Intra-cluster Replication for Apache Kafka

Jun Rao

About myself

- Engineer at LinkedIn since 2010
- Worked on Apache Kafka and Cassandra
- Database researcher at IBM

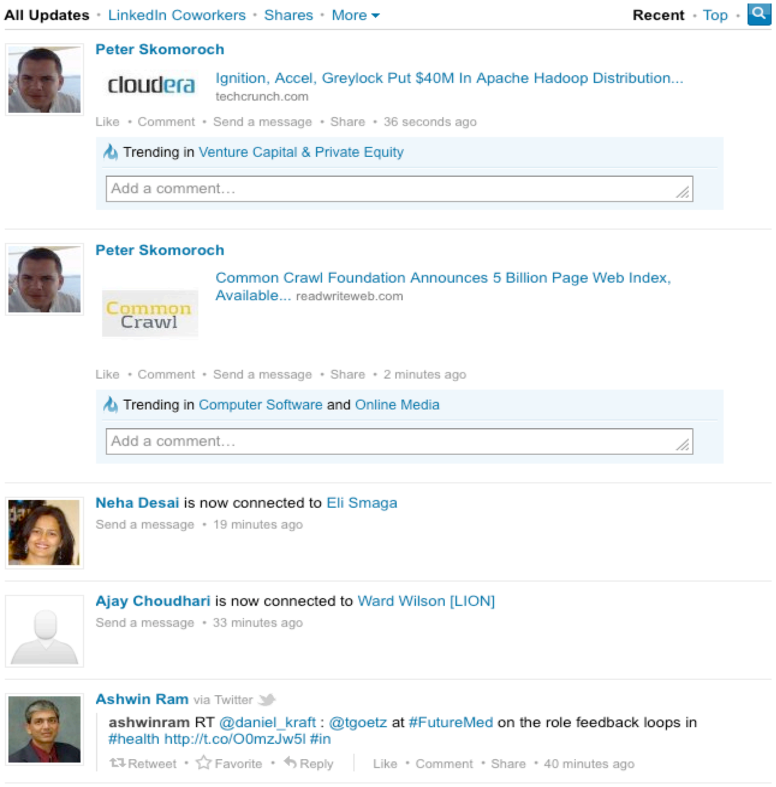
Outline

- Overview of Kafka
- Kafka architecture
- Kafka replication design
- Performance
- Q/A

What’s Kafka

- A distributed pub/sub messaging system
- Used in many places
  - LinkedIn, Twitter, Box, FourSquare …
- What do people use it for?
  - log aggregation
  - real-time event processing
  - monitoring
  - queuing

Example Kafka Apps at LinkedIn



Peter Skomoroch

cloudera — Ignition, Accel, Greylock Put $40M In Apache Hadoop Distribution... techcrunch.com

Like · Comment · Send a message · Share · 36 seconds ago

Trending in Venture Capital & Private Equity

Add a comment...

Peter Skomoroch

Common Crawl — Common Crawl Foundation Announces 5 Billion Page Web Index, Available... readwriteweb.com

Like · Comment · Send a message · Share · 2 minutes ago

Trending in Computer Software and Online Media

Add a comment...

Neha Desai is now connected to Eli Smaga

Send a message · 19 minutes ago

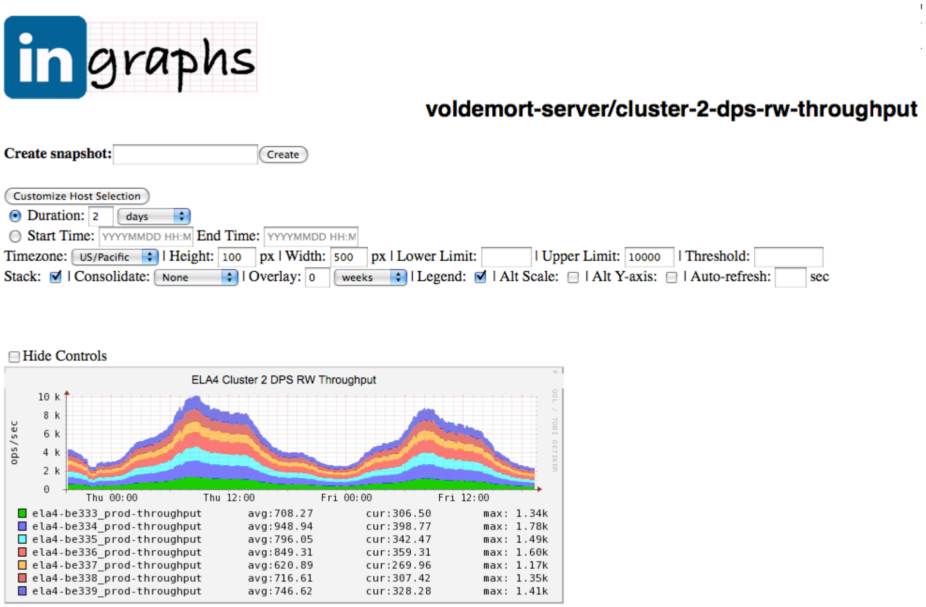
Ajay Choudhari is now connected to Ward Wilson [LION]

Send a message · 33 minutes ago

Ashwin Ram via Twitter

ashwinram RT @daniel_kraft : @tgoetz at #FutureMed on the role feedback loops in #health http://t.co/O0mzJw5l #in

Retweet · Favorite · Reply | Like · Comment · Share · 40 minutes ago



voldemort-server/cluster-2-dps-rw-throughput

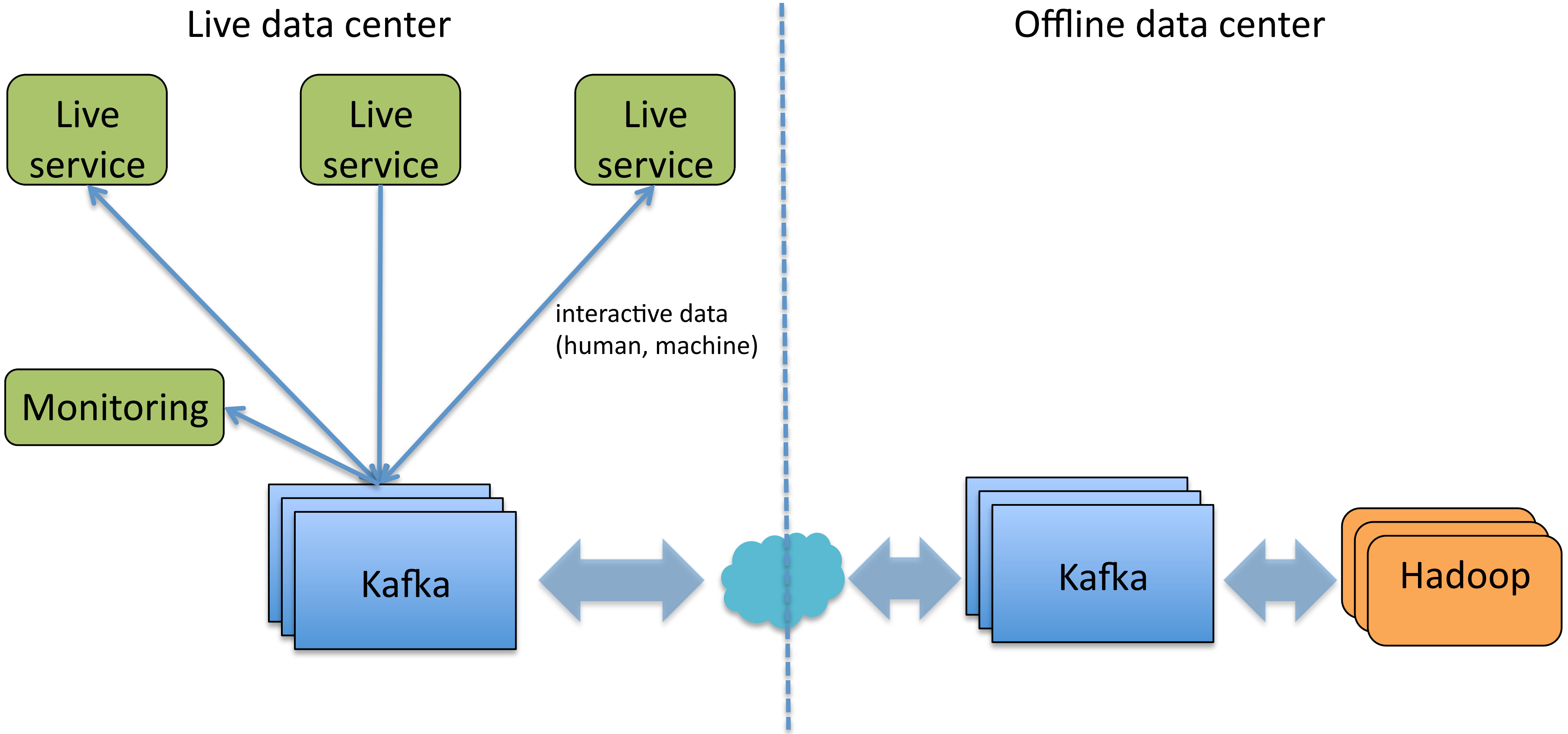
Create snapshot:

● Duration: 2 days
○ Start Time: YYYYMMDD HH:M End Time: YYYYMMDD HH:M
Timezone: US/Pacific | Height: 100 px | Width: 500 px | Lower Limit: | Upper Limit: 10000 | Threshold:
Stack: ☑ | Consolidate: None | Overlay: 0 weeks | Legend: ☑ | Alt Scale: □ | Alt Y-axis: □ | Auto-refresh: sec

□ Hide Controls

```
🟩 ela4-be333_prod-throughput      avg:708.27      cur:306.50      max: 1.34k
🟦 ela4-be334_prod-throughput      avg:948.94      cur:398.77      max: 1.78k
🟦 ela4-be335_prod-throughput      avg:796.05      cur:342.47      max: 1.49k
🟥 ela4-be336_prod-throughput      avg:849.31      cur:359.31      max: 1.60k
🟧 ela4-be337_prod-throughput      avg:620.89      cur:269.96      max: 1.17k
🟥 ela4-be338_prod-throughput      avg:716.61      cur:307.42      max: 1.35k
🟦 ela4-be339_prod-throughput      avg:746.62      cur:328.28      max: 1.41k
```

Kafka Deployment at LinkedIn



Per day stats

- writes: 10+ billion messages (2+TB compressed data)
- reads: 50+ billion messages

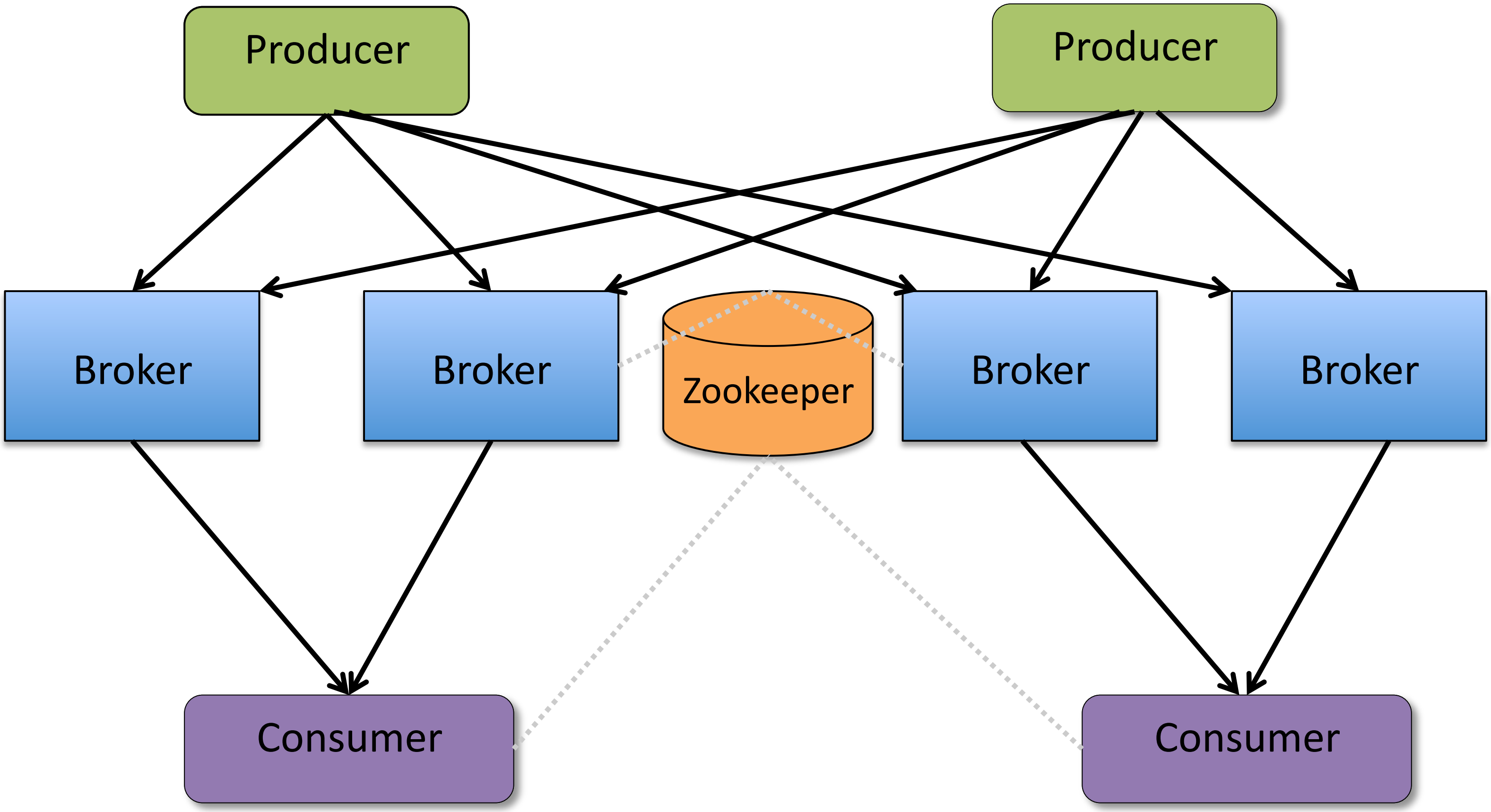
Kafka vs. Other Messaging Systems

- Scale-out from groundup
- Persistence to disks
- High throughput (10s MB/sec per server)
- Multi-subscription

Outline

- Overview of Kafka
- Kafka architecture
- Kafka replication design
- Performance
- Q/A

Kafka Architecture



Terminologies

- Topic = message stream
- Topic has partitions
  - partitions distributed to brokers
- Partition has a log on disk
  - message persisted in log
  - message addressed by offset

API

- Producer

```
messages = new List<KeyedMessage<K,V>>();
messages.add(new KeyedMessage(“topic1”, null, “msg1”);
send(messages);
```

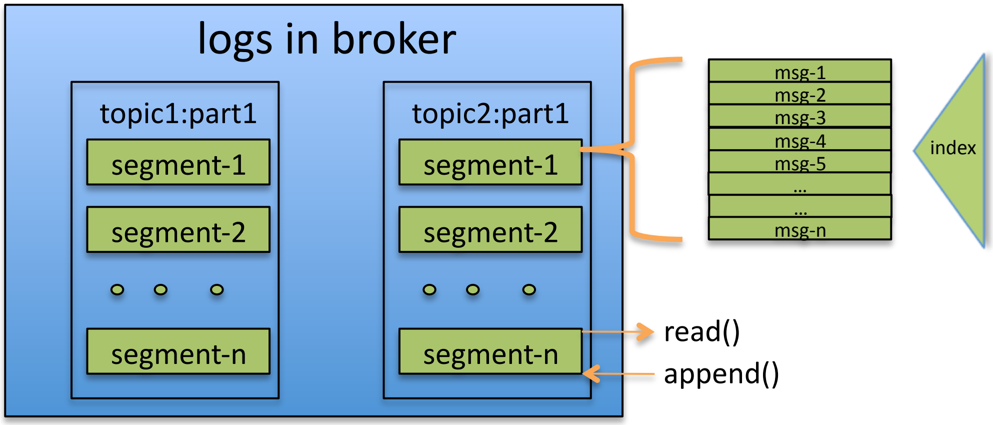
- Consumer

```
streams[] = Consumer.createMessageStream(“topic1”, 1);

for(message: streams[0]) {
   // do something with message
}
```

Deliver High Throughput

- Simple storage



- Batched writes and reads
- Zero-copy transfer from file to socket
- Compression (batched)

Outline

- Overview of Kafka
- Kafka architecture
- Kafka replication design
- Performance
- Q/A

Why Replication

- Broker can go down
  - controlled: rolling restart for code/config push
  - uncontrolled: isolated broker failure
- If broker down
  - some partitions unavailable
  - could be permanent data loss
- Replication → higher availability and durability

CAP Theorem

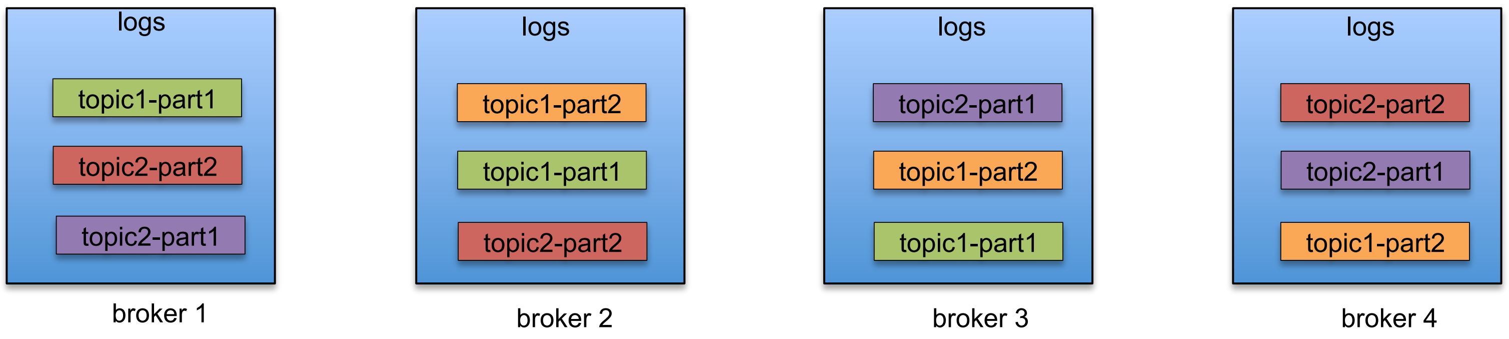
- Pick two from
  - consistency
  - availability
  - network partitioning

Kafka Replication: Pick CA

- Brokers within a datacenter
  - i.e., network partitioning is rare
- Strong consistency
  - replicas byte-wise identical
- Highly available
  - typical failover time: < 10ms

Replicas and Layout

- Partition has replicas
- Replicas spread evenly among brokers



logs

topic1-part1

topic2-part2

topic2-part1

broker 1

logs

topic1-part2

topic1-part1

topic2-part2

broker 2

logs

topic2-part1

topic1-part2

topic1-part1

broker 3

logs

topic2-part2

topic2-part1

topic1-part2

broker 4

Maintain Strongly Consistent Replicas

- One of the replicas is leader
- All writes go to leader
- Leader propagates writes to followers in order
- Leader decides when to commit message

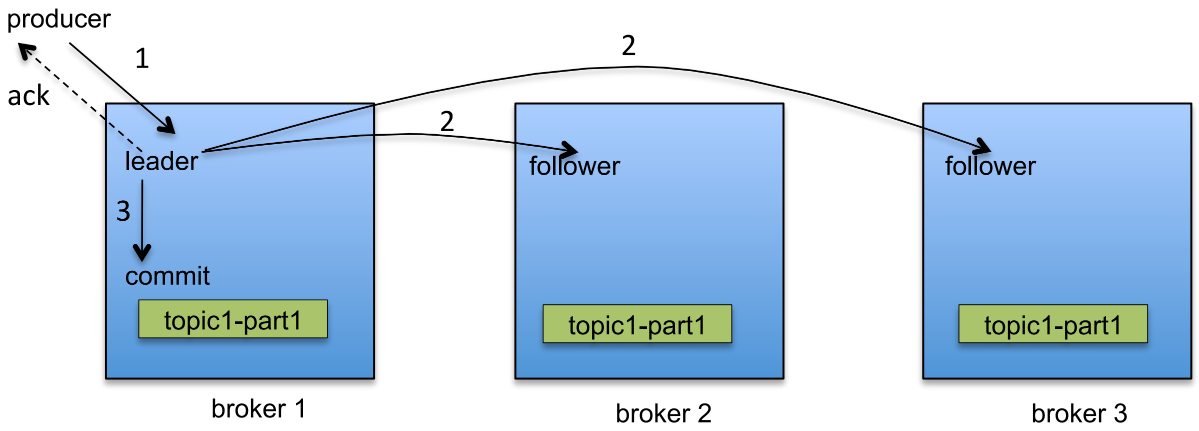
Conventional Quorum-based Commit

- Wait for majority of replicas (e.g. Zookeeper)
- Plus: good latency
- Minus: 2f+1 replicas → tolerate f failures
  - ideally want to tolerate 2f failures

Commit Messages in Kafka

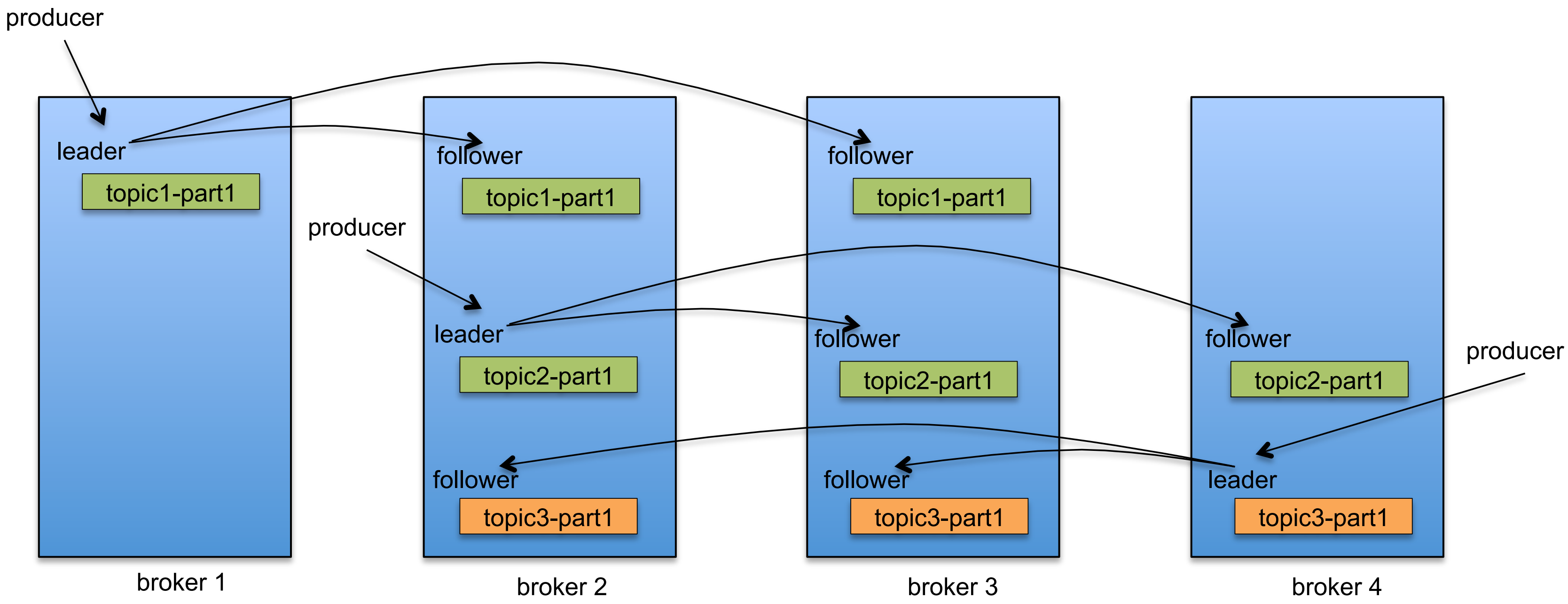
- Leader maintains in-sync-replicas (ISR)
  - initially, all replicas in ISR
  - message committed if received by ISR
  - follower fails → dropped from ISR
  - leader commits using new ISR
- Benefit: f replicas → tolerate f-1 failures
  - latency less an issue within datacenter

Data Flow in Replication



When producer receives ack	Latency	Durability on failures
no ack	no network delay	some data loss
wait for leader	1 network roundtrip	a few data loss
wait for committed	2 network roundtrips	no data loss

Extend to Multiple Partitions



- Leaders are evenly spread among brokers

Handling Follower Failures

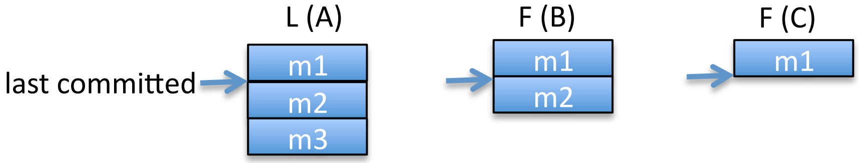
- Leader maintains last committed offset
  - propagated to followers
  - checkpointed to disk
- When follower restarts
  - truncate log to last committed
  - fetch data from leader
  - fully caught up → added to ISR

Handling Leader Failure

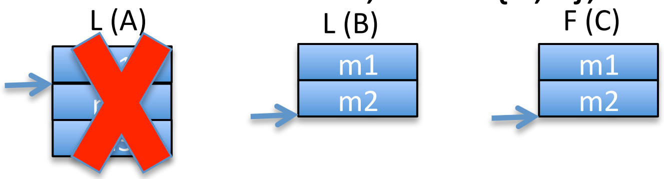
- Use an embedded controller (inspired by Helix)
  - detect broker failure via Zookeeper
  - on leader failure: elect new leader from ISR
  - committed messages not lost
- Leader and ISR written to Zookeeper
  - for controller failover
  - expected to change infrequently

Example of Replica Recovery

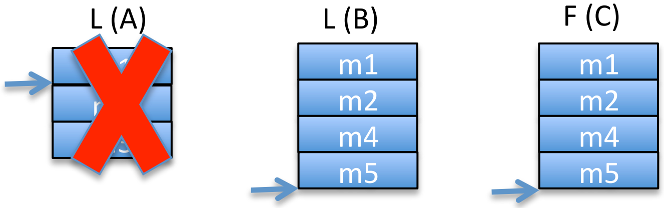
1. ISR = {A,B,C}; Leader A commits message m1;



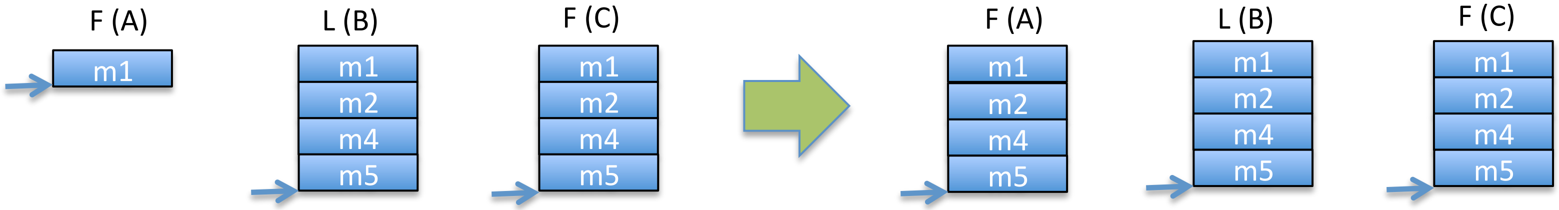
2. A fails and B is new leader; ISR = {B,C}; B commits m2, but not m3



3. B commits new messages m4, m5



4. A comes back, truncates to m1 and catches up; finally ISR = {A,B,C}



Outline

- Overview of Kafka
- Kafka architecture
- Kafka replication design
- Performance
- Q/A

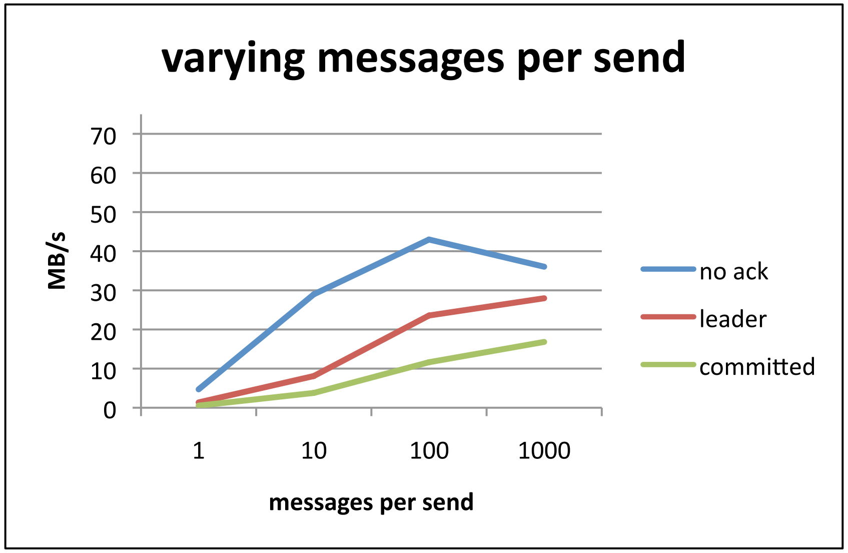
Setup

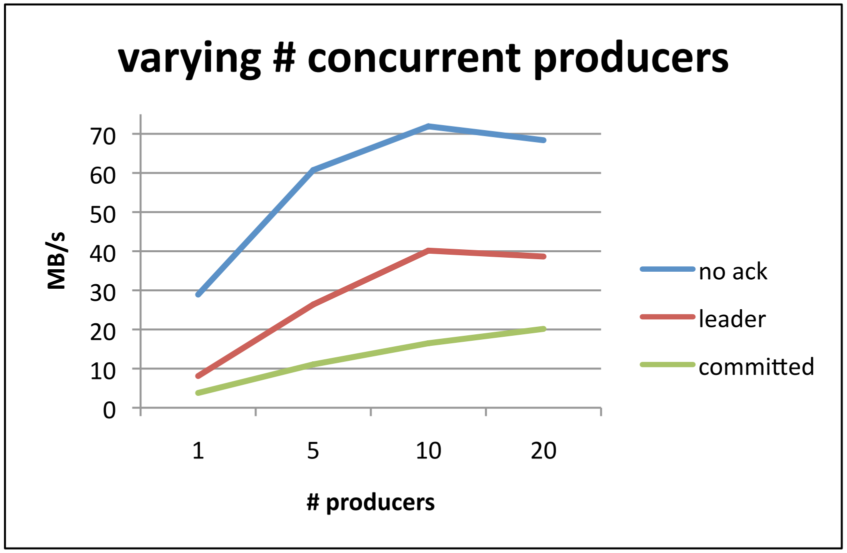
- 3 brokers
- 1 topic with 1 partition
- Replication factor=3
- Message size = 1KB

Choosing btw Latency and Durability

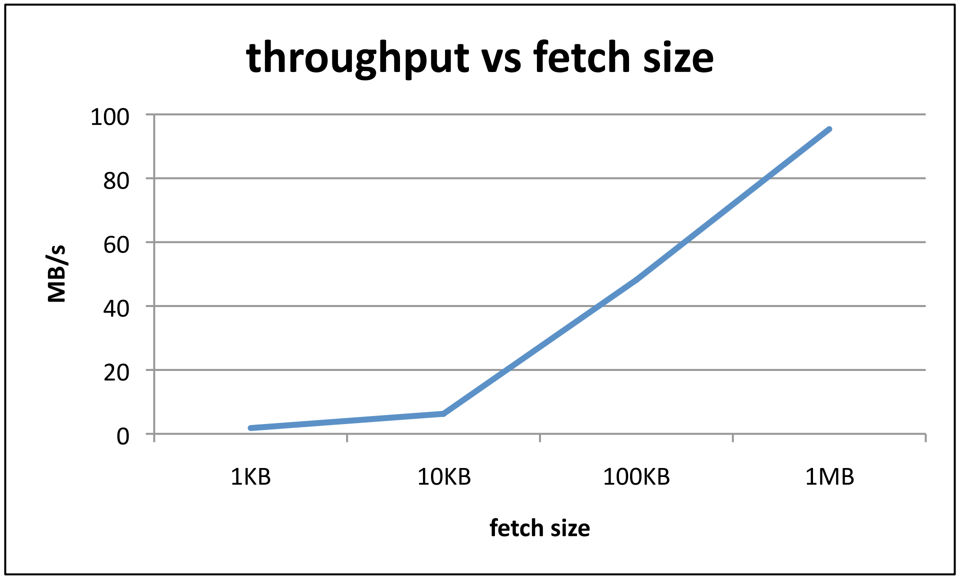
When producer receives ack	Time to publish a message (ms)	Durability on failures
no ack	0.29	some data loss
wait for leader	1.05	a few data loss
wait for committed	2.05	no data loss

Producer Throughput





Consumer Throughput



Q/A

- Kafka 0.8.0 (intra-cluster replication)
  - to be released in Mar
  - various performance improvements in the future
- Checkout more about Kafka
  - http://kafka.apache.org/
- Kafka meetup tonight