Ambry: LinkedIn’s Scalable Geo-Distributed Object Store

Shadi A. Noghabi*, Sriram Subramanian+, Priyesh Narayanan+, Sivabalan Narayanan+, Gopalakrishna Holla+, Mammad Zadeh+, Tianwei Li+, Indranil Gupta*, Roy H. Campbell*

+ LinkedIn Corporation
  Data Infrastructure team
* University of Illinois at Urbana-Champaign

SRG: http://srg.cs.illinois.edu and DPRG: http://dprg.cs.uiuc.edu
Massive Media Objects are Everywhere

100s of Millions of users

100s of TBs to PBs

From all around the world
Media has a unique access pattern

- Uploaded **once**
- Rarely deleted
- Never modified
- *Immutable*

Frequently accessed from all around the world

*Read-heavy*
Skewed toward recent data
How to handle all these objects?

We need a *geographically distributed* system that stores and retrieves these *read-heavy immutable objects* in an *efficient* and *scalable* manner.

**Ambry**

In production for >2 years and >400 M users!

Open source: [https://github.com/linkedin/ambry](https://github.com/linkedin/ambry)
Challenges

**Wide diversity**

- 10s of KBs to few GBs

**Fast, durable, and highly available processing**

- Geo-replication with low latency

**Evergrowing data and requests**

- Data rarely gets deleted
- > 800 M req/day (~120 TB)
- Rate doubled in 12 months

**Load imbalance**

- Workload variability
- Cluster expansions
Related Work

• **Distributed file system**: NFS, HDFS, Ceph, etc.
  – High metadata overhead
  – Unnecessary additional capabilities

• **Key value stores**: Cassandra, Dynamo, Bigtable, etc.
  – Not designed for very large objects

• **Blob stores**: Haystack, f4, Twitter’s blob store
  – Not resolving load imbalance
  – Not designed specifically for geo-distributed design
Design Goals

Wide diversity
- 10s of KBs to few GBs

Low latency and high throughput for all objects

Evergrowing data and requests
- Data rarely gets deleted
- > 800 M req/day (~120 TB)
- Rate doubled in 12 months
- Scalable

Fast, durable, and highly available processing
- Geo-distributed
- Geo-replication with low latency

Load imbalance
- Workload variability
- Cluster expansions
- Load balance
Ambry’s Design

<table>
<thead>
<tr>
<th>Low latency and high throughput</th>
<th>Scalable</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Logical grouping of objects (partition)</td>
<td>• Decentralized design</td>
</tr>
<tr>
<td>• Segmented Indexing</td>
<td>• Independent components with little interaction</td>
</tr>
<tr>
<td>• Exploiting OS caches</td>
<td>• Separation of logical and physical placement</td>
</tr>
<tr>
<td>• Zero cost failure detection</td>
<td>• Chunking and parallel read/write</td>
</tr>
<tr>
<td>• Chunking and parallel read/write</td>
<td></td>
</tr>
</tbody>
</table>
Outline

- Low latency and high throughput for all objects
- Scalable
- Geo-distributed
- Load balance
### High Throughput and Low Latency

**Small** objects: low metadata overhead per object

**Large** objects: sequential writes/reads of objects

#### Partition $p$

<table>
<thead>
<tr>
<th>0</th>
<th>200</th>
<th>700</th>
<th>850</th>
<th>900</th>
<th>980</th>
<th>1040</th>
<th>100 GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj id 30</td>
<td>obj id 20</td>
<td>...</td>
<td>obj id 70</td>
<td>obj id 40</td>
<td>obj id 60</td>
<td>obj id 90</td>
<td></td>
</tr>
</tbody>
</table>

- Group object together into virtual units called *partition*
- Large pre-allocated file where objects are appended to the end
- Partitions replicated on multiple nodes, in a separate procedure
High Throughput and Low Latency

Other Optimizations:

- **Segmented Indexing**: finding location of object with low latency
  - Latest segment in memory
  - Other segment on disk
    - Bloom filters per segment

- Exploiting OS caches
- Zero cost failure detection
- Zero copy reads
- Chunking
Low latency and high throughput for all objects

Geo-distributed

Scalable

Load balance
Scalable Design

Maintain state of the cluster

Cluster Manager

Frontend Layer
- Frontend Node
- Router Library

Data Layer
- Datanode
  - Disk 1
  - Disk k

Data Center 1

Receive, verify and route requests
Scalable Design

Cluster Manager

Maintain state of the cluster

Frontend Layer

Frontend Node

Router Library

Data Layer

Datanode

Disk 1

Disk k

Receive, verify and route requests

Data Center 1

Req

Receive, verify and route requests

Frontend Node
**Scalable Design**

- **Frontend Layer**
  - Frontend Node
  - Router Library

- **Data Center 1**
  - Req

- **Data Layer**
  - Datanode
    - Disk 1
    - ... Disk k

- **Cluster Manager**
  - Maintain state of the cluster

- **Actions**
  - Receive, verify and route requests
  - Store/retrieve actual data
Scalable Design

Data Center 1

Frontend Layer
- Frontend Node
- Router Library

Cluster Manager

Maintain state of the cluster

Data Layer
- Datanode
- Disk 1
- ... Disk k

Receive, verify and route requests

Store/retrieve actual data
Scalable Design

- Frontend Layer
  - Frontend Node
  - Router Library

- Cluster Manager
  - Maintain state of the cluster

- Data Center 1
  - Receive, verify, and route requests
  - Store/retrieve actual data

- Data Layer
  - Datanode
    - Disk 1, ..., Disk k
Scalable Design

- Decentralized design
- Independent components with little interaction
- Maintain state of the cluster

Frontend Layer
- Frontend Node
- Router Library

Data Center 1
- Receive, verify and route requests
- Store/retrieve actual data

Cluster Manager
- Independent components with little interaction
Outline

- Low latency and high throughput for all objects
- Geo-distributed
- Scalable
- Load balance
Geo-Distribution

• **Asynchronous writes**
  – Write synchronously only to the datacenter serving the request
  – Asynchronously replicate data to other datacenters
  – Reduce **latency**
  – Minimize **cross-DC traffic**

• Proxy request
• Active-active design
• 2-phase background replication
• Journaling
Outline

Low latency and high throughput for all objects

Geo-distributed

Scalable

Load balance
Load Balance

**Static Cluster**

We use three techniques:

- **Random placement** of objects on large partitions
- **CDN** layer above Ambry
- **Chunking**

Using these techniques load imbalance < 5 % among nodes

**Dynamic Cluster**

Cluster expansions + skewed workload (recent data) $\rightarrow$ imbalance

We built a rebalancing mechanism moving popular data around with:

- Minimum data movement
- 6-10x improvement in **request balance**
- 9-10x improvement in **disk usage balance**
EVALUATION
Evaluation - setup

• **Small cluster**
  – A beefy single Datanode
    • 24 core CPU, 64 GB of RAM, 14 1TB HDD disks, and a full-duplex 1 Gb/s Ethernet network
  – **Workload:** Read only, 50-50 read-write, and write only
    • Fix size objects in each test
    • Randomly reading objects (worse-case scenario)

• **Production cluster**
  – 3 datacenters located across the US
  – Real world traffic

• **Simulations**
  – Long periods & very large scale
Throughput

- **Large objects:**
  - All cases saturate network
  - Read-write saturates both inbound and outbound link

- **Small objects:**
  - Writes saturates network
  - Read throughput reduced due to frequent disk seeks

**Random workload worse case**
Latency

Low overhead for large objects
Ambry is mainly intended for large objects!

Small objects:
• Reads dominated by disk seek
• In 50 KB objects, > 94% latency for disk seeks.
Conclusion

**Low latency and high throughput**
- Partitions + Indexing

**Scalable**
- Decentralized design
- Independent components

**Geo-distributed**
- Asynchronous writes

**Load Balance**
- Random placement
- Load Balancing mechanism

Open source: [https://github.com/linkedin/ambry](https://github.com/linkedin/ambry)
Back up slides
Zero-cost Failure Recovery

- No extra messages leveraging request messages
- Effective, simple, and consumes very little bandwidth.
Cluster manager

- Maintains the state of the cluster

### Logical Layout

<table>
<thead>
<tr>
<th>Partition id</th>
<th>Replica placement</th>
</tr>
</thead>
</table>
| id 10        | DC 1, node 5, disk 3  
  DC 3, node 10, disk 1  
   ...          |
| id 15        | DC 1, node 5, disk 3  
  DC 1, node 2, disk 3  
   ...          |
| ...          | ...               |

### Physical Layout

<table>
<thead>
<tr>
<th>Datacenter</th>
<th>Datanode</th>
<th>Disk</th>
</tr>
</thead>
</table>
| DC 1       | node 1   | disk 1  
  disk 2  
   ...     |
| DC 1       | node 2   | disk 1  
   ...     |
| ...        | ...      | ...    |
| DC n       | node j   | ...    |
Replication Bandwidth - Production

Inter Datacenter:
- Short tail
- 95th to 5th percentile ratio 3x
  → load balanced design

Intra datacenter:
- Longer tail
- Many zero values (omitted)
- Very small value (< 1KB/s)
Latency - Variance

CDF

Latency (ms)

Write-50KB
Read-50KB
Read-Write-50KB
Write-5MB
Read-5MB
Read-Write-5MB