Intro to Spark Streaming with sparklyr

INTRO

In my last presentation we talked about what is R, RStudio, Spark and Sparklyr in order to set up step by step a cluster and monitoring a streaming with spark.

Also, we explained the difference between .R script with the commands and the insertion of the chuck of these commands in a R notebook in Rmarkdown language.

So, today we are going to keep learning this useful and modern interface named Sparklyr and do kind of more advanced operations in our streaming process.

Let's check the output to see what is the main objective and why we should do monitoring.

        - sc <- spark_connect(master = "local", spark_version = "2.3.0") no need to specify version
        - See that the connection has been established and open the interface
        - Only one job with id 0 for the establishment but we are going to see more when the script is executed but we can look at the environment that can be useful and the completed queries, client,fifo,versions.
        - So we are going to look at this again to evaluate the jobs and know if the operation is successful or not so it is basically the spark control user interface by using the link supplied by our studio cluster.
    - Execute all and open t see that it is running and the grafic and each job is triggered by and action like if we were using memory, saving a file, but know we are doing a streaming.
    - Stop it to show how it stops a d disconnect

For understanding what we are going to do today, may I remind you the steps of spark to make a streaming:

    - Spark does it in 3 steps
        - 1. Read the data inside a folder that have multiple data files and new files that are being created and this is the streaming data transmitted.
        - 2. Manipulate the data if needed for example to predict something but is not used here.
        - 3. Save the output in different folder selected.

Last time, we omitted step 2, so no manipulation of the data occurred. As I have said, why to manipulate the data? Sometimes it not only the stremming the objective, mauve we have to predict

something or need to make operations to extract insights. In this session, our manipulation of the data is a processing, when?: THE INPUT DATA IS MANIPULATED BEFORE SAVING IT TO THE OUTPUT FOLDER.

EXPLAIN DE CODE:

1. Future: test generation function will run 100 files every 0.2 seconds. To run the tests “out-of-sync” with the current R session, the future package is used. Invisible : This function can be useful when it is desired to have functions return values which can be assigned, but which do not print when they are not assigned.Library future:

> The state of a future can either be *unresolved* or *resolved*. As soon as it is resolved, the value is available instantaneously. If the value is queried while the future is still unresolved, the current process is *blocked* until the future is resolved. It is possible to check whether a future is resolved or not without blocking.
> provide a very simple and uniform way of evaluating R expressions asynchronously, very useful for parallel programming.

2. **Sparklyr:** Basically is an interface for R language to connect to Apache spark, and is a package available at CRAN repository.
How to install with install.packages(“sparklyr”)

3. New! : library(dplyr, warn.conflicts = FALSE): **dplyr** is a new package which provides a set of tools for efficiently manipulating datasets in R. **dplyr** is the next iteration of plyr , focussing on only data frames. **dplyr** is faster.

4. Processing of data begins! What is the processing that is done here?
A new binary field is added to indicate if the value from x is over 400 or not, so is like we are going to check all values named as x and determine if they are over 400 and save this in a new field, it is recommended to save this information as binary.

Analyse the chunk.

%>% it can be misunderstanding defined by dplyr package, it can be seen as %.% but it is deprecated now and %<% is used because dplyr like has imported from another package but what we need to know is that is a pipe so it pipes the value of an expression into a function.

Mutate, also from dplyr package, it is for adding new variables and also preserve the existing ones.

Ft_binarizer, Apply thresholding to a column, such that values less than or equal to the threshold are assigned the value 0.0, and values greater than the threshold are assigned the value 1.0. Column output is numeric for compatibility with other modeling functions.

This is a good illustration of how dplyr can help simplify the manipulation needed during the processing stage.

5. `stream_view(write_output)`: It will monitor the status of the job that `write_output` is pointing to and provide information on the amount of data coming into the “source” folder and going out into the “source-out” folder.

REMARK: for using future package is going to provide info till x test are being completed.

6. Invisible : This function can be useful when it is desired to have functions return values which can be assigned, but which do not print when they are not assigned.

- Future: test generation function will run 100 files every 0.2 seconds. To run the tests “out-of-sync” with the current R session, the future package is used.

EXECUTE TILL stream_view and next, multiple times:

```
spark_read_csv(sc, "stream", "source-out", memory = FALSE) %>%
  group_by(over) %>%
 tally()
```

We are going to group “over” to make operations as a groups and then, uses tally() as a wrapper to obtain n that is the sum, totals will increase as the experiment progresses as we can see in the console.

But what is the main objective of this chunk?
The “source-out” folder can be treated as a if it was a single table within Spark. Using spark_read_csv(), the data can be mapped, but not brought into memory (memory = FALSE). This allows the current results to be further analyzed using regular dplyr commands.

FINISH: the monitoring will continue even the test are completed so we have to finish:

```
stream_stop(write_output)
spark_disconnect(sc)
```

OTHER TYPE OF MANIPULATING: AGGREGATION IN PROCESS AND OUTPUT TO MEMORY

What’s new? No need of souce-out folder because we are going to output to memory, in a in-memory Spark table.the table and its data will only exist while the Spark session is active.

The processing is different, The biggest advantage of using Spark memory as the target, is that it will allow for aggregation to happen during processing.

this example code will perform some aggregations to the current stream input and save only those summarized results into Spark memory:

timestamp variable that is then used in the group_by() command. This is required by Spark Stream to accept summarized results as output of the stream. The second step is to simply decide what kinds of aggregations we need to perform. In this case, a simply max, min and count are performed.

The spark_write_memory() function is used to write the output to Spark memory. The results will appear as a table of the Spark session with the name assigned in the name argument, in this case the name selected is: “stream”.

To query the current data in the “stream” table can be queried by using the dplyr tbl() command.

Now see the table and monitoring see that takes time the processing

---

Shiny: Shiny is an open source R package that provides an elegant and powerful web framework for building web applications using R. Shiny helps you turn your analyses into interactive web applications without requiring HTML, CSS, or JavaScript knowledge.

Notice that there is no stream_write_... command. The reason is that reactiveSpark() function contains the stream_write_memory() function.

In the server section, the reactiveSpark() function will update every time there’s a change to the stream and return a data frame. The results are saved to a variable called ps() in this script. Treat the ps() variable as a regular table that can be piped from, as shown in the example. In this case, the timestamp variable is converted to string for to make it easier to read.

Use runGadget() to display the Shiny app in the Viewer pane. This is optional, the app can be run using normal Shiny run functions.