CS553 Homework #5

Hadoop Installed

```
root@namenode:~# jps
10897 NodeManager
10753 ResourceManager
803 SecondaryNameNode
18663 SparkSubmit
18888 Jps
730 NameNode
18815 YarnCoarseGrainedExecutorBackend
```

```
root@datanode1:~# jps
946 NodeManager
19282 YarnCoarseGrainedExecutorBackend
19363 Jps
19207 ExecutorLauncher
877 DataNode
```

Spark Installed

```
root@namenode:~# spark-submit --version
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.4.3
      /_/

Using Scala version 2.12.17, OpenJDK 64-Bit Server VM, 11.0.22
Branch HEAD
Compiled by user centos on 2024-04-15T01:06:05Z
Revision 1eb558c3a6fbdd59e5a305bc3ab12ce748f6511f
Url https://github.com/apache/spark
Type --help for more information.
```

```
root@datanode1:~# spark-submit --version
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.4.3
      /_/

Using Scala version 2.12.17, OpenJDK 64-Bit Server VM, 11.0.22
Branch HEAD
Compiled by user centos on 2024-04-15T01:06:05Z
Revision 1eb558c3a6fbdd59e5a305bc3ab12ce748f6511f
Url https://github.com/apache/spark
Type --help for more information.
```

Problem: Need to generate 10 bytes blake3 hash using 6bytes nonce and sort and write to a binary file.

Methodology: Need to implement 2 methods. One is using Hadoop cluster MapReduce where you use multiple mappers to generate sets of data and use reduce to generate a single file. Second is using Apche Spark on top of Hadoop cluster with a number of partitions to generate data and coaelace to a single file.

Configuration:

core-site.xml

```xml
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://namenode:9000</value>
  </property>
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/var/hadoop/tmp</value>
  </property>
</configuration>
```

hdfs-site.xml

```xml
<configuration>
  <property>
  <name>dfs.namenode.handler.count</name>
  <value>100</value>
</property>
  <property>
      <name>dfs.replication</name>
      <value>1</value>
  </property>
  <property>
      <name>dfs.namenode.name.dir</name>
      <value>file:///usr/local/hadoop/data/namenode</value>
  </property>
  <property>
      <name>dfs.datanode.data.dir</name>
      <value>file:///usr/local/hadoop/data/datanode</value>
  </property>
</configuration>
```

Namenode will have only dfs.namenode.name.dir property
Datanodes will have only dfs.datanode.name.dir property

mapred-site.xml

```xml
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>yarn.app.mapreduce.am.env</name>
    <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
  </property>
  <property>
    <name>mapreduce.map.env</name>
    <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
  </property>
  <property>
    <name>mapreduce.reduce.env</name>
    <value>HADOOP_MAPRED_HOME=/usr/local/hadoop</value>
  </property>
</configuration>
```

yarn-site.xml

```xml
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>
  <property>
    <name>yarn.resourcemanager.hostname</name>
    <value>namenode</value>
  </property>
</configuration>
```

Experiment	hashgen	vault	Hadoop Sort	Spark Sort
1 small.instance, 16GB dataset, 2GB RAM	1259.325	126.23	N/A	N/A
1 small.instance, 32GB dataset, 2GB RAM	1589.432	259.00	N/A	N/A
1 small.instance, 64GB dataset, 2GB RAM	N/A	N/A	N/A	N/A
1 large.instance, 16GB dataset, 16GB RAM	1101.212	334.85	1978.432	1741.771
1 large.instance, 32GB dataset, 16GB RAM	921.371	283.24	2871.743	2563.644
1 large.instance, 64GB dataset, 16GB RAM	1387.142	653.49	11,727.577	10,621.673
6 small.instances, 16GB dataset	N/A	N/A	2389.242	1995.742
6 small.instances, 32GB dataset	N/A	N/A	3294.251	3010.631
6 small.instances, 64GB dataset	N/A	N/A	12,462.347	11,462.347

hashgen: I used 16 threads for hash generation, 16 threads for sorting and 16 threads for writing to a file.
HadoopSort: I used 8 mappers and 1 reducer at the final stage to generate 1 single file.
Spark Sort: I used 16 partitions to generate data.
With regard to my implementation I had to write data twice and read data once.
Using Hadoop or Spark is slower than my implementation.

Table Results: As you can see from the table using higher ram for generating dataset with hashgen/vault results in faster execution because more data can be stored in memory for sorting nad it reduces the number of disk I/O. Hadoop Sort and Spark Sort are much more slower than hashgen/vault. This is because Hadoop Sort and Spark Sort are higly dependent on parameter optimization and configuration of clusters along with overhead of managing tasks, whereas hashgen/vault can fully use memory and multi-threading capabilities for faster execution. Spark performs better than hadoop and is easier to use in terms of code simplicity.

After setting the RAM, hashgen/vault performs better in large.instance because of the availability of more RAM. Spark seems to be best at 1 node scale. Yes, there is a difference between 1 small.instance and 1 large.instance. A large.instance has significantly better performance than a small.instance. For large.instance all data storage and processing tasks are handled by a single node, which can become a bottleneck if the workload increases. The performance may be limited by the datanode's capacity to handle input/output operations and its computational power. For small.instances, with multiple datanodes, the data and processing tasks are distributed among several nodes. This distribution helps in balancing the load and improving performance because multiple nodes can read and process data simultaneously. It also reduces the risk of overloading a single node. Ideally, with scaling from 1 node to 6 nodes, performance should increase but for my implementation and the size of the dataset, 1 node is performing better. With strong scaling the execution time/sorting time reduces. With weak scaling, sorting time would remain almost same. I would need small.instances as much as the number of threads I am using for executing my hashgen to achieve permormance closer to hashgen. This applies to both Hadoop and Apache.
With scaling up with more nodes (strong scaling), Spark tends to provide better performance due to its efficient in-memory data processing. Hadoop MapReduce, while reliable for large-scale data processing, is generally slower due to its disk-intensive operations and I/O overhead between map and reduce stages. Spark would be best be it 100 small.instances or 1000 small.instances.

With the help of graph of cpu, memory and i/o, it would be evident that Spark would perform better than Hadoop by utilising more real memory and les i/o. Hadoop would be using more i/o. This results in slower performance. Spark does in memory data processing which would explain higher usage of real memory and less i/o compared to hadoop which does more i/o. This explains why Spark performs better.