Optimize your storage performance with Amazon S3, ft. Lyft

Tim Harris  
Principal Engineer  
Amazon S3  
Amazon Web Services

Matt Sidley  
Sr. Product Manager  
Amazon S3  
Amazon Web Services

Darek Grala  
Software Engineer  
Lyft Level 5

Giacomo Dabisias  
Software Engineer  
Lyft Level 5
**Scale**

**Netflix**
- Billions of hours of content delivered from S3.
- S3 serves as the data lake for their big data analytics solution.

**Fannie Mae**
- S3 storage supports mortgage processing in the US.
- 250,000 loans processed per day.

**Finra**
- Serves as an investigative body to ensure stable financial markets.
- Enables analytics on multi-petabyte datasets.
- 37 billion records captured, analyzed, and stored daily.

**Zillow**
- Image hosting and distribution is based on S3 and EC2.
- ~100 TB of storage used.
- >1 billion objects durably stored.

**Airbnb**
- Uses S3 for static content and backup.
- 10+ PB of user pictures
- Continually innovating new ways to analyze data stored on Amazon S3.

**Major League Baseball**
- MLB Advanced Media’s player tracking system is based on S3 and EC2.
- 7TB per Game x 2,430 Games
- ~17PB per MLB Season
## Amazon S3 infrastructure at a glance

<table>
<thead>
<tr>
<th>Scale</th>
<th>Reliability</th>
<th>Durability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exabytes stored across many millions of drives</td>
<td>trillions of objects</td>
<td>millions of requests per second</td>
</tr>
<tr>
<td>Our index contains trillions of objects</td>
<td></td>
<td></td>
</tr>
<tr>
<td>We regularly peak at millions of requests per second</td>
<td></td>
<td></td>
</tr>
<tr>
<td>In a single region, we process peaks of over 60 tbps in a day</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Amazon S3 infrastructure at a glance

<table>
<thead>
<tr>
<th>Scale</th>
<th>Reliability</th>
<th>Durability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>99.9% availability SLA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Operational performance is second only to security</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Regions designed to lose an entire data center</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AWS Regions are isolated to limit blast radius</td>
<td></td>
</tr>
</tbody>
</table>
# Amazon S3 infrastructure at a glance

<table>
<thead>
<tr>
<th>Scale</th>
<th>Reliability</th>
<th>Durability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Designed to provide <strong>99.999999999% durability</strong> of objects over a given year</td>
<td>Designed to sustain data in the event of an <strong>entire Amazon S3 Availability Zone loss</strong></td>
</tr>
</tbody>
</table>
Agenda

Architecting to scale Amazon S3 request rates

Lyft Level 5

Optimizing for throughput from Amazon S3

Achieving more predictable outlier performance

Serving distributed clients

Putting it all together
Architecting to scale Amazon S3 request rates
Scaling request rates on Amazon S3

Autonomous-driving use case

1. Create data
Cars drive around, collecting data

2. Connect to AWS
Cars pull into the garage, connect to the network

3. Upload into Amazon S3
Upload drive-time data into Amazon S3 data lake for processing

4. Repeat daily
Scaling request rates on Amazon S3

Autonomous-driving use case

The bucket name: awsexamplebucket/daily-uploads/<date>/<car>/drive-data

The object key name:

Prefixes

Name
Scaling request rates on Amazon S3

Autonomous-driving use case

/daily-uploads/<date>/<car>/drive-data
/daily-uploads/20191007/CAR01/drive-data
/daily-uploads/20191007/CAR02/drive-data
/daily-uploads/20191007/CAR03/drive-data
/daily-uploads/20191007/CAR04/drive-data
/daily-uploads/20191007/CAR05/drive-data

Goal: 3,000 PUTs/sec per car – 15,000 PUTs/sec
Scaling request rates on Amazon S3

Autonomous-driving use case

All cars get throttled around 3,500 PUTs/sec (total)

New index prefixes are created, raising max TPS

Scaling request rates on Amazon S3

Autonomous-driving use case

All cars get throttled around 3,500 PUTs/sec (total)

New index prefixes are created, raising max TPS

Scaling request rates on Amazon S3

Autonomous-driving use case

All cars get throttled around 3,500 PUTs/sec (total)

New index prefixes are created, raising max TPS
Scaling request rates on Amazon S3

Autonomous-driving use case

/daily-uploads/<date>/<car>/drive-data
/daily-uploads/20191007/CAR01/drive-data
/daily-uploads/20191007/CAR02/drive-data
/daily-uploads/20191007/CAR03/drive-data
/daily-uploads/20191007/CAR04/drive-data
/daily-uploads/20191007/CAR05/drive-data

Each car encounters throttling until five prefixes are created.
Scaling request rates on Amazon S3

Autonomous-driving use case

Amazon S3 identified five prefixes. Now we’re good, right?
Scaling request rates on Amazon S3

Autonomous-driving use case

/daily-uploads/<date>/<car>/drive-data
/daily-uploads/20191008/CAR01/drive-data
/daily-uploads/20191008/CAR02/drive-data
/daily-uploads/20191008/CAR03/drive-data
/daily-uploads/20191008/CAR04/drive-data
/daily-uploads/20191008/CAR05/drive-data

Well, until tomorrow anyway
Scaling request rates on Amazon S3

Autonomous-driving use case

/daily-uploads/<car>/<date>/drive-data
/daily-uploads/CAR01/20191008/drive-data
/daily-uploads/CAR02/20191008/drive-data
/daily-uploads/CAR03/20191008/drive-data
/daily-uploads/CAR04/20191008/drive-data
/daily-uploads/CAR05/20191008/drive-data

By moving cardinality forward, we prefix once
Request rates are allocated proportionally

**Example 1**

**Workload**

50% PUT and 50% GET operations in single prefix

**Total TPS**

\[
(50\% \times 3,500) + (50\% \times 5,500) = 1,750 + 2,750 = 4,500 \text{ TPS combined}
\]
Request rates are allocated proportionally

**Example 1**

<table>
<thead>
<tr>
<th>Workload</th>
<th>Total TPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>50% PUT and 50% GET operations in single prefix</td>
<td>(50% * 3,500) + (50% * 5,500) = 1,750 + 2,750 = 4,500 TPS combined</td>
</tr>
</tbody>
</table>

**Example 2**

<table>
<thead>
<tr>
<th>Workload</th>
<th>Total TPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>30% PUT and 70% GET operations in single prefix</td>
<td>(30% * 3,500) + (70% * 5,500) = 1,050 + 3,850 = 4,900 TPS combined</td>
</tr>
</tbody>
</table>
Key takeaways for request rates

Request rates start with the naming scheme

You start with high request rates (3,500 PUTs | 5,500 GETs)

Amazon S3 automatically creates prefixes under a sustained load

The total TPS for a prefix is a proportional
Lyft Level 5

Darek Grala
Software Engineer

Giacomo Dabisias
Software Engineer
Lyft’s self-driving strategy

**Democratize** and **accelerate** autonomous vehicle development by building the world’s leading self-driving ecosystem.

- **Open Platform** (Third-party)
- **Lyft Level 5** (First-party)

**Partners’ Self-Driving Systems + Vehicles**

**Lyft Self-Driving System**

Copyright © 2019 by Lyft
Lyft’s in-house AV development: Level 5

Within 18 months our teams launched an employee program in Palo Alto

- **400+** top engineers and technical leads from leading tech and AV companies
- **Global offices** in Palo Alto, Munich, and London
- **AV Pilot Program**: Employees can request rides live in their Lyft app
Summary

Challenge
- Amazon S3 production vs research trade-offs
- Scalability/rapid growth

Topics
- Random keys vs. file-system-like structure
- Deterministic prefix partitioning
Lyftbag ingestion

AV missions → Upload hardware → Amazon S3

Elasticsearch ← Amazon DynamoDB ← Ingest service ← Amazon SQS
### Amazon S3 structure

<table>
<thead>
<tr>
<th>Name</th>
<th>Last modified</th>
<th>Size</th>
<th>Storage class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000003f1c66cfedc1a2d174930e1e85_20180809T001940Z.lyftbag</td>
<td>Aug 8, 2018 6:54:29 PM GMT-0700</td>
<td>145.8 MB</td>
<td>Standard</td>
</tr>
<tr>
<td>0000017fff22f2e71bcb6d6e71ed4341_20190614T172936Z.lyftbag</td>
<td>Jun 14, 2019 11:59:45 AM GMT-0700</td>
<td>146.1 MB</td>
<td>Standard</td>
</tr>
<tr>
<td>000001b13ca3aa6d18bb1c9766a3ba61_20190709T144248Z.lyftbag</td>
<td>Jul 10, 2019 2:20:27 PM GMT-0700</td>
<td>146.2 MB</td>
<td>Standard</td>
</tr>
<tr>
<td>000002a333ef9fa315339cadcb96ea12_20191028T214536Z.lyftbag</td>
<td>Oct 28, 2019 4:56:17 PM GMT-0700</td>
<td>193.5 MB</td>
<td>Standard</td>
</tr>
<tr>
<td>000002a47875b13359f7a71c637c2c56_20190422T183214Z.lyftbag</td>
<td>Apr 22, 2019 1:09:10 PM GMT-0700</td>
<td>151.9 MB</td>
<td>Standard</td>
</tr>
<tr>
<td>000003c3f021281dca4bc94c1d5aa145_20191104T191118Z.lyftbag</td>
<td>Nov 4, 2019 5:48:09 PM GMT-0800</td>
<td>218.0 MB</td>
<td>Standard</td>
</tr>
</tbody>
</table>

**Flat structure**
Challenges

- **Backfills**
  - Months of data ingested in hours
  - *8,000 workers* → hit DynamoDB and Amazon S3 limits

- **Metadata TTL**
  - No metadata → hard to recover
  - **List** the bucket and post-filter relevant files
  - Data gaps after restoration

- **Elasticsearch scaling limits**
  - *25B+ documents* in the index
Optimize for the most common use cases

Three relevant fields: car ID, time range, sensors

```
s3://bucket/<minute>/<hour>/<day>/<month>/<year>/<car_id>/<type>/<md5>
```

Solution: List relevant files
Takeaways

High cardinality prefix

s3://bucket/<minute>/<hour>/<day>/<month>/<year>/<car_id>/<type>/<md5>
Research @ Level 5 London

Giacomo Dabisias
Software Engineer
Building visual maps
Building visual maps

Data collection

Distributed map building
Building visual maps

Data collection

Amazon S3

AWS Batch

Distributed map building

Amazon S3
Amazon S3: Research vs. production

/level-5/research/map1/
  images/20191008/image.jpeg
  feature/20191008/features.tar
  pointclouds/20191008/pointcloud.ply
  ...

Poor performance
Amazon S3: Optimizations

Use parallel calls whenever possible
Amazon S3: Optimizations

- Use retries with random exponential backoff
- Use parallel calls whenever possible
Amazon S3: Optimizations

- Use parallel calls whenever possible
- Use retries with random exponential backoff
- Avoid listing keys whenever possible and if needed create your own Amazon S3 index on Amazon S3
Amazon S3: Optimizations

- Use parallel calls whenever possible
- Use retries with random exponential backoff
- Avoid listing keys whenever possible and if needed create your own Amazon S3 index on Amazon S3
- Use a key-value abstraction to test different storage services
Amazon S3: Current performance

- Input data bucket size: **1 PB => 10 billion keys**

- Up to **100K cores** on AWS Batch hitting Amazon S3 for get/put operations

- Average map build:
  - **1.5 days** of computation
  - Put requests: **1M**
  - Get requests: **500M**
  - 5XX errors: **100**

Good performance
Amazon S3: Work in progress

/level-5/research/map1/
  images/20191008/image.jpeg
  feature/20191008/features.tar

pointclouds/20191008/pointcloud.ply
...

Amazon S3
Amazon S3: Work in progress

/level-5/research/map1/
  images/20191008/image.jpeg
  feature/20191008/features.tar

pointclouds/20191008/pointcloud.ply
...

Two-way hash

Amazon S3
Q: How do we choose the right Amazon S3 key pattern to follow?
Optimizing for throughput from Amazon S3
Compute and storage in the same region
Thinking about Amazon S3

- Trillions of objects and millions of requests per second
- Simple GET/PUT interface to your objects
- Distributed service with multiple endpoints
Scale connections horizontally

Downloading large objects: one concurrent request for each 85–90 MB/s of desired network throughput
Deep dive – IP diversity
Deep dive – IP diversity
Deep dive – IP diversity

awsexamplebucket.s3.amazonaws.com

52.217.36.164
Deep dive – IP diversity

awsexamplebucket.s3.amazonaws.com

52.217.36.164

GET /Key
Host: awsexamplebucket.s3.amazonaws.com
...

53
Deep dive – IP diversity

```
$ dig awsexamplebucket.s3.amazonaws.com A

; <<>> DiG 9.10.6 <<>> awsexamplebucket.s3.amazonaws.com A
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 36672
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; ANSWER SECTION:
s3-1-w.amazonaws.com. 48 IN CNAME s3-w.us-east-1.amazonaws.com.
s3-w.us-east-1.amazonaws.com. 5 IN A 52.217.36.164
```

5s TTL for caching DNS resolvers
Deep dive – IP diversity

52.217.36.164
52.217.0.220
52.216.99.147
52.216.81.208
...

...
Reuse TCP connections

- TCP handshake
- SSL handshake
- GET /Key

Amortize connection setup over multiple requests
Avoid repeating TCP slow-start on each request
Use parallel byte-range fetches

Use granularities 8–16 MB as a starting point
Parallel fetches within a single object can be used
Align with multipart boundaries if possible
Spread load over the key namespace

Distributes the load across the prefixes under awsexamplebucket/input

Strategy: Distribute objects or batches of objects between workers
Achieving more predictable outlier performance
When to retry – Slow or stalled requests

Fixed byte-range request size promotes stable response time
When to retry – Slow or stalled requests

Define a timeout, say slowest 1% of requests

Fixed byte-range request size promotes stable response time
When to retry – Slow or stalled requests

- Slow or stalled requests

Define a timeout, say slowest 1% of requests

Fixed byte-range request size promotes stable response time

Retry a request that exceeds the timeout
When to retry – Slow or stalled requests

- Slow or stalled requests

Define a timeout, say slowest 1% of requests

Fixed byte-range request size promotes stable response time

End-to-end response time

Request #

Retry a request that exceeds the timeout

Rate limit retries (~1% of requests)
When to retry – Slow or stalled requests

Define a timeout, say slowest 1% of requests

Fixed byte-range request size promotes stable response time

Retry a request that exceeds the timeout

Rate limit retries (~1% of requests)

Use a fresh connection
When to retry – Slow or stalled requests

- Slow or stalled requests

- Fixed byte-range request size promotes stable response time

- Define a timeout, say slowest 1% of requests

- Retry a request that exceeds the timeout

- Rate limit retries (~1% of requests)

- Use a fresh connection

- Ideally, use a fresh DNS lookup
When to retry – 503

Indicates high load on a prefix

Amazon S3 scales up automatically in response to load, introducing more prefixes

Assess key namespace design if you continue to see 503 responses
When to retry on errors

A very small percentage of 5XX errors are excepted during normal operation.

All 5XX errors can and should be retried, subject to rate limiting.

SDKs include exponential backoff algorithms, often with jitter.
Serving distributed clients
Amazon S3 replication automatically copies your data to the same or different AWS Region.
Amazon S3 Replication Time Control NEW!

Designed to replicate 99.99% of objects within 15 minutes

15-minute replication time backed by an AWS Service Level Agreement (SLA)

Monitor replication using Amazon CloudWatch metrics and event notifications
Amazon S3 Replication Time Control NEW!

Designed to replicate 99.99% of objects within 15 minutes

Monitor your replication with three new CloudWatch metrics

Optional: Set up alarms on your metrics

**Replication latency**
The maximum number of seconds by which the destination region is behind the source region for a given replication rule.

**Bytes pending replication**
The total number of bytes of objects pending replication for a given replication rule.

**Operations pending replication**
The number of operations pending replication for a given replication rule.
Accessing S3 outside Amazon EC2

Amazon S3 Transfer Acceleration

Amazon CloudFront

AWS Direct Connect
Use Amazon S3 Transfer Acceleration to minimize latency caused by distance

- Change your endpoint, not your code
- Leverages Amazon CloudFront global edge locations
- Optimized protocols
- No firewall exceptions
- No client software required
Using caching for frequently accessed content

If your workload is sending repeated GET requests for a common set of objects, you can use a cache to optimize performance.

Amazon CloudFront
Amazon ElastiCache
AWS Elemental MediaStore
How caching works in CloudFront

1. Request is routed to most optimal edge location
2. Non-cached content is retrieved from the origin
3. Origin content is transferred to CloudFront edge location for caching
4. Data is transferred to end user
Amazon ElastiCache

• Stores data in-memory, so it provides sub-millisecond latency and supports incredibly high requests per second

• Can be implemented as an application side cache
AWS Direct Connect

- Reduces your bandwidth costs
- Consistent network performance
- Compatible with all AWS services
- Private connectivity with your Amazon VPC
- Simple and elastic
Putting it all together
Putting it all together
Putting it all together

Measure and iterate
Putting it all together

Measure and iterate

Design key space to scale
Putting it all together

- Measure and iterate
- Design key space to scale
- Use horizontal scaling for throughput
Putting it all together

Measure and iterate

Design key space to scale

Use horizontal scaling for throughput

Monitor and retry stragglers
Putting it all together

- Measure and iterate
- Design key space to scale
- Use horizontal scaling for throughput
- Monitor and retry stragglers
- Use the latest SDKs
Learn storage with AWS Training and Certification

Resources created by the experts at AWS to help you build cloud storage skills

45+ free digital courses cover topics related to cloud storage, including:

- Amazon S3
- AWS Storage Gateway
- Amazon S3 Glacier

- Amazon Elastic File System (Amazon EFS)
- Amazon Elastic Block Store (Amazon EBS)

Classroom offerings, like Architecting on AWS, feature AWS expert instructors and hands-on activities

Visit aws.amazon.com/training/path-storage/
Thank you!

Tim Harris
tlh@amazon.com

Matt Sidley
sidley@amazon.com
Please complete the session survey in the mobile app.