Cloudscape: A Study of Storage Services in Modern Cloud Architectures

Sambhav Satija, Chenhao Ye, Ranjitha Kosgi, Aditya Jain, Romit Kankaria, Yiwei Chen, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, *University of Wisconsin–Madison;* Kiran Srinivasan, *NetApp*

https://www.usenix.org/conference/fast25/presentation/satija

This paper is included in the Proceedings of the 23rd USENIX Conference on File and Storage Technologies.

February 25–27, 2025 • Santa Clara, CA, USA

ISBN 978-1-939133-45-8

Open access to the Proceedings of the 23rd USENIX Conference on File and Storage Technologies is sponsored by

Cloudscape: A Study of Storage Services in Modern Cloud Architectures

Sambhav Satija Chenhao Ye Ranjitha Kosgi Aditya Jain Romit Kankaria Yiwei Chen
Andrea C. Arpaci-Dusseau Remzi H. Arpaci-Dusseau Kiran Srinivasan[†]

University of Wisconsin – Madison [†]*NetApp*

Abstract

We present *Cloudscape*, a dataset of nearly 400 cloud architectures deployed on AWS. We perform an in-depth analysis of the usage of storage services in cloud systems. Our findings include: S3 is the most prevalent storage service (68%), while file system services are rare (4%); heterogeneity is common in the storage layer; storage services primarily interface with Lambda and EC2, while also serving as the foundation for more specialized ML and analytics services. Our findings provide a concrete understanding of how storage services are deployed in real-world cloud architectures, and our analysis of the popularity of different services grounds existing research.

1 Introduction

The cloud has become a dominant platform for service development and deployment. In 2023, cloud computing infrastructure investment represented roughly 60% of all IT infrastructure spend worldwide; cost efficiency was cited as the main driver behind this expansion [61]. Further growth is expected, with some predicting a compound annual growth rate of 16% over the remainder of this decade [37].

Despite the large and increasing importance of cloud, little is known about how cloud-based systems are generally constructed. A key design decision is choosing the appropriate storage services that fit the architecture. While there are some well-known examples of cloud-based data systems (e.g., the Snowflake data warehouse [29]), there is no comprehensive study of the many other systems that are being built and deployed atop today's cloud where storage is one part, and not necessarily the central part, of the system. This deficiency is problematic both for practitioners, who lack a unified resource to learn from the collective experiences of the community, and for storage researchers, who do not know where to focus cloud-based research efforts in order to maximize their impact.

In this work, we present the first large-scale study of cloud-based system architectures. Our dataset consists of 396 systems, each built atop Amazon Web Services (AWS). Each architecture is described in detail in video and collected by Amazon (presumably) to showcase the breadth and scope of AWS-based systems; the videos are, to our knowledge, the largest collective data source on cloud-based systems. However, the videos in raw form do not readily facilitate analysis and exploration.

To remedy this limitation, we apply a rigorous process using a range of best practices [31, 73, 88] to transform the videos into a rich and detailed quantitative dataset, which we refer to as *Cloudscape*; we then can query *Cloudscape* to answer critical questions about the systems under study. For each architecture, *Cloudscape* contains a directed graph of the AWS services used to construct it, thus including rich information about how compute, storage, and other AWS services interact. Most architectures consist of a small number of interacting workflows (i.e., subsets of the graph) that cooperate to achieve the goals of the given system. We capture flow directions to understand the reads and writes serviced by storage systems. In addition, we augment our dataset with qualitative annotations, with a focus on the type of data stored inside storage systems, enabling us to ask and answer questions not readily discovered via quantitative analysis. While our dataset has inherent biases (as discussed in Section 2), we believe the data is valuable and is broadly representative of modern cloud-based systems.

Our analysis focuses on the impact of cloud-based architectures on storage services. We find that even though the functional goal of a cloud system influences the services chosen, S3 is unsurprisingly the dominant force, used in 68% of the systems we study. Structured data storage is also popular, with 40% of systems using a key-value store (e.g., DynamoDB [35]), and 30% using a SQL database (e.g., RDS [19]). However, most systems utilize a combination of storage services (about half use two or more), i.e., storage is heterogeneous. We further find that for the architectures in this dataset, distributed file systems (e.g., Elastic File System [12]) are rarely used; distributed file systems, in the context of web-based services, are not particularly important.

We find that S3 is critical for specialized services. ML services [17, 20] store their inputs and outputs in S3, making it their default interoperability layer. Analytics engines [13, 18] store both pre-processed and post-processed data in S3, often leading to duplication between S3 and other storage services.

We find that while many different cloud services interact with storage services, they predominantly serve the needs of Lambda, EC2, and other compute services. S3 interacts

meaningfully with services by acting as a bridge between workflows and is not a mere dumping ground. We also find that services host a rich variety of semantically different data, ranging from media to configuration files in S3, and system metadata to pointers to S3 data in DynamoDB.

Finally, we find that serverless compute offerings (*e.g.* Lambda) are widely utilized, and occur in roughly 60% of the systems we study; furthermore, nearly 20% of systems rely solely on serverless as a compute platform.

We believe our findings are useful to the systems research community for the following reasons. First, while key-value stores and databases are widely used, object storage via S3 is dominant; perhaps more research into object stores would be useful, to better understand the variety of use cases and how to better optimize them [3, 86]. Second, about half of the architectures use two or more data services (e.g., S3 and DynamoDB); understanding the new issues (e.g., consistency) that arise due to storage of data in multiple different services would be valuable [36, 81]. Third, new specialized ML and analytics services are decoupled from the object stores; it is important to co-design them to minimize data movement between services [89, 93]. Fourth, distributed file systems are not particularly important in this context. Finally, the prevalence of serverless compute within hundreds of architectures justifies the focus of researchers on serverless platforms [26, 30, 38, 47, 52, 67, 91].

The rest of the paper is presented as follows: In §2, we describe our methodology for converting raw videos into *Cloudscape*, and in §3 we provide a high-level overview of the resulting dataset. We present our analysis in §4, in which we ask three research questions: a) how are storage services used across architectures, and does an architecture's functionality goal affect the choice of storage services? b) how are storage services used by other services, and what data is stored in them? And c) how do specialized services use the storage layer? Finally, in §5, we discuss the next steps for the research community.

2 Dataset

This section describes the data we use and our methodology for converting video content into a structured and queryable dataset, called *Cloudscape*. We also describe how each architecture is represented in *Cloudscape*. The dataset and analysis scripts are available at `https://github.com/WiscADSL/Cloudscape`. The artifact is described in Appendix A.

2.1 Data Source

Cloudscape consists of 396 architectures that were obtained from a YouTube video playlist [21] maintained by AWS. Each video describes the architecture from a different company, focusing on one team within the company rather than the entire system. They are presented as interviews in which an AWS representative asks questions that allow the invited representative to explain and elaborate on the architecture

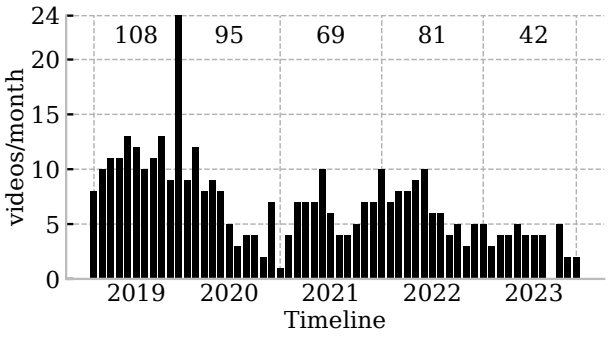


Figure 1: **Distribution of video release date:** Each bar denotes the number of videos released that month. The number at the top shows the number of videos released that year.

step by step. We build *Cloudscape* using videos published before 2024 and include most of them, barring a few that do not describe an architecture.

These architectures have rich and diverse contexts. They are described by businesses ranging from Fortune 500 companies to small startups. The timeline in Figure 1 shows that the videos span 5 years, from March 2019 to December 2023. While most videos were in English, they span 11 languages.

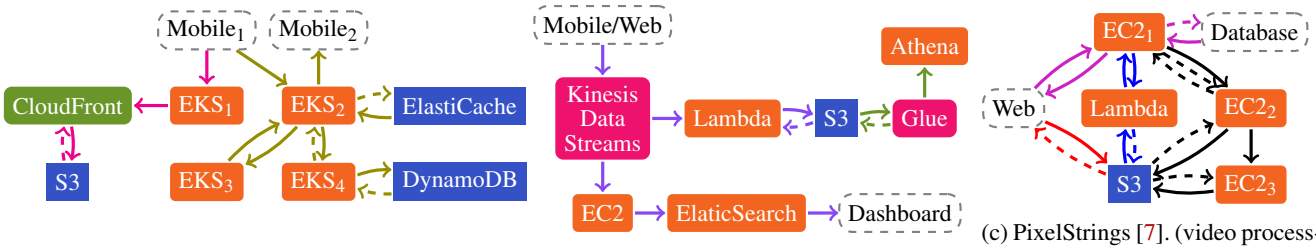
Although these videos are fairly short, with an average runtime of ~6 minutes, the interview format and visual aids used to describe the architecture result in information-dense videos. A typical video features a developer describing the AWS services used and how they interact. This is presented as a collection of images of AWS services on a whiteboard, and the developer draws connections between them while explaining workflows. A workflow describes a collection of services and interactions that fulfill some notion of work subjective to each video. A typical video consists of multiple workflows. Additionally, the developer might discuss how they use or configure a particular service. Such contextual information does not exist in all videos, and when it is available, developers might discuss from a broad range of topics.

We extract data from such videos into a structured dataset reliably. In the next section, we describe our methodology for capturing the services used, their interaction patterns, and workflows.

2.2 Structural Annotation

Each video represents one architecture in *Cloudscape*. An architecture is encoded as a graph and is a collection of nodes, workflows, and edges. Figure 2 presents three sample architectures from *Cloudscape*. We use them as running examples throughout this section.

Nodes The *nodes* in the graph reflect the services used in the architecture. Most nodes represent AWS services but some refer to users or the company's internal platforms. To faithfully encode the video, we keep duplicate nodes if



(a) Snap [10]. (message delivery) (b) Haptik [6]. (data lake) (c) PixelStrings [7]. (video processing)

Figure 2: **Typical architectures in *Cloudscape*.** Each node is a service used in the architecture. An edge encodes an interaction between services. Solid edges are *data* edges, and dotted edges are *meta* edges. Edges with the same color belong to one flow. Colored nodes denote AWS services of different capabilities: compute, storage, network and integration.

they appear multiple times in the video. We rarely add to or remove from the nodes described in the video. In Figure 2a, the architecture of Snap uses EKS, Elasticache, DynamoDB, S3 and CloudFront. It also includes users using the Snapchat mobile app. We categorize vaguely defined services, *e.g.* *Dashboard* in Figure 2b, as *ThirdParty* nodes.

Edges An edge conveys an interaction between two nodes. We classify edges as either *data* edges (solid lines) or *meta* edges (dotted lines). *Data* edges indicate movement of user data, *e.g.* in Figure 2c the data edge from S3 to Lambda indicates that Lambda reads from S3. *Meta* edges do not carry data but indicate a request trigger or an acknowledgment response.

Workflows We define a workflow as a synchronous sequence of invocations and data movements resulting from a trigger. Workflows are encoded as a sequence of edges. Most videos describe an architecture in terms of loosely coupled workflows, *e.g.* in Figure 2b, the first (purple) flow captures events from users being ingested into S3 and ElasticSearch using Kinesis Data Streams, Lambda and EC2. The second (green) flow captures how Athena uses Glue to read the schema of data stored in S3 when it needs to perform a query. Each workflow also imparts a sequence number to its edges, which enables the ordering of interactions.

We mostly capture these workflows as they are described in the videos. When the verbal description does not match what they drew, we consider the verbal description as the ground truth. In some cases, we introduce new workflows to keep them synchronous. For example, when architectures use a message bus like SQS to enqueue jobs, we create a new workflow when the message gets dequeued because SQS workflows are asynchronous. In the case of streaming workflows (Figure 2b), we show a single sequence of edges as data passes from one service to another. Workflows capture the full call graph of requests and responses (Figure 2a and 2c).

Annotation process We followed an iterative process to determine the right encoding that sufficiently captured most architectures. Three team members initially analyzed a subset of the dataset (50 architectures). We discussed and reiterated the annotation process [73, 88] until we finalized the definitions and methodology for capturing nodes, workflows, and edges. We then trained three more team members for the rest of the dataset, and their initial dataset additions were verified and discussed to ensure consistency and further solidify the annotations.

No. of Arch.	Graph + Notes	Quant.	Qual.
396	✔		
340	✔	✔	
176	✔	✔	✔

Table 1: ***Cloudscape* details.** This table provides a breakdown of the information captured in the architecture graphs. It outlines the number of architectures used for quantitative analysis (§3, §4) and qualitative (§4.6) analysis.

Our methodology works well for most of the dataset (340 out of 396 videos). The rest of the videos are vague regarding how services interact (*e.g.* they focus on infrastructure policies instead). We only consider these 340 architectures (85.9% of the dataset) in our analysis, and the percentages in the rest of the paper use 340 as the total number of architectures. Table 1 shows this breakdown.

Notes and qualitative annotation As described in §2.1, videos often provide contextual information about their architecture. *Cloudscape* captures this as textual notes for all the architectures. Additionally, a video would sometimes provide additional context that allowed us to capture the type of data stored in storage services. We were able to capture this for 176 videos, which we use for qualitatively understanding the breadth of data stored across services in §4.6. Table 1 summarizes this.

2.3 Limitations

Our dataset is derived from a collection of videos published by AWS. We were unable to identify a data source of comparable size and complexity for competing cloud providers. As a result, *Cloudscape* only includes architectures that utilize AWS cloud services. Nonetheless, we believe that the methodology and findings are generalizable to other providers. There is also potential for selection bias as the AWS team invites customers to

describe their architecture. However, *Cloudscape* captures architectures from 378 businesses, ranging from conglomerates to startups. Our methodology also benefits from the explanatory nature of the videos which describe entire workflows. This lets us capture all services in use, instead of those in focus. Our analysis should therefore be a reasonable proxy for real trends.

On the use of videos and manual annotation We considered other data repositories, but none described as many real-world deployments, or they missed the deployment nuances and instead presented architecture diagrams without workflows and configuration information. We tried using multimodal generative models with few shot examples and prompt tuning to help with encoding, but they had low accuracy. Our goal is to provide a dataset, built by applying best practices [31, 73, 88], that can be used to better understand deployments.

Summary *Cloudscape* consists of architectures captured from over 40 hours of video footage, taking more than 18 human-months of effort. It encodes the services used and their interactions as graphs, and captures contextual information as textual notes, facilitating analysis of deployed cloud architectures.

3 Overview of the Dataset

In this section, we provide an overview of the architectures captured in *Cloudscape*: which, and how many, services are used in these architectures, and what are these architectures built to do?

3.1 Composition of Architectures

To facilitate analysis, we first construct a service taxonomy and then examine which services are used to construct these architectures.

Services used across *Cloudscape* Across all architectures, we identify 134 unique AWS services. To understand architectures conceptually, we distill these services into their core capabilities and categorize them as:

- **Compute:** Services that enable general purpose compute, or services that do predefined compute (*e.g.* Textract, a transcription service). We classify 28 services as compute, the most popular being Lambda, EC2, and EKS.
- **Storage**: Services that store data for future retrieval. We also treat in-memory caches, *e.g.* ElastiCache, as storage services. We classify 14 services as storage and find S3, DynamoDB, and RDS to be most frequently used.
- **Network:** Services that route traffic from external users to other AWS services. We classify 20 services as network, the most popular being APIGateway, CloudFront, and ALB.
- **Integration:** Services that act as message-passing buses or generally enhance the interoperability between different AWS services. We classify 15 services as integration, the most popular being SQS, SNS, and Glue.
- **AWS Control Plane:** These 27 services deal with infrastructure administration, monitoring, security, identity, or compliance. The most common services were CloudWatch, Cognito, and CloudFormation.
- **Others:** The remaining 30 services cover various use cases. Some services are platforms for specific business cases (*e.g.*, Connect), some are developer tools (*e.g.*, XRay), and some are code/media repositories (*e.g.*, Model Registry), among other use cases.

Figure 3a shows the popularity of different capabilities of services across *Cloudscape*. Storage and compute services appear far more frequently than the rest of the service types, together constituting more than half of all the nodes. This figure also includes *User* and *ThirdParty* nodes; vaguely described nodes (*ThirdParty*) are a small part of *Cloudscape*.

Figure 3b presents the ten most popular AWS services. This reveals that multiple storage and compute services are popular enough to appear in the top ten services. In addition, we find that APIGateway, a network service, and SQS and SNS, both integration services, are present in more than 10% of the architectures.

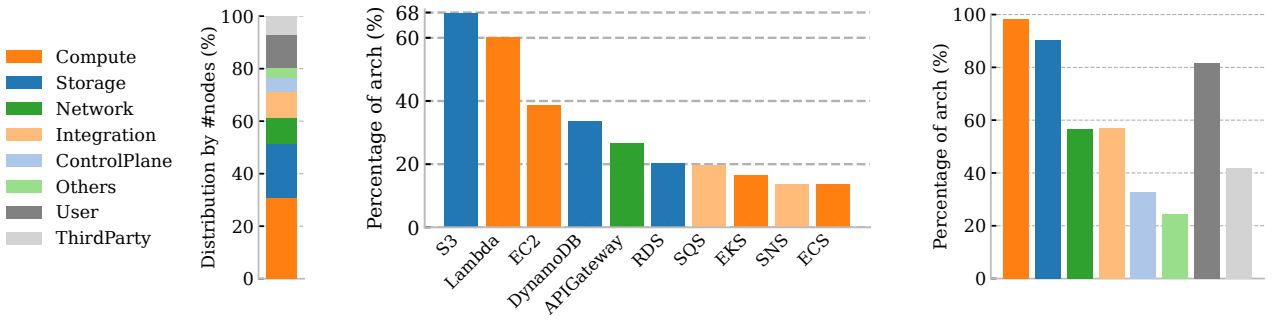
Services used in each architecture We now analyze the capabilities of services used in each architecture. Figure 3c shows that compute and storage services are used in almost all architectures. Further, almost ~60% of the architectures use network and integration services. More than 20% of architectures use at least one service categorized as *ControlPlane* or *Others* even though they collectively constitute less than 10% of the dataset (Figure 3a).

Adoption of S3 and Lambda S3 and Lambda are the most common AWS services, both being utilized in over 60% of architectures. While this wide adoption can be attributed to their flexibility, they are also functionally sufficient for many architectures: 22% of architectures employ S3 as their sole storage service, and 19% of architectures use only serverless services (Lambda, StepFunctions and LambdaAtEdge) for their compute layer.

Summary Modern cloud architectures are constructed using multiple specialized services; *Cloudscape* captures the use of 134 unique AWS services. Across the dataset, storage and compute services are most common and are used in more than 80% of the architectures. The deployment of S3 and Lambda across hundreds of architectures grounds ongoing research [1, 51, 74] and motivates the need for more work.

3.2 Functional Goals of Architectures

The architectures in *Cloudscape* are built to support a diverse set of business needs *e.g.* finance, games, ads, and medical research. Two architectures within the same business domain may have different computational and storage needs, making



(a) **Occurrence of service capabilities.** The stacked bars capture the proportion of nodes of each type found across all architectures in *Cloudscape*.

(b) **Popular AWS services.** This graph presents the ten most popular services in *Cloudscape*. Each bar represents the percentages of architectures that use a particular service. This graph shows that the most popular services span different core capabilities.

(c) **Percentage of architectures that use a capability.** Each bar indicates the percentage of architectures that use at least one service of a particular capability.

Figure 3: **Composition of Architectures.**

their comparison impractical. To better contextualize our analysis, we categorize architectures on the core functionalities they serve. An architecture may be classified under one or more of the following functional goals:

1. **Data Ingestion:** These architectures often perform ETL (Extract, Transform, Load) operations on a data stream. This involves processing the data stream and pushing it to different data stores. *e.g.* streaming pipelines built to handle analytics events, sensor data from edge devices, or access logs. Figure 2b shows one such architecture.
2. **Interactive:** These architectures are latency-sensitive and have an agent (human or otherwise) waiting for a response in real-time. *e.g.* video streaming, or a mobile app triggering a backend API. Figure 2a shows one such architecture.
3. **Compute Intensive:** These architectures are built to handle compute-intensive batch jobs. Their workload typically cares more about throughput and less about latency. *e.g.* video rendering, ML training, and simulation frameworks. Figure 2c shows one such architecture.
4. **Control Plane:** Most architectures in this category provision new AWS resources *e.g.* CI/CD pipelines. Many architectures also describe policies (*e.g.* security policies on S3 and VPC) and discuss how they manage AWS account resources. Architectures in this category stress AWS' control plane.
5. **Other:** All other architectures fall in this category.

Table 2 presents the distribution of goals across architectures. Our first observation is that almost 80% of the architectures in the dataset have one functional goal. The most prevalent goal in our dataset is *Data Ingestion* (45%), followed by *Interactive* (35%). 15% (51) architectures have two goals. Most of such architectures were tagged as *Interactive* and *Data Ingestion*. Only 2 architectures in our dataset span three goals. Both these architectures are tagged as *Data Ingest*, *Interactive*, and *Compute*. 17 architectures do not fit our classification; we tag them as *Others* and do not show them in the table.

	Data Ing.	Interactive	Compute	Control
Data Ing.	34% (116)	8% (28)	3% (9)	0% (0)
Interactive	8% (28)	23% (78)	2% (6)	2% (7)
Compute	3% (9)	2% (6)	10% (35)	0% (1)
Control	0% (0)	2% (7)	0% (1)	12% (41)
Total	45% (153)	35% (119)	15% (51)	14% (49)

Table 2: **Symmetric co-occurrence matrix of the number of architectures built for each functionality.** A cell at (**x**, **y**) shows the count of architectures with goals **x** and **y**. The diagonal cells show the architectures with a single function. Each cell is presented as the percentage of the dataset (number of architectures in brackets).

Summary We group architectures based on their core functionalities. Most architectures are built to do one thing well. Most of our dataset comprises architectures built to ingest a large volume of data or serve latency-sensitive clients.

4 Analysis of Storage Services

To understand the usage of the 14 storage services present in *Cloudscape*, we first analyze them from an architecture's perspective: how common are these services (§4.1), does the functional goal impact which service is used (§4.2), and how many services are used in an architecture (§4.3)? Next, we take a storage service's perspective and ask: which services interact with the storage layer (§4.4), how is S3, the most common storage service, used (§4.5), and what data is stored across different storage services (§4.6)? Finally, we consider how different specialized cloud services use storage implicitly:

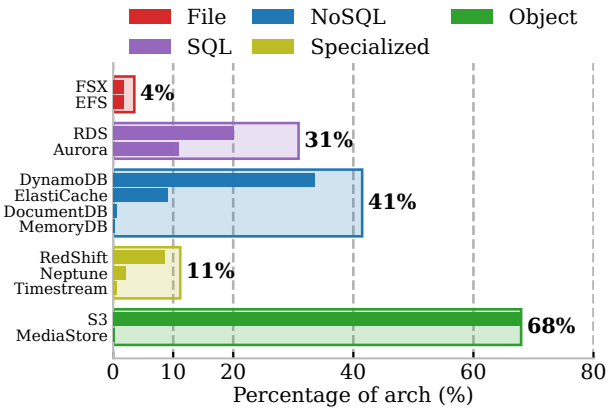


Figure 4: **Usage of different storage services**: Each thin horizontal bar represents the percentage of architectures that use a particular service. Services are grouped based on their schema. Each outer (wide) bar shows the percentage of architectures that use at least one of the services of that schema type.

ML services in §4.7, analytics engines in §4.8, and caching and queueing services in §4.9.

4.1 Popularity of Storage Services

Uncovering the popularity of different storage services can guide the efforts of the research community toward more impactful services. We group services with similar interfaces to allow comparison both across and within groupings. Services are categorized as one of:

1. **File System:** We group these services because they provide storage at the file level. We place EFS (Elastic File System) and FSX (File Server) in this bucket.
2. **SQL:** These services require the data to be structured in a relational model. We place RDS (Relational Database Service) and Aurora [84] in this bucket. RDS is a database service that lets the customer select their database engine (*e.g.* MySQL, PostgreSQL) and manages the infrastructure layer. Aurora is a proprietary database engine that is part of RDS but is often mentioned explicitly by the architectures.
3. **NoSQL:** These services store unstructured data. DynamoDB [35] is a key-value store and DocumentDB is a proprietary document store, similar to MongoDB. MemoryDB is a propriety Redis-compatible data store. ElastiCache is a caching service that uses the open-source versions of Redis and Memcached.
4. **Specialized:** These services are built for specific workloads: RedShift is a data-warehouse service with querying and storage capabilities, Neptune is a graph database, and Timestream is a timeseries database.
5. **Object:** These services (S3 and MediaStore) provide reads and writes to data on a per-object basis, often with a flat namespace. MediaStore is an object store that handles storing and streaming live media content.

Figure 4 shows the percentage of architectures that use a particular storage service. Most architectures use object stores, specifically S3. The figure also reveals that more than 40% architectures use NoSQL data stores, while fewer (~30%) use SQL services. Specialized services built for graphs and data warehousing are used in ~11% of the architectures. Surprisingly, a very small number of architectures (~4%) use distributed file-system services. We further observe a clear preference for services in each category: S3 among object stores, DynamoDB among NoSQL services, and RDS among SQL services. Consequently, these three services are the most commonly used storage services across all architectures.

One storage service missing from this analysis is Elastic Block Store (EBS). *Cloudscape* is unable to accurately capture EBS usage because developers rarely interface with it directly and, as a result, seldom mention it. Consequently, it appears in only 2% of architectures. This low usage is deceptive because EBS is tacitly used by other AWS services, such as EC2 [34], EKS [32], and RDS [33]. Notably, 60% of architectures make use of at least one of these services, thereby indirectly using EBS. This widespread reliance highlights the significant potential impact of ongoing advancements in block stores [92].

Implications Cloud architectures rarely interface with distributed filesystems directly, with most product-focused data being offloaded to more specialized storage services. In the face of these changing requirements, the community would benefit from revisiting the workloads faced by the filesystems deployed in the cloud. Further, it would be fruitful to distinguish between filesystem workloads generated by end-user applications (*e.g.* binaries on EC2 and Lambda) and those generated by overlying storage services like databases. Future filesystems can limit their scope to a few workloads instead of optimizing for everything [48, 72, 83], while still providing significant impact.

S3 should be considered the new default storage system for cloud architectures, playing the same role in cloud computing as the local filesystem does in an operating system [5]. The research community would greatly benefit from repeating the filesystem characterization studies for object stores [22, 44, 75, 76, 85]. Due to a lack of representative traces, new object stores are forced to use arbitrary, and differing, synthetic benchmarks [3, 86]. Other studies find that synthetic benchmarks do not fully capture all aspects of production traces [23, 28]. While Faa$T [74] provides a dataset of object accesses made in the wild, it lacks API parameters like RangeGet and MultiPartUpload, which could differ across use cases and have considerable performance impact.

Summary Architectures use object stores, NoSQL stores, and SQL stores, in that order. The most popular storage services are S3, DynamoDB, and RDS. Roughly 4% architectures explicitly mention using distributed file services. Future research should focus on understanding S3 workload and the APIs used.

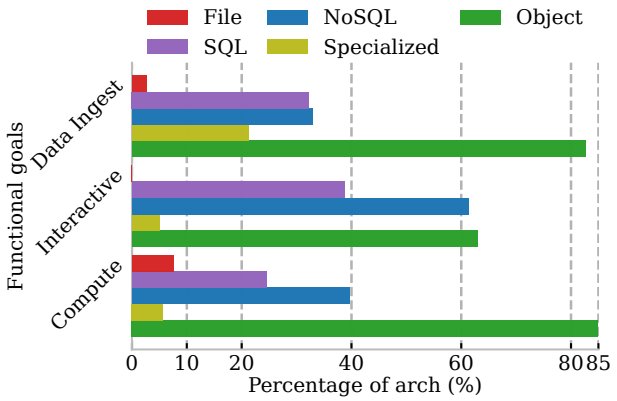


Figure 5: **Usage of storage services in architectures broken down by functional goals.** Each bar represents the percentage of architectures that use a particular type of storage service. The percentage is relative to the number of architectures with that particular goal.

4.2 Storage Usage Across Functional Goals

Next, we investigate how strongly an architecture's functional goal (§3.2) influences the storage services used. We capture this service preference in Figure 5. We observe that object stores are the most common storage type across functional goals, followed by SQL and NoSQL stores. However, their relative occurrences differ.

Data Ingestion architectures prefer NoSQL and SQL services equally. These architectures often perform ETL operations on data streams and distribute the transformed data to downstream services. The choice of these downstream storage services is driven by the nature of the queries that will be run on the transformed data, rather than the pipeline itself. SQL and NoSQL services are both used to answer downstream queries. We also find that the enriched dataset is often stored in S3 for future retrieval. Specialized data services such as Redshift, a data warehouse service, are more prevalent in *Data Ingestion* architectures.

Interactive architectures are less likely to use object stores. Though S3 is still used to serve static content like SPAs [66] and media content to users, architectures tend to use SQL and NoSQL services as they are better suited for transactional user-specific data.

Among the architectures that use file services, they are mostly tagged *Compute*. Thus, architectures that are built for compute-intensive batch jobs (*e.g.* simulations) are more likely to consider lower-level storage primitives like file services.

Implications Object stores are widely used across architectures regardless of functional goals. As a result, they must perform well under all workloads [3]. Optimizing either latency (for *Interactive* architectures) or throughput (for *Data Ingestion* architectures), while keeping the other stable, will have a significant impact. We find that NoSQL services are preferred over SQL services; this finding supports ongoing research into caching systems [24, 25, 58] and key-value stores [27, 56, 59]. The popularity of SQL services for *Interactive* architectures provides more evidence for the well-understood latency constraints imposed on such services.

≤ Services	0	1	2	3	4	5
Percentage (%)	10	47.65	86.76	98.24	99.71	100

(a) **Number of unique storage services used in each architecture.**

	SQL	NoSQL	Object
SQL	6% (19)	6% (21)	13% (45)
NoSQL	6% (21)	9% (30)	21% (70)
Object	13% (45)	21% (70)	28% (96)
Total	25% (85)	36% (121)	62% (211)

(b) **Symmetric co-occurrence matrix of the number of architectures using a combination of storage services.** Each cell (x, y) shows the number of architectures that use a service from both type x and y.

Flow interacts with ≥ storage	1	2	3	4
Percentage of arch. (%)	89.1	35.3	4.7	0.9

(c) **Percentage of architectures with one workflow interacting with multiple storage services.**

Table 3: **Combination of Storage Services**

Summary The functional goal of an architecture influences the storage services used. Filesystem services are more commonly used by architectures marked *Compute Intensive*, *e.g.* for video rendering.

4.3 Combination of Storage Services

We now consider whether architectures use one storage service as their primary data store or whether they spread their data across multiple services. In doing so, we wish to learn whether one data store is sufficient to model and service the data requirements of the architecture.

Table 3a shows the number of unique storage services used in an architecture. Most architectures use up to three different storage services. 10% architectures use no storage services, and few (~2%) use four or five.

We use Table 3b to understand which combinations of services are used across architectures. We limit our analysis to the three most common schema types. We observe that object services are often used in conjunction with NoSQL (21%) and SQL (13%) services. Similarly, SQL and NoSQL services are rarely used in isolation and are paired with another complementary service. 6% (20) architectures use services across all three types (not depicted in the table).

We also report that all architectures, except eight, used only one storage service per type. Of these eight exceptions,

seven used ElastiCache in the presence of DynamoDB or DocumentDB. One architecture used both S3 and MediaStore.

Servicing a single workflow With the presence of multiple storage services in an overarching architecture, learning how many are actively involved for a particular *task* would be useful. Such storage services are, in effect, loosely coupled, and finding common occurrences provides a deeper understanding of how the storage layer is used collectively. We analyze this using *workflows* provided by *Cloudscape*: we ask how many unique storage services are read from or written to during 1 workflow.

From Table 3c, 35% of the architectures in our dataset have at least one workflow that interacts with two different storage services. 5% architectures have a workflow that interacts with three or four. In the case of two storage interactions in a flow, in 75% of the occurrences S3 is touched with either a SQL or NoSQL store, while in 14% of the occurrences, the workflow calls a SQL and a NoSQL service.

Implications Cloud architectures use multiple storage services. While storage services do not directly interact with each other, they share the same fate because they often participate in the same workflow. This implicitly forces similar performance constraints on both services, *e.g.* from the perspective of an end-to-end task latency, a sluggish storage access could nullify any performance improvement made by a different latency-aware datastore. On the bright side, this finding motivates the need for understanding correlated workload patterns. Uncovering correlated patterns *across storage systems* can enable optimizations like shared-fate caching. Having one-third of the architectures in our dataset interact with different data stores across the same conceptual task further strengthens the need for providing cross-service consistency, an active area of research [36, 81].

Summary Most architectures use up to three storage services. SQL and NoSQL services are often paired with complementary storage services. A large number of workflows interact with multiple storage services.

4.4 Common Writers and Readers

We now analyze which services most frequently interact with storage services. In doing so, our goal is to find the common service pairs to optimize for. Using the direction of *data* edges provided by *Cloudscape*, Figure 6 shows the percentage of architectures in which an AWS service writes to or reads data from any storage service. This figure lets us draw three important conclusions.

First, the set of frequent writers has a high overlap with the set of frequent readers. Thus, these services would benefit from optimizations that improve both read/write performance.

Second, the most common interacting services are compute services, followed by integration services and S3. Of the eight compute services that appear in the top ten, five of them are general-purpose compute platforms (Lambda, EC2, EKS, ECS, Fargate); this indicates that recent cloud architectures decouple

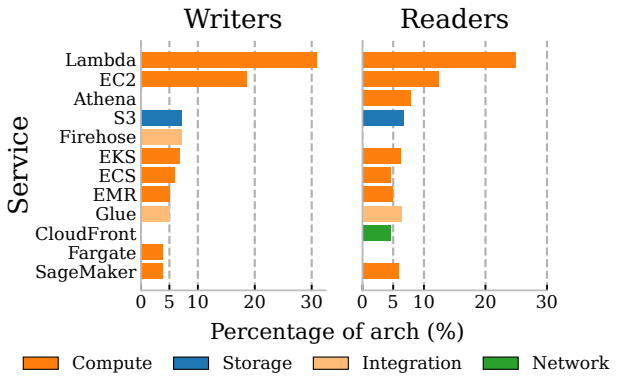


Figure 6: **Services writing and reading from storage services.** This shows the top ten services writing and reading to storage services. The services are sorted by write percentage.

their compute layer from the storage layer and have fully embraced managed data services. Besides general-purpose compute services, purpose-built analytics services like EMR and Athena also directly interface with storage services. Finally, SageMaker, an ML platform, uses existing AWS storage services (primarily S3) for both, storing models and reading data for training and inference. S3 appears in this figure because customers often describe cross-account transfers or cross-region data duplication inside S3, which results in edges between different S3 nodes. Additionally, we find that exposing data publicly to users via networking services is relatively uncommon, though some architectures, *e.g.* SundaySky [8], do serve SPAs [66] and media content from S3 via CloudFront.

Third, Lambda is the most common user of storage services by far, interacting almost twice as often as the next service. While it is natural for a stateless service to rely on external storage to provide state, this is the first work that provides a sense of the relative pressure stateless compute frameworks will place on storage services, as opposed to existing well-understood workloads from EC2 and EMR.

Implications Given the number of AWS-proprietary services frequently interacting with storage services (namely SageMaker, Glue, Firehose and Athena), there is ample opportunity to co-design these frameworks with storage services for better performance [79, 89]. Ongoing research in improving interactions between serverless compute and storage [1, 51] should significantly impact cloud architectures deployed in the wild.

Summary Storage services primarily interface with compute, networking, and other storage services. Lambda, EC2 and Athena are the most common services interacting with storage services. High usage of storage by other proprietary services like Athena, Firehose, and Glue highlights the opportunity for co-designing them.

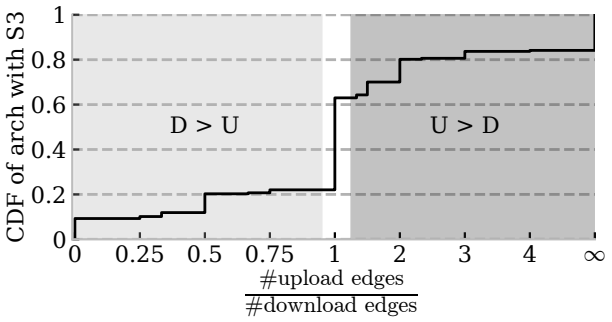


Figure 7: **Comparison of uploads and downloads to S3**. This shows the ratio of upload edges to download edges. Architectures falling on the left side of (x=1) have more services that download from S3 than those uploading to S3, while the ones on the right have more services that upload.

4.5 How Is S3 Used?

We attempt to better understand the usage of S3 given its widespread adoption. Our goal is to understand whether S3 is the dumping ground of data, or if it is intricately involved in workflows.

Within an architecture, S3 is more connected than other storage services. We report that on average, the number of unique services that upload to S3 is more than the services that upload to any other storage service that might be present in an architecture. This is true for the number of services that download from it as well. Additionally, on average, S3 is involved in more workflows than other storage services. This highlights a rich ecosystem of services interfacing with S3, likely due to significant efforts in ensuring compatibility [63].

To understand how an architecture uses S3, we consider the ratio of data-upload edges to data-download edges. We clarify that an upload edge means that a service is writing data to S3. Figure 7 shows that at one end (x=0), ~10% architectures do not explain how data reached S3; they simply download from it. Conversely, ~20% architectures dump information into S3 (x=∞) and do not interact with it further, often to enable additional workflows outside the architecture or for archival purposes. The remaining 70% of the architectures interact more meaningfully by both downloading and uploading data (note that *Cloudscape* is unable to capture the volume of data or track if the same object is moved across workflows). 10% of the architectures are skewed towards downloads, with the download-to-upload ratio ranging from $(1,4]$, presented as $(0.25,1)$ in the graph. Around 20% of architectures upload more often than they download, with the upload-to-download ratio ranging from $(1,4]$, presented as $(1,4)$ in the graph. The remaining 40% of the architectures have the same number of upload and download edges from S3 (x=1).

Implications S3 participates in multiple workflows in most architectures. While we could not objectively capture it, we observed during our data collection phase that S3 was often used as a staging area to pass objects across services and workflows. This trend would see considerable benefits from the recently launched S3 Express One Zone [11] that trades availability for performance by storing data in a single availability zone. We believe more research on end-to-end object tracing as objects get transferred across workflows will provide the community with a better understanding of how S3 is used, and therefore must be improved. Future research can consider allowing users to sacrifice high redundancy in favor of performance and cost if the workload permits. Generally, more progress on how best to present tradeoffs between performance, cost, and availability with a simple API, would be useful to practitioners.

Summary S3 is used across multiple workflows to both read and write objects from multiple services; it facilitates data movement across workflows. Future research on object tracking across workflows is necessary to get an accurate picture of the lifecycle of objects [76].

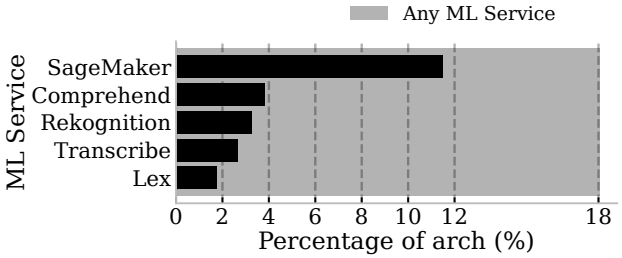
4.6 What Data Is Stored Where?

We now present a qualitative analysis (§2.2) on what types of data are stored in these services. Our goal is to identify the semantic meaning of the data stored, making it an independent factor to optimize for. Our analysis focuses on clearly identifiable use cases, skipping vague or general descriptions like "user data."

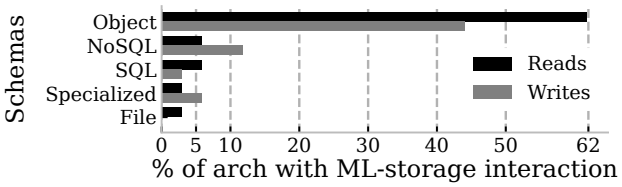
Among identifiable use cases, S3 is used for a wide spectrum of purposes: logs (16 cases), images (16 cases), web (11 cases), videos (11 cases), executable code or ML models (9 cases), metadata (7 cases) and database backup or data archival (6 cases).We also observe that S3 is widely used in the data processing pipeline: it stores the raw input data in 24 cases and post-processed data in 18 cases.

Other than storing general data, DynamoDB is also commonly used to store metadata (16 cases), *e.g.*, configuration files, and pointers to data on S3. Interestingly, we also found it is used in 12 cases to store the system state (*e.g.*, job status) for orchestration and coordination purposes. Such functionality is dominated by DynamoDB: there are only 3 use cases in ElastiCache, 1 in RDS, and 1 in Aurora.

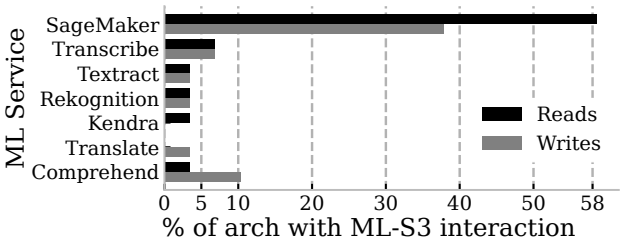
Implications Storage services are used to store data not only of varying types and sizes but also of different importance. For example, we found that S3 stores analytics files (*e.g.* log files) as well as configuration files. Downstream systems fetching them might respond to transient failures differently; failure to fetch a log file can be rescheduled in the context of a larger batch operation [90], while failure to fetch a configuration file might halt an infrastructure setup. Crucially, storage services are forced to treat each datum with the same strict constraints; future research should consider whether encoding expectations of availability can improve the effective reliability of storage services. Studies that compare the usage of different MIME types on object stores will also be useful.



(a) **Presence of ML services in architectures**. The bar graph shows the percentage of architectures that use the five most common ML services. The grayed larger bar shows the percentage of architectures that use at least one of the 11 ML services.



(b) **Popularity of storage services that interact with ML services.** The distribution is normalized against the number of architectures having at least one ML service interacting with any storage service.



(c) **Breakdown distribution of ML services that interact with S3.** The distribution is normalized against the number of architectures having at least one ML service interacting with S3.

Figure 8: **The role of storage in ML services.**

Summary S3 stores a wide spectrum of data, ranging from logs to executable code. Outside customer data, DynamoDB is often used for tracking metadata of the rest of the system. Future work should categorize popular data formats and requirements to better understand use-cases.

4.7 The Role of Storage in ML Services

The demand for intelligent products has risen dramatically over the past decade, and cloud operators have answered that call by spinning up multiple dedicated ML services. *Cloudscape* captures 11 ML services used across all architectures (we categorize a service to be ML using AWS' classification [16]). Figure 8a captures the presence of the five most popular ML services. It is of note that 18% of all architectures use some ML service. It is therefore important to study how such services fit in the larger context of architectures, and how they interact with the storage layer.

From Figure 8b, we observe that ML services prefer to read from and write to S3 compared to other storage services. Given this strong preference, Figure 8c focuses on the ML services that interact with S3. SageMaker is a general-purpose model training and inference platform, which provides some intuition to its popularity and its use of S3 for reading training data and storing models. Most of the other ML services are specialized for language and video processing, *e.g.* Transcribe, Rekognition, and Textract. Such services often require their input data to reside in S3 and store their output back in S3 as well [15].

Implications Understanding how ML services integrate into larger cloud architectures is critical, as their adoption is expected to increase significantly. S3 has become the de-facto layer for holding training data. While research has mostly focused on improving training performance by building systems on top of object stores [53, 69], future work that targets object stores directly and selectively enables workload-aware optimizations would have a considerable impact on platforms like SageMaker and opensource alternatives [62]. Conversely, it is also important to consider if popular formats used to store training data (*e.g.* CSV, JSON, Parquet [4] and TFRecord [50]) fully exploit the properties of S3.

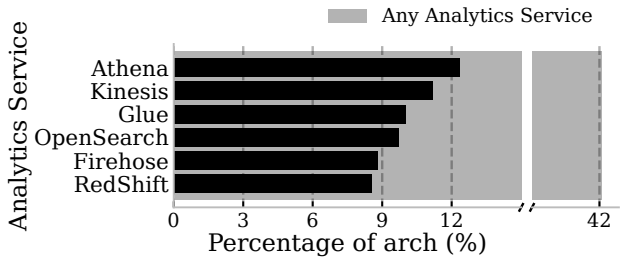
In the context of specialized ML services, S3 has also become the layer for exchanging data between asynchronous ML inference operations. As *Cloudscape* shows, cloud architectures stitch together specialized compute, networking, and storage services; we similarly expect more specialized ML services to be included in the mix, instead of a single service that handles all types of ML queries. Getting the interaction layer right for these services is crucial for the next decade as the adoption grows. Future research should question when this interoperability layer requires replication or even durability.

Summary Object stores are used for storing training data, models, and also the input and output for ML inference services. Future research should study how to best design cloud native storage formats, and question whether S3 is the right API for providing interoperability between ML services.

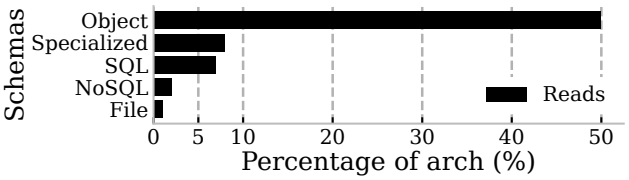
4.8 Storage for Analytics Frameworks

Analytics is important for most architectures; indeed, Figure 9a shows that 42% of the architectures use some analytics service. We define a service to provide analytics using AWS' categorization [16]. Third-party services like Snowflake [29] appear only in a few architectures, so we do not include them. Understanding the current deployment practices of cloud analytics services will help focus future efforts. For this, we split the services into ones that provide batch or ad-hoc queries and those that enable streaming analytics.

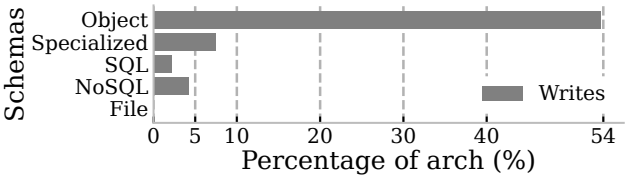
Athena, Glue, RedShift, OpenSearch and QuickSight are the most popular analytics services that enable batch and ad-hoc queries. We found that compute services often



(a) **Presence of analytics services in architectures**. The bar graph shows the percentage of architectures that use the most common analytics services. The grayed larger bar shows the percentage of architectures that use at least one of the 12 analytics services



(b) **Popularity of storage services read by analytics services.** The distribution is normalized against the number of architectures with at least one batch service.



(c) **Popularity of storage services written by streaming services.** The distribution is normalized against the number of architectures with at least one streaming service.

Figure 9: **Storage for analytics frameworks.**

directly write to RedShift and OpenSearch as these services have dedicated data stores. However, the rest of the services primarily read from storage services. Figure 9b shows these batch analytics services mostly read from object stores instead of pulling in data from other storage services.

Kinesis, Firehose, EMR, and MSK are the most popular streaming analytics services. We found that these services primarily receive data from compute services and write to storage services. Figure 9c shows that services predominately write to object stores instead of specialized data services.

Implications We found two broad classes of analytics storage engines: vertically integrated analytics engines (Redshift, and OpenSearch) that ingest the data, and remote-storage centric analytics engines (*e.g.* Glue and Athena). Vertically integrated analytics engines impose product-lockin; future research focusing on narrowing the performance gap between them and remote-storage engines is necessary to remedy this. Open source file formats often used by remote-storage engines lack rich tooling [43] that prevents data analysts from quickly groking data. Future research that discusses how to provide a rich library of thin tools similar to coreutils [41] would provide a flexible alternative to frameworks.

Given the popularity of object stores as the data lake, it has become important to consider cloud-native file formats. *e.g.* many architectures use the Parquet [4] format which tightly couples the data and metadata (*e.g.* minimum and maximum values of columns) in a single file. As analytics engines read metadata to prevent scanning the full data, this still results in object accesses, which will interfere with S3 auto-tiering [9] mechanisms that operate at a coarser object-level granularity.

Finally, writing processed data to object stores instead of specialized storage services suggests an underlying challenge developers face in effectively modeling their processed data. This also often results in data duplicated in SQL/NoSQL services for interactive querying and object stores for future analytics.

Summary Analytics services are used in almost half the architectures. Services read primarily from S3, future work should focus on minimizing data duplication and building S3-specific file formats.

4.9 Storage-Adjacent Services

Cloud architectures often rely on services that are closely related to the storage layer. In this section, we examine three such families of services: caches, queues, and container registries. Caches improve the performance of the storage layer, queues provide durable buffers between workflows, and container registries power containerized deployments by serving images.

ElastiCache is the sole caching service in *Cloudscape*. Caches appear to be most suited for web systems, with over 70% of ElastiCache usage observed in *Interactive* architectures. Among storage services, ElastiCache is primarily used to cache responses from RDS. Overall, caching is more commonly employed by generalized compute services such as EC2, EKS, Fargate, and ECS, rather than stateless applications like Lambda.

For queuing, architectures use SQS, SNS, EventBridge, and MSK. These services provide temporal yet durable storage for messages and are designed for latency-sensitive production and consumption patterns. We now analyze the prevalence of these services and the common producers and consumers.

36% architectures use at least one messaging service. SQS and SNS are the most popular ones, used in 67 and 46 architectures, respectively. Note that a single architecture might utilize the same service multiple times, *e.g.* SQS appears 87 times across 67 architectures. The fan-out of these services is generally small: 53/87 cases of SQS, 19/47 cases of SNS, 6/17 cases of EventBridge, and 5/12 cases of MSK are single-producer-single-consumer. Among these 83 single-producer-single-consumer use cases, there is a relatively flat distribution of producer/consumer service pairs, with a long tail. 42 unique producer/consumer combinations were observed. The most common pair is Lambda→Lambda with only 9 cases, followed by S3→Lambda (6 cases) and EC2→EC2 (4 cases).

In AWS, container images are hosted on the Elastic Container Registry (ECR) service. Similar to the case of EBS (§4.1), developers seldom mention ECR explicitly, resulting in it being referenced in fewer than 5% architectures. However, we can better estimate its usage by identifying architectures that use ECS, EKS, or Fargate – all of which involve running containers. Using this method, we find that 35% of architectures rely on ECR, making it an attractive candidate for future research.

Implications We observe that Lambda rarely interacts with caching services. Despite recent work demonstrating the benefit of caching for serverless workloads [1, 65], traditional caching services do not seem to be deployed. This likely reflects the limitations of general-purpose caching services in the context of serverless workloads, and motivates the need for caching services integrated with serverless platforms [74].

On the other hand, Lambda is the most common consumer for queueing services, used in 38% of SQS cases. Serverless platforms must therefore optimize their invocation strategy when triggered by queues. However, queues must balance these optimizations, ensuring they do not favour Lambda at the expense of other services. The diversity of producer/consumer pairs highlights the broad range of usecases queues must serve; indeed this shows that they are highly effective at combining different types of asynchronous workflows.

Summary Caching services are primarily placed in front of relational datastores. Most messaging services are operated in a single-producer-single-consumer manner, but there is a large distribution of services acting as producers or consumers. Container registries power more than one-third of the architectures.

5 Discussion

From our study of cloud deployments, we now discuss the future directions for research.

Open-source storage services and testbeds for the research community Most cloud services are based on proprietary closed-source software, which becomes an obstacle for system researchers to conduct in-depth studies; as a result, the researchers often need to build prototypes atop some open-source alternatives. The major alternatives for S3 include MinIO [60], Ceph [86], and SeaweedFS [55]. Among them, MinIO [60] is the only system designed with S3-compatibility as its primary goal, while Ceph [86] and SeaweedFS [55] expose S3-compatible APIs despite different internal architectures. Given the dominance of S3, the research community must pay more attention to these object stores. Any functionality and reliability discrepancy between S3 and these alternatives must also be studied, as this impacts the transferability of new techniques from open-source systems to real production systems like S3.

Alongside systems, the research community requires representative testbeds. The popular microservice testbed DeathStarBench [39] closely aligns with our findings. It uses three different storage services: Redis, MongoDB, and Memcached. Additionally, some workflows read from more than one storage service, *e.g.* the ReadMovieReviews API reads from MongoDB and Redis, consistent with our findings in §4.3. However, DeathStarBench surprisingly does not use any object store. Instead, static content is served via instances using Lua scripts within nginx. We believe that a more representative setup should instead use an open-source object store like MinIO. Other serverless testbeds like vhive-serverless [82] and FunctionBench [49] use object stores but do not use any other storage service; researchers must therefore be careful not to treat any single testbed as completely representative.

Developer rationale *Cloudscape* shows that S3 and Lambda have seen strong adoption. Both services share several advantages compared to their counterparts: they are cost-effective, require minimal developer oversight, and enjoy deep integration with a host of other cloud services. However, as alternative storage and compute services offer better performance for specific use-cases, a qualitative study understanding developers' reasons for choosing particular services would be highly valuable. As services vary in performance, price, and ease of use, understanding developers' priorities would guide future research.

Simplifying seemingly simple APIs To effectively store data at scale in S3, a developer must consider versioning, storage classes, cross-region replication, permissions, lifecycle rules, and more. Notably, we found several studies devoted subsections of their architecture to ensuring proper management policies were applied. Thus, while the S3 data access API appears straightforward, much of its complexity lies within the management API. Middleware and library solutions that understand common needs and offer easily selectable policy profiles for various workloads would benefit practitioners.

Optimal architectures in the face of ever-changing infrastructure Modern cloud architectures are built on an ever-changing wave of services built by cloud operators. The high number of cloud services to learn about and select from makes designing an optimal system exponentially harder. This problem is exacerbated by the inability to pin cloud services to specific versions. As cloud services constantly improve, it results in gradual software rot even though the code and product goals have not changed, *e.g.* S3 recently introduced conditional writes [14], which would obviate key decisions made by architectures built before this feature existed. Future research that constantly evaluates workloads and changing service capabilities and suggests surgical changes to the architecture would be useful. Such models can unearth cross-service optimizations that would otherwise require a developer to be up-to-date about all services.

As the usage of specialized services increases, data increasingly hops through multiple systems before completing a logical workflow. This exposes the need to enable cross-service consistency, which the community has started

considering actively [36, 81]. Further, having data cross multiple service boundaries raises security and compliance concerns, which need to be tackled per service. While systems exist that enable security for individual cloud services [2, 46], we lack systematic cross-service integration. Thus, future research should evaluate the benefit of having services describe consistency and security SLAs in their API.

6 Related Work

Previous work has studied real-world systems with different focuses. Google [71] published an analysis of their clusters in 2012, revealing the highly dynamic and heterogeneous nature of workloads. More recently, Guo et al. [42] and Liu and Yu [54] analyzed production traces from Alibaba data centers to understand resource utilization and characterize workload colocation. These studies focus on the overall resource consumption pattern instead of how these systems are used and interact with each other.

The interaction among large-scale production systems has been studied but on the lower stack of the infrastructure and mostly from cloud service providers' perspective. Huye et al. [45] studied the microservice architectures within Meta's data center, analyzing the topology and flows of microservices, as did Luo et al. [57] for Alibaba. Google released a characterization of RPCs in their infrastructures [77]. Many distributed storage systems, e.g., GFS [40], F4 [64], Haystack [23], Tectonic [68], were built based on the company-internal observation of usage patterns but are primarily used for private cloud with internal customers, not for public cloud.

Earlier studies examining public cloud usage patterns primarily focus on compute services, such as serverless computing [70, 78] and MLaaS [87]. In contrast, our work studies the usage of public cloud storage services from the users' perspective. A related dataset that provides insights into cloud service usage is the StackOverflow dataset [80], although it does not easily allow an understanding of service interactions. On analyzing the AWS tags in this dataset, we found S3, EC2, and Lambda tags to be the most common, significantly more popular than the rest. This observation aligns with our findings (Figure 3b) and reinforces the validity of our dataset.

7 Conclusion

We present *Cloudscape*, a large-scale dataset describing real-world architectures deployed on AWS. Our analysis of deployed storage services reveals a high adoption of S3 and the use of multiple services to meet the storage needs of an architecture. We find that storage services are used for a wide range of semantically distinct data and play a crucial role in specialized ML and analytics services.

The research community would benefit from understanding object store workloads and tracking the lifecycle and movement of objects. As architectures become more heterogeneous and storage services become loosely tied with both storage and non-storage services, it is worth exploring how storage services can adapt to this evolving landscape.

We hope our analysis informs future storage research and that our dataset enables future studies on other critical classes of services. The dataset is made available at https://github.com/WiscADSL/Cloudscape.

Acknowledgments

We thank our shepherd, Cesar A. Stuardo, and the anonymous reviewers of FAST '25 for their valuable feedback and comments. This work was supported by the NSF under award number CNS-2402859 and NetApp. We also thank Microsoft and InfluxData for their generous support. Additionally, we are grateful to Sachin Ashok, Naman Gupta, and the students of the ADSL group at UW-Madison for their insightful discussions.

References

[1] Mania Abdi, Samuel Ginzburg, Xiayue Charles Lin, Jose Faleiro, Gohar Irfan Chaudhry, Inigo Goiri, Ricardo Bianchini, Daniel S Berger, and Rodrigo Fonseca. Palette load balancing: Locality hints for serverless functions. In *Proceedings of the Eighteenth European Conference on Computer Systems*, EuroSys '23, page 365–380, New York, NY, USA, 2023. Association for Computing Machinery.

[2] Sebastian Angel, Aditya Basu, Weidong Cui, Trent Jaeger, Stella Lau, Srinath Setty, and Sudheesh Singanamalla. Nimble: Rollback Protection for Confidential Cloud Services. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 193–208, Boston, MA, July 2023. USENIX Association.

[3] Ali Anwar, Yue Cheng, Aayush Gupta, and Ali R. Butt. MOS: Workload-aware Elasticity for Cloud Object Stores. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '16, page 177–188, New York, NY, USA, 2016. Association for Computing Machinery.

[4] Apache. Parquet. https://parquet.apache.org, 2024. [Accessed 10-09-2024].

[5] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books, 1.00 edition, August 2018.

[6] AWS. Haptik: Data Lake for Conversational AI — youtube.com. https://www.youtube.com/watch?v=r2xfxJ-sXMY, 2019. [Accessed 22-05-2024].

[7] AWS. PixelStrings by Cinnafilm: A Flexible & Scalable Platform for Video/Audio Optimization & Conversion — youtube.com. https://www.youtube.com/watch?v=BZ32w0SSAoY, 2020. [Accessed 22-05-2024].

[8] AWS. SundaySky: Create Personalized Videos in Real Time on GPU based Spot Instances for Video Rendering. https://www.youtube.com/watch?v=6CgqEzyWpeA, 2020. [Accessed 22-05-2024].

[9] AWS. Amazon S3 Intelligent-Tiering Storage Class. https://aws.amazon.com/s3/storage-classes/intelligent-tiering, 2022. [Accessed 10-09-2024].

[10] AWS. Snap: Journey of a Snap on Snapchat Using AWS — youtube.com. https://www.youtube.com/watch?v=Cgv0kfp_6xQ, 2022. [Accessed 22-05-2024].

[11] AWS. Amazon S3 Express One Zone. https://aws.amazon.com/s3/storage-classes/express-one-zone/, 2023. [Accessed 10-09-2024].

[12] AWS. Amazon Elastic File System. https://aws.amazon.com/efs/, 2024.

[13] AWS. Amazon Redshift. https://aws.amazon.com/redshift/, 2024. [Accessed 10-09-2024].

[14] AWS. Amazon S3 now supports conditional writes. https://aws.amazon.com/about-aws/whats-new/2024/08/amazon-s3-conditional-writes/, 2024. [Accessed 10-09-2024].

[15] AWS. Amazon Transcribe: Data input and output. https://docs.aws.amazon.com/transcribe/latest/dg/how-input.html, 2024. [Accessed 10-09-2024].

[16] AWS. AWS services by category. https://docs.aws.amazon.com/whitepapers/latest/aws-overview/amazon-web-services-cloud-platform.html, 2024. [Accessed 16-05-2024].

[17] AWS. Machine Translation - Amazon Transcribe. https://aws.amazon.com/translate/, 2024. [Accessed 10-09-2024].

[18] AWS. Managed Kafka. https://aws.amazon.com/msk/, 2024. [Accessed 10-09-2024].

[19] AWS. Managed SQL Database - Amazon Relational Database Service (RDS) - AWS. https://aws.amazon.com/rds/, 2024. [Accessed 22-05-2024].

[20] AWS. Speech To Text - Amazon Transcribe. https://aws.amazon.com/transcribe/, 2024. [Accessed 10-09-2024].

[21] AWS. This Is My Architecture | Amazon Web Services. https://www.youtube.com/playlist?list=PLhr1KZpdzukdeX8mQ2qO73bg6UKQHYsHb, 2024.

[22] Mary G Baker, John H Hartman, Michael D Kupfer, Ken W Shirriff, and John K Ousterhout. Measurements of a Distributed File System. In *Proceedings of the thirteenth ACM symposium on Operating systems principles*, pages 198–212, 1991.

[23] Doug Beaver, Sanjeev Kumar, Harry C Li, Jason Sobel, and Peter Vajgel. Finding a needle in Haystack: Facebook's photo storage. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*, 2010.

[24] Benjamin Berg, Daniel S. Berger, Sara McAllister, Isaac Grosof, Sathya Gunasekar, Jimmy Lu, Michael Uhlar, Jim Carrig, Nathan Beckmann, Mor Harchol-Balter, and Gregory R. Ganger. The CacheLib Caching Engine: Design and Experiences at Scale. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 753–768. USENIX Association, November 2020.

[25] Nathan Bronson, Zach Amsden, George Cabrera, Prasad Chakka, Peter Dimov, Hui Ding, Jack Ferris, Anthony Giardullo, Sachin Kulkarni, Harry Li, Mark Marchukov, Dmitri Petrov, Lovro Puzar, Yee Jiun Song, and Venkat Venkataramani. TAO: Facebook's Distributed Data Store for the Social Graph. In *2013 USENIX Annual Technical Conference (USENIX ATC 13)*, pages 49–60, San Jose, CA, June 2013. USENIX Association.

[26] Sebastian Burckhardt, Chris Gillum, David Justo, Konstantinos Kallas, Connor McMahon, and Christopher S. Meiklejohn. Durable Functions: Semantics for Stateful Serverless. *Proceedings of the ACM on Programming Languages*, 5(OOPSLA):1–27, 2021.

[27] Badrish Chandramouli, Guna Prasaad, Donald Kossmann, Justin Levandoski, James Hunter, and Mike Barnett. FASTER: A Concurrent Key-Value Store with In-Place Updates. In *Proceedings of the 2018 International Conference on Management of Data*, pages 275–290, 2018.

[28] Yu Chen, Wei Tong, Dan Feng, and Zike Wang. Mass: Workload-Aware Storage Policy for OpenStack Swift. In *Proceedings of the 49th International Conference on Parallel Processing*, pages 1–11, 2020.

[29] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Ava nes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huan g, and Allison W. Lee. The Snowflake Elastic Data Warehouse. SIGMOD '16, June 2016.

[30] Martijn de Heus, Kyriakos Psarakis, Marios Fragkoulis, and Asterios Katsifodimos. Distributed Transactions on Serverless Stateful Functions. In *DEBS '21: The 15th ACM International Conference on Distributed and Event-based Systems*, pages 31–42, July 2021.

[31] Delve. How To Do Open, Axial and Selective Coding in Grounded Theory. https://delvetool.com/blog/openaxialselective, Feb 2022. [Accessed 04-05-2024].

[32] AWS docs. Amazon EBS CSI migration frequently asked questions. https://docs.aws.amazon.com/eks/latest/userguide/ebs-csi-migration-faq.html#csi-migration-faq-default-storageclass, 2024. [Accessed 20-01-2025].

[33] AWS docs. Amazon RDS DB instance storage. https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/CHAP_Storage.html, 2024. [Accessed 20-01-2025].

[34] AWS docs. Storage options for your Amazon EC2 instances. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Storage.html, 2024. [Accessed 20-01-2025].

[35] Mostafa Elhemali, Niall Gallagher, Bin Tang, Nick Gordon, Hao Huang, Haibo Chen, Joseph Idziorek, Mengtian Wang, Richard Krog, Zongpeng Zhu, et al. Amazon DynamoDB: A Scalable, Predictably Performant, and Fully Managed NoSQL Database Service. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 1037–1048, 2022.

[36] João Ferreira Loff, Daniel Porto, João Garcia, Jonathan Mace, and Rodrigo Rodrigues. Antipode: Enforcing Cross-Service Causal Consistency in Distributed Applications. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 298–313, New York, NY, USA, 2023. Association for Computing Machinery.

[37] Fortune Business Insights. Cloud Computing Market Size, Share, and Industry Analysis, April 2024.

[38] Apache Software Foundation. OpenWhisk Architecture. https://cwiki.apache.org/confluence/display/OPENWHISK/System+Architecture (Last Accessed January 2023), 2023.

[39] Yu Gan, Yanqi Zhang, Dailun Cheng, Ankitha Shetty, Priyal Rathi, Nayan Katarki, Ariana Bruno, Justin Hu, Brian Ritchken, Brendon Jackson, et al. An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 3–18, 2019.

[40] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, 2003.

[41] GNU. coreutils/coreutils | Github. https://github.com/coreutils/coreutils, 2010. [Accessed 10-09-2024].

[42] Jing Guo, Zihao Chang, Sa Wang, Haiyang Ding, Yihui Feng, Liang Mao, and Yungang Bao. Who limits the resource efficiency of my datacenter: an analysis of Alibaba datacenter traces. In *Proceedings of the International Symposium on Quality of Service*, IWQoS '19, New York, NY, USA, 2019. Association for Computing Machinery.

[43] HackerNews. Querying Parquet with Precision Using DuckDB | HackerNews. https://web.archive.org/web/20210626071921/https://news.ycombinator.com/item?id=27634840, 2021. [Accessed 10-09-2024].

[44] Tyler Harter, Chris Dragga, Michael Vaughn, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. A File is Not a File: Understanding the I/O Behavior of Apple Desktop Applications. *ACM Transactions on Computer Systems (TOCS)*, 30(3):1–39, 2012.

[45] Darby Huye, Yuri Shkuro, and Raja R. Sambasivan. Lifting the veil on Meta's microservice architecture: Analyses of topology and request workflows. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 419–432, Boston, MA, July 2023. USENIX Association.

[46] Deepak Sirone Jegan, Liang Wang, Siddhant Bhagat, and Michael Swift. Guarding Serverless Applications with Kalium. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4087–4104, Anaheim, CA, August 2023. USENIX Association.

[47] Zhipeng Jia and Emmett Witchel. Nightcore: Efficient and Scalable Serverless Computing for Latency-Sensitive, Interactive Microservices. ASPLOS '21, April 2021.

[48] Rohan Kadekodi, Se Kwon Lee, Sanidhya Kashyap, Taesoo Kim, Aasheesh Kolli, and Vijay Chidambaram. SplitFS: Reducing software overhead in file systems for persistent memory. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 494–508, 2019.

[49] Jeongchul Kim and Kyungyong Lee. FunctionBench: A Suite of Workloads for Serverless Cloud Function Service. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 502–504. IEEE, 2019.

[50] Jong Wook Kim. Anatomy of TFRecord. https://jongwook.kim/blog/Anatomy-of-TFRecord.html, 2024. [Accessed 10-09-2024].

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

[52] Peter Kraft, Qian Li, Kostis Kaffes, Athinagoras Skiadopoulos, Deeptaanshu Kumar, Danny Cho, Jason Li, Robert Redmond, Nathan W. Weckwerth, Brian S. Xia, Peter Bailis, Michael J. Cafarella, Goetz Graefe, Jeremy Kepner, Christos Kozyrakis, Michael Stonebraker, Lalith Suresh, Xiangyao Yu, and Matei Zaharia. Apiary: A DBMS-Backed Transactional Function-as-a-Service Framework. *CoRR*, abs/2208.13068, 2022.

[53] Abhishek Vijaya Kumar and Muthian Sivathanu. Quiver: An Informed Storage Cache for Deep Learning. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 283–296, Santa Clara, CA, February 2020. USENIX Association.

[54] Qixiao Liu and Zhibin Yu. The Elasticity and Plasticity in Semi-Containerized Co-locating Cloud Workload: a View from Alibaba Trace. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '18, page 347–360, New York, NY, USA, 2018. Association for Computing Machinery.

[55] Chris Lu. SeaweedFS. https://seaweedfs.com.

[56] Lanyue Lu, Thanumalayan Sankaranarayana Pillai, Hariharan Gopalakrishnan, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. WiscKey: Separating Keys from Values in SSD-Conscious Storage. *ACM Transactions On Storage (TOS)*, 13(1):1–28, 2017.

[57] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '21, page 412–426, New York, NY, USA, 2021. Association for Computing Machinery.

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

[59] Microsoft. Garnet. https://microsoft.github.io/garnet/, 2024.

[60] MinIO, Inc. Minio. https://min.io/.

[61] MIT Technology Review Insight. 2023 Global Cloud Ecosystem. https://www.technologyreview.com/2023/11/16/1078645/2023-global-cloud-ecosystem/, November 2023.

[62] MLflow.org. MLflow. https://mlflow.org/, 2024. [Accessed 10-09-2024].

[63] MonadCloud. The Importance of AWS S3 Compatibility. https://www.monadcloud.com/blog/2012/1/30/aws-s3-compatibility.html, 2012. [Accessed 10-09-2024].

[64] Subramanian Muralidhar, Wyatt Lloyd, Sabyasachi Roy, Cory Hill, Ernest Lin, Weiwen Liu, Satadru Pan, Shiva Shankar, Viswanath Sivakumar, Linpeng Tang, and Sanjeev Kumar. f4: Facebook's Warm BLOB Storage System. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 383–398, Broomfield, CO, October 2014. USENIX Association.

[65] Djob Mvondo, Mathieu Bacou, Kevin Nguetchouang, Lucien Ngale, Stéphane Pouget, Josiane Kouam, Renaud Lachaize, Jinho Hwang, Tim Wood, Daniel Hagimont, Noël De Palma, Bernabé Batchakui, and Alain Tchana. OFC: An Opportunistic Caching System for FaaS Platforms. In *Proceedings of the Sixteenth European Conference on Computer Systems*, EuroSys '21, page 228–244, New York, NY, USA, 2021. Association for Computing Machinery.

[66] Mozilla Developer Network. SPA (Single-page application) - MDN Web Docs Glossary: Definitions of Web-related terms | MDN — developer.mozilla.org. https://developer.mozilla.org/en-US/docs/Glossary/SPA. [Accessed 14-05-2024].

[67] Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea Arpaci-Dusseau, and Remzi Arpaci-Dusseau. SOCK: Rapid Task Provisioning with Serverless-Optimize Containers. USENIX Annual Technical Conference (USENIX ATC '18), June 2018.

[68] Satadru Pan, Theano Stavrinos, Yunqiao Zhang, Atul Sikaria, Pavel Zakharov, Abhinav Sharma, Shiva Shankar P, Mike Shuey, Richard Wareing, Monika Gangapuram,

Guanglei Cao, Christian Preseau, Pratap Singh, Kestutis Patiejunas, JR Tipton, Ethan Katz-Bassett, and Wyatt Lloyd. Facebook's Tectonic Filesystem: Efficiency from Exascale. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 217–231. USENIX Association, February 2021.

[69] Christian Pinto, Yiannis Gkoufas, Andrea Reale, Seetharami Seelam, and Steven Eliuk. Hoard: A Distributed Data Caching System to Accelerate Deep Learning Training on the Cloud. *arXiv preprint arXiv:1812.00669*, 2018.

[70] Giuseppe Raffa, Jorge Blasco Alis, Dan O'Keeffe, and Santanu Kumar Dash. AWSomePy: A Dataset and Characterization of Serverless Applications. In *Proceedings of the 1st Workshop on SErverless Systems, Applications and MEthodologies*, SESAME '23, page 50–56, New York, NY, USA, 2023. Association for Computing Machinery.

[71] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, New York, NY, USA, 2012. Association for Computing Machinery.

[72] Yujie Ren, Changwoo Min, and Sudarsun Kannan. CrossFS: A Cross-layered Direct-Access File System. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 137–154. USENIX Association, November 2020.

[73] K Andrew R Richards and Michael A Hemphill. A Practical Guide to Collaborative Qualitative Data Analysis. *Journal of Teaching in Physical education*, 37(2):225–231, 2018.

[74] Francisco Romero, Gohar Irfan Chaudhry, Íñigo Goiri, Pragna Gopa, Paul Batum, Neeraja J Yadwadkar, Rodrigo Fonseca, Christos Kozyrakis, and Ricardo Bianchini. Faa$T: A Transparent Auto-Scaling Cache for Serverless Applications. In *Proceedings of the ACM symposium on cloud computing*, pages 122–137, 2021.

[75] Drew Roselli, Jacob R Lorch, and Thomas E Anderson. A Comparison of File System Workloads. In *2000 USENIX Annual Technical Conference (USENIX ATC 00)*, 2000.

[76] M. Satyanarayanan. A Study of File Sizes and Functional Lifetimes. *SIGOPS Oper. Syst. Rev.*, 15(5):96–108, dec 1981.

[77] Korakit Seemakhupt, Brent E. Stephens, Samira Khan, Sihang Liu, Hassan Wassel, Soheil Hassas Yeganeh, Alex C. Snoeren, Arvind Krishnamurthy, David E. Culler, and Henry M. Levy. A Cloud-Scale Characterization of Remote Procedure Calls. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 498–514, New York, NY, USA, 2023. Association for Computing Machinery.

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

[79] Muthian Sivathanu, Midhul Vuppalapati, Bhargav S Gulavani, Kaushik Rajan, Jyoti Leeka, Jayashree Mohan, and Piyus Kedia. INSTalytics: Cluster Filesystem Co-design for Big-data Analytics. *ACM Transactions on Storage (TOS)*, 15(4):1–30, 2020.

[80] StackOverflow. Stack Overflow Data. https://archive.org/download/stackexchange, 2024. [Accessed 21-01-2025].

[81] Lilia Tang, Chaitanya Bhandari, Yongle Zhang, Anna Karanika, Shuyang Ji, Indranil Gupta, and Tianyin Xu. Fail through the Cracks: Cross-System Interaction Failures in Modern Cloud Systems. In *Proceedings of the Eighteenth European Conference on Computer Systems*, EuroSys '23, page 433–451, New York, NY, USA, 2023. Association for Computing Machinery.

[82] Dmitrii Ustiugov, Plamen Petrov, Marios Kogias, Edouard Bugnion, and Boris Grot. Benchmarking, Analysis, and Optimization of Serverless Function Snapshots. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 559–572, 2021.

[83] Tarasov Vasily. Filebench. https://github.com/filebench/filebench, 2011. [Accessed 10-09-2024].

[84] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. Amazon Aurora: Design Considerations for High Throughput Cloud-native Relational Databases. In *SIGMOD 2017*, 2017.

[85] Werner Vogels. File system usage in Windows NT 4.0. *SIGOPS Oper. Syst. Rev.*, 33(5):93–109, dec 1999.

[86] Sage A Weil. Ceph: Reliable, Scalable, and High-performance Distributed Storage. 2007.

[87] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale heterogeneous GPU clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 945–960, Renton, WA, April 2022. USENIX Association.

[88] David Wicks. The Coding Manual For Qualitative Researchers. *Qualitative research in organizations and management: an international journal*, 12(2):169–170, 2017.

[89] Pengtao Xie, Jin Kyu Kim, Qirong Ho, Yaoliang Yu, and Eric Xing. Orpheus: Efficient Distributed Machine Learning via System and Algorithm Co-design. In *Proceedings of the ACM Symposium on Cloud Computing*, SoCC '18, page 1–13, New York, NY, USA, 2018. Association for Computing Machinery.

[90] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. In *2nd USENIX workshop on hot topics in cloud computing (HotCloud 10)*, 2010.

[91] Tian Zhang, Dong Xie, Feifei Li, and Ryan Stutsman. Narrowing the Gap between Serverless and its State with Storage Functions. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 1–12, 2019.

[92] Weidong Zhang, Erci Xu, Qiuping Wang, Xiaolu Zhang, Yuesheng Gu, Zhenwei Lu, Tao Ouyang, Guanqun Dai, Wenwen Peng, Zhe Xu, Shuo Zhang, Dong Wu, Yilei Peng, Tianyun Wang, Haoran Zhang, Jiasheng Wang, Wenyuan Yan, Yuanyuan Dong, Wenhui Yao, Zhongjie Wu, Lingjun Zhu, Chao Shi, Yinhu Wang, Rong Liu, Junping Wu, Jiaji Zhu, and Jiesheng Wu. What's the Story in EBS Glory: Evolutions and Lessons in Building Cloud Block Store. In *22nd USENIX Conference on File and Storage Technologies (FAST 24)*, pages 277–291, Santa Clara, CA, February 2024. USENIX Association.

[93] Mark Zhao, Satadru Pan, Niket Agarwal, Zhaoduo Wen, David Xu, Anand Natarajan, Pavan Kumar, Shiva Shankar P, Ritesh Tijoriwala, Karan Asher, Hao Wu, Aarti Basant, Daniel Ford, Delia David, Nezih Yigitbasi, Pratap Singh, Carole-Jean Wu, and Christos Kozyrakis. Tectonic-Shift: A Composite Storage Fabric for Large-Scale ML Training. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 433–449, Boston, MA, July 2023. USENIX Association.

A Artifact Appendix

Abstract

Cloudscape is a dataset of 396 publicly described cloud architectures, capturing the usage and interaction of services in each architecture.

Scope

The artifact includes a Jupyter notebook that reproduces the claims of this paper and is useful for understanding how to query the dataset. We hope the dataset will facilitate future studies on the composition of cloud architectures and other services not covered in our paper.

Contents

The dataset is a collection of GraphML files. Each architecture is encoded in the GraphML format as a MultiDiGraph. The nodes of the graph correspond to services, and edges indicate service interaction. The artifact package contains a `README.md` file which describes the additional attributes associated with each graph, node, and edge.

Apart from a Jupyter notebook that replicates results presented in this paper, the artifact includes an interactive web-based explorer for the architecture diagrams.

Hosting

The artifact is available at https://github.com/wiscADSL/Cloudscape on the `master` branch (`#572a151`).