Google

Fast ACS: Low-Latency File-Based Ordered Message Delivery at Scale

Google LLC

fast-acs-wp@google.com

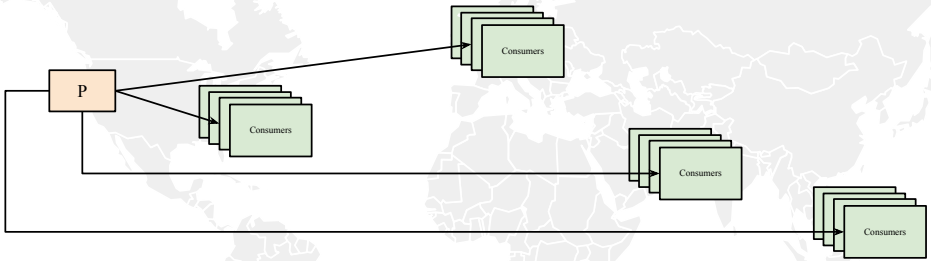
07/07/2025

Problem Statement

Deliver messages from producer to consumer(s) with these guarantees:

- **Ordered:** Messages are delivered in the exact order they were produced (total order).
- **At least once delivery:** Messages are delivered at least once.

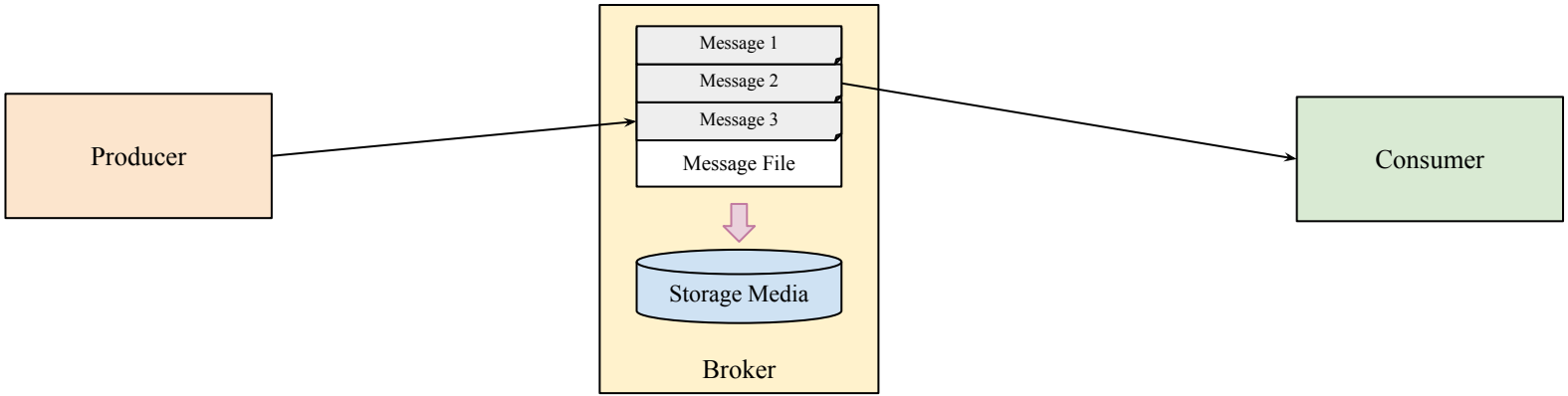
Problem Statement



Requirements:

- Support geographically distributed consumers across **dozens of clusters**.
- Support **tens of thousands of consumers per cluster**, totaling hundreds of thousands globally.
  - Consumer read traffic in **Tbps** range.
- Achieve end-to-end **sub-second tail latency**.

Problem Statement



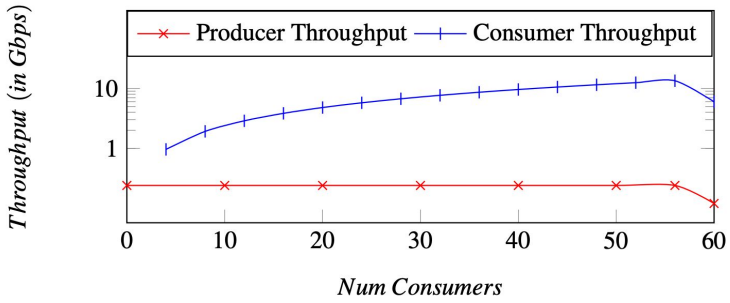
Traditional ordered message delivery (based on files):

- Producer serialize and writes messages to file(s) on storage, called **message files**.
- Consumers read message file bytes sequentially and deserialize messages.

Limitations:

- Massive **tail-reading**, cannot scale to tens of thousands of consumers.
- Storage media (e.g., SSDs) have bandwidth limits (e.g., 4.84 Gbps for SATA).

Problem Statement



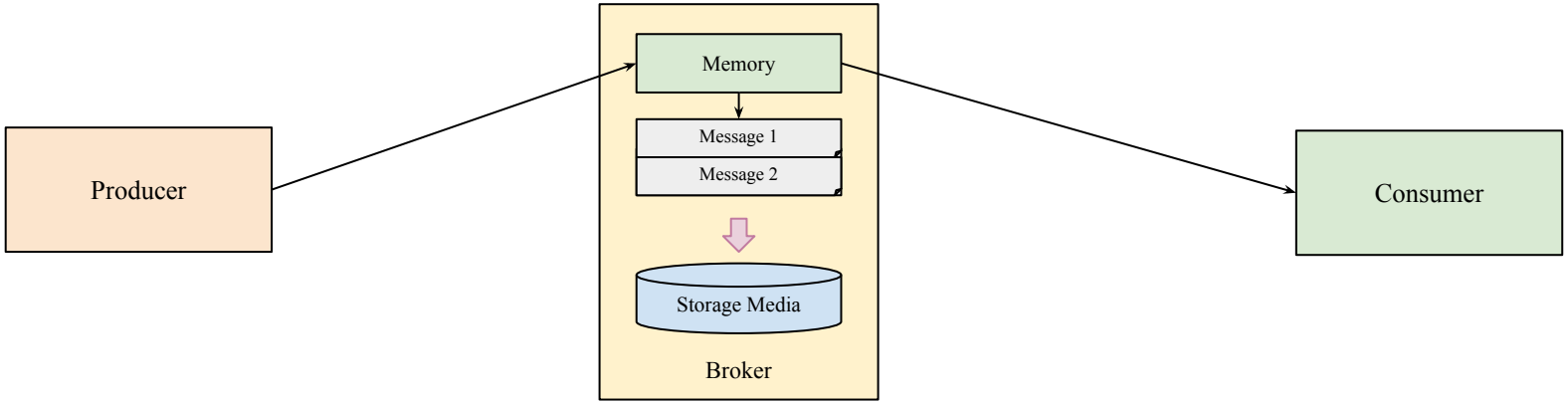
Kafka Experiment (on Google Cloud):

- 3 brokers in a zone, each configured with 8 vCPUs, 32GB of RAM, and NVMe SSD.
- 1 topic with 9 non-replicated partitions (even load distribution).
- **Producer byte rate** - 240 Mbps (across 9 partitions).

Results:

- Max supported consumers - 56.
- **Peak consumer byte rate** - 14 Gbps.

Solution 1: Retain Recent Messages In-Memory



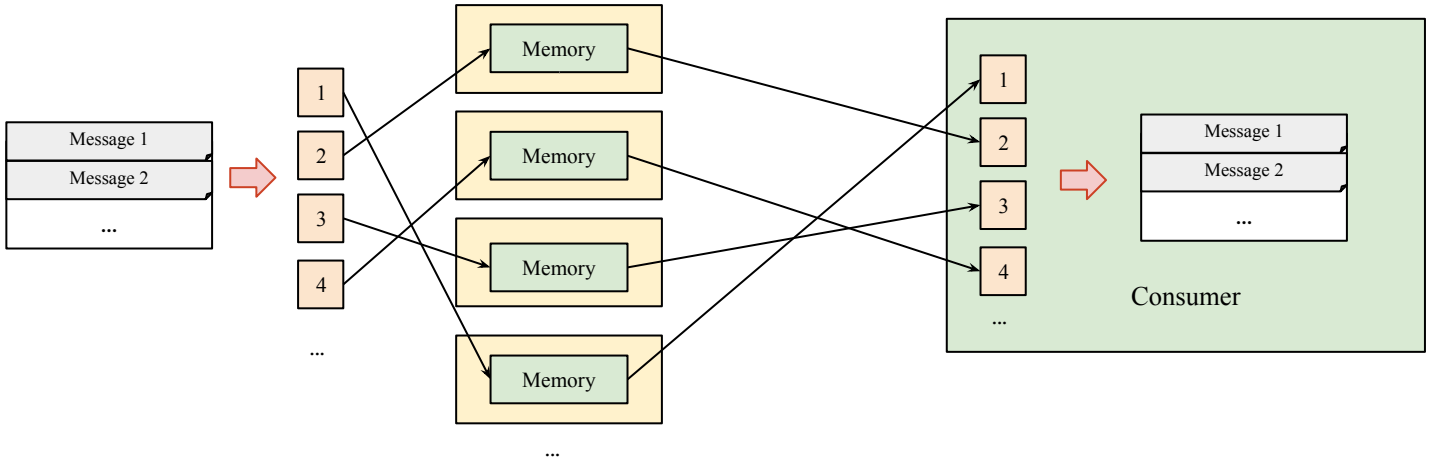
Retain recent messages in memory:

- Broker store the recent, “hot” messages in memory to avoid storage tail-reads.
- Storage media acts as the “fallback” for old messages.

Limitations:

- Massive **tail-reading**, still cannot scale to tens of thousands of consumers.
- Network Interface Cards (NICs) have transmit limits (e.g., 10 Gbps).

Solution 2: Chunking & Distributing File Bytes



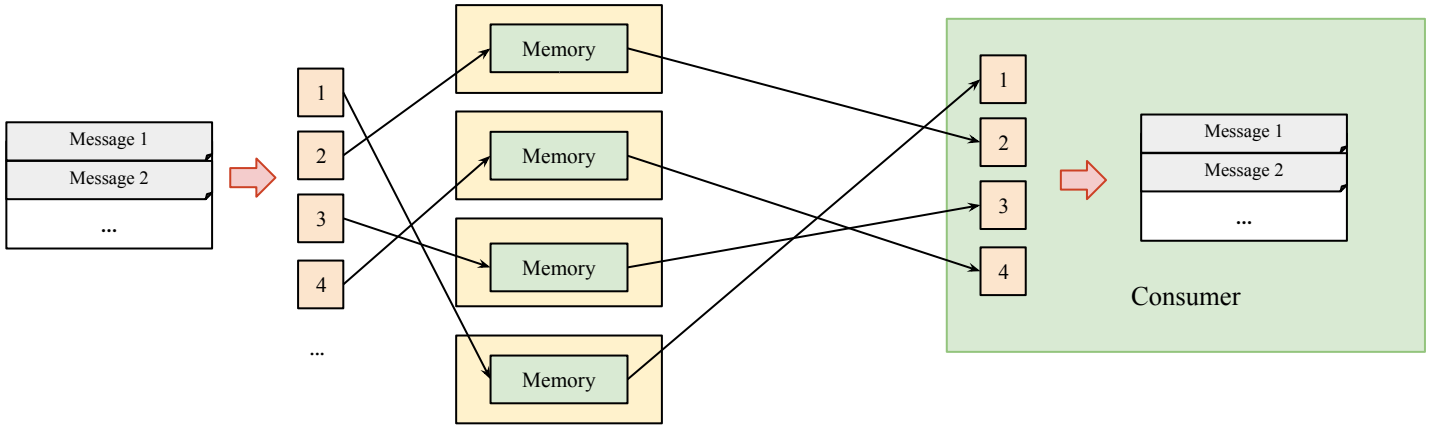
Divide message file bytes into small chunks:

- Distribute the chunks consistently across several brokers (resolves hotspotting).
- Consumers can then fetch and assemble those chunks into file bytes.

Limitations:

- Computing resources can be costly as we scale out the system horizontally.

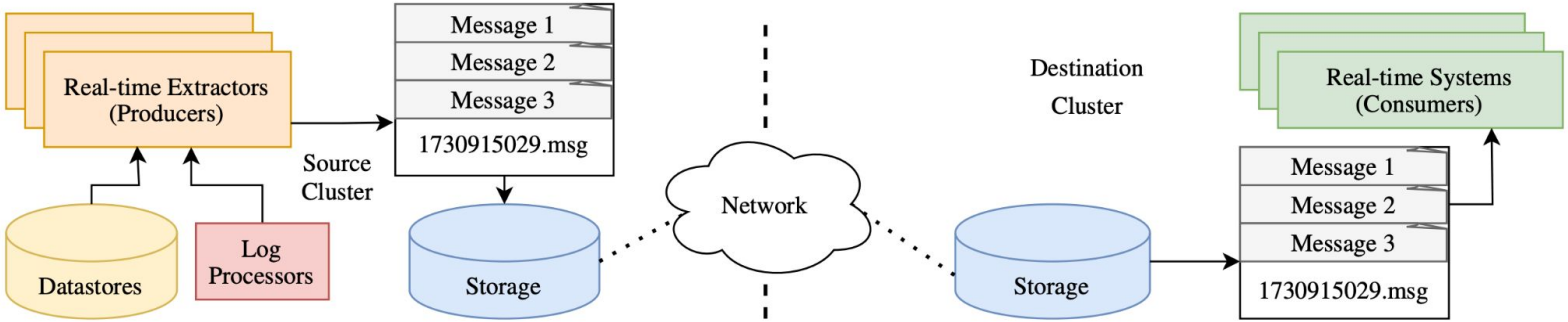
Solution 3: Remote Direct Memory Access



Consumers use RDMA to fetch chunks:

- Scaling out resources horizontally only costs little RAM overheads.
- Efficiently utilizes the power of several NICs in a cluster.

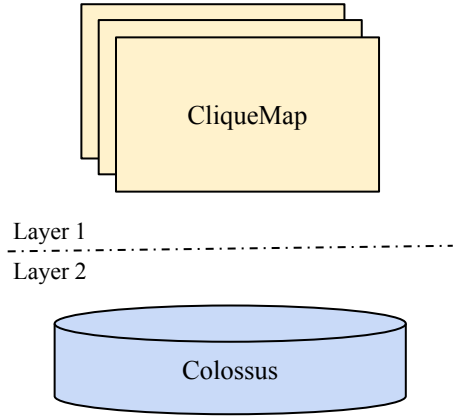
Fast ACS (Ads Copy Service)



- **Real-time extractors** (a.k.a. producers) generate **message streams**.
- Message streams are sharded:
  - Messages within a shard are totally-ordered.
- Message stream shards are stored as files:
  - Files for a single message stream shard are within a single directory.
  - Files rollover upon reaching certain limits.
- Files in message stream shards are tail-copied and bytes are delivered to consumers.

Multi-Layered Storage

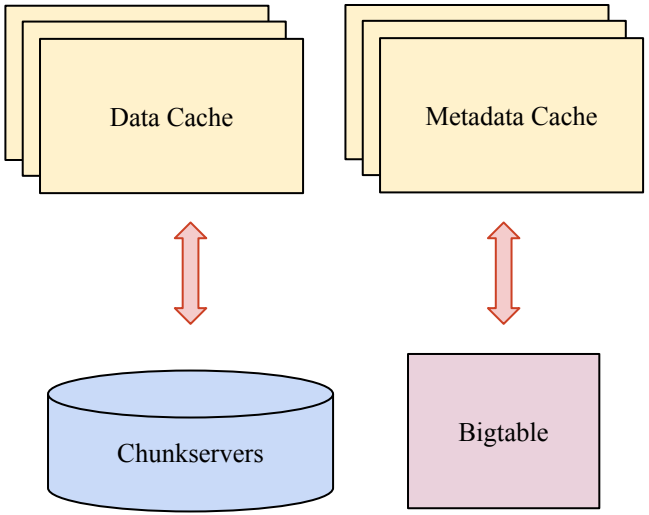
- **Colossus (L2) (Ghemawat et. al, ‘03)**
  - A distributed file system (successor of Google File System, 2003).
  - Stores message files in hierarchical directories.
    - Helps in discovery.
  - Files are append-only.
    - Acts as the source of truth.
  - All reads and writes are RPC-based.



- **CliqueMap (L1) (Akella et. al, ‘21)**
  - A RDMA-based volatile distributed key-value store (like Pilaf):
    - All writes via RPC and all reads are via RDMA.
  - Uses consistent hashing for key distribution.
  - Replicated (supports r=3.2).
  - Minimally replicates Colossus:
    - Only retains the most recent, “hot” message file chunks.

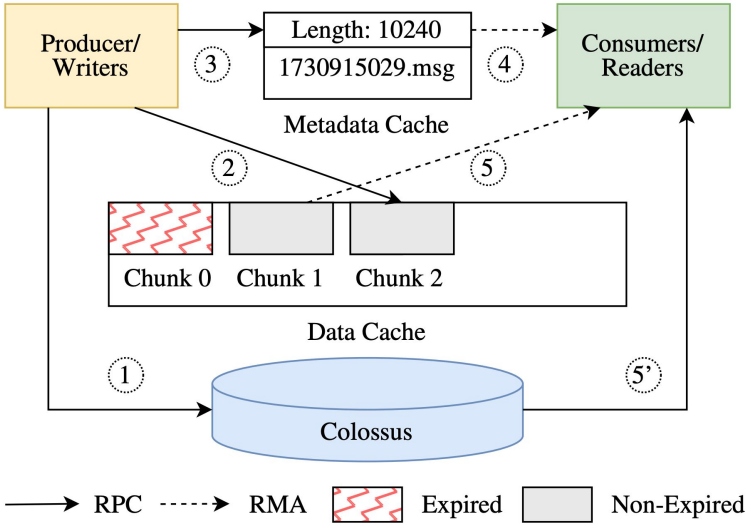
The Cache Layer

- **Data Cache (analogous to chunkservers)**
  - Hold file bytes as chunks.
    - Fixed size - **4 KB** - the MTU in cluster fabric.
  - Uses **Last-Recently Modified** policy to remove old chunks of a file.
  - Last chunk of a file may be *partially-filled*.
  - ***Chunk key = hash(file path, chunk sequence no.)***
- **Metadata Cache (analogous to Bigtable)**
  - Holds last position (i.e. **length**) for both data cache chunks and corresponding Colossus file.
  - And **file lock**:
    - For best-effort avoidance of dual writers.
- Unlike Colossus, the cache is:
  - Volatile.
  - Has no server-side processing.

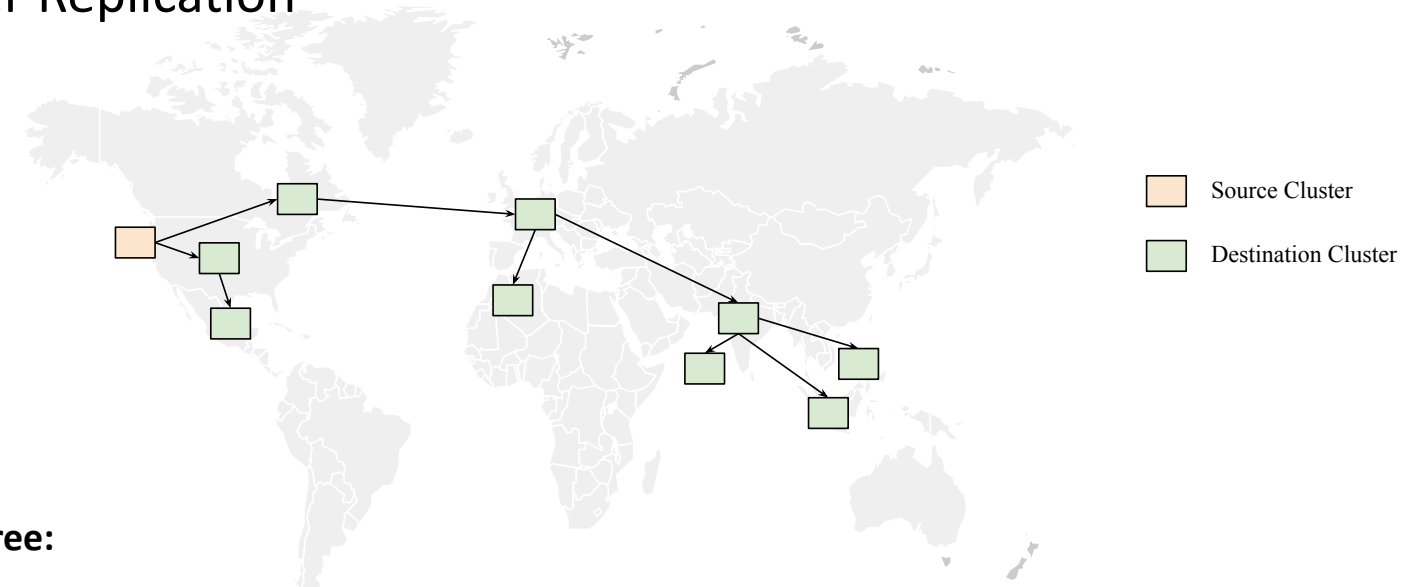


Basic Flow (No Cross Cluster Replication)

- Producers write the bytes to Colossus files **(1)**.
  - And then “shadows” the bytes as chunks to the data cache **(2)**.
  - And writes the current position of data cache chunks and Colossus file length to the metadata cache **(3)**.
- Consumers poll and read the length from the metadata cache **(4)**.
  - “Preferably” reads the file bytes from the data cache **(5)**.
  - Colossus acts as a “fallback” when caches or chunks in data cache are unavailable **(5’)**.

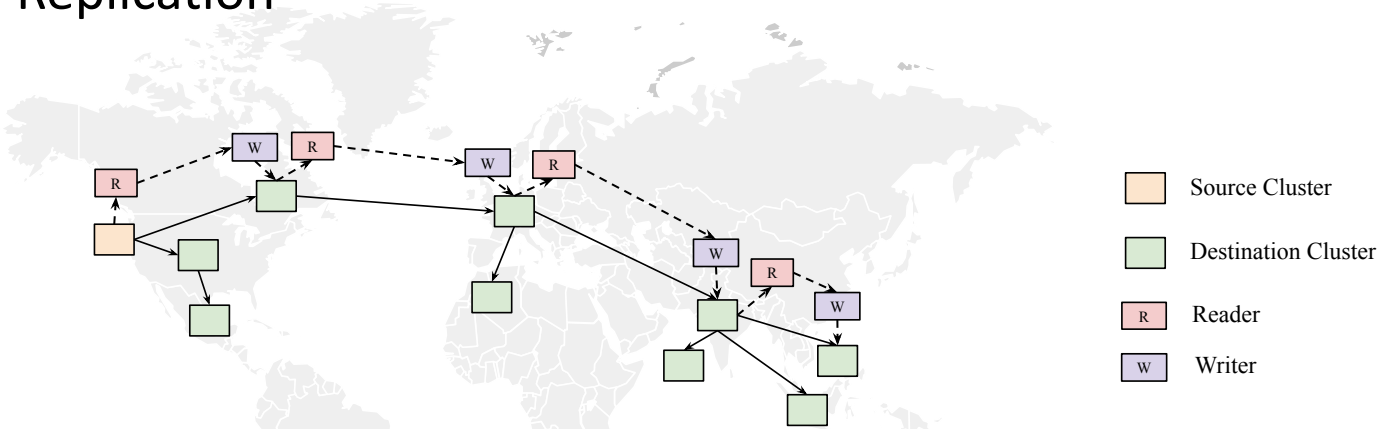


Cross Cluster Replication



- **Copy Tree:**
  - Minimum spanning tree of clusters (Prim-based):
    - Weights based on bandwidth cost.
    - Misc constraints - max depth, max fanout per upstream cluster, etc.
  - A message stream is routed along the copy tree.
- A unique copy tree per message stream.
- Typical copy tree depth - 4

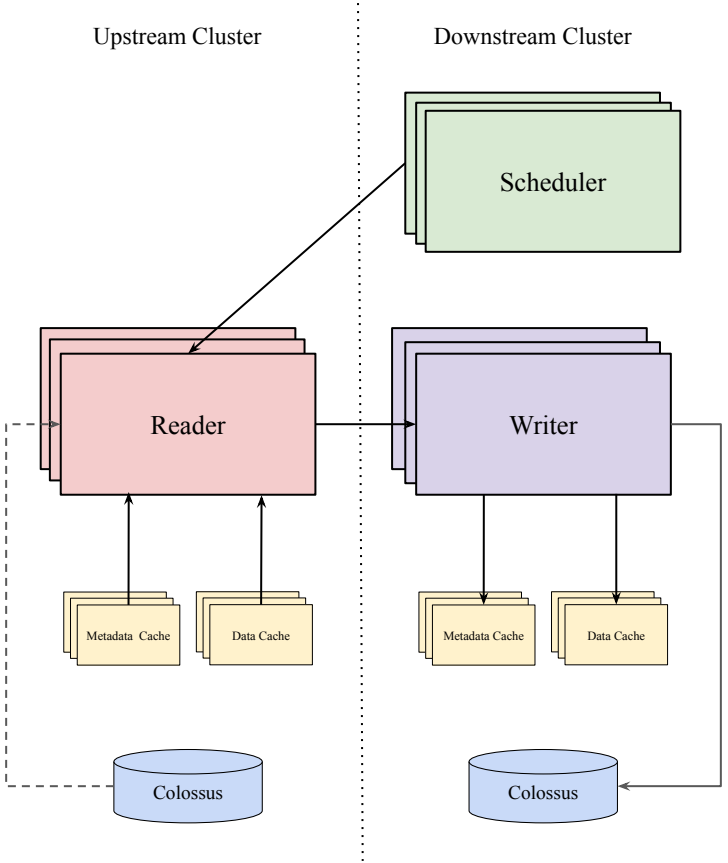
Cross Cluster Replication



- Two worker jobs along each hop:
  - **Readers**: Polls and reads bytes and stream the bytes to the next hop(s).
  - **Writers**: Writes bytes to the destination cluster storage.
- A unique **operation** pair per file - one for cache and one for Colossus.
  - Doubles bandwidth utilization.
  - Cache stream is optimized for latency and throughput (parallel, out-of-order writes, etc).
  - Colossus stream acts as a (delayed) “fallback”.

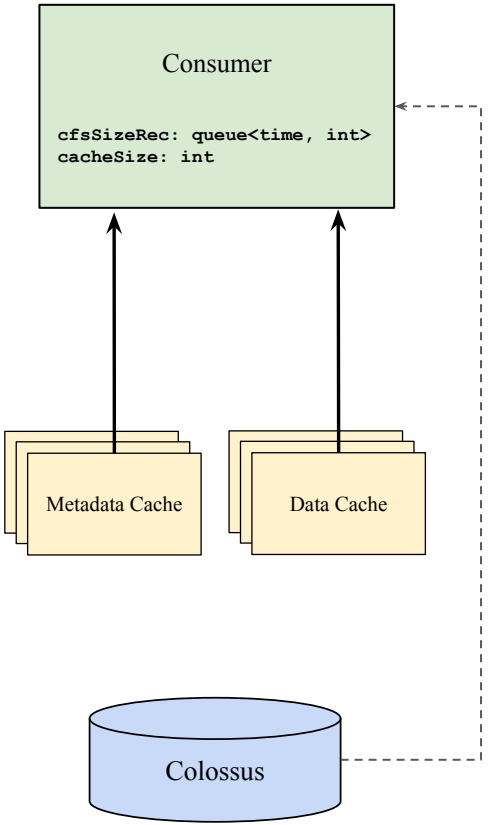
Cross Cluster Replication

- **Scheduler**
  - Schedules operations in response to incoming notifications.
  - Manages the life-cycle of an operation.
  - Scheduler's availability critical for ensuring low-latency.
  - Immediately reschedules disrupted operations (say when workers terminate).
- ***Make-before-break*** strategy
  - Existing operation can operate in **orphaned** mode when scheduler is unavailable.
  - Scheduler schedules new operations that **overtake** existing operations.
  - Simple for Colossus operations as the chunkservers implement "file locks".
  - Complex for cache operations (details in paper).



Consumers

- Prefer RDMA-based reads from cache:
  - Colossus reads burn disk-time resources.
  - SSDs infeasible due to cost reasons.
- **Delayed fallback reads**:
  - Cache operations may sometimes lag behind.
  - Reads from Colossus are delayed till message delivery delay is at risk.
    - To maximize the reads from the cache and save valuable disk-time Colossus resources.
  - Typically small - 1s.



Evaluation

- **Metrics under considerations**:
  - **Delivery Delay**: Measured from the time when producer creates a message in memory to the time consumers deserialize the message.
    - Comprehensive end-to-end measurements.
  - **Fallback**: Number of reads from Colossus.

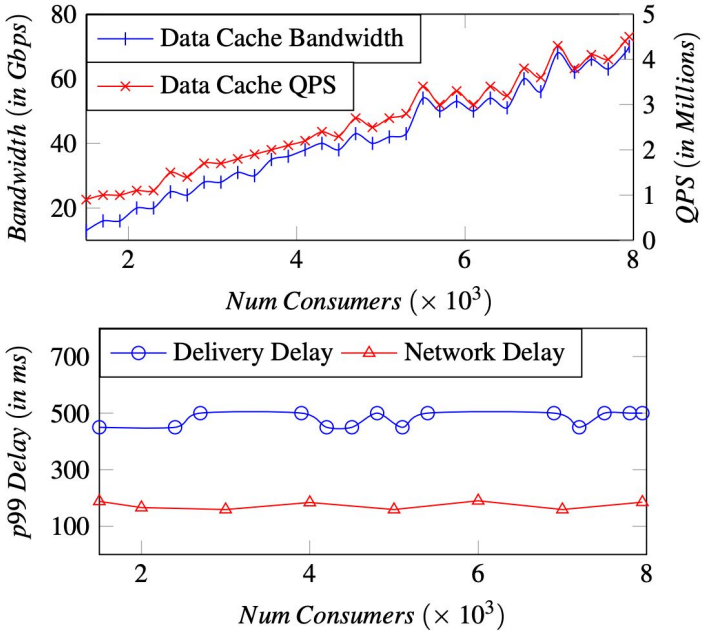
- **Setup**:
  - Shared borg:
    - Subject to same noise as production.
    - High priority CPU resources.
  - Pony Express (2019):
    - Software RDMA.
  - B4 network (cheaper, low-priority, non-user-facing traffic).
  - Producers buffered messages for 100ms to amortize storage write cost.

Experiment 1

- **Message streams:**
  - Two 120-way sharded message streams originating from us-central and euro-west region.
  - Each shard has a pair of files - **data** and **index**.
  - 240 Mbps average write rate.
  - 15 destination clusters per stream.
- **No scaling:**
  - Fixed data cache (9 replicas) and metadata cache (6 replicas) sizes.
- **Consumers:**
  - Polling at 50ms.
  - 4 shards per consumers (8 Mbps per consumer).

Experiment 1(a)

- Scaled up consumers steadily from 0 to 7,950 in a particular cluster.
  - No Colossus fallback observed.
  - Delay:
    - Delivery delay (p99) - **500ms**.
    - Producer buffering - 120ms.
    - Network delay - 180ms.
    - Consumer polling - 100ms.
  - Consumer bandwidth - **70 Gbps**.
- Other experiments - 1(b) - 1(d).

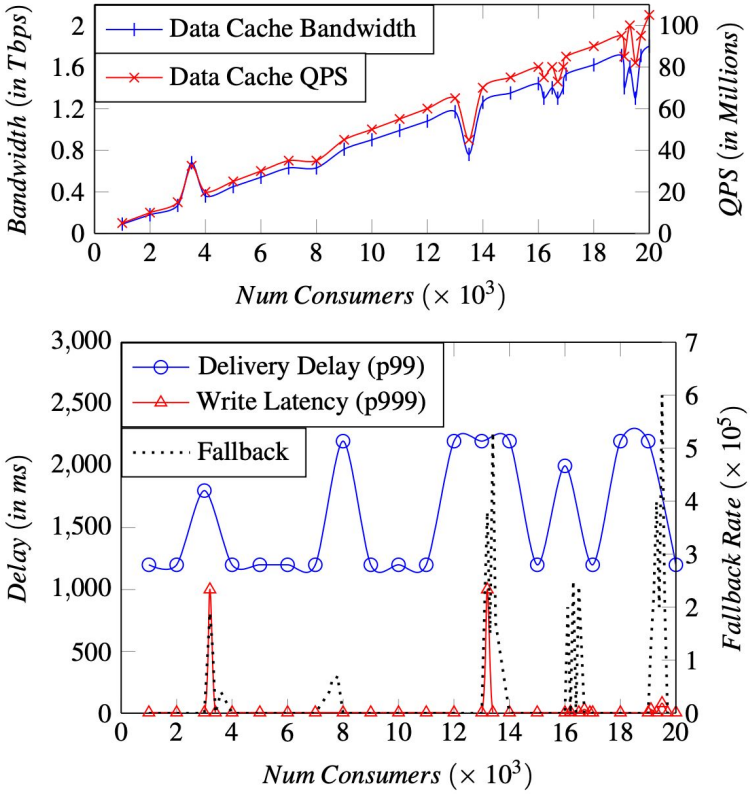


Experiment 2

- **Message streams:**
  - Two 120-way sharded message streams originating from us-central and euro-west region.
  - Each shard has a pair of files - **data** and **index**.
  - 480 Mbps average write rate.
  - 15 destination clusters per stream.
- Caches allowed to scale horizontally.
- **Consumers**:
  - 20 shards per consumers (80 Mbps per consumer).

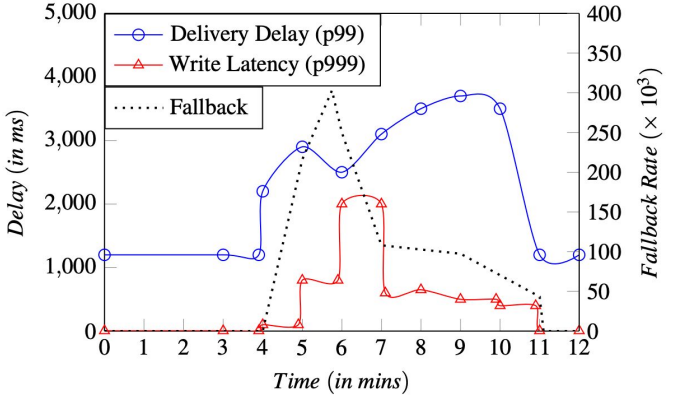
Experiment 2(a)

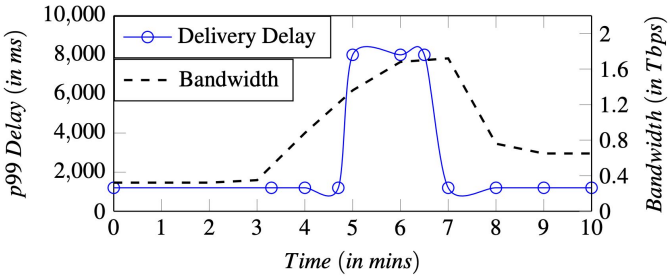
- Scaled up the consumers steadily from 1,000 to 20,000 in a leaf cluster:
  - Both data and metadata cache scaled up triggering transient failures due to hash modulo change.
- Consumer bandwidth - **1.8 Tbps**.



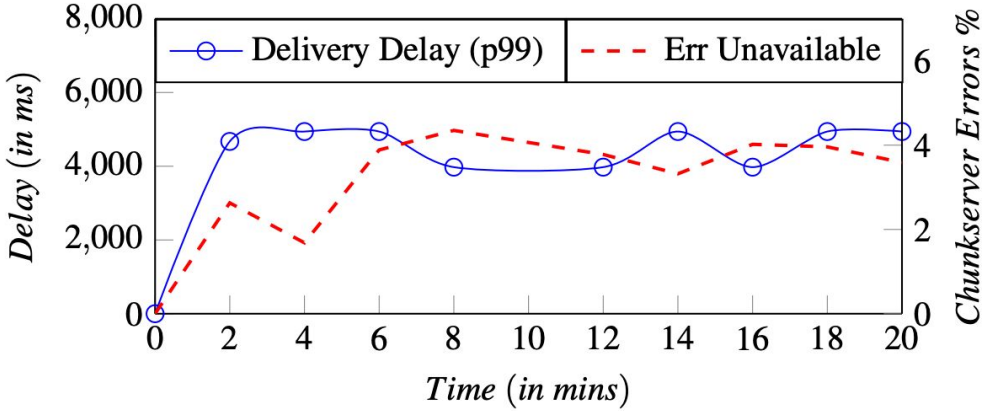
Experiment 2(b) and 2(c)

- Scaled up consumers suddenly from 0 to 6,500:
  - Such large surge happens during regional outages.
  - Large failures as caches scaled up.
  - Delivery delay - **4s**.
  - Eventually recovered within 7 minutes.
- Producers turned down and turned back up after 15 minutes.
  - A huge backlog generated (25+ GB).
  - Recovered in 2 minutes.





Ablation Study



- All components are essential except the cache layer.
- Conducted an experiment removing the data cache:
  - Delayed fallback reads helped manage QPS and hence disk time.
  - Majority of UNAVAILABLE errors (99%) from Colossus were due to network congestion.

Development Experience

- About 25,000 lines of non-test C++ code.
- Several sources of delays:
  - **Rollouts were tricky to handle**:
    - We do a slow rollout such that scheduler is available during worker rollout and vice-versa.
    - We never rollout caches and worker jobs simultaneously.
  - **Machine issues were hard to detect**:
    - We reschedule jobs based on system level metrics like:
      - Network queueing delays.
      - RDMA unavailability.
  - Inter-continental network delays have led to several SLO misses.

Production Experience

- **Service Level Objectives on**:
  - Delivery delay (p99)
  - Cache byte availability (p999)
- **Continuous monitoring**:
  - For byte-level correctness.
  - For delivery delay.

Delivery Delay*	
Percentile	**Latency**
p95	500ms
p99	630ms
p999	730ms
p9999	5.71s

Cache Byte Availability*	
Component	**% available**
Readers	96%
Consumers	99.9%

* - Measured over a day across all message streams.

Questions?