Distributed Systems

26. Messaging: Kafka

Paul Krzyzanowski
Rutgers University
Spring 2020

Message Processing

How do we design a computing cluster to process huge, never-ending streams of messages from multiple sources?

Apache Kafka

Kafka is
• Open-source
• High-performance
• Distributed
• Durable
• Fault-tolerant
• Publish-subscribe messaging system

Messages may be anything: IoT (Internet of Things) reports, logs, alerts, user activity, data pipelines, …

Publish-Subscribe Messaging

• Publishers send streams of messages = producers
• Subscribers receive messages = consumers
• Messaging system = message broker
  – Provides a loose coupling between producers & consumers

Publish-Subscribe – Topics

• We will often have multiple message streams
  – Different purposes (e.g., IoT temperature reports, error logs, page views, …)
  – Different consumers will be interested in different streams
• Streams are identified by a topic
  – Publishers send messages to a topic and subscribers subscribe to a topic
Kafka runs as a cluster on one or more servers. Each server is called a broker. A Kafka deployment may have anywhere from 1 to 1000s of brokers. Kafka can feed messages to:

- **Real-time systems**: e.g., Spark Streaming
- **Batch processing**: e.g., store to Amazon S3 or HDFS & then use MapReduce or Spark

**Partitioned log**

- Each topic is stored as a partitioned log
  - One message log is broken up (partitioned) into multiple smaller logs
  - Each chunk is a partition and can be stored on a different server
- A partitioned log enables messages for a topic to scale beyond the capacity of a single server.

**Fault Tolerance & Replication**

- Messages in a partition are **durable**: written to disk
  - Persist for a configurable time period – then erased
- One server is elected to be the **leader** for a partition
  - 0 or more other servers are **followers**
  - Replication amount is configurable
  - Leader handles all read/write requests (like Raft)
  - Clients do not communicate with followers

**Achieving Scale**

**Producers**

- Clients choose which partition to write message to
  - Default: round-robin distribution to balance load evenly across multiple brokers
- Create more partitions for a topic = more load distribution

**Consumers**

- **Consumer group** = one or more consumers
- Group members share the same message queue for the topic
  - Messages to the topic get distributed among the members of the consumer group
- More consumers in a group = more processing capacity
Queuing vs. Publish-Subscribe

Queuing model
- Pool of consumers that take messages from a shared queue
- When any consumer gets a message, it is out of the queue
- Only one consumer gets each message
- Great for distributing processing among multiple subscribers

Publish-Subscribe model
- Each consumer that subscribes to a topic will get every message for that topic
- Allows multiple clients to share the same data … but does not scale

Zookeeper
Kafka uses Apache Zookeeper for coordination
- Zookeeper = Google Chubby
  - Getting heartbeats from brokers
  - Leader election
  - Configuring replication settings
  - Tracking members of cluster
  - Etc.

Producers
- Use it to find partitions for a topic
Consumers
- Use it to track the current index # (offset) of the next message in each partition they’re reading

Disk storage
Kafka provides **durable** message logs
- Messages will not be lost if the system dies and restarts
**But disks are slow!**
- Not necessarily
- Huge performance difference between random block access and sequential access
- Kafka optimizes for large sequential writes & reads
  - Disk operations can be thousands of times faster than random access

Apache Kafka is
- **Open-source**
  - Developed by LinkedIn and donated to the Apache Software Foundation, written in Scala and Java
- **High-performance**
  - Scalable to handle huge volumes of incoming messages by partitioning each message queue (log) among multiple servers
  - Partitioned log enables the log to be larger than the capacity of any one server
  - Consumer groups enable the scaling of message processing
- **Distributed**
  - Each message queue (log) is divided among multiple servers
- **Durable**
  - Message logs are written to disk (via large streaming writes for best performance)
- **Fault-tolerant**
  - Support for redundancy with a leader & followers per partition
- **Publish-subscribe messaging system**
  - Publish & subscribe to topics
Kafka summary

- Solved the problem of dealing with continuous data streams
- Solves the scaling problem by using partitioned logs
- Supports both single queue & publish-subscribe models
- Message ordering is guaranteed per-partition only
- Well-used, proven performance
  - Activision, Airbnb, Tinder, Pinterest, Uber, Netflix, LinkedIn, Microsoft, most banks, …