whereas GCP allows to configure CPU and memory independently. We have three key findings. First, all three clouds provide a bandwidth of a few hundred Mbps between different regions, which allows users to transfer a small object in seconds with a single function. Second, the links between geographically close regions are generally faster, although local access may not be the fastest. Third, there is a sweet spot for each platform where one cannot achieve a higher bandwidth with a more expensive configuration.

Opportunity #2: near-linear performance scaling. However, a naive serverless approach is unviable for arbitrary objects. Cloud functions have a hard execution time limit(e.g., 15 minutes for AWS Lambda [23]). While a single function can achieve a bandwidth of several hundred Mbps, this still imposes an upper bound on the size of an object that can be

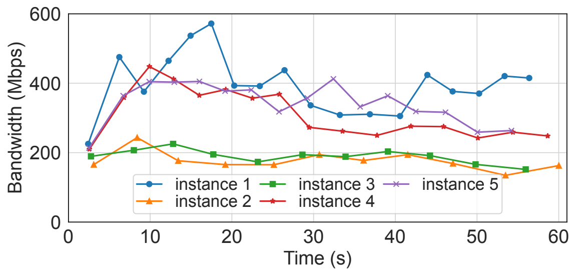


Figure 9. Performance variability of function instances.

replicated in an invocation. To allow replication of a large object and concurrent replication of many objects, we next verify whether serverless computing can scale with the number of functions. In Figure 7, we run a given number of functions in parallel and sum up their aggregate bandwidth. The results show that i) the aggregate bandwidth increases near-linearly with the number of functions for all three platforms; ii) we can easily reach over a few Gbps aggregate bandwidth with 64 or fewer functions, even for those slow links.

Challenge #1: asymmetric performance of clouds/regions. Unlike Skyplane which creates VMs in both the source and destination regions, data replication with cloud functions has to be one-sided. The reason is that two cloud functions are not addressable by each other. In Figure 8, we replicate a 1GB object pairwise between AWS us-east-1, Azure eastus, and GCP us-east1. Surprisingly, the replication speeds do not only depend on the source and destination pair but also vary based on where the cloud functions are run. Both the average speed and the variance may differ between platforms. Therefore, a replication system has to choose the right platform and region to run the functions in order to meet its target SLO.

Challenge #2: performance variability of instances. Although the aggregate performance can grow with increased concurrency in Figure 7, each function instance does not contribute to the aggregate bandwidth equally. In Figure 9, we run five function instances and each of them transfers a 1GB object from AWS us-east-1 to Azure eastus repetitively. The bandwidth between instances differs by more than a factor of 2. Replication speeds can vary significantly, even when the source and destination regions are identical and the instances have exactly the same configuration. There are no clear patterns indicating which function instance is more likely to be impacted. From a user's perspective, it appears completely random. This means that when there is a large amount of function instances, some of them can be expected to run slowly.

4 λReplica Overview

λReplica is a serverless object replication system that achieves sub-minute replication delay with minimal cost. As Figure 10 shows, at the heart of λReplica is a replication strategy planner based on its distribution-aware performance model and a distributed replication engine that employs decentralized part-granularity scheduling.

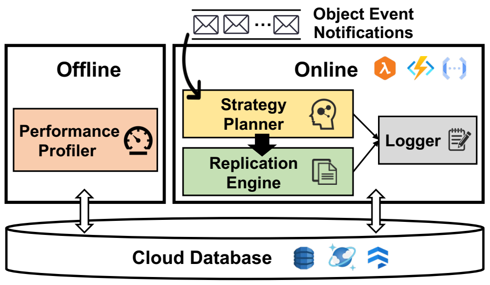


Figure 10. λReplica overview.

Replication engine. The replication engine replicates an object or parts of it from the source bucket to the destination bucket with one or more cloud functions. When an object is replicated in a distributed manner, the replication engine mitigates the performance variability among function instances with autonomous subtask distribution (§5.1). The replication engine also guarantees eventual consistency (§5.2).

Performance profiler. When a cloud platform or a user wants to onboard a new cloud platform or a new cloud region to λReplica, it requires offline profiling to collect necessary performance metrics. The performance model that λReplica relies on tunes its parameters for new platforms/regions according to the profiling results (§5.3).

Strategy planner. Before replicating an object, λReplica's strategy planner decides the appropriate region to execute the task and the level of parallelism based on the performance model with a goal to meet the SLO (§5.3). The strategy planner opportunistically optimizes replication cost with changelog propagation and SLO-bounded batching (§5.4).

Logger. Because the transfer rates between regions may change after offline profiling, in order to keep the performance model accurate over time, a logger keeps track of the replication time of representative tasks and periodically updates the parameters of the performance model.

5 λReplica Design

In this section, we first introduce the workflow of λReplica's replication engine, especially how λReplica handles performance variability in distributed replication (§5.1) and how it ensures eventual consistency (§5.2). Next, we formulate the replication time of λReplica's workflow as a distribution-aware analytical model and show how to derive an SLO-compliant replication plan from it (§5.3). Finally, we describe our opportunistic cost optimization strategy of changelog propagation and SLO-bounded batching (§5.4).

5.1 Variability-Tolerant Replication Engine

The workflow of cross-cloud/region replication. Figure 11 shows the four stages of replicating an object from one region

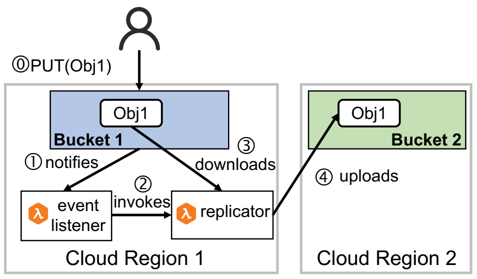


Figure 11. The workflow of serverless object replication.

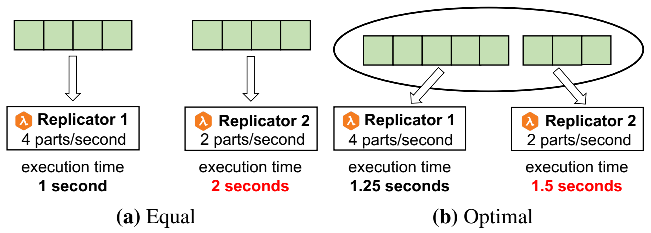


Figure 12. Sub-optimality of equal distribution of data parts.

to another when ⓪ an object is created. ① Cloud notification: when an object is created or deleted, a JSON-format notification is generated by the cloud platform [17, 22, 25]. The notification includes the metadata of an object, which allows a function/VM to post-process it accordingly. ② Function invocation: one or more replicator functions are invoked either at the source region or at the destination region and become ready for data replication. ③ Data download: a replicator function downloads data from the source bucket to its local storage. ④ Data upload: a replicator function uploads data from its local storage to the destination bucket.

Distributed replication. While most objects can be replicated with a single function, replication of a relatively large object (e.g., > 64MB) can be significantly accelerated by downloading and uploading parts of it in a distributed manner with additional functions. The key problem is how the data parts are distributed to the replicator functions. Because an orchestrator can not directly communicate with the functions after invoking them, a straightforward approach is to assign a fixed set of parts to each function at invocation, which incurs minimal scheduling overhead. This approach works fairly well when there is little performance variability. However, as Figure 9 shows, the performance of concurrent function instances can be very diverse. Figure 12 provides an illustrative example of how performance variability impacts the replication time of distributed replication. Replicator 1 can replicate four data parts per second, while Replicator 2 can only process two per second. The optimal plan is to assign five data parts to Replicator 1 and three to Replicator 2 rather than giving each function four parts. When many performance-variable function instances cooperate on a replication task, it is more likely to observe one or more extremely slow instances.

Algorithm 1 Decentralized part-granularity scheduling

```
1:  function ORCHESTRATOR(obj, num_func)
2:      task_id ← init_replication(obj)
3:      completed_parts[task_id] = 0
4:      num_parts ← create_part_pool(obj)
5:      for $i$ = 1..num_func do
6:          invoke(REPLICATOR(task_id, num_parts))
7:  function REPLICATOR(task_id, num_parts)
8:      while part_pool $\neq \emptyset$ do
9:          part ← get_part_from_pool()
10:         download_and_upload(task_id, part)
11:         completed_parts[task_id] += 1
12:         if completed_parts[task_id] = num_parts then
13:             finish_replication(task_id)
```

Essentially, our goal for distributed replication is to minimize the maximum execution time across all the replicator functions. However, it is practically difficult to achieve optimal scheduling because the performance of each function instance is not static, and we have no means to accurately forecast it. Therefore, to achieve the best possible scheduling under uncertainty, we have to schedule at data part granularity instead of scheduling all the parts at once.

Decentralized part-granularity scheduling. Our key idea is to assign a data part to an available function instance as soon as possible. Unlike using VMs, an orchestrator function cannot directly provide additional instructions to the replicator functions once the replicator functions are invoked. Instead, as Algorithm 1 depicts, a replication task is first assigned a unique task_id at line 2, we then create a pool of data parts with their metadata kept in shared external storage (i.e., cloud databases) at line 4. Whenever a replicator function becomes available, it actively retrieves another part from the pool and replicates it by calling cloud APIs to download and upload the parts (lines 9-10). When all the parts are replicated, the replicator function that delivers the last part concludes the replication task (line 13).

The size of these parts is a critical parameter. There is a fundamental trade-off: larger parts are more efficient by avoiding extra API calls but limit scheduling flexibility, as a slow function can delay the entire process if it gets stuck on a large part. Conversely, smaller parts allow for finer-grained load balancing but increase the overhead per part. Through empirical analysis, we find that a part size of 8MB strikes an effective balance, as we observe only marginal overhead reduction beyond this size.

Decentralized part-granularity scheduling allows fast function instances to process more data parts than slow ones so that the execution time across the function instances is more balanced. Furthermore, it requires no interaction between the participants and triggers only two external storage accesses per data part (one to claim the part and another to update its

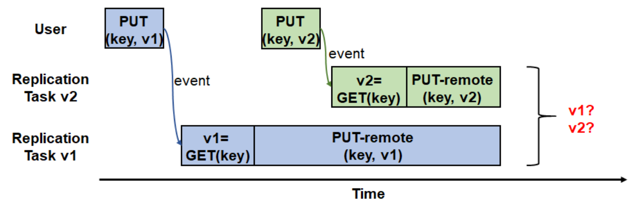


Figure 13. Concurrent replications of the same object.

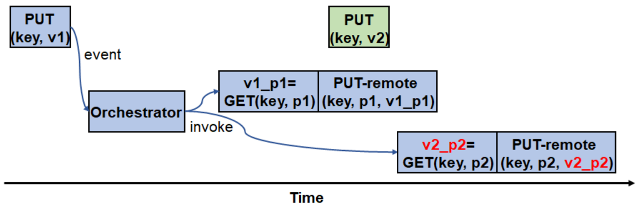


Figure 14. Invalid object with inconsistent parts.

status), achieving the benefits with minimal overhead. Using managed NoSQL databases like Amazon DynamoDB [8], these operations typically complete with single-digit millisecond latency. The monetary cost (e.g., \$0.6250 per million writes for Amazon DynamoDB in the us-east-1 region) is also negligible compared to the replication itself (single-digit dollars per 100GB.

5.2 Eventual Consistency Guarantees

λReplica's replication engine must ensure the objects are eventually replicated to the destination correctly. Existing bulk replication systems [60, 81] assume that the replicated objects are static and there are no concurrent replication tasks. Alternatively, the cross-region bucket replication feature [35, 37, 38] provided by cloud platforms relies on object versioning where each version of an object is static and can be replicated without interfering with others.

Although enabling versioning ensures correctness, it incurs consequential storage cost. For example, if each object is updated once a day, versioning at least doubles the storage cost because the lifecycle rules are at day-granularity (i.e., a non-current version must wait for at least a day to expire). Our goal is to achieve eventual consistency without incurring significant extra cost or changing user behaviors.

There are two race conditions which can break eventual consistency without versioning enabled under λReplica's design. First, in Figure 13, if two PUT operations are called on the same object in the source bucket, two concurrent PUTs will be triggered on the destination object. Object storage does not have a deterministic behavior on concurrent writes on the same object. Possible results include: i) all of the requests succeed and the resulting object can come from any of them; ii) some or all of the requests fail. Therefore, we must avoid concurrent replications. Second, in Figure 14, when a large object is split into parts and handled by multiple replicators,

Algorithm 2 Replication Lock

```
1:  function LOCK(obj_key, etag, seq)
2:      lock ← try_lock(obj_key)
3:      if lock.status = success then
4:          return true
5:      else if lock.seq = NULL or lock.seq < seq then
6:          lock.seq = seq; lock.etag = etag
7:          return false
8:  function UNLOCK(obj_key, lock)
9:      etag ← lock.etag; seq ← lock.seq
10:     release_lock(lock)
11:     if etag ≠ NULL then
12:         cur_etag ← get_etag(obj_key)
13:         if etag ≠ cur_etag then
14:             trigger_replication(obj_key, etag, seq)
```

if another PUT request is made successfully in the middle, the replicated object can be assembled from inconsistent parts.

Object-granularity replication lock. To prevent concurrent PUTs in replication tasks, we enforce serial replications with a distributed lock which can be easily implemented with a cloud database (e.g., Amazon DynamoDB [9]). In Algotithm 2, when there is an ongoing replication task, we keep track of the ETag of the latest version in sequential attempts. ETag is essentially a platform-generated content hash of the object [11]. When we release the lock when a replication task finishes, we compare the pending ETag with the most recently replicated ETag. If they mismatch, we invoke the orchestrator again in case the latest version is not replicated yet.

Optimistic replication with validation. To avoid replicating inconsistent parts, each replicator checks that ETag matches the one provided by the orchestrator. If a mismatch is found, the ongoing replication task is aborted. We expect a retry will go through unless an extremely large object is updated frequently which should be rare. In that case, the object must be locked in order to let λReplica replicate it successfully.

5.3 Dynamic Replication Plan Generation

With the workflow of λReplica explained, we next show how the replication plan is decided in λReplica. To decide where to run the replication task and how many parallel functions should be invoked, a performance model that predicts the replication time of a plan is necessary.

The cloud platform fully owns the cloud notification stage, and λReplica takes over the replication task after receiving the notification. If the user-defined SLO for replicating an object is SLO, and it takes T_n to deliver the notification after an object is created, our goal is to provide a bound of the aggregate time of the remaining three stages T_{rep}, which should be within $SLO - T_n$. Note that if a user defines an unreasonably tight SLO, the SLO might have been already violated when λReplica receives the notification.

There are two requirements for designing the performance model of λReplica. First, although common objects can be replicated with a single cloud function, replicating relatively large objects requires distributed replication with multiple cloud functions to avoid timeouts and meet the SLOs. Therefore, the performance model of λReplica must be twofold, covering both cases. Second, as previous work points out [68, 88], to meet a target SLO when using cloud functions, the performance model must be distribution-aware so that performance variability is taken into account.

Single replicator function. We decompose the replication time T_{rep} into two parts: the time to start the replicator function T_{func} and the time to download and upload the object $T_{transfer}$. T_{rep} is represented as:

$$T_{rep} = T_{func} + T_{transfer}$$

When an object is small enough, the orchestrator that receives the notification can handle the replication locally. In that case, T_{func} is zero. When another replicator function is necessary, the replicator becomes ready after the API invocation time I and a delay of D. These two parameters are related to where the function is run. Therefore, T_{func} can be expressed as:

$$T_{func} = \begin{cases} 0 & size(obj) < threshold \\ I(loc) + D(loc) & otherwise \end{cases}$$

On the other hand, $T_{transfer}$ is positively correlated to the object size. We notice that an overhead of S exists for the cloud clients to become ready for data transfer. We note the time of transferring a unit of data (chunk size c) as C. The total time of transferring the object is C times the number of chunks in the object. $T_{transfer}$ then becomes:

$$T_{transfer} = S(src, dst, loc) + C(src, dst, loc) \times \lceil size(obj)/c \rceil$$

Parallel replicator functions. We next extend our model to reflect the performance of distributed replication. It is harder to estimate the replication time when multiple functions are involved. However, the model is allowed to overestimate the replication time to some extent as long as we can find an SLO-compliant replication plan. We still decompose T_{rep} to T_{func} and $T_{transfer}$ for simplicity.

When starting n functions, we assume that each asynchronous invocation is pipelined, so we do not need to add up the delay D for each function. However, the scheduling behavior of each cloud platform is an additional factor that we need to consider. When a large number of function calls are received, it can take a few seconds for a cloud platform's scheduler to add new instances. For example, the scheduler of Google Cloud Run Functions runs every five seconds [7]. We observe a similar behavior on Azure functions. We denote the scheduling postponement as P, T_{func} then becomes:

$$T_{func} = I(loc) \times n + D(loc) + P(loc)$$

Algorithm 3 Dynamic replication strategy planning

```
    p: percentile of distribution
    n_max: the maximum parallelism
 1: function GENERATE_PLAN(obj, SLO_e2e, p)
 2:     SLO_rep ← SLO_e2e - (now - obj.timestamp)
 3:     best_time, best_plan ← ∞, NULL
 4:     for $i = 1, 2, ...,$n_max do
 5:         for loc ∈ $\{src, dst\}$ do
 6:             time ← T_rep(i, loc, obj, p)
 7:             if time < best_time then
 8:                 best_time, best_plan ← time, (i, loc)
 9:             if best_time < SLO_rep then
10:                 return best_plan
11:     return best_plan
```

For $T_{transfer}$, the exact formulation should be the maximum execution time across all the function instances. However, how many chunks each function replicates is uncertain under part-granularity scheduling. We assume that the slowest function processes fewer chunks than average because it takes a longer time to process a chunk. The time consumed per chunk is also different for distributed replication because it also involves cloud database accesses. Therefore, a practical upper bound of $T_{transfer}$ is:

$$T_{transfer} = \max_{1 \le i \le n} \{S(src, dst, loc) + C'(src, dst, loc) \times \left\lceil \frac{size(obj)}{c \times n} \right\rceil\}$$

The performance model above can predict the replication time for any object size, and we intentionally make the parameters easy and affordable to profile.

Awareness of distribution. In our formulation, the profiler is responsible for identifying the parameters I, D, P, S, C, and C'. Because we notice certain clouds and regions have high performance variability as shown in Figure 9, we need to describe these parameters as distributions instead of a trivial values. The profiler needs to collect enough samples to generate a representative distribution.

For simplicity, unless we clearly notice an unusually long tail, we fit the samples to a normal distribution. Because our performance model is mostly weighted sums of the parameters, the results also follow a normal distribution. We can easily calculate the desired percentile with statistics libraries. One exception is $T_{transfer}$ for multiple replicator functions because it is the maximum across all function instances. For most values of n, we use Monte Carlo methods to obtain a distribution of $T_{transfer}$. Rather than running for each planning request, the simulation is an on-demand process triggered in two scenarios: first, to bootstrap the model for a given replication path and parameters, and second, when the system detects a significant, persistent deviation between its predictions and actual task performance. However, for large n, resampling will be too time-consuming. Instead, based on the extreme

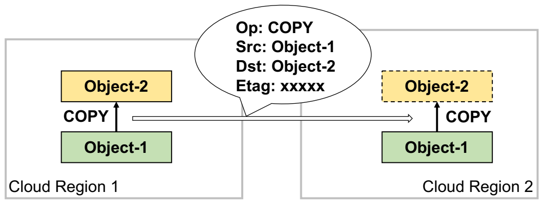


Figure 15. Propagating COPY operation.

value theory, we can use Gumbel distribution to represent the maximum of n i.i.d. random variables for large n, which is significantly faster than Monte Carlo methods.

SLO-compliant plan generation. With the performance model established, we can compare the estimated replication time of different plans. The goal of λReplica's strategy planner is to generate an SLO-compliant replication plan with the lowest cost rather than the fastest one.

However, it is unnecessary to calculate the exact cost of each plan. When more function instances are invoked, additional API calls are triggered. It also incurs extra performance overhead, which increases the aggregate function execution time. Therefore, in Alogithm 3, we start with the single-function strategy and iterate through different parallelisms exponentially. For each level of parallelism, we compare the plan of utilizing functions at the source region with the plan of utilizing functions at the destination region. As soon as we identify an SLO-compliant plan, we return it immediately without evaluating all the remaining possible plans. If there is no SLO-compliant plan, the fastest plan is returned.

The algorithm takes a user-defined percentile (e.g., p90 or p99) as an input. The performance model will output the replication time t where $P(T \leq t) \geq p$. Because we use Monte Carlo methods to generate the distribution for parallel functions, we restrict the resampling time to avoid violating the SLO due to slow plan generation.

5.4 Opportunistic Replication Reduction

We further optimize the cost of cross-cloud/region object replication where data egress cost dominates after unnecessary compute cost is opted out by replacing VMs with cloud functions. Our principle for data egress cost reduction is to avoid unnecessary data replication as much as possible.

There are two key insights that we can leverage to avoid unnecessary replication. First, object storage is unaware of how a new object is generated because it only provides a simple PUT write interface. For example, when text content is stored in object storage, copy/move/concatenation/append from existing files are common operations. And when object storage is used as a substitute for block storage [55, 78], a partial update can also occur. It does not create a difference cost-wise when the data is only stored in one cloud region. However, when cross-cloud/region replication is required, it forces the replication system to copy the entire object instead

Algorithm 4 SLO-bounded batching

```
1:  function BATCH(obj, $SLO_{e2e}$)
2:      deadline ← obj.timestamp + $SLO_{e2e}$
3:      if now + $T_{rep}$(obj) + $\epsilon \geq$ deadline then
4:          if obj.etag ∈ pending_etags[obj.key] then
5:              pending_etags[obj.key].clear()
6:              $obj_{new}$ ← latest_object_metadata(obj.key)
7:              schedule_replicate($obj_{new}$)
8:      else
9:          pending_etags[obj.key].push(obj.etag)
10:         delay(BATCH(obj, $SLO_{e2e}$), deadline - $T_{rep}$(obj))
```

of just the changes. Therefore, we argue that additional hints from the cloud users help reduce the cost for such scenarios.

Second, it is unnecessary to replicate every single version of an object. For example, if the SLO is 60 seconds and an object is updated once per second, we can buffer and merge the updates to replicate the object once or twice a minute, while still meeting the SLO. By doing so, the cost can be reduced in proportion to the update frequency.

Changelog propagation. Instead of replicating the full content of the new object, we propagate the changelog to the destination to opportunistically reduce the cross-cloud/region transmission. A changelog is generated at the user program as a hint to λReplica, which can be created by the user or automated by program analysis. In Figure 15, we show how a COPY operation is mirrored from Region 1 to Region 2. When a new object version is created, λReplica will find a corresponding COPY changelog in the cloud database. λReplica propagates only the changelog to Region 2, where the changelog is applied locally.

A caveat here is that λReplica may already replicate a newer version of the source object to Region 2. So the changelog must include the ETag of the source object. Before a changelog can be applied, λReplica checks if the current ETag matches the requested ETag to ensure correctness.

SLO-bounded batching. When the target SLO is relatively loose for certain object sizes, it creates extra space for cost optimization. We do not need to replicate an object immediately. Instead, we take the opportunity to delay the replication towards its deadline so multiple updates can be aggregated into one. As Algorithm 4 depicts, a delayed replication task always replicates the newest version. Other replication tasks which find their versions or newer versions have already been replicated can just quit without taking any further actions.

6 Discussion

Fault tolerance and consistency guarantee. A cloud function can timeout or fail mid-execution. λReplica relies on the auto-retry mechanism provided by the cloud platforms [19, 24, 44] on unexpected faults. λReplica can retry

Table 1. Replication delay and cost from AWS us-east-1

Cloud			AWS			Azure			GCP		
Region			ca-central-1	eu-west-1	ap-northeast-1	eastus	uksouth	southeastasia	us-east1	europe-west6	asia-northeast1
1MB	Delay (second)	λReplica	1.5	1.5	1.6	1.3	2.0	2.5	1.4	2.2	3.3
		Skyplane	76.2	84.7	90.2	134.3	146.5	149.4	109.4	126.5	115.2
		S3 RTC	21.3	24.1	24.5	N/A	N/A	N/A	N/A	N/A	N/A
		Δ	**-92.79%**	**-93.62%**	**-93.60%**	**-99.00%**	**-98.66%**	**-98.32%**	**-98.76%**	**-98.26%**	**-97.09%**
	Cost (10^{-4}\$)	λReplica	0.3	0.3	0.3	0.9	1.0	1.0	1.0	1.0	1.1
		Skyplane	541.6	541.6	586.9	768.9	1104.9	1152.9	771.9	1238.1	845.2
		S3 RTC	0.4	0.4	0.4	N/A	N/A	N/A	N/A	N/A	N/A
		Δ	**-32.25%**	**-31.43%**	**-28.55%**	**-99.88%**	**-99.91%**	**-99.91%**	**-99.88%**	**-99.92%**	**-99.87%**
128MB	Delay (second)	λReplica	2.7	3.8	5.0	2.2	3.6	7.5	3.4	6.8	8.6
		Skyplane	82.8	88.7	92.5	139.4	151.8	159.2	116.2	127.8	131.0
		S3 RTC	15.3	16.1	17.1	N/A	N/A	N/A	N/A	N/A	N/A
		Δ	**-82.52%**	**-76.49%**	**-70.77%**	**-98.39%**	**-97.61%**	**-95.28%**	**-97.08%**	**-94.68%**	**-93.47%**
	Cost (10^{-4}\$)	λReplica	26.8	27.6	28.1	113.5	114.5	116.6	114.3	116.1	117.0
		Skyplane	567.4	567.3	612.6	881.0	1217.0	1521.1	1143.4	1350.6	1289.9
		S3 RTC	44.3	44.3	44.3	N/A	N/A	N/A	N/A	N/A	N/A
		Δ	**-39.59%**	**-37.72%**	**-36.59%**	**-87.12%**	**-90.59%**	**-92.33%**	**-90.00%**	**-91.40%**	**-90.93%**
1GB	Delay (second)	λReplica	3.6	6.5	10.3	3.4	4.3	13.5	3.8	7.2	10.3
		Skyplane	82.9	91.6	95.2	141.8	153.3	165.9	117.2	131.6	145.3
		S3 RTC	24.6	25.2	26.6	N/A	N/A	N/A	N/A	N/A	N/A
		Δ	**-85.37%**	**-74.28%**	**-61.11%**	**-97.63%**	**-97.21%**	**-91.89%**	**-96.79%**	**-94.52%**	**-92.91%**
	Cost (10^{-4}\$)	λReplica	212.4	218.9	222.9	907.7	913.9	929.8	912.4	924.5	934.2
		Skyplane	749.0	748.4	793.3	1927.5	2008.2	2312.3	1937.0	2144.1	2083.5
		S3 RTC	353.9	353.6	353.4	N/A	N/A	N/A	N/A	N/A	N/A
		Δ	**-39.99%**	**-38.08%**	**-36.92%**	**-52.90%**	**-54.49%**	**-59.79%**	**-52.90%**	**-56.88%**	**-55.16%**

without extra error handling because object storage's PUT API is idempotent. To avoid repetitively retrying on permanent failures, the failed event is moved to a dead-letter queue after exceeding the maximum retries.

λReplica provides eventual consistency as the existing replication systems [35, 37, 38, 60], which can support use cases like disaster recovery and content delivery. While λReplica is non-intrusive to user applications, achieving stronger consistency will require customizd APIs.

Resource limitations and overlay networks. Although cloud platforms can virtually provide unlimited resources, a user account usually has a set of static limits that are adjustable (e.g., the number of concurrent function instances). By default, AWS and Azure provide 1,000 concurrent instances [20, 43], which are enough for common use cases. A user can also request a quota increase if necessary.

An overlay network can accelerate cross-cloud/region replication at extra cost, when there is a certain resource limit as in Skyplane. It is orthogonal to λReplica and can become useful when a user's target throughput is extremely high and the resource limit cannot be lifted further.

Emerging Use Cases. Beyond traditional applications like disaster recovery, fast cross-cloud/region replication has the potential to play a role for emerging data-intensive workloads that rely on object storage. A prominent example is the global distribution of machine learning models and other AI artifacts [58, 75]. As organizations deploy models across multiple regions and clouds to serve a global user base, the ability to rapidly and cost-effectively replicate massive model files (often tens to hundreds of gigabytes) is paramount for minimizing deployment times and ensuring consistent performance. Similarly, geo-distributed training workflows [76] depend on the efficient synchronization of large datasets across sites. The on-demand, highly parallel nature of λReplica is particularly well-suited for these bursty, large-scale data movement tasks, addressing the communication bottlenecks in the modern AI/ML lifecycle.

7 Implementation

We implement a system prototype of λReplica with ~5200 lines of code in Python. The offline performance profiler is a set of test cases that run a few times for each distinct configuration. The replication strategy planner and replication engine are two individual code modules packaged with the SDKs of cloud platforms.

The prototype supports data replication between any public regions of AWS, Azure, and GCP. We deploy the replication strategy planner and replication engine as cloud functions to AWS Lambda [13], Azure Functions [18], and Google Cloud Run Functions [28], respectively. Any intermediate states of a replication task are stored in Amazon DynamoDB [8], Azure Cosmos DB [16], and Google Firestore [31], which are all serverless databases on a pay-as-you-go basis. We rely on cloud-managed serverless workflows to realize SLO-bounded batching [14, 32, 42].

8 Evaluation

In this section, we evaluate our prototype of λReplica from the following aspects: i) replication delay and cost against both open-source and cloud proprietary baselines (§8.1); (ii) effectiveness of individual techniques (§8.2); (iii) performance on a real-world object storage trace (§8.3).

Setup. We conduct our experiments on AWS, Azure, and GCP. We manually configure cloud functions so that they achieve the best performance at the lowest cost. On AWS, the memory sizes of AWS Lambda are between 512MB and

Table 2. Replication delay and cost from Azure eastus

Cloud			AWS			Azure			GCP		
Region			us-east-1	eu-west-1	ap-northeast-1	westus2	uksouth	southeastasia	us-east1	europe-west6	asia-northeast1
1MB	Delay (second)	λReplica	0.1	0.7	0.9	0.8	0.9	1.7	0.7	1.4	2.2
		Skyplane	113.2	114.1	119.6	131.6	142.2	147.9	136.3	149.7	142.9
		AZ Rep	N/A	N/A	N/A	61.3	61.3	64.0	N/A	N/A	N/A
		Δ	**-99.93%**	**-99.37%**	**-99.25%**	**-98.77%**	**-98.48%**	**-97.28%**	**-99.51%**	**-99.04%**	**-98.49%**
	Cost (10^{-4}\$)	λReplica	1.0	1.1	1.2	0.4	0.8	0.8	1.0	1.0	1.2
		Skyplane	768.9	827.6	1174.2	768.2	1104.5	1152.5	1030.9	1238.0	1177.4
		AZ Rep	N/A	N/A	N/A	0.2	0.5	0.5	N/A	N/A	N/A
		Δ	**-99.87%**	**-99.87%**	**-99.90%**	**+97.43%**	**+44.64%**	**+46.71%**	**-99.90%**	**-99.92%**	**-99.90%**
128MB	Delay (second)	λReplica	3.7	6.1	6.0	8.2	12.1	15.8	6.6	14.1	16.1
		Skyplane	115.3	122.4	124.8	140.6	147.9	150.1	138.3	150.6	149.5
		AZ Rep	N/A	N/A	N/A	61.2	60.8	70.3	N/A	N/A	N/A
		Δ	**-96.80%**	**-95.00%**	**-95.23%**	**-86.55%**	**-80.02%**	**-77.56%**	**-95.24%**	**-90.63%**	**-89.25%**
	Cost (10^{-4}\$)	λReplica	110.9	116.1	112.0	27.3	64.7	78.5	113.9	121.5	124.0
		Skyplane	1133.7	1192.4	1283.0	1049.4	1167.0	1215.0	1139.6	1346.8	1286.2
		AZ Rep	N/A	N/A	N/A	25.4	63.0	63.0	N/A	N/A	N/A
		Δ	**-90.21%**	**-90.26%**	**-91.27%**	**+7.38%**	**+2.66%**	**+24.58%**	**-90.01%**	**-90.98%**	**-90.36%**
1GB	Delay (second)	λReplica	4.8	3.8	8.4	17.6	16.8	24.3	10.0	15.8	19.8
		Skyplane	123.5	123.9	129.3	140.6	152.5	174.9	145.8	175.5	170.9
		AZ Rep	N/A	N/A	N/A	62.4	62.8	75.4	N/A	N/A	N/A
		Δ	**-96.14%**	**-96.90%**	**-93.47%**	**-71.78%**	**-73.24%**	**-67.78%**	**-93.14%**	**-91.02%**	**-88.39%**
	Cost (10^{-4}\$)	λReplica	883.5	885.1	889.9	301.1	612.4	637.6	956.5	1031.0	1042.1
		Skyplane	1900.8	1959.4	2049.7	1227.2	1607.9	1976.1	1906.7	2476.4	2053.2
		AZ Rep	N/A	N/A	N/A	203.2	503.9	504.0	N/A	N/A	N/A
		Δ	**-53.52%**	**-54.83%**	**-56.59%**	**+48.23%**	**+21.54%**	**+26.51%**	**-49.84%**	**-58.37%**	**-49.25%**

1GB. The memory sizes of Azure Functions are configured to 2048MB, which is the minimum. Google Cloud Run Functions is configured to 1024MB memory and 1-2 vCPUs.

Baselines. We compare λReplica with the v0.3.2 release of Skyplane [40, 60, 81] and the proprietary replication services AWS S3 Replication Time Control (S3 RTC) [34] and Azure object replication (AZ Rep) [35]. Skyplane allows object replication between public regions of major clouds. S3 RTC supports replication between two AWS buckets with a 15-minute SLO, while Azure object replication has no SLO guarantee. As a prerequisite for using AWS S3 RTC and Azure object replication, object versioning is enabled.

Metrics. The main metrics that we focus on are replication delay and cost. For a fair comparison of the systems, we define the replication delay as the time from completion of a PUT request and a successful retrieval of the version or its subsequent versions in the destination region. The VM deprovisioning time is excluded for Skyplane. The cost is comprehensively estimated based on the listed prices of the cloud products and metered usage from the recorded logs.

8.1 Replication Delay and Cost

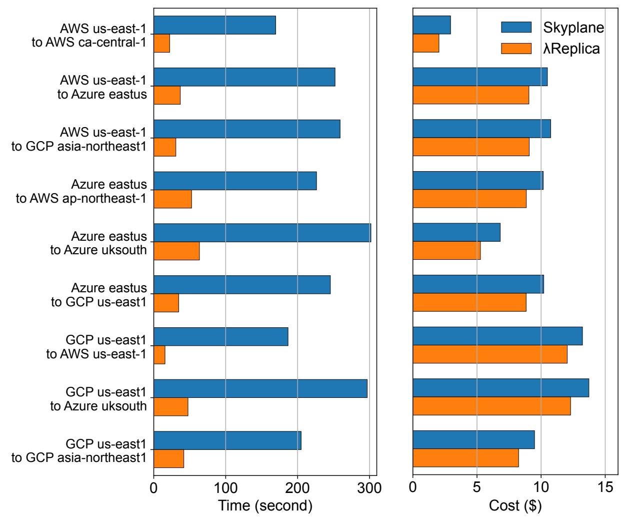
We first compare the replication delay and cost of λReplica under different object sizes. Three representative object sizes (1MB, 128MB, and 1GB) are chosen for comparison. The sizes of ~80% objects are equal to or below 1MB in the IBM COS trace. For a 128MB object, distributed replication usually outperforms a single replicator function. And over 99.99% of the objects are below 1GB in the IBM COS trace, which covers the vast majority of the objects. To show that λReplica can also be used as an efficient bulk replication tool, we also compare the replication time and cost with Skyplane at 100GB. For Skyplane, we use one VM per region by default and eight VMs per region for the 100GB experiment. For objects smaller than a few gigabytes, provisioning additional VMs fails to improve performance. Moreover, this approach unnecessarily increases costs, while the significant variability in VM startup times can paradoxically extend the end-to-end replication delay. To show the best performance that λReplica can achieve, the SLO is set to zero for these experiments so that λReplica always chooses the fastest replication strategy.

AWS results. Table 1 shows the replication delay and cost of λReplica from AWS us-east-1 to the other nine regions. λReplica outperforms the best baseline in every experiment and reduces the replication delay by 61%-99%. The replication delay of S3 RTC is between 15 and 26 seconds, while it takes Skyplane at least 76 seconds to replicate an object. λReplica is able to keep a single-digit replication delay for most of the experiments, except for some Asian regions, which have slower links. The replication delay to Asian regions can be further optimized with finer-granularity manual tuning of the chunk size and level of parallelism. λReplica also reduces the cost by 28.5%-99.9%. Using λReplica with underlying Lambda and DynamoDB is more cost-effective than S3 RTC, providing 28.5%-39.9% cost savings.

Azure results. Table 2 shows the replication delay and cost of λReplica from Azure eastus to the other nine regions. Overall, the replication delay is reduced by 67%-99%. Azure object replication consistently exhibits >60 seconds delay. Skyplane on Azure is also slower because it takes longer to provision Azure VMs. λReplica on Azure is not as fast as on AWS because Azure functions's cross-region links are relatively unstable when multiple functions are invoked. λReplica is more expensive than Azure object replication which is free of charge while providing no SLO guarantee.

Table 3. Replication delay and cost from GCP us-east1

Cloud			AWS			Azure			GCP		
Region			us-east-1	eu-west-1	ap-northeast-1	eastus	uksouth	southeastasia	us-west1	europe-west6	asia-northeast1
1MB	Delay (second)	λReplica	0.2	1.0	2.3	1.9	2.5	3.8	1.7	2.7	3.9
		Skyplane	100.0	105.8	112.6	138.1	149.3	160.1	106.8	119.3	125.5
		Δ	**-99.81%**	**-99.04%**	**-97.99%**	**-98.62%**	**-98.34%**	**-97.61%**	**-98.41%**	**-97.73%**	**-96.88%**
	Cost (10^{-4}\$)	λReplica	1.5	1.6	1.8	1.6	1.8	2.1	0.7	1.2	1.5
		Skyplane	516.2	545.5	921.5	1031.2	1111.2	1159.2	518.2	1243.6	1183.3
		Δ	**-99.71%**	**-99.70%**	**-99.80%**	**-99.84%**	**-99.84%**	**-99.81%**	**-99.87%**	**-99.91%**	**-99.87%**
128MB	Delay (second)	λReplica	4.2	6.2	9.5	6.5	11.8	14.4	6.0	6.1	8.1
		Skyplane	102.6	112.5	120.2	140.9	148.2	172.5	113.0	121.3	128.3
		Δ	**-95.90%**	**-94.49%**	**-92.08%**	**-95.38%**	**-92.06%**	**-91.66%**	**-94.65%**	**-95.00%**	**-93.70%**
	Cost (10^{-4}\$)	λReplica	153.1	155.4	156.7	159.4	168.7	180.1	32.5	78.1	117.4
		Skyplane	665.9	980.6	1330.2	1180.4	1260.5	1567.5	543.9	1306.6	1283.4
		Δ	**-77.01%**	**-84.15%**	**-88.22%**	**-86.49%**	**-86.61%**	**-88.51%**	**-94.02%**	**-94.02%**	**-90.85%**
1GB	Delay (second)	λReplica	10.7	7.9	19.3	13.4	18.3	20.1	8.2	14.5	35.4
		Skyplane	106.1	119.4	122.3	149.2	154.6	174.6	115.7	127.7	133.5
		Δ	**-89.89%**	**-93.38%**	**-84.21%**	**-91.00%**	**-88.17%**	**-88.50%**	**-92.90%**	**-88.61%**	**-73.50%**
	Cost (10^{-4}\$)	λReplica	1215.4	1224.3	1243.2	1253.8	1406.7	1466.4	262.9	579.7	889.2
		Skyplane	1978.0	2295.6	2385.9	2233.4	2314.2	2621.3	1242.9	1750.1	1989.4
		Δ	**-38.56%**	**-46.67%**	**-47.89%**	**-43.86%**	**-39.21%**	**-44.06%**	**-78.85%**	**-66.87%**	**-55.30%**



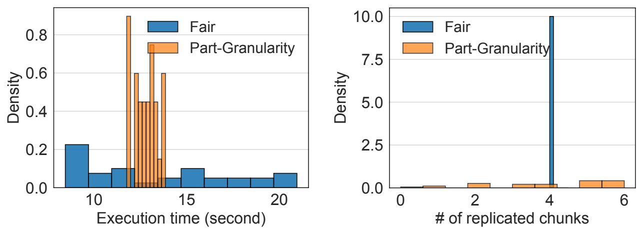
(a) Time **(b)** Cost

Figure 16. Replication time and cost of a 100GB object.

Google Cloud Platform results. Table 3 shows the replication delay and cost of λReplica from GCP us-east1 to the other nine regions. Overall, the replication delay is reduced by 73%-99% compared to Skyplane. Like Azure Functions, Google Cloud Run Functions's link speeds are unstable with parallelism, causing relatively high replication delay on 128MB and 1GB objects. Despite providing 38.5%-99.9% cost savings, using λReplica on GCP is generally less cost-effective because Firestore and Cloud Run are more expensive.

We observe that λReplica assigns a single function instance for 1MB objects, 4-8 for 128MB, and 32-64 for 1GB in most cases. λReplica consistently chooses to run functions on AWS when possible, as its performance model determines that AWS Lambda generally offers the lowest replication latency.

Bulk replication results. Figure 16 shows the replication time and cost of λReplica and Skyplane for replicating a 100GB object. The notification delay is not included in these experiments. Even for replicating a 100GB object, Skyplane still suffers from the non-negligible VM provisioning time.



(a) Execution time **(b)** Replicated chunks

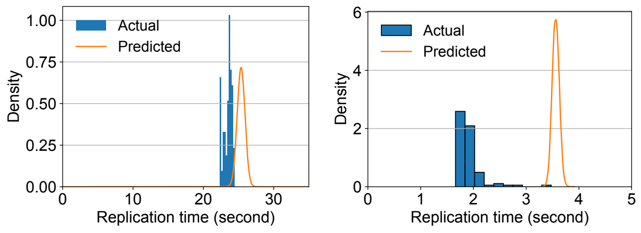
Figure 17. Time and chunk distribution among replicators.

When multiple VMs are used and one starts slowly, the others must wait, increasing replication time and cost. In contrast, λReplica can replicate the 100GB object in a minute, improving the replication time by 76%-91%. λReplica does not reduce the cost significantly compared to Skyplane because the fixed data egress cost dominates for large objects. For 100GB objects, λReplica uses 128-512 function instances to replicate between the reported region pairs.

8.2 Ablation Study

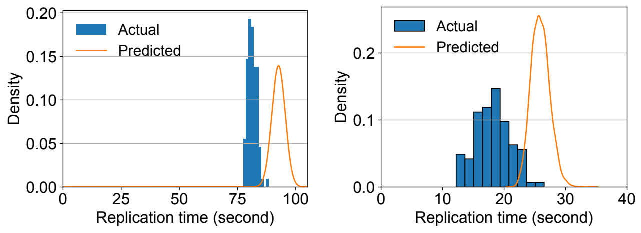
Effectiveness of decentralized part-granularity scheduling. In Figure 17, we show the distributions of execution time and number of replicated chunks among function instances when replicating a 1GB object from Azure eastus to GCP asia-northeast1 with 32 function instances. In contrast to fair task dispatching which distributes parts equally among function instances, decentralized part-granularity scheduling allows the function instances to finish running at approximately the same time, which in turn significantly reduces the end-to-end replication time. Some slow instances never replicate a chunk, and the fastest instances replicate six chunks.

Accuracy of the performance model. In Figure 18-19, we show the actual replication times and the distributions generated by λReplica's performance model for replicating a 1GB object. We configure λReplica to run functions at the source region for 100 times. The links between AWS us-east-1 and



(a) 1 function instance **(b)** 32 function instances

Figure 18. AWS us-east-1 to Azure eastus replication time.



(a) 1 function instance **(b)** 32 function instances

Figure 19. Azure eastus to GCP asia-northeast1 replication time.

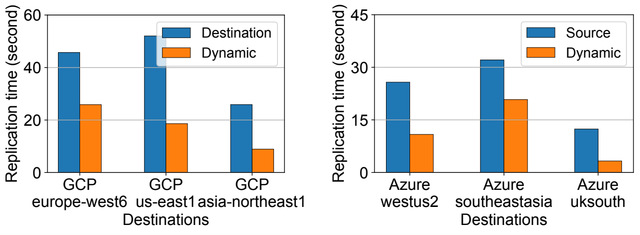
Azure eastus are relatively fast and stable, while those between Azure eastus and GCP asia-northeast1 are slow and fluctuating. We further evaluate our model across six region pairs with 32 function instances, comparing the predicted mean and standard deviation of the replication time against the actual measured results as shown in Table 4. Although our performance model tends to overestimate the replication time in general, it reflects the relative performance of different strategies. Nevertheless, our performance model also captures the variance differences across various cases.

Effectiveness of dynamic region selection. We conduct an experiment to replicate a 128MB object between each pair of regions with a relaxed SLO which lets λReplica to use a single function instance. And we discover there are certain regions which possess very distinct characteristics. In Figure 20, we show the replication time from Azure southeastasia and from GCP europe-west6. Neither statically using the source region nor the destination region can provide optimal performance. Dynamically selecting where to execute the functions can significantly reduce the replication time.

Effectiveness of changelog propagation. In Figure 21, we show the time and cost of COPY operation. Changelog propagation does not improve the replication time significantly because the experiments are conducted between us-east-1 and us-east-2, and the difference between inter-region and intra-region bandwidths is minimal. However, it dramatically reduces the cost by completely avoiding unnecessary cross-cloud/region object replication.

Table 4. Predicted replication time (green) vs. measured (red).

src \ dst	AWS us-east-1	Azure westus2	GCP europe-west6
AWS us-east-1	/	7.01±0.99 5.90±0.97	9.21±1.31 7.08±0.88
Azure westus2	7.22±1.48 5.99±2.23	/	17.87±1.95 12.06±1.68
GCP europe-west6	16.54±3.95 12.47±7.34	72.73±17.32 62.89±13.84	/



(a) From Azure southeastasia **(b)** From GCP europe-west6

Figure 20. Effectiveness of dynamic replication strategy.

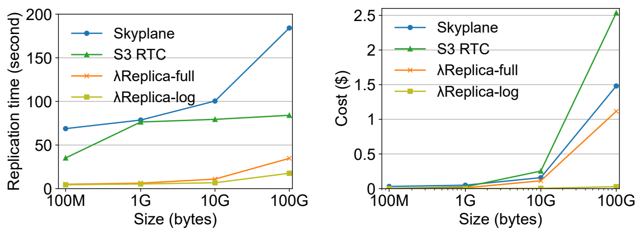
Effectiveness of SLO-bounded batching. In Figure 22, we show the effectiveness of SLO-bounded batching. We replicate a 100 MB object with different update frequencies and set the SLO to 30 seconds. SLO-bounded batching is able to maintain the SLO with very few violations, and the cost is almost constant as the update frequency changes. The cost increases significantly if the mechanism is absent. It stops increasing beyond 50 times per minute because it reaches the maximum replication frequency λReplica can provide.

8.3 Real-World Object Storage Trace

We evaluate λReplica's replication delay using a busy, one-hour segment of the week-long IBM COS trace [33]. After removing non-replicating GET and HEAD operations, the trace contains ~0.99 million PUT and DELETE requests. To compare λReplica with S3 RTC, we replicate objects from AWS us-east-1 to us-east-2, replaying the trace at a high rate using 32 m5.8xlarge EC2 instances, each running 16 parallel clients. Skyplane is omitted as it can not handle this replication rate. As shown in Figure 23, S3 RTC's replication time is typically around 20 seconds, but its p99.99 delay exceeds 30 seconds during traffic bursts. In contrast, λReplica 's elasticity and adaptivity keeps the p99.99 replication time under 10 seconds for the entire period, despite dynamic bursts and varying object sizes. λReplica achieves this by dynamically scaling to hundreds of concurrent function instances to absorb the traffic bursts.

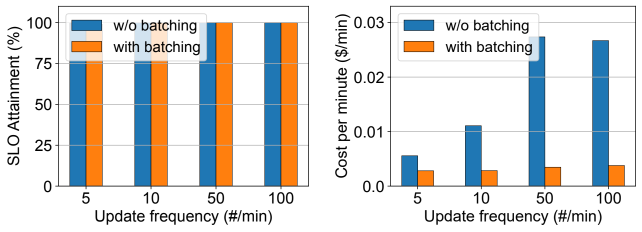
9 Related Work

WAN replication. The key challenges of replicating data over a wide-area network are its limited bandwidth, unstable links, and high cost. Peer-to-peer protocol is a classic approach to improve performance and robustness by allowing a client to download a file from multiple peers [52, 54, 64]. With more IT infrastructures consolidated into datacenters, many existing efforts focus on replication between datacenters [58,



(a) Time **(b)** Cost

Figure 21. Replication time and cost of COPY operation.



(a) SLO Attainment **(b)** Cost

Figure 22. Effectiveness of opportunistic batching.

65, 75, 82, 87]. Although the nodes may directly transfer data to each other in a distributed manner, many inter-DC replication systems [58, 65, 82, 87] possess a centralized view of available resources across the datacenters and utilize them efficiently towards different performance/cost goals.

The most related work includes SPANStore [83] and Skyplane [60, 81]. They can be deployed on demand without native support from cloud providers as λReplica, and allow fast and cost-effective cross-cloud/region storage/replication by taking the data egress pricing model of cloud platforms into account in their schedulers. Unlike existing Inter-DC or cross-cloud/region storage/replication systems, λReplica is highly elastic without assuming a resource limit because serverless computing can span many cloud functions instantaneously. Macaron [70] and SkyStore [67] take an alternative approach that caches objects for acceleration and cost saving, which is orthogonal to λReplica.

There are also commercial solutions that also address multi-cloud data management. Major clouds provide managed services that support object replication across regions [26, 35, 37, 38]. Rubrik [36], an independent solution provider, offers a comprehensive, VM-based cluster management platform which focuses on protecting data via periodic snapshots, whereas λReplica is a fully serverless solution designed for real-time replication of individual objects to reduce latency and cost. Exostellar [27] focuses on cloud cost optimization without detailing their replication methods.

Serverless applications. As a new computing paradigm, serverless computing brings easy deployment, high elasticity, and potential cost savings. Many traditional serverful applications are migrating to serverless platforms, including data analytics[61, 71, 86], video processing [49, 59], and machine

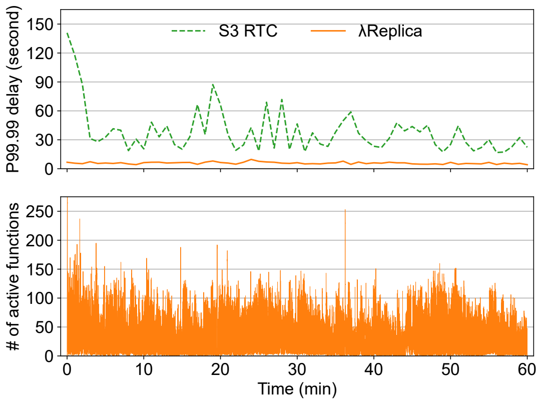


Figure 23. Replication time on the IBM production trace.

learning [50, 77, 79, 85]. λReplica is the first attempt to apply serverless computing to cross-cloud/region data replication.

There is also a large body of prior work that characterizes the performance of serverless workloads [61, 68, 74, 86, 88]. Based on these observations, different approaches including parallel execution [51, 66] and configuration optimization[47, 61, 68, 80, 86, 88, 89] are applied to improve the performance and reduce the cost of serverless applications. This paper presents the first comprehensive characterization of serverless cross-cloud/region data replication, critically analyzing its performance dynamics to inform the design of λReplica.

10 Conclusion

This paper presents λReplica, a serverless platform-independent object replication system across clouds/regions. λReplica proactively decides the replication plan based on a distribution-aware performance model and applies decentralized part-granularity scheduling during runtime to react to performance variability. It avoids unnecessary costs by adopting changelog propagation and SLO-bounded batching. We implement and evaluate a λReplica prototype on three major cloud platforms and show that λReplica reduces the replication delay by 61%-99% with a cost saving of up to three orders of magnitude on common object sizes.

Acknowledgments

We thank our shepherd Ken Birman and the anonymous reviewers for their valuable feedback. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB4500700, the Scientific Research Innovation Capability Support Project for Young Faculty under Grant ZYGXQNJSKYCXNLZCXM-I1, the Fundamental Research Funds for the Central Universities, Peking University, and the National Natural Science Foundation of China under Grant 62172008 and Grant 62325201. Xin Jin is the corresponding author. Junyi Shu, Xiaolong Huang, Gang Huang, Hong Mei, Xuanzhe Liu, and Xin Jin are with the Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education.

References

[1] 2018. Building for Durability in Amazon S3 and Glacier. https://www.youtube.com/watch?v=nLyppihvhpQ. (2018). Retrieved May 4, 2025.

[2] 2021. Tapping the power of unstructured data. https://mitsloan.mit.edu/ideas-made-to-matter/tapping-power-unstructured-data. (2021). Retrieved May 4, 2025.

[3] 2023. Alibaba Cloud suffers second service outage in a month. https://www.reuters.com/technology/alibaba-cloud-suffers-second-service-outage-month-2023-11-28/. (2023). Retrieved May 4, 2025.

[4] 2024. A joint statement from UniSuper CEO Peter Chun, and Google Cloud CEO, Thomas Kurian. https://www.unisuper.com.au/about-us/media-centre/2024/a-joint-statement-from-unisuper-and-google-cloud. (2024). Retrieved May 4, 2025.

[5] 2024. The BBC Preserves 100 Years of History Using Amazon S3. https://aws.amazon.com/solutions/case-studies/bbc-s3-case-study/. (2024). Retrieved Sep 10, 2025.

[6] 2024. Using latency-based routing with Amazon CloudFront for a multi-Region active-active architecture. https://aws.amazon.com/blogs/networking-and-content-delivery/latency-based-routing-leveraging-amazon-cloudfront-for-a-multi-region-active-active-architecture/. (2024). Retrieved Sep 10, 2025.

[7] 2025. About instance autoscaling in Cloud Run services. https://cloud.google.com/run/docs/about-instance-autoscaling. (2025). Retrieved May 4, 2025.

[8] 2025. Amazon DynamoDB. https://aws.amazon.com/dynamodb/. (2025). Retrieved May 4, 2025.

[9] 2025. Amazon DynamoDB Lock Client. https://github.com/awslabs/amazon-dynamodb-lock-client. (2025). Retrieved May 4, 2025.

[10] 2025. Amazon S3. https://aws.amazon.com/s3/. (2025). Retrieved May 4, 2025.

[11] 2025. Amazon S3 API Reference - Object. https://docs.aws.amazon.com/AmazonS3/latest/API/API_Object.html. (2025). Retrieved May 4, 2025.

[12] 2025. AWS Health Dashboard. https://health.aws.amazon.com/health/status. (2025). Retrieved May 4, 2025.

[13] 2025. AWS Lambda. https://aws.amazon.com/lambda/. (2025). Retrieved May 4, 2025.

[14] 2025. AWS Step Functions - Wait workflow state. https://docs.aws.amazon.com/step-functions/latest/dg/state-wait.html. (2025). Retrieved May 4, 2025.

[15] 2025. Azure Blob Storage. https://azure.microsoft.com/en-us/products/storage/blobs. (2025). Retrieved May 4, 2025.

[16] 2025. Azure Cosmos DB. https://azure.microsoft.com/en-us/products/cosmos-db. (2025). Retrieved May 4, 2025.

[17] 2025. Azure Event Grid trigger for Azure Functions. https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-event-grid-trigger#event-schema. (2025). Retrieved May 4, 2025.

[18] 2025. Azure Functions. https://azure.microsoft.com/en-us/products/functions. (2025). Retrieved May 4, 2025.

[19] 2025. Azure Functions error handling and retries. https://learn.microsoft.com/en-us/azure/azure-functions/functions-bindings-error-pages. (2025). Retrieved May 4, 2025.

[20] 2025. Azure Functions hosting options. https://learn.microsoft.com/en-us/azure/azure-functions/functions-scale#service-limits. (2025). Retrieved May 4, 2025.

[21] 2025. Azure Status History. https://azure.status.microsoft/en-gb/status/history/. (2025). Retrieved May 4, 2025.

[22] 2025. CloudEvents format - HTTP protocol binding. https://cloud.google.com/eventarc/docs/cloudevents. (2025). Retrieved May 4, 2025.

[23] 2025. Configure Lambda function timeout. https://docs.aws.amazon.com/lambda/latest/dg/configuration-timeout.html. (2025). Retrieved Sep 10, 2025.

[24] 2025. Enable event-driven function retries. https://cloud.google.com/functions/docs/bestpractices/retries. (2025). Retrieved May 4, 2025.

[25] 2025. Event message structure. https://docs.aws.amazon.com/AmazonS3/latest/userguide/notification-content-structure.html. (2025). Retrieved May 4, 2025.

[26] 2025. Exoscale Object storage - Replication. https://community.exoscale.com/product/storage/object-storage/how-to/replication/. (2025). Retrieved Sep 10, 2025.

[27] 2025. Exostellar - Self-managed, Al Infrastructure Orchestration. https://www.exostellar.ai/. (2025). Retrieved Sep 10, 2025.

[28] 2025. Google Cloud Run Functions. https://cloud.google.com/functions?hl=en. (2025). Retrieved May 4, 2025.

[29] 2025. Google Cloud Service Health. https://status.cloud.google.com/summary. (2025). Retrieved May 4, 2025.

[30] 2025. Google Cloud Storage. https://cloud.google.com/storage. (2025). Retrieved May 4, 2025.

[31] 2025. Google Firestore. https://cloud.google.com/firestore. (2025). Retrieved May 4, 2025.

[32] 2025. Google Workflows - Wait using polling. https://cloud.google.com/workflows/docs/sleeping. (2025). Retrieved May 4, 2025.

[33] 2025. IBM Object Store traces. http://iotta.snia.org/traces/key-value/36305. (2025). Retrieved May 4, 2025.

[34] 2025. Meeting compliance requirements using S3 Replication Time Control (S3 RTC). https://docs.aws.amazon.com/AmazonS3/latest/userguide/replication-time-control.html. (2025). Retrieved May 4, 2025.

[35] 2025. Object replication for block blobs - Azure. https://learn.microsoft.com/en-us/azure/storage/blobs/object-replication-overview. (2025). Retrieved May 4, 2025.

[36] 2025. Protecting AWS with Rubrik Security Cloud. https://www.youtube.com/watch?v=1GNQPF5jMNE. (2025). Retrieved Sep 10, 2025.

[37] 2025. Redundancy across regions - Google. https://cloud.google.com/storage/docs/availability-durability#cross-region-redundancy. (2025). Retrieved May 4, 2025.

[38] 2025. Replicating objects overview - AWS. https://docs.aws.amazon.com/AmazonS3/latest/userguide/replication.html. (2025). Retrieved May 4, 2025.

[39] 2025. RocksDB-Cloud. https://github.com/rockset/rocksdb-cloud. (2025). Retrieved May 4, 2025.

[40] 2025. skyplane-0.3.2. https://github.com/skyplane-project/skyplane/tree/0.3.2. (2025). Retrieved May 4, 2025.

[41] 2025. Snowflake - Expand to More Regions and Clouds with Zero Additional Egress Cost. https://www.snowflake.com/en/blog/new-regions-egress-cost-optimizer/. (2025). Retrieved May 4, 2025.

[42] 2025. Timers in Durable Functions (Azure Functions). https://learn.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-timers. (2025). Retrieved May 4, 2025.

[43] 2025. Understanding Lambda function scaling. https://docs.aws.amazon.com/lambda/latest/dg/lambda-concurrency.html. (2025). Retrieved May 4, 2025.

[44] 2025. Understanding retry behavior in Lambda. https://docs.aws.amazon.com/lambda/latest/dg/invocation-retries.html. (2025). Retrieved May 4, 2025.

[45] 2026. Simplifying data management with Amazon S3 & Salesforce Data Cloud | Amazon Web Services. https://www.youtube.com/watch?v=tKEhr2US5gg. (2026). Retrieved Sep 10, 2025.

[46] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. 2020. Firecracker: Lightweight Virtualization for Serverless Applications .

In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 419–434. https://www.usenix.org/conference/nsdi20/presentation/agache

[47] Nabeel Akhtar, Ali Raza, Vatche Ishakian, and Ibrahim Matta. 2020. COSE: Configuring Serverless Functions using Statistical Learning. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. 129–138. https://doi.org/10.1109/INFOCOM41043.2020.9155363

[48] Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, and Volker Hilt. 2018. SAND: Towards High-Performance Serverless Computing. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 923–935. https://www.usenix.org/conference/atc18/presentation/akkus

[49] Lixiang Ao, Liz Izhikevich, Geoffrey M. Voelker, and George Porter. 2018. Sprocket: A Serverless Video Processing Framework. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '18)*. Association for Computing Machinery, New York, NY, USA, 263–274. https://doi.org/10.1145/3267809.3267815

[50] Joao Carreira, Pedro Fonseca, Alexey Tumanov, Andrew Zhang, and Randy Katz. 2019. Cirrus: a Serverless Framework for End-to-end ML Workflows. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '19)*. Association for Computing Machinery, New York, NY, USA, 13–24. https://doi.org/10.1145/3357223.3362711

[51] Benjamin Carver, Jingyuan Zhang, Ao Wang, Ali Anwar, Panruo Wu, and Yue Cheng. 2020. Wukong: a scalable and locality-enhanced framework for serverless parallel computing. In *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC '20)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3419111.3421286

[52] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. 2003. SplitStream: high-bandwidth multicast in cooperative environments. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP '03)*. Association for Computing Machinery, New York, NY, USA, 298–313. https://doi.org/10.1145/945445.945474

[53] Sarah Chasins, Alvin Cheung, Natacha Crooks, Ali Ghodsi, Ken Goldberg, Joseph E. Gonzalez, Joseph M. Hellerstein, Michael I. Jordan, Anthony D. Joseph, Michael W. Mahoney, Aditya Parameswaran, David Patterson, Raluca Ada Popa, Koushik Sen, Scott Shenker, Dawn Song, and Ion Stoica. 2022. The Sky Above The Clouds. *arXiv preprint arXiv:2205.07147* (2022).

[54] Bram Cohen. 2003. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer systems*, Vol. 6. 68–72.

[55] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. 2016. The Snowflake Elastic Data Warehouse. In *Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 215–226. https://doi.org/10.1145/2882903.2903741

[56] Dong Du, Tianyi Yu, Yubin Xia, Binyu Zang, Guanglu Yan, Chenggang Qin, Qixuan Wu, and Haibo Chen. 2020. Catalyzer: Sub-millisecond Startup for Serverless Computing with Initialization-less Booting. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20)*. Association for Computing Machinery, New York, NY, USA, 467–481. https://doi.org/10.1145/3373376.3378512

[57] Ohad Eytan, Danny Harnik, Effi Ofer, Roy Friedman, and Ronen Kat. 2020. It's Time to Revisit LRU vs. FIFO. In *12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 20)*. USENIX Association. https://www.usenix.org/conference/hotstorage20/presentation/eytan

[58] Jason Flinn, Xianzheng Dou, Arushi Aggarwal, Alex Boyko, Francois Richard, Eric Sun, Wendy Tobagus, Nick Wolchko, and Fang Zhou. 2022. Owl: Scale and Flexibility in Distribution of Hot Content. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 1–15. https://www.usenix.org/conference/osdi22/presentation/flinn

[59] Sadjad Fouladi, Riad S. Wahby, Brennan Shacklett, Karthikeyan Vasuki Balasubramaniam, William Zeng, Rahul Bhalerao, Anirudh Sivaraman, George Porter, and Keith Winstein. 2017. Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 363–376. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/fouladi

[60] Paras Jain, Sam Kumar, Sarah Wooders, Shishir G. Patil, Joseph E. Gonzalez, and Ion Stoica. 2023. Skyplane: Optimizing Transfer Cost and Throughput Using Cloud-Aware Overlays. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 1375–1389. https://www.usenix.org/conference/nsdi23/presentation/jain

[61] Chao Jin, Zili Zhang, Xingyu Xiang, Songyun Zou, Gang Huang, Xuanzhe Liu, and Xin Jin. 2023. Ditto: Efficient Serverless Analytics with Elastic Parallelism. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 406–419. https://doi.org/10.1145/3603269.3604816

[62] Eric Jonas, Qifan Pu, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. 2017. Occupy the cloud: distributed computing for the 99%. In *Proceedings of the 2017 Symposium on Cloud Computing (SoCC '17)*. Association for Computing Machinery, New York, NY, USA, 445–451. https://doi.org/10.1145/3127479.3128601

[63] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, Joseph E. Gonzalez, Raluca Ada Popa, Ion Stoica, and David A. Patterson. 2019. *Cloud Programming Simplified: A Berkeley View on Serverless Computing*. Technical Report. EECS Department, University of California, Berkeley.

[64] Dejan Kostić, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat. 2003. Bullet: high bandwidth data dissemination using an overlay mesh. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (SOSP '03)*. Association for Computing Machinery, New York, NY, USA, 282–297. https://doi.org/10.1145/945445.945473

[65] Nikolaos Laoutaris, Michael Sirivianos, Xiaoyuan Yang, and Pablo Rodriguez. 2011. Inter-datacenter bulk transfers with netstitcher. In *Proceedings of the ACM SIGCOMM 2011 Conference (SIGCOMM '11)*. Association for Computing Machinery, New York, NY, USA, 74–85. https://doi.org/10.1145/2018436.2018446

[66] David H. Liu, Amit Levy, Shadi Noghabi, and Sebastian Burckhardt. 2023. Doing More with Less: Orchestrating Serverless Applications without an Orchestrator. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 1505–1519. https://www.usenix.org/conference/nsdi23/presentation/liu-david

[67] Shu Liu, Xiangxi Mo, Moshik Hershcovitch, Henric Zhang, Audrey Cheng, Guy Girmonsky, Gil Vernik, Michael Factor, Tiemo Bang, Soujanya Ponnapalli, Natacha Crooks, Joseph E. Gonzalez, Danny Harnik, and Ion Stoica. 2025. SkyStore: Cost-Optimized Object Storage Across Regions and Clouds. *Proc. VLDB Endow.* 18, 7 (August 2025), 2084–2096. https://doi.org/10.14778/3734839.3734846

[68] Ashraf Mahgoub, Edgardo Barsallo Yi, Karthick Shankar, Sameh Elnikety, Somali Chaterji, and Saurabh Bagchi. 2022. ORION and the Three Rights: Sizing, Bundling, and Prewarming for Serverless DAGs.

In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 303–320. https://www.usenix.org/conference/osdi22/presentation/mahgoub

[69] Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. 2018. SOCK: Rapid Task Provisioning with Serverless-Optimized Containers. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, 57–70. https://www.usenix.org/conference/atc18/presentation/oakes

[70] Hojin Park, Ziyue Qiu, Gregory R. Ganger, and George Amvrosiadis. 2024. Reducing Cross-Cloud/Region Costs with the Auto-Configuring MACARON Cache. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles (SOSP '24)*. Association for Computing Machinery, New York, NY, USA, 347–368. https://doi.org/10.1145/3694715.3695972

[71] Qifan Pu, Shivaram Venkataraman, and Ion Stoica. 2019. Shuffling, Fast and Slow: Scalable Analytics on Serverless Infrastructure. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 193–206. https://www.usenix.org/conference/nsdi19/presentation/pu

[72] Rohan Basu Roy, Tirthak Patel, and Devesh Tiwari. 2022. IceBreaker: warming serverless functions better with heterogeneity. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22)*. Association for Computing Machinery, New York, NY, USA, 753–767. https://doi.org/10.1145/3503222.3507750

[73] Johann Schleier-Smith, Vikram Sreekanti, Anurag Khandelwal, Joao Carreira, Neeraja J. Yadwadkar, Raluca Ada Popa, Joseph E. Gonzalez, Ion Stoica, and David A. Patterson. 2021. What Serverless Computing is and Should Become: The Next Phase of Cloud Computing. *Commun. ACM* 64, 5 (April 2021), 76–84. https://doi.org/10.1145/3406011

[74] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 205–218. https://www.usenix.org/conference/atc20/presentation/shahrad

[75] Chijun Sima, Yao Fu, Man-Kit Sit, Liyi Guo, Xuri Gong, Feng Lin, Junyu Wu, Yongsheng Li, Haidong Rong, Pierre-Louis Aublin, and Luo Mai. 2022. Ekko: A Large-Scale Deep Learning Recommender System with Low-Latency Model Update. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 821–839. https://www.usenix.org/conference/osdi22/presentation/sima

[76] Foteini Strati, Zhendong Zhang, George Manos, Ixeia Sánchez Périz, Qinghao Hu, Tiancheng Chen, Berk Buzcu, Song Han, Pamela Delgado, and Ana Klimovic. 2025. Sailor: Automating Distributed Training over Dynamic, Heterogeneous, and Geo-distributed Clusters. *arXiv preprint arXiv:2504.17096* (2025).

[77] John Thorpe, Yifan Qiao, Jonathan Eyolfson, Shen Teng, Guanzhou Hu, Zhihao Jia, Jinliang Wei, Keval Vora, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. 2021. Dorylus: Affordable, Scalable, and Accurate GNN Training with Distributed CPU Servers and Serverless Threads. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, 495–514. https://www.usenix.org/conference/osdi21/presentation/thorpe

[78] Midhul Vuppalapati, Justin Miron, Rachit Agarwal, Dan Truong, Ashish Motivala, and Thierry Cruanes. 2020. Building An Elastic Query Engine on Disaggregated Storage. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, Santa Clara, CA, 449–462. https://www.usenix.org/conference/nsdi20/presentation/vuppalapati

[79] Hao Wang, Di Niu, and Baochun Li. 2019. Distributed Machine Learning with a Serverless Architecture. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 1288–1296. https://doi.org/10.1109/INFOCOM.2019.8737391

[80] Zhaojie Wen, Yishuo Wang, and Fangming Liu. 2022. StepConf: SLO-Aware Dynamic Resource Configuration for Serverless Function Workflows. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*. 1868–1877. https://doi.org/10.1109/INFOCOM48880.2022.9796962

[81] Sarah Wooders, Shu Liu, Paras Jain, Xiangxi Mo, Joseph E. Gonzalez, Vincent Liu, and Ion Stoica. 2024. Cloudcast: High-Throughput, Cost-Aware Overlay Multicast in the Cloud. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 281–296. https://www.usenix.org/conference/nsdi24/presentation/wooders

[82] Bingyang Wu, Kun Qian, Bo Li, Yunfei Ma, Qi Zhang, Zhigang Jiang, Jiayu Zhao, Dennis Cai, Ennan Zhai, Xuanzhe Liu, and Xin Jin. 2023. XRON: A Hybrid Elastic Cloud Overlay Network for Video Conferencing at Planetary Scale. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 696–709. https://doi.org/10.1145/3603269.3604845

[83] Zhe Wu, Michael Butkiewicz, Dorian Perkins, Ethan Katz-Bassett, and Harsha V. Madhyastha. 2013. SPANStore: cost-effective geo-replicated storage spanning multiple cloud services. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13)*. Association for Computing Machinery, New York, NY, USA, 292–308. https://doi.org/10.1145/2517349.2522730

[84] Zongheng Yang, Zhanghao Wu, Michael Luo, Wei-Lin Chiang, Romil Bhardwaj, Woosuk Kwon, Siyuan Zhuang, Frank Sifei Luan, Gautam Mittal, Scott Shenker, and Ion Stoica. 2023. SkyPilot: An Intercloud Broker for Sky Computing. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 437–455. https://www.usenix.org/conference/nsdi23/presentation/yang-zongheng

[85] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. 2019. MArk: Exploiting Cloud Services for Cost-Effective, SLO-Aware Machine Learning Inference Serving. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, Renton, WA, 1049–1062. https://www.usenix.org/conference/atc19/presentation/zhang-chengliang

[86] Hong Zhang, Yupeng Tang, Anurag Khandelwal, Jingrong Chen, and Ion Stoica. 2021. Caerus: NIMBLE Task Scheduling for Serverless Analytics. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, 653–669. https://www.usenix.org/conference/nsdi21/presentation/zhang-hong

[87] Yuchao Zhang, Junchen Jiang, Ke Xu, Xiaohui Nie, Martin J. Reed, Haiyang Wang, Guang Yao, Miao Zhang, and Kai Chen. 2018. BDS: a centralized near-optimal overlay network for inter-datacenter data replication. In *Proceedings of the Thirteenth EuroSys Conference (EuroSys '18)*. Association for Computing Machinery, New York, NY, USA, Article 10, 14 pages. https://doi.org/10.1145/3190508.3190519

[88] Zili Zhang, Chao Jin, and Xin Jin. 2024. Jolteon: Unleashing the Promise of Serverless for Serverless Workflows. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 167–183. https://www.usenix.org/conference/nsdi24/presentation/zhang-zili-jolteon

[89] Zhuangzhuang Zhou, Yanqi Zhang, and Christina Delimitrou. 2022. AQUATOPE: QoS-and-Uncertainty-Aware Resource Management for Multi-stage Serverless Workflows. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3567955.3567960