CS523-BDT-Final Project on SOCCOR ANALYSER

SUBMITTED BY:

PUSKAR BUDHATHOKI, MHRETEAB ADHANOM BERHE , NATNAEL BERHE

SUBMITTED TO: PROF. MRUDULA MUKADAM

Project Overview

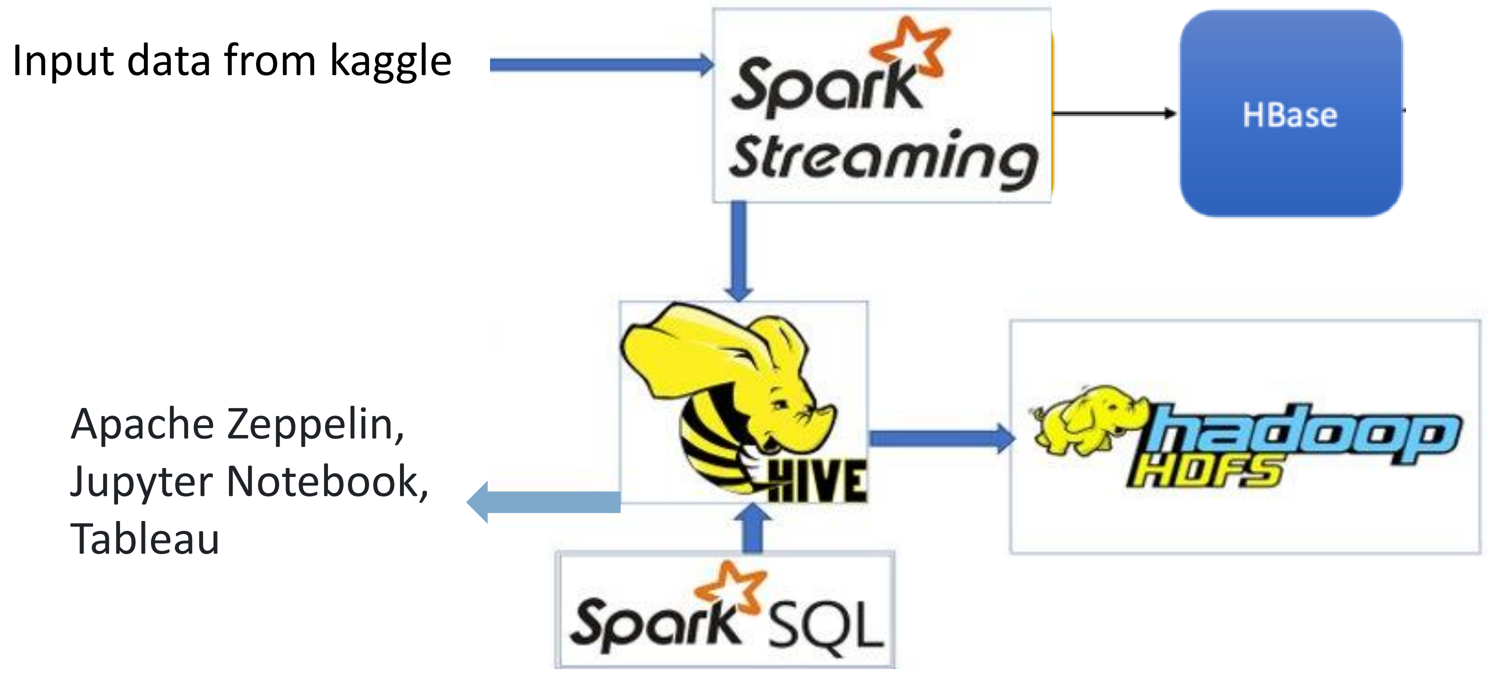
Dataset: we have used International Football Results from 1872 to 2023 dataset from popular website kaggle (https://www.kaggle.com/datasets/martj42/international-football-results-from-1872-to-2017?datasetId=4305)

Kafka: Distributed messaging system which is used for streaming analytics and data integration(Demo Project)

Spark: It is used for getting the dataset and performing some filter and aggregation

Hive/Hbase: The enriched result is saved in Hbase/Hive.

Project Overview

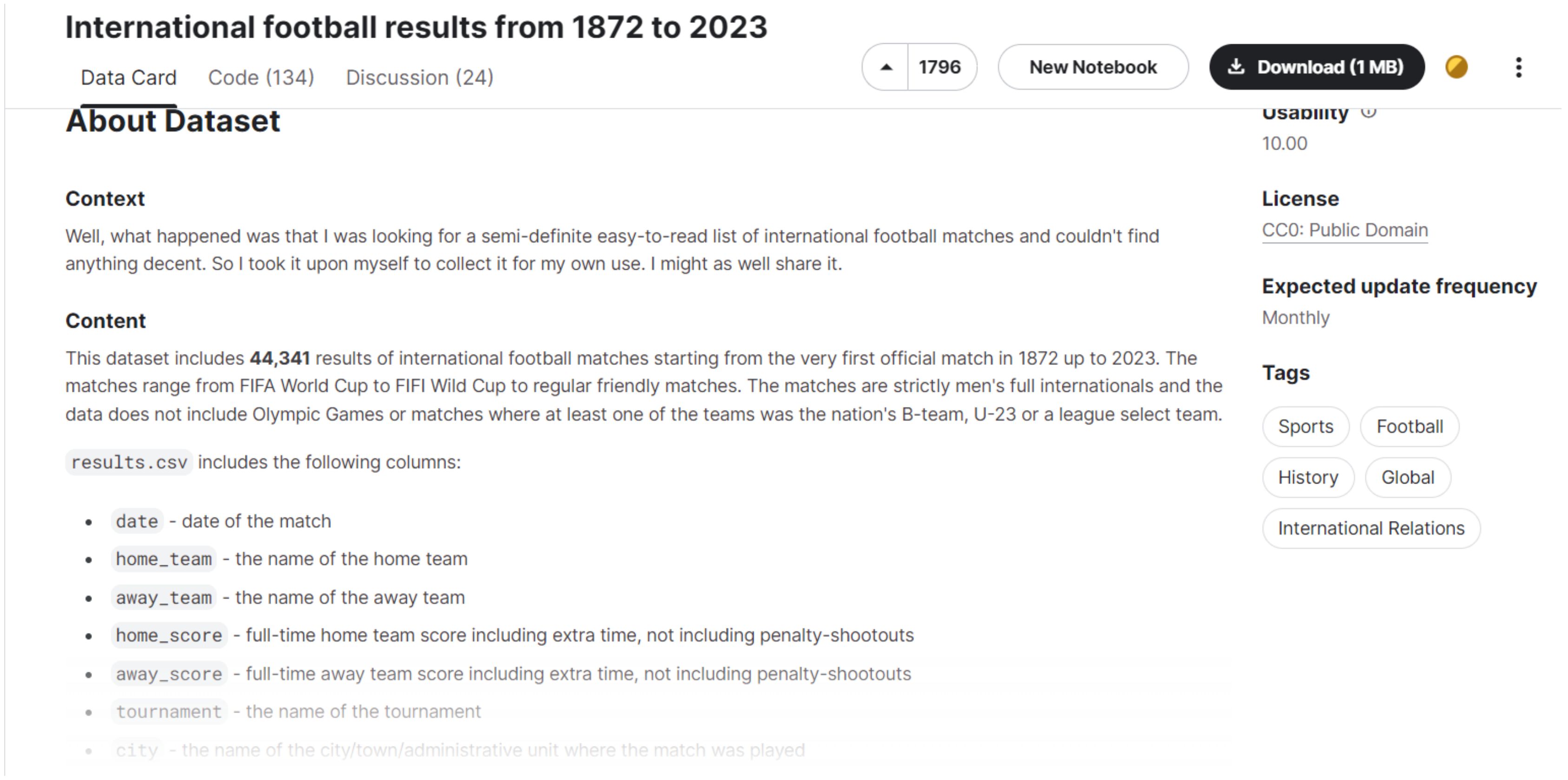


Technology stack used

- Kafka – xxx
- Spark streaming – 2.4.4
- Spark core – 2.4.4
- Java - 8
- Hadoop – 2.6.0
- Hbase- 2.2.5
- Hive – xxx

Dataset used

- Used static football dataset from kaggle
- https://www.kaggle.com/datasets/martj42/international-football-results-from-1872-to-2017?datasetId=4305
- Created a program to split the whole dataset into smaller datset to use apache data streaming



International football results from 1872 to 2023

About Dataset

Context

Well, what happened was that I was looking for a semi-definite easy-to-read list of international football matches and couldn't find anything decent. So I took it upon myself to collect it for my own use. I might as well share it.

Content

This dataset includes **44,341** results of international football matches starting from the very first official match in 1872 up to 2023. The matches range from FIFA World Cup to FIFI Wild Cup to regular friendly matches. The matches are strictly men's full internationals and the data does not include Olympic Games or matches where at least one of the teams was the nation's B-team, U-23 or a league select team.

`results.csv` includes the following columns:

- `date` - date of the match
- `home_team` - the name of the home team
- `away_team` - the name of the away team
- `home_score` - full-time home team score including extra time, not including penalty-shootouts
- `away_score` - full-time away team score including extra time, not including penalty-shootouts
- `tournament` - the name of the tournament
- `city` - the name of the city/town/administrative unit where the match was played

Usability
10.00

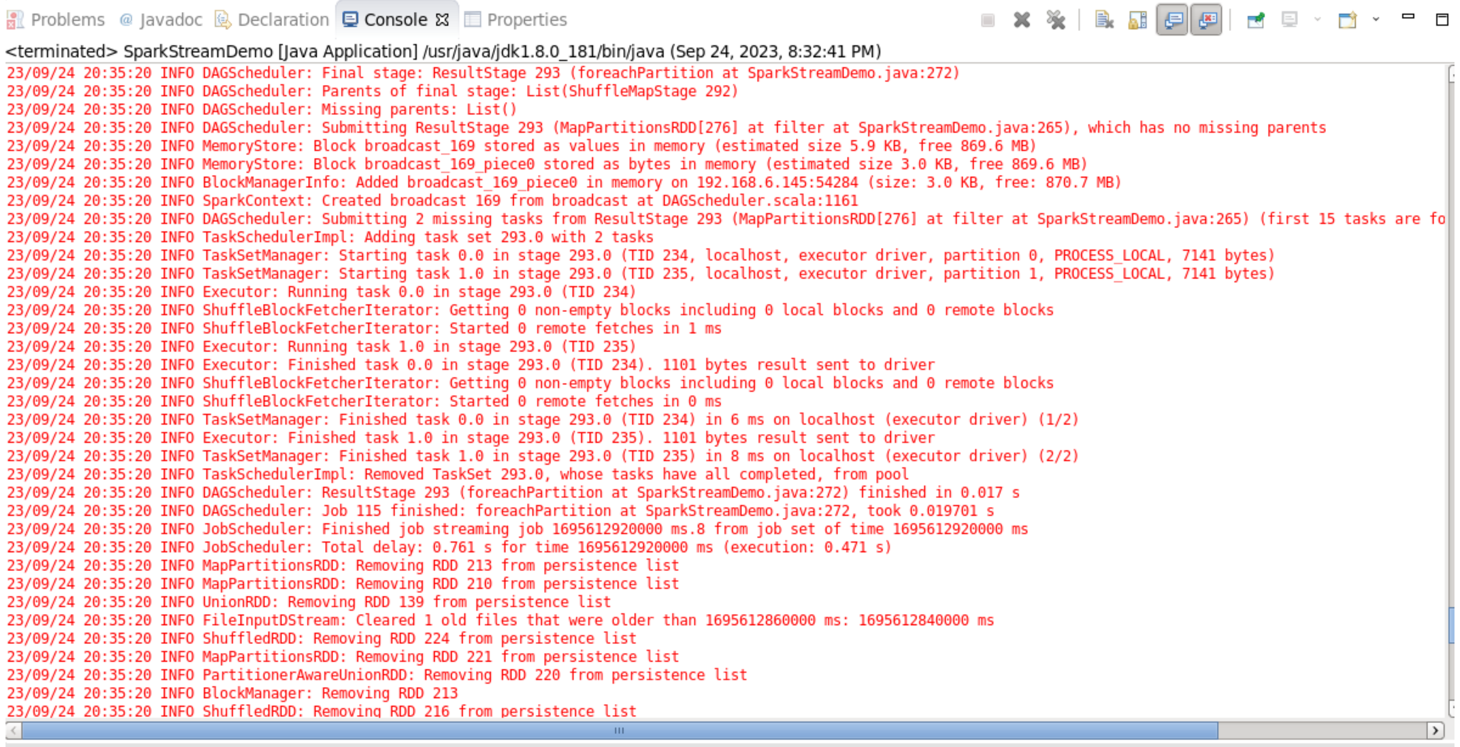
License
CC0: Public Domain

Expected update frequency
Monthly

Tags
Sports
Football
History
Global
International Relations

Apache Data Streaming sample

- Received csv file as text stream input and allocate infrastructure to complete the allocated jobs



```
<terminated> SparkStreamDemo [Java Application] /usr/java/jdk1.8.0_181/bin/java (Sep 24, 2023, 8:32:41 PM)
23/09/24 20:35:20 INFO DAGScheduler: Final stage: ResultStage 293 (foreachPartition at SparkStreamDemo.java:272)
23/09/24 20:35:20 INFO DAGScheduler: Parents of final stage: List(ShuffleMapStage 292)
23/09/24 20:35:20 INFO DAGScheduler: Missing parents: List()
23/09/24 20:35:20 INFO DAGScheduler: Submitting ResultStage 293 (MapPartitionsRDD[276] at filter at SparkStreamDemo.java:265), which has no missing parents
23/09/24 20:35:20 INFO MemoryStore: Block broadcast_169 stored as values in memory (estimated size 5.9 KB, free 869.6 MB)
23/09/24 20:35:20 INFO MemoryStore: Block broadcast_169_piece0 stored as bytes in memory (estimated size 3.0 KB, free 869.6 MB)
23/09/24 20:35:20 INFO BlockManagerInfo: Added broadcast_169_piece0 in memory on 192.168.6.145:54284 (size: 3.0 KB, free: 870.7 MB)
23/09/24 20:35:20 INFO SparkContext: Created broadcast 169 from broadcast at DAGScheduler.scala:1161
23/09/24 20:35:20 INFO DAGScheduler: Submitting 2 missing tasks from ResultStage 293 (MapPartitionsRDD[276] at filter at SparkStreamDemo.java:265) (first 15 tasks are fo
23/09/24 20:35:20 INFO TaskSchedulerImpl: Adding task set 293.0 with 2 tasks
23/09/24 20:35:20 INFO TaskSetManager: Starting task 0.0 in stage 293.0 (TID 234, localhost, executor driver, partition 0, PROCESS_LOCAL, 7141 bytes)
23/09/24 20:35:20 INFO TaskSetManager: Starting task 1.0 in stage 293.0 (TID 235, localhost, executor driver, partition 1, PROCESS_LOCAL, 7141 bytes)
23/09/24 20:35:20 INFO Executor: Running task 0.0 in stage 293.0 (TID 234)
23/09/24 20:35:20 INFO ShuffleBlockFetcherIterator: Getting 0 non-empty blocks including 0 local blocks and 0 remote blocks
23/09/24 20:35:20 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 1 ms
23/09/24 20:35:20 INFO Executor: Running task 1.0 in stage 293.0 (TID 235)
23/09/24 20:35:20 INFO Executor: Finished task 0.0 in stage 293.0 (TID 234). 1101 bytes result sent to driver
23/09/24 20:35:20 INFO ShuffleBlockFetcherIterator: Getting 0 non-empty blocks including 0 local blocks and 0 remote blocks
23/09/24 20:35:20 INFO ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
23/09/24 20:35:20 INFO TaskSetManager: Finished task 0.0 in stage 293.0 (TID 234) in 6 ms on localhost (executor driver) (1/2)
23/09/24 20:35:20 INFO Executor: Finished task 1.0 in stage 293.0 (TID 235). 1101 bytes result sent to driver
23/09/24 20:35:20 INFO TaskSetManager: Finished task 1.0 in stage 293.0 (TID 235) in 8 ms on localhost (executor driver) (2/2)
23/09/24 20:35:20 INFO TaskSchedulerImpl: Removed TaskSet 293.0, whose tasks have all completed, from pool
23/09/24 20:35:20 INFO DAGScheduler: ResultStage 293 (foreachPartition at SparkStreamDemo.java:272) finished in 0.017 s
23/09/24 20:35:20 INFO DAGScheduler: Job 115 finished: foreachPartition at SparkStreamDemo.java:272, took 0.019701 s
23/09/24 20:35:20 INFO JobScheduler: Finished job streaming job 1695612920000 ms.8 from job set of time 1695612920000 ms
23/09/24 20:35:20 INFO JobScheduler: Total delay: 0.761 s for time 1695612920000 ms (execution: 0.471 s)
23/09/24 20:35:20 INFO MapPartitionsRDD: Removing RDD 213 from persistence list
23/09/24 20:35:20 INFO MapPartitionsRDD: Removing RDD 210 from persistence list
23/09/24 20:35:20 INFO UnionRDD: Removing RDD 139 from persistence list
23/09/24 20:35:20 INFO FileInputDStream: Cleared 1 old files that were older than 1695612860000 ms: 1695612840000 ms
23/09/24 20:35:20 INFO ShuffledRDD: Removing RDD 224 from persistence list
23/09/24 20:35:20 INFO MapPartitionsRDD: Removing RDD 221 from persistence list
23/09/24 20:35:20 INFO PartitionerAwareUnionRDD: Removing RDD 220 from persistence list
23/09/24 20:35:20 INFO BlockManager: Removing RDD 213
23/09/24 20:35:20 INFO ShuffledRDD: Removing RDD 216 from persistence list
```

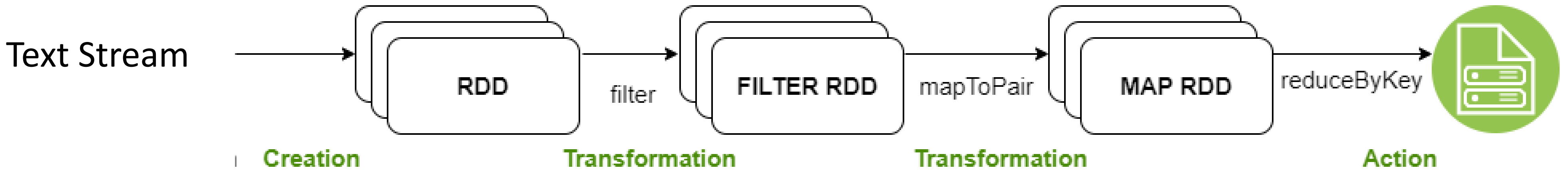
Fig 2: Apache data stream console

Apache Kafka

- Consist xxx brokers (Kafka cluster) and manage by zookeeper
- Detail description and screenshot goes here

Spark Streaming

- Get international football statistics from the split dataset
- Filter required columns like home_team, away_team, home_goal, away_goal, city, match_type etc
- Calculate total scores by each country, frequency of match type, organizing city with frequency
- Filter data based on threshold values for different use cases



Apache HBASE/Hive

- The output data from spark (aggregated) is persist in HBase/Hive database
- Detail description and screenshot goes here

Project execution steps

Step 1(Data preparation):

- Created a account in kaggle (https://www.kaggle.com/)
- Downloaded static dataset related to international football
- split the dataset into smaller files so that we can apply data streaming and processing operations

Step 2(Data feed to Apache Spark Data Streaming):

- The pipeline is written in shellscript iwhich fetch data from local directory in cloudera VM to the Apache Data Streaming in certain interval of time

Step 3(Real-time Analysis with Apache Spark Streaming):

- Ingest data in every 20 seconds.
- Text file streaming is used to load data.
- Different stream function like filter, maptoPair, Union, reducebykey etc are used.
- complex data structure with multiple Tuple are used to get the aggregate data from input data.

Project execution steps

Step 4(Data Processing with Apache Hive):

- Schema Definition: Defined a Hive schema that matches the structure of the output result from Apache data streaming

- Data Loading: Load the CSV data into a Hive table using Hive's LOAD DATA command or other methods for bulk data loading.

- Data Transformation: Use Hive's SQL-like language, HiveQL, to perform various data transformations and aggregations. we calculated statistics, filter data by date, tournament, or teams, and perform other necessary preprocessing.

- Querying: We ran Hive queries to extract insights from the data. For example, we found the countries with the most international goals, city and country organizing most games, the pattern of football match in international tournament, and more.

Project execution steps

Step 5(Data Export to HBase):

- For the demonstration, the aggregate soccor result is exported in HBase table SoccorScoreTable. Every aggregated result from RDD partitions are saved to HBase column-family for use case1 soccor result.

Demo steps (Demo project related)

- Please add details here

Future tasks

Output and Visualization:

- We planned to do visualization of some statistics.
- However, due to time constraints, we were not able to do it.
- In future, We plan to visualize the real-time data and insights using data visualization tools like Apache Zeppelin, Jupyter Notebook, or Tableau(connect with Cloudera Hadoop using ODBC client and fetching data from the Hive Tables).
- This can help us provide live updates to football enthusiasts.

Thank You!