Paper Review –5

Kafka

Title: Kafka: A Distributed Messaging System for Log Processing

Summary

The paper presents Kafka a distribute messaging system that was developed for collecting and delivering high volumes of log data with low latency. Kafka is suitable for both offline and online message consumption unlike traditional log processing systems. Capturing events such as clicks, operational metrics are dynamic in nature and this data is required for clients in LinkedIn to process in real time. Kafka also provides an API similar to messaging system and allows applications to listen to API events.

Good Aspects

Kafka has addressed one of the main important issues that exist in conventional log processing system and that they are not able to perform analytics in real time. Hence Kafka has managed to address this issue and yet provide some of the offline log aggregation facility. Kafka uses Zoo Keeper to help facilitate coordination amongst different consumers and producers. Kafta also uses simple policies hence reduces complexity. It also is based on the pull model so that application need not get completely overwhelmed by data as opposed to other counterpart's like Flume, Scribe and Data highway.

Review

Following details will cover all aspect of the paper providing crucial and also interesting aspects of the paper that appealed the reader.

Kafka is a distributed publish-subscribe messaging system that is designed to be fast, scalable and durable. Kafka uses the concept of consumer and producer to describe who consumes the data and serves the data to Kafka cluster. Kafka uses a concept of topic to group message of same type thus builds efficient mechanism to dispatch it to multiple consumers via Pull mechanism.

Architecture

- **Concepts**
  - **Topic** – A stream of messages of the same type.
  - **Producer** – Can publish message.
  - **Consumer** – Consumes the message by subscription.
  - **Brokers** – Producers messages and these messages are stored as partitions in brokers which are a set of servers.
- **Sample Producer code**

```
producer = new Producer(…);
message = new Message(“test message str”.getBytes());
set = new MessageSet(message);
producer.send(“topic1”, set);
```

- **Sample Consumer code**.

```
streams[] = Consumer.createMessageStreams(“topic1”, 1)
for (message : streams[0]) {
bytes = message.payload();
// do something with the bytes
}
```

- **Storage Layout**
  - Each partition of a topic is called Logical Log.
  - Physically a log is implemented as a segment of files.
  - Each message does not have an id but instead has an offset in every log.
  - To get the id of next message, add current message id + length of current message
  - The broker keeps offset of files inside memory to help efficient lookup.
  - The messages are self expiring and don’t remove after all consumers have consumed it hence it allows for consumers that failed to recover lost messages within 7 days.

- **Efficient Transfer**
  - An iterator is provided that helps in iterating through all messages.

- Each iterator transfers about message size of 100 KBs.
- Kafka does not cache any messages in the server.

- **Stateless broker**
  - State information of each consumer is not maintained by broker.
  - The ability to handle states is offloaded from broker to Consumer hence it must keep state of all messages received.

Distributed Coordination

- **Consumer Group –** Consists of consumers that consume messages of same topic. Each of these messages are delivered to each of the consumers within the group.
- **Design Decisions –**
  - Distribute messages stored in brokers evenly to consumers to avoid hotspots.
  - Partitions made inside a topic to better control parallelism.
  - Partitions are done in such a way that in a given point in time, all messages in one partition are consumed by only one consumer.
  - This avoids unnecessary locking and state handling implementation from client.
  - There is no notion of a master node and in order to facilitate coordination, it uses zookeeper which has a very simple file system like API. Following are the uses of Zookeeper
    - A watch can be registered with any path to get notified of path gets changed.
    - A path can be ephemeral.
    - Zookeeper replicates data to multiple servers.
    - Internal rebalance done by the consumer to determine how much data it can consume now.

```
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 $P_T$ = partitions available in all brokers under topic T
        compute $C_T$ = all consumers in G 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 $O_p$ = the offset of partition p stored in the offset registry
        invoke a thread to pull data in partition p from offset $O_p$
    }
}
```

Delivery Guarantees –

- Kafka guarantees that messages from single partition are delivered to consumer in order.

- To avoid corruption in log, kafka stores CRC and removes messages whose CRC does not match.

Current Usage-

- Kafka is currently used in LinkedIn by its frontend services and also stored offline for data analytics.
- Kafka loads data via Map Reduce jobs that take inputs to read directly from Kafka.
- Uses AVRO serialization protocol as it supports schema evolution and is quite efficient.

Conclusion and Summary

The following paper has presented Kafka that is a novel system for processing huge volume of log data streams. Like a messaging system, Kafka employs a Pull based consumption model that allows an application to consume data at its own rate rather than overwhelming the client.

Kafka offers a highly distributed and scalable system that combines the benefits of traditional log aggregators and messaging systems.

Kafka is currently in use by LinkedIn and use by various other companies since it has been open sourced.

Kafka keeps things simple and provides Consumer and Producer concept that handles most of Kafka interaction to keep it clean for clients.

In a way Kafka has managed to solve the crisis at LinkedIn to help them tackle real time data and providing a highly efficient distributed system with a very high throughput.