a high throughput messaging system for log processing

Presenter: Hao Tan
h26tan@uwaterloo.ca
What is log data

- Tech companies nowadays are dealing with various types of log data
- user activities: likes, login records, comments, queries
- operational metrics: CPU, memory, disk utilisation
Log data is valuable

- Companies need those data to improve user experience of their services:
  - recommendation system
  - news feed aggregation
  - search relevance
  - ad targeting
  - spam detection
Problem

• large data volume: TB level

• Build a specialised pipeline between data producer and data consumer is not scalable
At the beginning:

Source → Hadoop
Then, we have more data sources to process..
More consumer come...
Previous Systems

Enterprise messaging systems:

• Overkill features: IBM WebSphere MQ provide API to insert message to multiples queues atomically
• Throughput is not the top concern: JMS has no batch delivery, one message per network round trip
• Not distributed
• Assume immediate consumption of the message

Log aggregator:

• Mostly designed for offline data consumption
• use a push model
Kafka introduction

• Initially developed in LinkedIn, now become part of Apache

• Decouples data pipelines from producers and consumers

• Pull model instead of push model

• Support both online and offline data consumption

• Scalable, fault-tolerant and focuses on throughput
Key terminology

- **Topic**: a stream of messages of a particular type

- **Producer**: a process that publishes messages to a Kafka topic

- **Broker**: a server that stores message data, Kafka runs on a cluster of brokers

- **Consumer**: process that subscribes one or more topics and pulls messages from brokers
Kafka Architecture

Sample Producer Code

```java
import java.util.*;
import kafka.javaapi.producer.Producer;
import kafka.producer.KeyedMessage;
import kafka.producer.ProducerConfig;

public class TestProducer {
    public static void main(String[] args) {
        long events = Long.parseLong(args[0]);
        Random rnd = new Random();

        Properties props = new Properties();
        props.put("metadata.broker.list", "broker1:9092,broker2:9092");
        props.put("serializer.class", "kafka.serializer.StringEncoder");
        props.put("partitioner.class", "example.producer.SimplePartitioner");
        props.put("request.required.acks", "1");

        ProducerConfig config = new ProducerConfig(props);

        new Producer<String, String> producer = new Producer<>(config);

        for (long nEvents = 0; nEvents < events; nEvents++) {
            long runtime = new Date().getTime();
            String ip = "192.168.2." + rnd.nextInt(255);
            String msg = runtime + ",", www.example.com," + ip;
            KeyedMessage<String, String> data = new KeyedMessage<String, String>("page_visits", ip, msg);
            producer.send(data);
        }

        producer.close();
    }
}
```

reference: https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+Producer+Example
FetchRequest req = new FetchRequestBuilder()
   .clientId(clientName)
   .addFetch(a_topic, a_partition, readOffset, 100000)
   .build();
FetchResponse fetchResponse = consumer.fetch(req);

long numRead = 0;
for (MessageAndOffset messageAndOffset : fetchResponse.messageSet(a_topic, a_partition)) {
   long currentOffset = messageAndOffset.offset();
   if (currentOffset < readOffset) {
      System.out.println("Found an old offset: " + currentOffset + " Expecting: " + readOffset);
      continue;
   }
   readOffset = messageAndOffset.nextOffset();
   ByteBuffer payload = messageAndOffset.message().payload();

   byte[] bytes = new byte[payload.limit()];
   payload.get(bytes);
   System.out.println(String.valueOf(messageAndOffset.offset()) + ":" + new String(bytes, "UTF-8");
   numRead++;
   a_maxReads--;
}

if (numRead == 0) {
   try {
      Thread.sleep(1000);
   } catch (InterruptedException ie) {
   }
}

reference: https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+SimpleConsumer+Example
What’s under the hood

- A partition consists of a set of segment files
  - roughly 1GB per segment file
- When producer publish a message to a partition, broker appends it to the end of the last segment file
- Segment files are flushed to disk after accumulating certain number of messages.
- Message id is its offset in each segment file.
- An in-memory index to support fast lookups
Storage Layout

consumer 1

consumer 2

consumer 3

producer
Efficiency

- Relies on OS page cache

- achieves great performance due to sequential access to segment files and lagging between broker and consumer

- Leverage Linux `sendfile` system call for faster data transfer
Stateless Brokers

- Consumer maintains the offset for consumed messages (in ZooKeeper)
- Messages will be automatically deleted
- Consumer has a chance to rewind back:
  - make consumers more resilient to errors
Coordination

• Consumer group

• No coordination between consumer groups

• Partition is the smallest unit for parallelism

• Coordination is only needed for load balancing when a broker or consumer is removed/added

• Decentralised coordination via ZooKeeper
Rebalancing workload

**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 $N = \frac{|P_T|}{|C_T|}$
  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 $O_p$
}
Delivery Guarantee

- Kafka guarantee at least once delivery
- Message from a single partition will be delivered to consumer in order
- No order guarantee on messages from different partitions
- When broker is down, all not yet consumed messages are lost
- Later version of Kafka supports replication of partition across brokers
Experiment and Performance

Figure 4. Producer Performance

Figure 5. Consumer Performance
Discussion

- Any weak point of Kafka?
- No exact-once guarantee
- No order guarantee for messages from multiple partitions
- Pull model vs push model
Thank you very much