Infrastructure Technologies for Large-Scale Service-Oriented Systems

Kostas Magoutis
magoutis@csd.uoc.gr
http://www.csd.uoc.gr/~magoutis
Kafka

- **Data logged**
  - User activity (logins, page views, clicks, likes, sharing, comments, search queries)
  - Operational metrics (call latency, errors, system metrics)

- **Uses**
  - Search relevance
  - Recommendations driven by item popularity or co-occurrence in activity stream
  - Ad targeting and reporting
  - Security applications
  - Newsfeed of user status for friends / connections to read
Challenges

• High event rates
  – Search, recommendations, and advertising require computing granular click-through rates
  – China Mobile 5-7TB of phone call records / day
  – Facebook gathers ~6TB of various user activity events / day

• Traditional enterprise messaging systems too strict
  – Unnecessarily rich set of delivery guarantees
    • IBM WebSphere MQ: allow atomic inserts into multiple queues
    • JMS spec: ack each individual message after consumption
  – Performance issues: No API to batch messages (JMS)
  – No easy way to partition and store msgs on many machines
  – Assuming near-immediate consumption of messages
**Kafka architecture**

**Sample producer code:**

```java
producer = new Producer(...);
message = new Message("test message str".getBytes());
set = new MessageSet(message);
producer.send("topic1", set);
```

**Sample consumer code:**

```java
streams[] = Consumer.createMessageStreams("topic1", 1)
for (message : streams[0]) {
    bytes = message.payload();
    // do something with the bytes
}
```
• Each partition of a topic corresponds to a logical log
• Flush to disk after configurable number of published messages
Efficiency of single partition

- **Simple storage**
  - Consumer acknowledges message offsets
  - Under the cover, consumer issues async pull requests
  - Broker locates segment file, sends data back to consumer

- **Efficient transfer**
  - No user-space caching by brokers, reduces JVM GC costs
  - Direct transfer from files to network sockets

- **Stateless broker**
  - Does not know whether all subscribers have consumed msg
  - Automatic message deletions after 7 days
  - Subscribers can rewind and replay messages
Consumer groups

• One or more consumers that jointly consume a set of subscribed topics
  – Each message delivered to only one consumer within CG

• No coordination needed across CGs

• Goal is to divide messages stored in brokers evenly among consumers

• All messages from one partition consumed by single consumer in a CG
  – Multiple consumers of a partition would need to coordinate
  – To balance load, multiple partitions per consumer
Coordination service: ZooKeeper

- Simple file-like API on znodes
- Can register watcher on a path, get notified
- Ephemeral vs. persistent paths
- Highly available service

Image courtesy of https://zookeeper.apache.org
Kafka data structures in ZooKeeper

Broker registry
Broker hostname/port, set of topics/partitions it stores

Consumer registry
CG consumer belongs to, set of topics it subscribes to

Ownership registry
Partition-to-consumer mapping

Offset registry
Offset of last consumed message per partition

Partitions
[partitionId]

Topics
[topic]

Brokers

Consumers

State

Ownership

Consumer ids

Group ids

Partition ids

Broker ids

Offset ids

Offset

Ownership registry

Partition-to-consumer mapping

Broker registry

Broker hostname/port, set of topics/partitions it stores

Consumer registry
CG consumer belongs to, set of topics it subscribes to

Ownership registry
Partition-to-consumer mapping

Offset registry
Offset of last consumed message per partition

Partitions
[partitionId]

Topics
[topic]

Brokers

Consumers

State

Ownership

Consumer ids

Group ids

Partition ids

Broker ids

Offset ids

Offset
Rebalancing partitions

- Detect the addition or removal of brokers or consumers
- Trigger a re-balance process when that happens

**Algorithm 1**: rebalance process for consumer $C_i$ in group $G$

For each topic $T$ that $C_i$ subscribes to {
  remove partitions owned by $C_i$ from the ownership registry
  read the broker and the consumer registries from Zookeeper
  compute $P_T = \text{partitions available in all brokers under topic } T$
  compute $C_T = \text{all consumers in } G \text{ that subscribe to topic } T$
  sort $P_T$ and $C_T$
  let $j$ be the index position of $C_i$ in $C_T$ and let $N = |P_T|/|C_T|$
  assign partitions from $j*N$ to $(j+1)*N - 1$ in $P_T$ to consumer $C_i$
  for each assigned partition $p$ {
    set the owner of $p$ to $C_i$ in the ownership registry
    let $O_p = \text{the offset of partition } p \text{ stored in the offset registry}$
    invoke a thread to pull data in partition $p$ from offset $O_p$
  }
}
Typical Kafka deployment