ET4310: SuperComputing For Big Data
Lab Assignment 3

Georgios Dimitropoulos: 4727657
Emmanouil Manousogiannis: 4727517
Group 29

November 2018

Outline of the Code

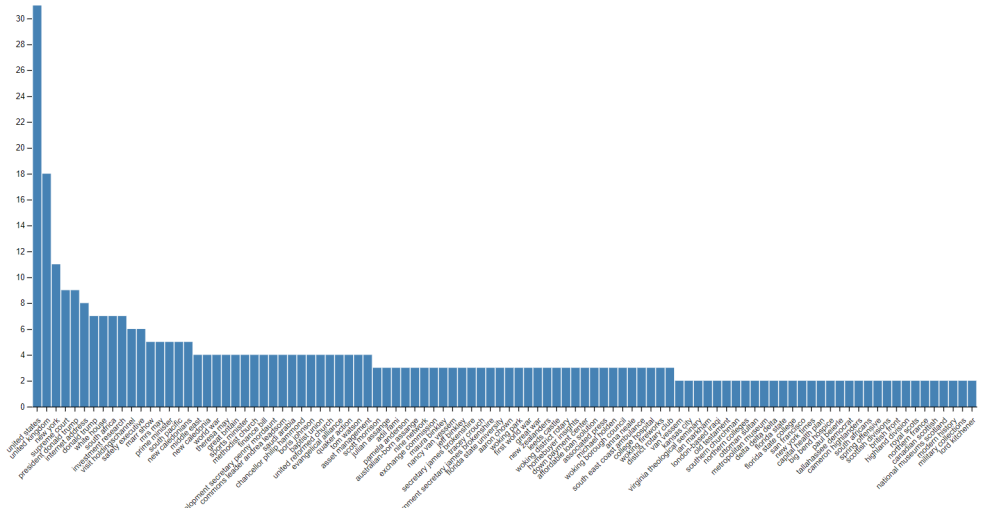

Figure 1: Screen-shot of the created histogram with top 100 topics

In this assignment we had to create a histogram that presents the top 100 discussed topics during the last hour, based on the GDelt data. Using Apache Kafka, our implementation architecture is described below.

The Kafka Producer, which was already provided, is reading the incoming data as a stream and stores it in a topic named 'gdelt'. Our main task is involved in the Transformer object, where we had to take the incoming stream, filter it so that we only keep the **topic value** and the **key** (String, String) and then transform it and finally store it in the 'GDelt histogram' topic as a pair of (String, Long), where String corresponds to the topic name and Long is the number of times this topic occurs.

As we already said, our first step was to filter the incoming stream and only keep the 'allNames' column value, which included the topic titles. This was almost the same as our task in lab assignment 1. The next step was to create a persistent KeyValueStore **state store** object, which is needed so that the processor can remember recently arrived records, perform queries in it etc.

After the state store was created, we had to implement the actual **Transformer** class , which is responsible for transforming our input stream from (key: String,value: String) format, to a (topic:String, value:Long) output stream. In the **transform** function of this class, for every arriving record, we queried its value from the created state store. For instance if a record with value "United States" arrived, we made a query to the state store to get how many times united states has been already seen as a topic. If the topic was not present, the value returned was 0. In both cases we increased the value of this topic by 1, and then updated the KeyValueStore state store

accordingly: **this.mystate.put(name, counter)**.

Finally we used the **context.scheduler()**, in order to schedule a decrease of this topic value, exactly one hour after. This is because we want the top 100 topics within the last hour to be present. After all the above is done, the output stream record is stored in "gdelt-histogram" topic.

Questions

General Kafka questions

1. What is the difference, in terms of data processing, between Kafka and Spark?

Answer: Apache Spark is a system that is mainly used for batch processing. Specifically, it can be used in order to process previously collected data in a long batch. Therefore, based on the fact that Spark processes the data in a batch-processing nature it is not well suited for real-time streaming. On the other hand, Kafka is a distributed streaming platform. Thus, it means that it processes the data in a streaming-fashion as they occur. For this reason, Kafka is a well-suited candidate for building real-time streaming applications.

2. What is the difference between replications and partitions?

Answer: Firstly, a topic is a stream of messages that belongs to a particular category. After that, the topics are split into partitions and the partition have messages in an unchangeable ordered sequence. The number of these partitions is equal to the number of the consumers that are in a consumer-group and the larger this number is, the greatest the parallelism that we have. On the other side of the spectrum, Replications are just copies of the partitions and do not write or read data. Hence, they are used for data redundancy as backups of partitions in order to prevent data loss.

3. What is Zookeeper's role in the Kafka cluster? Why do we need a separate entity for this? (max. 50 words)

Answer: Information is shared via a Zookeeper cluster by the Kafka servers and critical information about brokers, topics and so on is stored in Zookeeper. Zookeeper replicates this data and any failure of Zookeeper does not affect the state of the Kafka cluster. Therefore we need a separate entity for this.

4. Why does Kafka by default not guarantee exactly once delivery semantics on producers? (max. 100 words)

Answer: The default delivery semantics on producers is the at-least-once where the sender and the receiver participate to be sure that each message is delivered at least once and no message will end up unprocessed, though it could be processed multiple times. The holy grail is the the exact-once where all messages will be delivered exactly once. However, to guarantee exactly-once you should process a message and commit an offset in an atomic operation which is very hard to be done for instance in applications that run in parallel on multiple machines where failures are more likely.

5. Kafka is a binary protocol (with a reference implementation in Java), whereas Spark is a framework. Name two (of the many) advantages of Kafka being a binary protocol in the context of Big Data. (max. 100 words)

Answer: The first advantage of Kafka being a binary protocol in the context of Big data is that it offers a greater parallelism. More specifically, TCP connections are opened by a a Kafka client in multiple brokers in a cluster and the data are in parallel across a variety of partitions of a topic. A second advantage is that it provides a better network utilization as it is very compact in comparison to HTTP headers for instance that aggregate a larger size to messages.

Questions specific to the assignment

1. On average, how many bytes per second does the stream transformer have to consume? How many does it produce?

Answer: In this question, we thought it would be a good idea to monitor the performance of the Kafka producers and consumers. Based on that, we found the corresponding metrics: **"kafka.server:type=BrokerTopicMetrics,name=TotalProduceRequestsPerSec"** and **"kafka.network:type=RequestMetrics,name=TotalTimeMs, request=ProduceFetchConsumer|FetchFollow"** which compute the total produce request rate and the total time in ms to serve the specified request respectively. However, we did not manage to implement these computations.

2. Could you use a Java/Scala data structure instead of a Kafka State Store to manage your state in a processor/transformer? Why, or why not? (max. 50 words)

Answer: One alternative solution would be to to use a HashMap data structure. However, this would make our state non-persistent and there is a high probability to be lost in case the stream processor crashes. Thus, when the processing restarts, the content of HashMap will be vacant.

3. Given that the histogram is stored in a Kafka StateStore, how would you extract the top 100 topics? Is this efficient? (max. 75 words)

Answer: We could perform a query to get the first 100 entries in the statestore and sort them based on their value. Then we compare every other entry in the statestore with the minimum 'retrieved' 100 and exchange them if the value of the new entry is larger and repeat. However, we should also be able to store all of our data in one partition, otherwise we would have to do that computation for all the created partitions. This of course is not efficient, because storing everything in one partition does not allow parallelism.

4. The visualizer draws the histogram in your web browser. A Kafka consumer has to communicate the current 'state' of the histogram to this visualizer. What do you think is an efficient way of streaming the 'state' of the histogram to the web-server? (max. 75 words)

Answer: In order to expose the 'state' of the histogram to the web-server we could add a Remote Procedure Call (RPC) layer to our application and then expose the RPC endpoints via the application.server so that we can make the 'state' of our histogram discoverable by the web-server.

5. What are the two ways you can scale your Kafka implementation over multiple nodes? (max. 100 words)

Answer: The first way to scale our Kafka implementation over multiple nodes would be to add more partitions to the created topics and distribute those partitions in the available nodes. Then we could use multiple consumers, who would 'read' from those nodes in parallel.

Another method, would be to create a number of stream tasks, which would be equal to the number of the topic partitions. In that way each task would be assigned with one partition from the input topic, and as a result all of them would be processed in parallel.

6. How could you use Kafka's partitioning to compute the histogram in parallel? (max. 100 words)

Answer: The most common technique in topic partitioning is the random one. All records are randomly split to different partitions of the topic. However, since we want to count how many times each specific topic appears in the stream, we prefer to use partition by aggregate. In that way, every record that corresponds to a specific query identifier, will be assigned to the same partition. Hence, to count the number of times that a topic occurs, it will be enough to perform computations in the partition that all relevant records are assigned to. Otherwise, we should combine data from different partitions.

References

[1] *https://jack-vanlightly.com/blog/2017/12/15/rabbitmq-vs-kafka-part-4-message-delivery-semantics-and-guarantees*

[2] *https://developer.lightbend.com/blog/2017-11-01-atotm-akka-messaging-part-2/index.html*

[3] *https://www.confluent.io/blog/enabling-exactly-kafka-streams/*

[4] *https://kafka.apache.org/intro*