Spring
2019

Reddit comment analysis using Spark

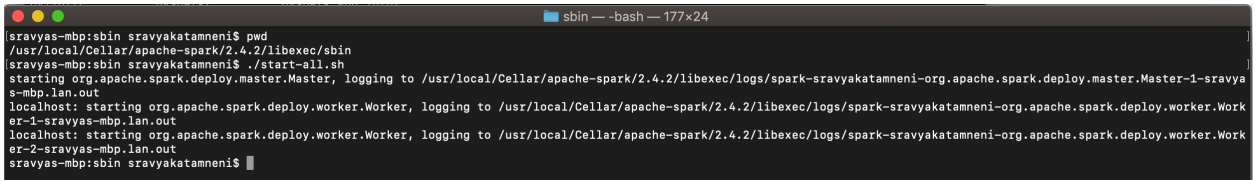
STA-9760 FINAL PROJECT

SRAVYA KATAMNENI

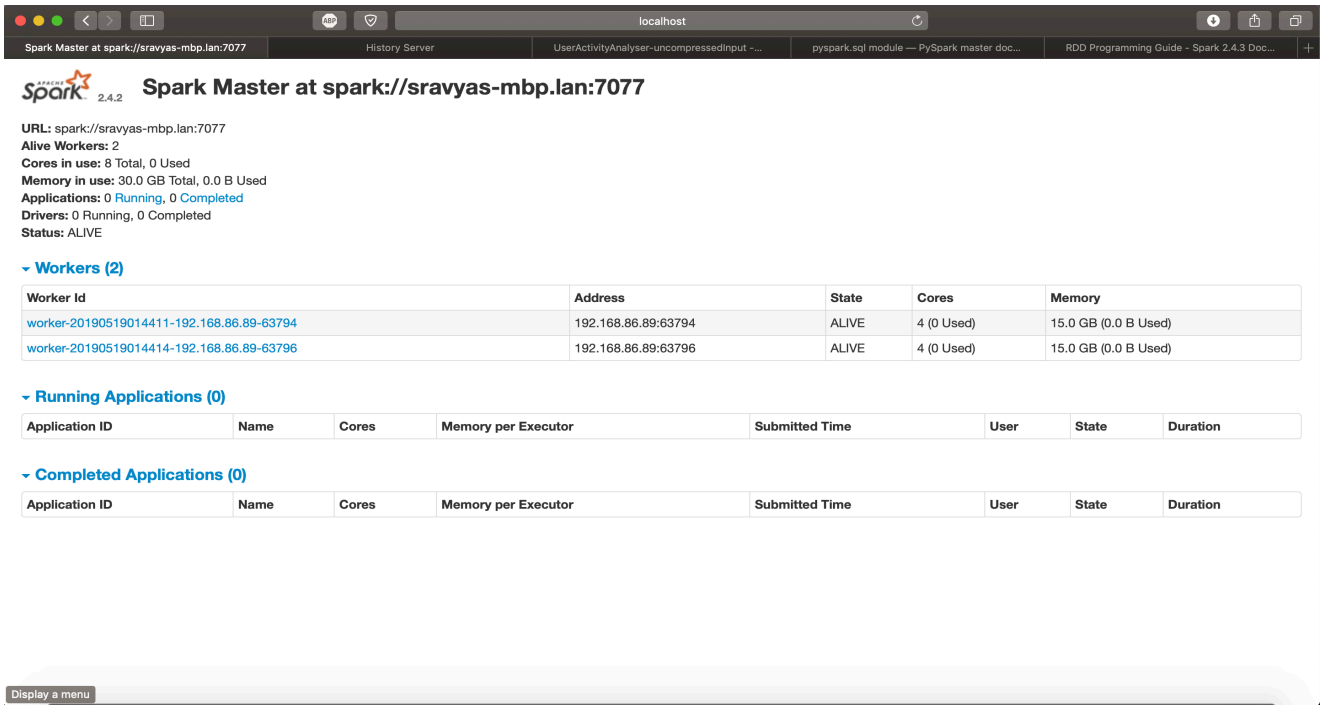
Spark Setup

A standalone Spark cluster is run on local machine with two executors. Each executor was configured to have 4 cores and no-memory limit has been assigned.

Spark cluster was started using start-all command



Spark Master and Executor setup:



Input Data

Public comments from Reddit users in JSON format is used as input to Spark application. Data is hosted on PushShift platform and is available for each month.

- https://files.pushshift.io/reddit/comments/

For a manageable performance on standalone Spark the JSON input file of Reddit comments data (from Dec 2009) was used in this Spark application.

Format	File Size
BZ2	222MB
GZ	327MB
Uncompressed	1.4GB

Spark Application

Description

- Python based Spark application is built to analyze Reddit comments data
- This application uses Spark Context and SQL Context to load raw input and further
  - select specific JSON fields to create Spark data-frame
  - apply transformation and action to achieve group-by, order-by and select top-5 authors and sub-reddits
  - apply Spark SQL queries to achieve group-by, order-by and select top-5 authors and sub-reddits

Results

Author	Total-UpVotes
Scarker	18,497
P-Dub	17,714
big80smullet	12,541
amazingkris	11,937
borez	9,313

Subreddit	Total-Comments
AskReddit	418,224
reddit.com	194,442
pics	129,841
politics	98,868
IAmA	89,123

Spark Job Performance

Runtime Performance Results

	Configuration		Time Taken (sec)			
Input	Worker Cores	Executor Memory	Spark Context	First Line	Total Line Count	Total Time
Uncompressed	1	2GB	1.55	2.84	34.07	137.12
	1	4GB	1.59	2.82	33.54	137.38
	2	2GB	1.73	3.35	25.66	107.03
	2	4GB	1.70	3.25	24.89	99.87
	4	4GB	1.62	3.07	23.79	94.04
Compressed (bz2)	1	2GB	1.52	2.85	132.13	519.53
	1	4GB	1.52	2.95	131.40	491.09
	2	2GB	1.64	3.59	91.06	335.42
	2	4GB	1.61	3.46	90.45	335.40
	4	4GB	1.66	3.47	61.59	231.52
	8	4GB	1.68	3.41	51.45	196.87
Compressed (gz)	1	2GB	1.50	2.80	31.47	120.36
	1	4GB	1.48	2.86	33.21	121.78
	2	2GB	1.66	3.31	32.82	123.14
	2	4GB	1.58	3.36	31.61	118.23
	4	4GB	1.59	3.52	32.89	118.39
	8	4GB	1.63	3.35	32.85	113.92

Granular Performance Analysis

- Performed data load into Spark Dataframe using SQL Context
- Transformation performed on Spark Dataframe to identify top subreddit by comments and popular user by number of up-votes
- Used Spark SQL with group-by and order-by clauses to achieve same result i.e. top subreddit by comments and popular user by number of up-votes

	Configuration		Sub-Reddit Comment#			User Up-Votes	
Input	Worker Cores	Executor Memory	Spark DF	Spark SQL	To CSV	Spark SQL	To CSV
Uncompressed	1	2GB	18.15	16.54	19.16	19.29	28.32
	1	4GB	17.98	16.50	19.21	17.96	27.75
	2	2GB	15.28	12.87	14.57	13.98	19.57
	2	4GB	13.67	12.38	14.26	12.32	17.37
	4	4GB	12.85	11.81	12.18	12.27	16.42
Compressed (bz2)	1	2GB	69.05	65.02	67.51	99.91	81.52
	1	4GB	68.81	68.66	70.57	69.20	77.96
	2	2GB	46.03	45.37	47.84	44.69	55.19
	2	4GB	46.60	46.15	48.49	46.65	51.96
	4	4GB	32.25	30.98	32.82	31.22	37.50
	8	4GB	26.72	26.16	27.43	27.03	32.97
Compressed (gz)	1	2GB	15.48	14.04	16.35	14.85	23.84
	1	4GB	15.25	13.68	16.13	14.97	24.17
	2	2GB	17.30	14.25	16.45	14.93	22.39
	2	4GB	15.08	13.49	17.42	14.79	20.86
	4	4GB	15.65	13.54	16.01	14.65	20.52
	8	4GB	13.74	13.12	15.47	13.57	20.16

Conclusions

Cores

- Increasing the number of cores had most impact on the compressed input files
- 35% improvement in total time was observed on BZ2 input file while 22% improvement was observed on uncompressed input

Memory per Executor

- Memory per executor didn't have a consistent effect on the performance and it was in the order of 5%

Input Compression

- When input is in BZ2, best compression, performance was dependent on number of available cores
- When input is uncompressed, performance improvement was relatively less sensitive to number of cores and memory
- G-Zip format offered balance of file compression benefits without eroding performance

Serialization

- Default serializer, org.apace.spark.serializer.JavaSerializer, is used as default serializer
- PySpark stores data as byte array which are quick to serialize, so it doesn't benefit from using Kryo serializer

RDD Compression

- As Spark was run as standalone it doesn't benefit from using RDD compression

Performance Summary

Run Time			
	BZ2	**G-Zip**	**Uncompressed**
2GB			
1-Core	519.53	120.36	137.12
2-Cores	335.42	123.14	107.03
4GB			
1-Core	491.09	121.78	137.38
2-Cores	335.40	118.23	99.87
4-Cores	231.52	118.39	94.04
8-Cores	196.87	113.92	

Takeaways

- Cluster with more nodes (>5) where each node has >=8 cores would allow for greater performance optimizations to be applied
- RDDs can be compressed as they are distributed across the workers and stored on the cluster
- Various serialization options can be implemented on Spark when run on Java/Scala
- Compression block sizes can be set to higher values if enough memory resources are available to be allocated to Spark nodes
- Serialization buffer size can be set to higher values to handle serialization of bigger objects without impacting runtime performance