Big Data Assignment 5

1. Perform random data lookup and streaming in SPARK?

When processing streaming data, the raw data from the events are often not sufficient. Additional data must be added in most cases, for example metadata for a sensor, of which only the ID is sent in the event.

I would like to discuss various ways to solve this problem in Spark Streaming. The examples assume that the additional data is initially outside the streaming application and can be read over the network – for example in a database. All samples and techniques refer to Spark Streaming and not to Spark Structured Streaming. The main techniques are

broadcast: static data

mapPartitions: for volatile data

mapPartitions + connection broadcast: effective connection handling

mapWithState: speed up by a local state

- **Broadcast**

Spark has an integrated broadcasting mechanism that can be used to transfer data to all worker nodes when the application is started. This has the advantage, in particular with large amounts of data, that the transfer takes place only once per worker node and not with each task.

However, because the data can not be updated later, this is only an option if the metadata is static. This means that no additional data, for example information about new sensors, may be added, and no data may be changed. In addition, the transferred objects must be serializable.

In this example, each sensor type, stored as a numerical ID (1,2, …), is to be replaced by a plain-text name in the stream processing (tire temperature, tire pressure, ..). It is assumed that the assignment type ID -> name is fixed.

- **MapPartitions**

The first way to read non-static data is in a map() operation. However, not map() should be used but mapPartitions(). mapPartitions() is not called for every single element, but for each partition, which then contains several elements. This allows to connect to the database only once per partition and then to reuse the connection for all elements.

There are two different ways to query the data: Use a bulk API to process all elements of the partition together, or an asynchronous variant: an asynchronous, non-blocking query is issued for each entry and the results are then collected.

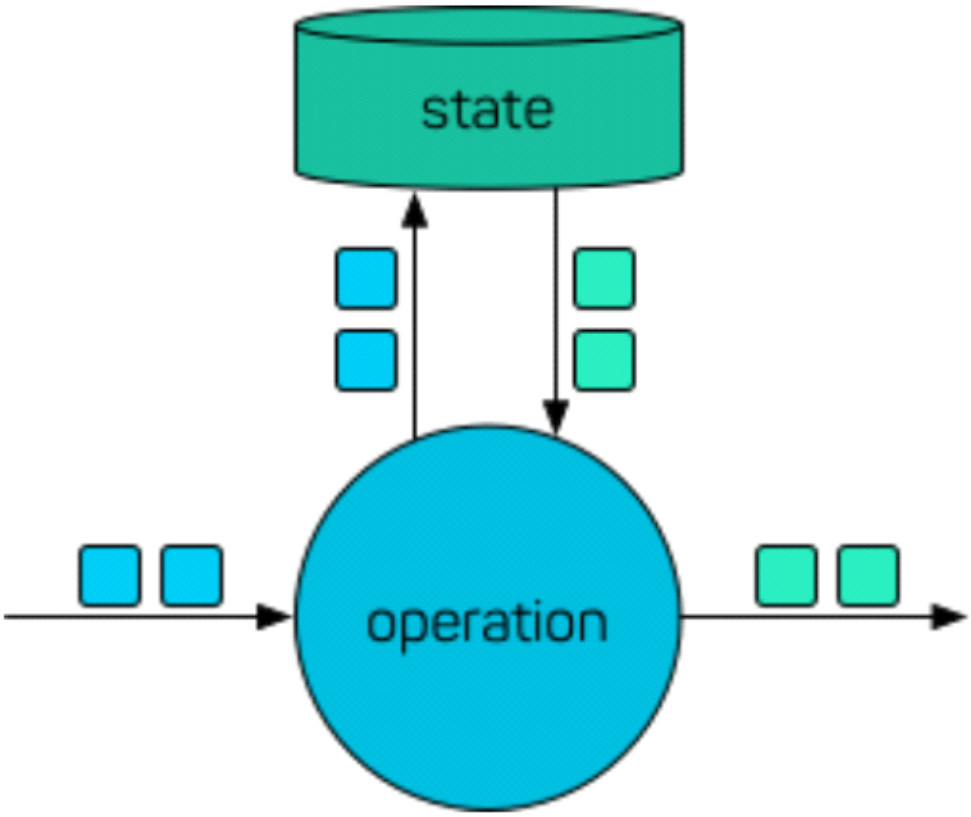
- **Broadcast Connection + MapPartitions**

However, it is a good idea not to rebuild the connection for each partition, but only once per worker node. To achieve this, the connection is not broadcasted because it is not serializable (see above), but instead a factory that builds the connection on the first call and then returns this connection on all other calls. This function is then called in mapPartitions() to get the connection to the database.

In Scala it is not necessary to use a function for this. Here a lazy val can be used. The lazy val is defined within a wrapper class. This class can be serialized and broadcasted. On the first call, an instance of the non-serializable connection class is created on the worker node and then returned for every subsequent call.

- **MapWithState()**

All solution approaches shown so far retrieve the data from a database, if necessary. This usually means a network call for each entry or at least for each partition. It would be more efficient to have the data directly in-memory available.

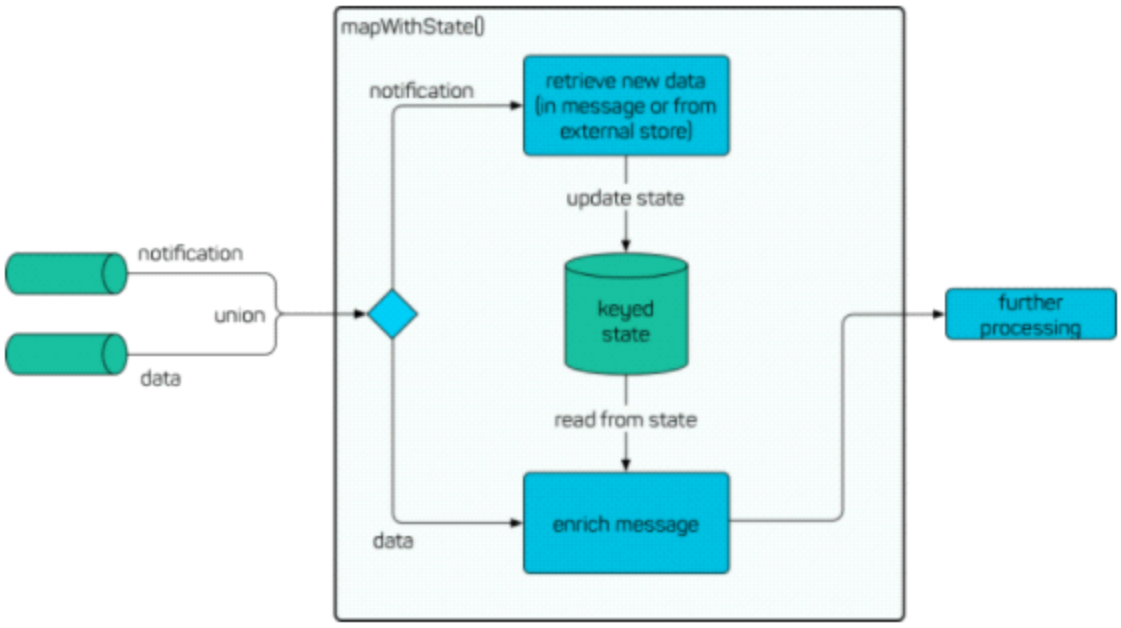


With mapWithState() Spark itself offers a way to change data by means of a state and, in turn, also to adjust the state. The state is managed by a key. This key is used to distribute the data in the cluster, so that all data must not be kept on each worker node. An incoming stream must therefore also be constructed as a key-value pair.

This keyed state can also be used for a lookup. By means of initialState(), an RDD can be passed as an initial state. However, any updates can only be performed based on a key. This also applies to deleting entries. It is not possible to completely delete or reload the state.

To update the state, additional notification events must be present in the stream. These can, for example, come from a separate Kafka topic and must be merged with the actual data stream (union()). The amount of data sent, can range from a simple notification with an ID, which is then used to read the new data, to the complete new data set.

Messages are published to the Kafka topic, for example, if metadata is updated or newly created. In addition, timed events can be published to the Kafka topic or can be generated by a custom receiver in Spark itself.



A simple implementation can look like this. First, the Kafka topics are read and the keys are additionally supplemented with a marker for the data type (data or notification). Then, both streams are merged into a common stream and processed in mapWithState(). The state was previously specified by passing the function of the state to the StateSpec.

The lookupWithState function describes the processing in the state. The following parameters are passed:

batchTime: the start time of the current microbatch

key: the key, in this case the original key from the stream, together with the type marker (data or notification)

valueOpt: the value to the key in the stream

state: the value stored in the state for the key

A tuple consisting of the original key and the original value as well as a number will be returned. The number is taken from the state or – if not already present in the state – is chosen randomly.

In addition, the timeout mechanism of the mapWithState() can also be used to remove events after a certain time without updating from the state.

2. What are the benefits of partitioning in Hive? Could you give an

example?

Apache Hive organizes tables into partitions. Partitioning is a way of dividing a table into related parts based on the values of particular columns like date, city, and department.

Each table in the hive can have one or more partition keys to identify a particular partition. Using partition it is easy to do queries on slices of the data.

Hive Partitioning Benefits

1. Partitioning in Hive distributes execution load horizontally.
2. In partition faster execution of queries with the low volume of data takes place. For example, search population from Vatican City returns very fast instead of searching entire world population.

Hive Data Partitioning Example

Now let's understand data partitioning in Hive with an example. Consider a table named Tab1. The table contains client detail like id, name, dept, and yoj(year of joining). Suppose we need to retrieve the details of all the clients who joined in 2012.

Then, the query searches the whole table for the required information. But if we partition the client data with the year and store it in a separate file, this will reduce the query processing time. The below example will help us to learn how to partition a file and its data-

The file name says file1 contains client data table:

[php]tab1/clientdata/file1

id, name, dept, yoj

1, sunny, SC, 2009

2, animesh, HR, 2009

3, sumeer, SC, 2010

4, sarthak, TP, 2010[/php]

Now, let us partition above data into two files using years

[php]tab1/clientdata/2009/file2

1, sunny, SC, 2009

2, animesh, HR, 2009

tab1/clientdata/2010/file3

3, sumeer, SC, 2010

4, sarthak, TP, 2010[/php]

Now when we are retrieving the data from the table, only the data of the specified partition will be queried. Creating a partitioned table is as follows:

[php]CREATE TABLE table_tab1 (id INT, name STRING, dept STRING, yoj INT) PARTITIONED BY (year STRING);

LOAD DATA LOCAL INPATH tab1'/clientdata/2009/file2'OVERWRITE INTO TABLE studentTab PARTITION (year='2009');

LOAD DATA LOCAL INPATH tab1'/clientdata/2010/file3'OVERWRITE INTO TABLE studentTab PARTITION (year='2010');[/php]

3. What commands should I use to start and stop Hadoop daemons?

start-all.sh & stop-all.sh : Used to start and stop hadoop daemons all at once.

Issuing it on the master machine will start/stop the daemons on all the nodes of a cluster. Deprecated as you have already noticed.

4. What exactly is in-memory computing, and how does Spark achieve it?

In-memory cluster computation enables Spark to run iterative algorithms, as programs can checkpoint data and refer back to it without reloading it from disk; in addition, it supports interactive querying and streaming data analysis at extremely fast speeds.

Because Spark is compatible with YARN, it can run on an existing Hadoop cluster and access any Hadoop data source, including HDFS, S3, HBase, and Cassandra.

5. Could you elaborate on Hadoop configuration files?

Configuration Files are the files which are located in the extracted tar.gz file in the etc/hadoop/ directory.

1) **HADOOP-ENV.sh->>** It specifies the environment variables that affect the JDK used by Hadoop Daemon (bin/hadoop). We know that Hadoop framework is wriiten in Java and uses JRE so one of the environment variable in Hadoop Daemons is $Java_Home in Hadoop-env.sh.

2) **CORE-SITE.XML->>** It is one of the important configuration files which is required for runtime environment settings of a Hadoop cluster.It informs Hadoop daemons where the NAMENODE runs in the cluster. It also informs the Name Node as to which IP and ports it should bind.

3) **HDFS-SITE.XML->>** It is one of the important configuration files which is required for runtime environment settings of a Hadoop. It contains the configuration settings for NAMENODE, DATANODE, SECONDARYNODE. It is used to specify default block replication. The actual number of replications can also be specified when the file is created,

4) **MAPRED-SITE.XML->>** It is one of the important configuration files which is required for runtime environment settings of a Hadoop. It contains the configuration settings for MapReduce . In this file, we specify a framework name for MapReduce, by setting the MapReduce.framework.name.

5) **Masters->>** It is used to determine the master Nodes in Hadoop cluster. It will inform about the location of SECONDARY NAMENODE to Hadoop Daemon.

The Mater File on Slave node is blank.

6) **Slave->>** It is used to determine the slave Nodes in Hadoop cluster.

The Slave file at Master Node contains a list of hosts, one per line.

The Slave file at Slave server contains IP address of Slave nodes.