The secret sauce behind LinkedIn's self-managing Kafka clusters

Jiangjie (Becket) Qin @LinkedIn
Agenda

- Kafka introduction and terminologies
- Problems to solve
- Our solution
  - Cruise Control Architecture
  - Challenges and Solutions
- Insights and Generalization
  - Problem Generalization
  - Workload Model Generalization
- Q&A
Agenda

- **Kafka introduction and terminologies**
- Problems to solve
- Our solution
  - Cruise Control Architecture
  - Challenges and Solutions
- Insights and Generalization
  - Problem Generalization
  - Workload Model Generalization
- Q&A
What is Kafka

- An open source distributed stream processing platform
  - High throughput
  - Low latency
  - Message persistency
  - Partitioned data
  - Ordering within partitions
  - ...
Key Concepts and Terminologies

- Each **Topic** has multiple **Partitions**
- Each Partition has a few **Replicas** (one **leader**, 0+ **followers**)
- Each **Broker** (Server) hosts many replicas
Key Concepts and Terminologies

- **Producer** clients and **consumer** clients
  - The producers and consumers are only served by Leader Replicas
  - Follower replicas are only for failover
Agenda

- Kafka introduction and terminologies
- **Problems to solve**
- Our solution
  - Cruise Control Architecture
  - Challenges and Solutions
- Insights and Generalization
  - Problem Generalization
  - Workload Model Generalization
- Q&A
Operation Challenges

- The scale of Kafka deployment @LinkedIn
  - 2,100+ brokers
  - ~ 60,000 topics
  - ~ 4.2 million partitions
  - > 4.5 trillion messages / day

- Huge operation overhead
  - Hardware failures are norm
  - Workload skews
Requirements for Kafka Cluster Management

- Dynamic Load Balancing
  - CPU, Disk, Network IO, SLA, Rack Aware...
  - Heterogeneous brokers support
- Failure detection and self-healing
  - Reassign the replicas on the dead brokers
  - Reduce the window of under-replication
- Other admin operations
  - Add / Decommission brokers, manual leader movement...
Problem to solve

- How to manage the Kafka clusters to meet all the requirements?
Two basic operations

- Replica Movement
  - Expensive – require data copy
  - Impact on all hardware resources

- Leader Movement
  - Cheap – no data copy
  - Impact on CPU and network bytes out
The questions to answer

- Which partition should be moved?
- What should be moved?
  - Leader Movement
  - Replica Movement
- Where to move?
  - Move to which broker
- How much should be moved?
- Moving cost?
Agenda

- Kafka introduction and terminologies
- Problems to solve
- **Our solution**
  - Cruise Control Architecture
  - Challenges and Solutions
- Insights and Generalization
  - Problem Generalization
  - Workload Model Generalization
- Q&A
Cruise Control Architecture

- **Metrics reporter** collects the standard Kafka metrics and send them to a Kafka topic (CruiseControlMetrics).
Cruise Control Architecture

- **Load Monitor** generates a Cluster Load Model to describe the workload of the cluster.
Cruise Control Architecture

- **Load Monitor** generates a Cluster Load Model to describe the workload of the cluster
  - **Metric Sampler** – Periodically (e.g. every 5 min) samples the cluster workload and generates *partition workload samples*. Default implementation reads from the cruise control metrics topic.
Cruise Control Architecture

- **Load Monitor** generates a Cluster Load Model to describe the workload of the cluster
  - **Metric Sampler** – Periodically (e.g. every 5 min) samples the cluster workload and generates *partition workload samples*. Default implementation reads from the cruise control metrics topic.
  - **Sample Store** – Save per partition workload samples to the workload history topic as backup for failure recovery.
Cruise Control Architecture

- **Cluster Load Model**
  - Topology
  - Replica distribution
  - Leader distribution
  - Workload
  - Granular workload (CPU, Disk, Network, etc.) per replica for each monitoring Window (e.g. an hour)

![Diagram of Cruise Control Architecture](image)
Cruise Control Architecture

- **Cluster Load Model**
  - Interface to **simulate**
  - Replica movement
  - Leader movement
  - The simulation could be tricky
  - More about this later

![Diagram of Cruise Control Architecture]

- **Sample Store**
- **Metric Sampler**
- **Cluster Load Model**
- **Send Partition Workload**
- **Load Partition Workload**
- **Workload History Topic**
- **Standard Metrics**
- **Send Partition Workload**
- **Load Partition Workload**
- **REST API**
- **Analyzer**
  - Goal 0
  - Goal 1
  - Goal 2
  - ...
- **Detector**
  - Goal Violation Detector
  - Failure Detector
  - Anomaly Notifier
- **Executor**
- **Kafka Cluster**
  - **Brocker Failures**
  - **Throttled Proposal Execution**

©2018 LinkedIn Corporation. All Rights Reserved.
• **Analyzer** is responsible for generating optimization proposals to achieve the pluggable **goals**.
  - Heuristic solution
    - Fast
    - Not globally optimal
    - But is usually good enough

---

Cruise Control Architecture

- **Analyzer** is responsible for generating optimization proposals to achieve the pluggable **goals**.
- **Heuristic solution**
  - Fast
  - Not globally optimal
  - But is usually good enough
Cruise Control Architecture

- **Goals**
  - Are **pluggable**
    - Impl. of a public interface
    - Easy to add new goals
  - With different **priorities**
    - High priority goals run first
  - **Hard** Goal or **Soft** Goal
    - Hard Goals – must be satisfied, otherwise optimization fails
    - Soft Goals – best effort
  - Some Example goals
    - Rack Aware
    - Resource Utilization Balance
    - AVG $\pm$ 5%

- Pluggable Component
Cruise Control Architecture

- **Detector** detects anomalies
  - Two types of anomalies
    - Goal violations
    - Broker failures
  - Anomaly Notifier
    - Anomaly notification
      - E.g. email
    - Decision making
      - Fix
      - Delayed Check
      - Ignore

---

**Cruise Control**

- **Sample Store**
- **Metric Sampler**
- **Load Monitor**
  - Cluster Load Model
  - Send Partition Workload
    - Load Partition Workload

**Detector**

- **Goal 0**
- **Goal 1**
- **Goal 2**
- ... (Goal 
- **Violation Detector**
- **Failure Detector**
- **Anomaly Notifier**

**Executor**

- **Cluster Load Model**
- **Throttled Proposal**
- Execution

**Kafka Cluster**

- **Workload History Topic**
- **Metrics reporter**
- **Standard Metrics**
- **Broker Failures**
- **Throttled Proposal Execution**

**REST API**

- **Analyzer**
  - Goal 0
  - Goal 1
  - Goal 2
  - ...

**Pluggable Component**
**Cruise Control Architecture**

- **Executor** carries out the proposals generated by the analyzer
  - The execution
    - Should not impact existing user traffic
    - Should be interruptible
Cruise Control Architecture

- REST API – User interaction
- GUI is in development
Agenda

- Kafka introduction and terminologies
- Problems to solve
- Our solution
  - Cruise Control Architecture
  - Challenges and Solutions
- Insights and Generalization
  - Problem Generalization
  - Workload Model Generalization
Challenges

- Trustworthy Workload Modeling (Load Monitor)
- Complexity of Dynamic Workload Balancing (Analyzer)
- Fast Optimization Resolution (Analyzer)
- False Alarm in Failure (Failure Detector)
- Controlled Balancing Execution (Executor)
- And so on...

- See detailed discussion: https://www.slideshare.net/JiangjieQin/introduction-to-kafka-cruise-control-68180931
Challenges

- Trustworthy Workload Modeling (Load Monitor) ➔ Good Simulation
- Complexity of Dynamic Workload Balancing (Analyzer)
- Fast Optimization Resolution (Analyzer)
- False Alarm in Failure (Failure Detector)
- Controlled Balancing Execution (Executor)
- And so on...

- See detailed discussion: https://www.slideshare.net/JiangjieQin/introduction-to-kafka-cruise-control-68180931
Trustworthy Workload Modeling – An Example

- Optimize CPU for the following case
  - **Broker 0:** CPU=80%
    - T0P0: DISK=1 GB, NW_IN=10 MB/s, NW_OUT=30 MB/s
    - T1P0: DISK=1 GB, NW_IN=20 MB/s, NW_OUT=40 MB/s
    - T0P1: ..., T1P1: ...
  - **Broker 1:** CPU=30%
    - T0P0: DISK=1 GB, NW_IN=10 MB/s, NW_OUT=0 MB/s
    - T1P1: DISK=2 GB, NW_IN=30 MB/s, NW_OUT=60 MB/s
    - T1P0: ..., T0P1: ...

---

### Broker 0
```
+----------------+----------------+
| Topic0-Partition0 (Leader) |
| Topic1-Partition0 (Leader) |
| Topic0-Partition1 (Follower) |
| Topic1-Partition1 (Follower) |
+----------------+----------------+
```

### Broker 1
```
+----------------+----------------+
| Topic0-Partition0 (Follower) |
| Topic1-Partition0 (Follower) |
| Topic0-Partition1 (Leader) |
| Topic1-Partition1 (Leader) |
+----------------+----------------+
```
Trustworthy Workload Modeling – An Example

- Optimize CPU for the following case
  - Broker 0: CPU=80%
    - T0P0: DISK=1 GB, NW_IN=10 MB/s, NW_OUT=30 MB/s
    - T1P0: DISK=1 GB, NW_IN=20 MB/s, NW_OUT=40 MB/s
    - T0P1: ..., T1P1: ...
  - Broker 1: CPU=30%
    - T0P0: DISK=1 GB, NW_IN=10 MB/s, NW_OUT=0 MB/s
    - T1P1: DISK=2 GB, NW_IN=30 MB/s, NW_OUT=60 MB/s
    - T1P0: ..., T0P1: ...

- Move something from Broker 0 to Broker 1!
  - Let’s move the leader of Topic0-Partition0 to broker 1.
Trustworthy Workload Modeling – An Example

- Optimize CPU for the following case
  - Broker 0: CPU=80%
    - T0P0: DISK=1 GB, NW_IN=10 MB/s, NW_OUT=30 MB/s
    - T1P0: DISK=1 GB, NW_IN=20 MB/s, NW_OUT=40 MB/s
    - T0P1: ..., T1P1: ...
  - Broker 1: CPU=30%
    - T0P0: DISK=1 GB, NW_IN=10 MB/s, NW_OUT=0 MB/s
    - T1P1: DISK=2 GB, NW_IN=30 MB/s, NW_OUT=60 MB/s
    - T1P0: ..., T0P1: ...

- Move something from Broker 0 to Broker 1!
  - Let’s move the leader of Topic0-Partition0 to broker 1.

- What is the CPU utilization of Broker 1 after the movement? Should we move more?
Trustworthy Workload Modeling

- Some metrics can be easily aggregated
  - E.g. Bytes In Rate, Bytes Out Rate, Messages In Rate, etc.
- Some metrics are difficult to “aggregate”
  - E.g. CPU, Memory, Latency
  - We need to estimate or predict
- Cluster workload model should provide reliable simulation
Agenda

- Kafka introduction and terminologies
- Problems to solve
- How do we solve it
  - Cruise Control Architecture
  - Challenges and solutions

- **Insights and Generalization**
  - Problem generalization
  - Workload Model Generalization
Problem Generalization

- Rethink of the problem to solve
  - Given a topology
    - E.g. replica distribution, leader distribution
  - and associated metrics,
    - E.g. Partition Bytes In Rate, Partition Bytes Out Rate, Messages In Rate, Request Rate, etc.
  - optimize for some specific metrics
    - E.g. Broker CPU Usage, Broker DISK Usage, Broker Network IO Usage, Broker Memory Usage, Request Latency, etc.
Understand the metrics – Native Metrics

- Some of the metrics are natural attributes of a given system
  - E.g. the partition bytes in is only driven by the applications (assuming not using quota)
- Those metrics are Native Metrics
  - The values cannot be changed
  - The distribution CAN be changed
Understand the metrics – Causal Relationship

- The **causal relationship** between metrics
  - Some metrics are **caused by** other metrics
    - E.g. Broker CPU utilization is caused by Broker Bytes In Rate, Broker Messages In Rate, Broker Bytes Out Rate, Broker Request Rate, etc.
    - E.g. Broker Bytes In Rate is caused by Partition Bytes In Rate.
  - The metrics that causes other metrics are **Causal Metrics**
  - The metrics that are caused by other metrics are **Resultant Metrics**
**Metrics Dependency DAG**

![Diagram showing the dependency between different metrics such as Partition Message Rate, Broker Message Rate, CPU Utilization, Latency, and Memory Consumption, with arrows indicating the flow of dependency.](image)

- **Native/Causal (Partition Level)**
- **Causal/resultant (Broker Level)**
- **Resultant (Broker Level)**
Understand the metrics – Causal Relationship

- The causal relation has different representations
  - Simple aggregation
    - \( \text{BrokerBytesInRate} = \text{AllPartitionBytesInOnBroker} \)
  - More complicated function
    - \( \text{BrokerCpuUsage} = f(\text{BrokerBytesInRate}, \text{BrokerBytesOutRate}, \ldots) \)
Understand the metrics – Causal Relationship

- Define the causal relationship with a linear function:
  - Given Causal Metrics \( CM_1, CM_2, \ldots \) and Resultant Metric \( RM \)
  - \( RM = a_0 + a_1 \cdot CM_1 + a_2 \cdot CM_2 + \ldots \)

- A polynomial function can also be used
  - Can still be achieved through linear regression

- More complicated models are also possible
Understand the metrics

- Optimize CPU for the following case
  - Broker 0: **CPU=80%**
    - T0P0: DISK=1 GB, NW_IN=10 MB/s, NW_OUT=30 MB/s
    - T1P0: DISK=1 GB, NW_IN=20 MB/s, NW_OUT=40 MB/s
    - T0P1: ..., T1P1: ...
  - Broker 1: **CPU=30%**
    - T0P0: DISK=1 GB, NW_IN=10 MB/s, NW_OUT=0 MB/s
    - T1P1: DISK=2 GB, NW_IN=30 MB/s, NW_OUT=60 MB/s
    - T1P0: ..., T0P1: ...

- Move something from Broker 0 to Broker 1!
  - Let’s move the leader of Topic0-Partition0 to broker 1.

- What is the CPU utilization of Broker 1 after the move? Should we move more?
  - Derive CPU utilization from the causal relationship function
Resource Estimation Experiment

- **CPU estimation**
  - CPU usage: 0% - 35%
  - Synthetic traffic
  - Causal Metrics
    - LEADER_BYTES_IN
    - LEADER_BYTES_OUT
    - REPLICATION_BYTES_IN
    - REPLICATION_BYTES_OUT
    - MESSAGES_IN_RATE
    - PRODUCE_RATE
    - FETCH_RATE
- Online training and verification
Problem Generalization

- Rethink of the problem to solve
  - Given a topology
    - E.g. replica distribution, leader distribution
  - and associated metrics,
    - E.g. Partition Bytes In Rate, Partition Bytes Out Rate, Messages In Rate, Request Rate, etc.
  - Optimize for some specific metrics
    - E.g. Broker CPU Usage, Broker DISK Usage, Broker Network IO Usage, Broker Memory Usage, Request Latency, etc.
  - By changing topology (causal metric distribution)
    - E.g. leader movement, replica movement
  - Or changing configurations (causal relationship function)
    - E.g. Compression type, caching policy, etc.
Agenda

- Kafka introduction and terminologies
- Problems to solve
- How do we solve it
  - Cruise Control Architecture
  - Challenges and solutions
- **Insights and Generalization**
  - Problem generalization
  - **Workload Model Generalization**
Workload Model Generalization

Tree Structure Topology

Id: 0
Groups: G1
Tier: 0
Tags: ...

Id: 1
Groups: G2
Tier: 1
Tags: ...

Id: 2
Groups: G3
Tier: 1
Tags: ...

Id: 3
Groups: G4
Tier: 1
Tags: ...

Id: 4
Groups: G1, G4
Tier: 2
Tags: ...

Id: 5
Groups: G1, G3
Tier: 2
Tags: ...

Group Registry

G1

G2

G3

G3

Arbitrarily group the nodes in the topology together.
Workload Model Generalization

- **Topology Tree**
  - Physical hierarchy of the system
  - Ownership among nodes

- **Group Registry**
  - Logical grouping
  - Flat structure, no ownership
  - Quick access to a set of nodes

- **Key-value based model, easy to scale**
Insights and Generalization

- The architecture still works after removing Kafka specific parts.
Comparison to existing solutions

- Automatic cluster load balancing for stateful systems
  - Cloud management system (Kubernetes, Docker, etc.)
    - Balancing by moving the entire process
    - Application unaware
  - Highly customized system (e.g. Microsoft Azure Storage)
    - Partial state movement
    - Tightly coupled with a specific system
  - Cruise Control
    - Application aware
    - Generalized distributed system model
    - Partial state movement
    - Estimation and prediction
Future Work

- **Scalability**
  - Currently everything is in memory
  - Would like to abstract the cluster load model to use a K-V based interface
- **Integration with more projects (Apache Samza, Apache Helix, etc)**
- **Parallel computation of the optimization proposals**
- **GUI and multi-cluster management**
Links

- Cruise Control
  - [https://github.com/linkedin/cruise-control](https://github.com/linkedin/cruise-control) (github repo)
  - [https://gitter.im/kafka-cruise-control/Lobby](https://gitter.im/kafka-cruise-control/Lobby) (gitter room for questions)

- Other LinkedIn open source projects
  - [https://github.com/linkedin/](https://github.com/linkedin/)

- We are hiring!!!!!