Building Durable Real-time Data Pipeline

Apache BookKeeper at Twitter

@sijieg | Twitter
Agenda

- Background
- Layered Architecture
- Design Details
- Performance
- Scale @Twitter
- Q & A
Online services - 10s of milliseconds

Transaction log, Queues, RPC

Near real-time processing - 100s of milliseconds

Change propagation, Stream Computing

Data delivery for analytics - seconds~minutes

Log collection, aggregation
Twitter Messaging at 2012

Core Business Logic (tweets, fanouts …)

Deferred RPC

Kestrel

Gizzard

Kestrel

Database

Book Keeper

Search

MySQL

Kafka

Scribe

HDFS
Kestrel

Simple

Perform well (as long as queue fits in memory)

Fan-out Queues: One per subscriber

Reliable reads - per item transaction

Cross DC replication
Kestrel Limitations

Durability is hard to achieve - Each queue is a separate file

Adding subscribers is expensive

Separate physical copy of the queue for each fanout

Read-behind degrades performance - Too many random I/Os

Scales poorly as #queues increase
Kafka

Throughput/Latency through sequential I/O with small number of topics

Avoid data copying - Direct Network I/O (sendfile)

Batch Compression

Cross DC replication (Mirroring)
Kafka Limitation

Relies on filesystem page cache

Limit #topics: Ideally one or handful topics per disk

Performance degrades when subscriber falls behind - Too much random I/O

No durability and replication (0.7)
Each of the systems came with their maintenance overhead

Software Components - backend, clients and interop with the rest of Twitter stack

Manageability and Supportability - deployment, upgrades, hardware maintenance and optimization

Technical know-how
Rethink the messaging architecture

Unified Stack - tradeoffs for various workloads
Durable writes, intra-cluster and geo-replication
Multi tenancy
Scale resources independently - Cost efficiency
Ease of manageability
Layered Architecture

Data Model
Software Stack
Data Flow
Log Stream

Entry:

a batch of records

sequence of bytes

DLSN: (LSSN, Eid, Sid)

Sequence ID

Transaction ID

Log Segment X

Log Segment X + 1

Log Segment Y
Layered Architecture

APPLICATION

STATELESS SERVING

CORE

PERSISTENT STORAGE

Write Clients

Ownership Tracker

Write Proxy

Writer

Read Clients

Routing Service

Read Proxy

Reader

Metadata Store (ZooKeeper)

BookKeeper

Cold Storage (HDFS)

Bookie
1. Write records
2. Transmit buffer
3. Flush - Write a batched entry to bookies
4. Acknowledge
5. Commit - Write Control Record
6. Long poll read
7. Speculative Read
8. Cache Records
9. Long poll read
Design Details

Consistency
Global Replicated Log
Consistency

LastAddConfirmed => Consistent views among readers
Fencing => Consistent views among writers
Consistency - LastAddPushed

Writer

Add entries

0 1 2 3 4

7 8 9 10 11 12

LastAddPushed
Consistency - LastAddConfirmed

Ownership Changed

Ack Adds

Add entries

Fencing

LastAdd Confirmed

LastAdd Pushed

Reader

Reader
Consistency - Fencing

0. Ownership Changed

New Writer

Old Writer

Completed Log Segment X

Inprogress Log Segment X+1

Completed Log Segment X

Inprogress Log Segment X+1

Completed Log Segment X+1

Inprogress Log Segment X+2

Bookie

Bookie

Bookie

Bookie

1. Get Log Segments

2.1 Fence Inprogress LogSegment

2.2 write rejected

3 new inprogress
Ownership Tracking (Leader Election)

ZooKeeper Ephemeral Znodes (leases)

Aggressive Failure Detection (within a second)

TickTime = 500 (ms)

Session Timeout = 1000 (ms)
Global Replicated Log

Region Aware Data Placement

Cross Region Speculative Reads
Hierarchical Data Placement

Data is spread uniformly across available regions

Each region uses rack aware placement policy

Acknowledge only when the data is persisted in majority of regions
Reader consults data placement policy for read order

First: the bookie node that is closest to the client

Second: the closest node that is in a different failure domain - different rack

Third: the bookie node in a different closest region

...
Performance

- Latency vs Throughput
- Scalability
- Efficiency
Support various workloads with latency/throughput tradeoffs

Latency vs Throughput (RF=3)

- periodical-10k-write
- periodical-10k-delivery
- periodical-0k-write
- periodical-0k-delivery
- immediate-write
- immediate-delivery
Under 100k rps, latency increased with number of streams increased on a single hybrid proxy.

Each stream writes 100 rps. Throughput increased linearly with number of streams.
Analytic application writes **2.45GB** per second, while the data has been fanout to **40x** to the readers.
DistributedLog

@ Twitter

Use Cases
Deployment
Scale
Applications at Twitter

Manhattan Key-Value Store

Durable Deferred RPC

Real-time search indexing

Self-Served Pub-Sub System / Stream Computing

Reliable cross datacenter replication

...
One global cluster, and a few local clusters each dc

$O(10^3)$ bookie nodes

$O(10^3)$ global log streams and $O(10^4)$ local log streams

$O(10^6)$ live log segments

Data is kept from hours to days, even up to a year

Pub-Sub: deliver $O(1)$ trillion records per day, roughly accounting for $O(10)$ PB per day
Lessons that we learned

Make foundation durable and consistent

Don’t trust filesystem

Think of workloads and I/O isolation

Keep persistent state as simple as possible

...
DistributedLog is the new messaging foundation

Layered Architecture

Separated *Stateless Serving* from *Stateful Storage*

Scale *CPU/Memory/Network* (shared mesos) independent of *Storage* (hybrid mesos)

Messaging Design

Writes / Reads Isolation

Scale *Writes* (Fan-in) independent of *Reads* (Fan-out)

Global Replicated Log
It is on Github!!

https://github.com/twitter/distributedlog

Apache Incubating ...

Twitter Messaging Team

@l4stewar @sijieg

https://about.twitter.com/careers