Documentation for Spring Boot application kafka-stream-count

1. Solution

1. Kafka installation
2. Topics created
3. Kafka producer from Kafka produces given test data to created topic
4. Application provides multiple api points for processing data from kafka
    a. /api/print_stream - consumes test data from topic and prints data to stdout
    b. /api/kafka_pipe - provides simple counting mechanism, using HashSet, HyperLogLog and Linear counting
    c. /api/read_produce - consumes data from original topic and produces data to new topic
5. Estimation
    a. /api/estimate_data - reads data from Kafka topic, and produces estimator from input stream
    b. /api/get_estimator – consumes estimated data from Kafka topic

Kafka installation

tar -xzf kafka_2.12-2.3.0.tgz

cd kafka_2.12-2.3.0

bin/zookeeper-server-start.sh config/zookeeper.properties

bin/kafka-server-start.sh config/server.properties

Kafka topic creation

bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic tamedia-topic-source

bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic tamedia-topic-destination

bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic tamedia-topic-estimation

List Kafka topics

bin/kafka-topics.sh --list --zookeeper localhost:2181

Delete Kafka topic

bin/kafka-topics.sh --delete --zookeeper localhost:2181 --topic test

Sending test data to Kafka topic using Kafka producer

zcat stream.jsonl.gz | head -1000 | bin/kafka-console-producer.sh --broker-list localhost:9092 --topic tamedia

View topic stream

bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic TOPIC_NAME --from-beginning

2. Use of suitable data structure

Cardinality estimation algorithms trade space for accuracy. The most used example is counting the number of distinct words in data using three different counting techniques, assuming that input dataset has extra data in it so the cardinality is higher than the standard reference answer to this question. The three techniques used are Java HashSet, Linear Probabilistic Counter, and a Hyper LogLog Counter. Results:

Counter	**Bytes Used**	**Count**	**Error**
HashSet	10447016	67801	0%
Linear	3384	67080	1%
HyperLogLog	512	70002	3%

The table shows that we can count the words with a 3% error rate using only 512 bytes of space. Compare that to a perfect count using a HashMap that requires nearly 10 megabytes of space and you can easily see why cardinality estimators are useful. In applications where accuracy is not paramount, which is true for most web scale and network counting scenarios, using a probabilistic counter can result in tremendous space savings.

3. Estimation

Estimator is apropriate bitmap structure (hash table) available in HyperLogLog and Linear counting algorithms. Idea is to produce estimator every minute (minimum time interval).

Record structure produced directly (online) from Kafka topic (stream reader) can be:

{“ts”:<timestamp>, “range“:<range>,”ec”:<ecvalue>,”est”:<estimator>}

Value <timestamp> unix time.
Value <range> is in minutes
Value <ecvalue> is parameter of HyperLogLog (<=30) or Linear counting (>30).
Value <estimator> is hex dump of serialized object (bitmap).

Output is written to resulting Kafka topic every minute.

To sum estimators, “ec” parametar (bitmap number of bytes in the case of Linear counting) should be the same.

Estimator addition is scalable. Cardinality estimation is based on resulting estimator for given time range. Resulting estimator has the same size as any estimator used in calculation.

Expected cardinality

Expected cardinality is the most important parameter of HyperLogLog and Linear counting algorithms.
HyperLogLog - an improved version of LogLog that is capable of estimating the cardinality of a set with accuracy = 1.04/sqrt(m) where expected cardinality is **m = 2^b**. So we can control accuracy vs space usage by increasing or decreasing **b** <= 30.

Linear counting – parameter n is size of bit array in bytes. Expected cardinality is **m = 8*n** So we can control accuracy vs space usage by increasing or decreasing **n**.

Query from application reads all records that match given criteria and performs estimator addition to produce resulting cardinality estimation.

To minimize error, resulting cardinality should be less than expected cardinality (provided as input parametar).