Lab 5.8: Kafka Custom Partitioner

Welcome to the session 5 lab 8. The work for this lab is done in `~/kafka-training/lab5.8`. In this lab, you are going to set up a Kafka custom Partitioner.

Please refer to the Kafka course notes for any updates or changes to this lab.

Find the latest version of this lab here.

Lab Write Custom Partitioner

Next let's create a `StockPricePartitioner`. The `StockPricePartitioner` will implement a priority queue. It will treat certain stocks as important and send those stocks to the last partition. The `StockPricePartitioner` implements the Kafka interface `Partitioner`. The `Partitioner` interface is used to pick which partition a record lands. We will need to implement the `partition()` method to choose the partition. And we will need to implement the `configure()` method so we can read the `importantStocks` config property to setup `importantStocks` set which we use to determine if a stock is important and needs to be sent to the important partition. To do this we need to configure new `Partitioner` in Producer config with property `ProducerConfig.INTERCEPTOR_CLASSES_CONFIG`, and pass config property `importantStocks`.

Producer Partitioning

To set a custom partitioner set the Producer config property `partitioner.class`. The default `partitioner.class` is `org.apache.kafka.clients.producer.internals.DefaultPartitioner`. All Partitioner class implements the Kafka `Partitioner` interface and have to override the `partition()` method which takes topic, key, value, and cluster and then returns partition number for record.

StockPricePartitioner configure

`StockPricePartitioner` implements the `configure()` method with `importantStocks` config property. The `importantStocks` gets parsed and added to `importantStocks` HashSet which is used to filter the stocks.

~/kafka-training/lab5.8/src/main/java/com/cloudurable/kafka/producer/StockPriceKafkaProducer.java

Kafka Producer: StockPriceKafkaProducer configure partitioner

```
package com.cloudurable.kafka.producer;

import org.apache.kafka.clients.producer.Partitioner;
import org.apache.kafka.common.Cluster;
import org.apache.kafka.common.PartitionInfo;

import java.util.*;

public class StockPricePartitioner implements Partitioner{

    private final Set<String> importantStocks;
    public StockPricePartitioner() {
        importantStocks = new HashSet<>();
    }
    ...
```

```
    @Override
    public void configure(Map<String, ?> configs) {
        final String importantStocksStr = (String) configs.get("importantStocks");
        Arrays.stream(importantStocksStr.split(","))
                .forEach(importantStocks::add);
    }

}
```

ACTION - EDIT StockPricePartitioner and implement Partitioner interface

ACTION - EDIT StockPricePartitioner and finish configure method to read importantStocks

StockPricePartitioner partition()

IMPORTANT STOCK: If stockName is in the importantStocks HashSet then put it in partitionNum = (partitionCount -1) (last partition). REGULAR STOCK: Otherwise if not in importantStocks set then not important use the the absolute value of the hash of the stockName modulus partitionCount -1 as the partition to send the record `partitionNum = abs(stockName.hashCode()) % (partitionCount - 1)` .

~/kafka-training/lab5.8/src/main/java/com/cloudurable/kafka/producer/StockPricePartitioner.java

Kafka Producer: StockPricePartitioner partition

```
package com.cloudurable.kafka.producer;

public class StockPricePartitioner implements Partitioner{

    private final Set<String> importantStocks;
    public StockPricePartitioner() {
        importantStocks = new HashSet<>();
    }

    @Override
    public int partition(final String topic,
                         final Object objectKey,
                         final byte[] keyBytes,
                         final Object value,
                         final byte[] valueBytes,
                         final Cluster cluster) {

        final List<PartitionInfo> partitionInfoList =
                cluster.availablePartitionsForTopic(topic);
        final int partitionCount = partitionInfoList.size();
        final int importantPartition = partitionCount -1;
        final int normalPartitionCount = partitionCount -1;

        final String key = ((String) objectKey);

        if (importantStocks.contains(key)) {
            return importantPartition;
```

```
        } else {
            return Math.abs(key.hashCode()) % normalPartitionCount;
        }

    }
    ...
}
```

ACTION - EDIT StockPricePartitioner and finish partition method as described and show above.

Producer Config: Configuring Partitioner

~/kafka-training/lab5.8/src/main/java/com/cloudurable/kafka/producer/StockPricePartitioner.java

Kafka Producer: StockPriceKafkaProducer createProducer()

```
public class StockPriceKafkaProducer {

    private static Producer<String, StockPrice>
                                    createProducer() {
        final Properties props = new Properties();
        setupBootstrapAndSerializers(props);
        setupBatchingAndCompression(props);
        setupRetriesInFlightTimeout(props);

        ...

        props.put("importantStocks", "IBM,UBER");

        props.put(ProducerConfig.PARTITIONER_CLASS_CONFIG,
                StockPricePartitioner.class.getName());

        return new KafkaProducer<>(props);
    }
```

Configure the new `Partitioner` in Producer config with property `ProducerConfig.INTERCEPTOR_CLASSES_CONFIG`. Pass config property to `importantStocks`. The `importantStock` are the ones that go into priority queue. Run it as before. The important stocks are IBM and UBER in this example and are the only ones that will go into the last partition.

ACTION - EDIT StockPriceKafkaProducer and add `importantStocks` to the Producer config

ACTION - EDIT StockPriceKafkaProducer and set `PARTITIONER_CLASS_CONFIG` to `StockPricePartitioner`

Review of lab work

You implemented custom `ProducerSerializer`. You tested failover configuring broker/topic `min.insync.replicas`, and `acks`. You implemented batching and compression and used metrics to see how it was or was not working. You implemented retires and timeouts, and tested that it worked. You setup max inflight messages and retry back off. You implemented a `ProducerInterceptor`. You implemented a custom partitioner to implement a priority queue for important stocks.

ACTION - START ZooKeeper and Kafka Brokers as needed

ACTION - Run StockPriceKafkaProducer from the IDE

ACTION - Run SimpleStockPriceConsumer from the IDE

It should all work. :)