A2Q2 Report

Name: Viswa Nihar Nukala
UB Name: viswanih
UB Number: 50414392

This code is to get the id of a connected component of a forest, such that id of each node is the root of the connected component it belongs to.

Initially I get a map of all the children of one node and its parent, and I then map all its children to the parent such that after every iteration the child points to one level closer to the node. After a few iterations all the nodes point to the root. The termination condition is to find the sum of all the parents of all the nodes.

I have used combineByKey to map the data and used leftOuterJoin to join the 2 RDDs and then flatMap to reduce the data.

For the termination condition, I have used sum.
I also print the time difference between right before the SparkContext has begun and ended.

The following is a snapshot of the spark args in my slurm file:

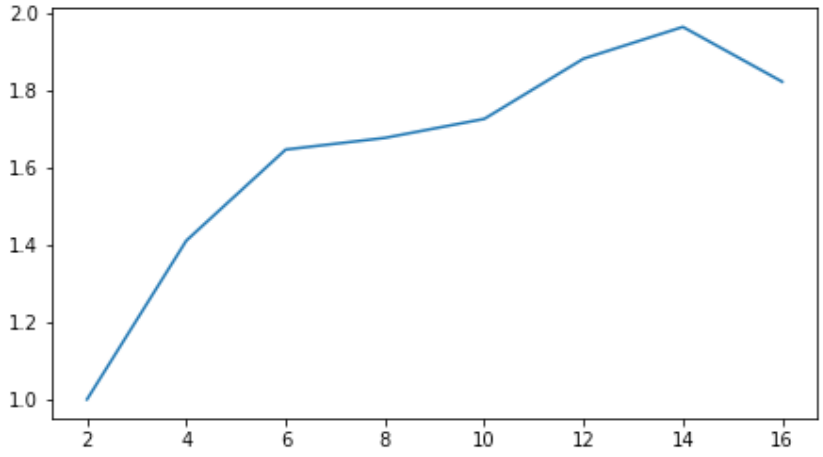
```
# SET YOUR COMMAND AND ARGUMENTS
PROG="a2.py"
#ARGS="infile.txt out"
ARGS="/panasas/scratch/grp-jzola/intropdp/T0.txt /panasas/scratch/grp-cse570f21/viswanih/out"
#ARGS="/panasas/scratch/grp-jzola/intropdp/T.txt"


# SET EXTRA OPTIONS TO spark-submit
# HERE YOU CAN CONTROL HOW MANY EXECUTORS YOU WANT TO RUN PER NODE, ETC.
# EXAMPLE OPTIONS:
# --num-executors
# --executor-cores
# --executor-memory
# --driver-cores
# --driver-memory
# --py-files
#SPARK_ARGS="--num-executors 80 --executor-cores 10 --executor-memory 24G"
SPARK_ARGS="--conf spark.default.parallelism=400 --num-executors 40 --executor-cores 10 --executor-memory 12G"
```

I have chosen strong scaling to do profiling.

Following are the measurements from keeping the number of executors per node constant (at 5 per node) and 10 cores:

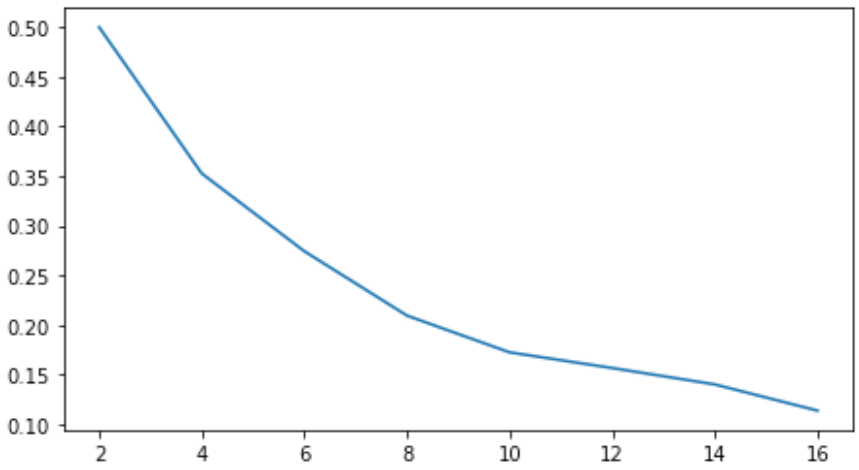
Nodes	Executors	Time
2	10	2044.7
4	20	1449.5
6	30	1242.2
8	40	1219.8
10	50	1185.1
12	60	1086.9
14	70	1041.5
16	80	1122.5

Following are my speed up and efficiency graphs for the above data:



Speed Up

As we reach 16 nodes, the speed increases a little bit.



Efficiency

Bottleneck:

As we reach towards the end of the iterations, the time taken to map all the children to its parent takes a lot of time as the number of children for a single parent keeps increasing as the pointer jumping progresses. Hence even if we increase the number of nodes there is no significant decrease in time.

Following are the screenshots to support the above claim:

▾ Active Jobs (1)

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
24	runJob at SparkHadoopWriter.scala:78 runJob at SparkHadoopWriter.scala:78 (kill)	2021/11/26 10:23:58	7 s	0/49	244/28232 (21 running)

▾ Completed Jobs (24)

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
23	sum at /user/viswanih/A2/a2.py:73 sum at /user/viswanih/A2/a2.py:73	2021/11/26 10:22:08	1.8 min	3/3 (46 skipped)	1800/1800 (26432 skipped)
22	sum at /user/viswanih/A2/a2.py:73 sum at /user/viswanih/A2/a2.py:73	2021/11/26 10:21:12	57 s	3/3 (44 skipped)	1800/1800 (25232 skipped)
21	sum at /user/viswanih/A2/a2.py:73 sum at /user/viswanih/A2/a2.py:73	2021/11/26 10:20:14	57 s	3/3 (42 skipped)	1800/1800 (24032 skipped)
20	sum at /user/viswanih/A2/a2.py:73 sum at /user/viswanih/A2/a2.py:73	2021/11/26 10:19:28	46 s	3/3 (40 skipped)	1800/1800 (22832 skipped)
19	sum at /user/viswanih/A2/a2.py:73 sum at /user/viswanih/A2/a2.py:73	2021/11/26 10:18:42	46 s	3/3 (38 skipped)	1800/1800 (21632 skipped)
18	sum at /user/viswanih/A2/a2.py:73 sum at /user/viswanih/A2/a2.py:73	2021/11/26 10:17:58	45 s	3/3 (36 skipped)	1800/1800 (20432 skipped)
17	sum at /user/viswanih/A2/a2.py:73 sum at /user/viswanih/A2/a2.py:73	2021/11/26 10:17:14	44 s	3/3 (34 skipped)	1800/1800 (19232 skipped)
16	sum at /user/viswanih/A2/a2.py:73 sum at /user/viswanih/A2/a2.py:73	2021/11/26 10:16:31	43 s	3/3 (32 skipped)	1800/1800 (18032 skipped)
15	sum at /user/viswanih/A2/a2.py:73 sum at /user/viswanih/A2/a2.py:73	2021/11/26 10:15:50	41 s	3/3 (30 skipped)	1800/1800 (16832 skipped)
14	sum at /user/viswanih/A2/a2.py:73 sum at /user/viswanih/A2/a2.py:73	2021/11/26 10:15:12	38 s	3/3 (28 skipped)	1800/1800 (15632 skipped)
13	sum at /user/viswanih/A2/a2.py:73 sum at /user/viswanih/A2/a2.py:73	2021/11/26 10:14:36	36 s	3/3 (26 skipped)	1800/1800 (14432 skipped)
12	sum at /user/viswanih/A2/a2.py:73	2021/11/26 10:13:58	38 s	3/3 (24 skipped)	1800/1800 (13232 skipped)

▾ Completed Stages (3)

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
623	sum at /user/viswanih/A2/a2.py:73 +details	2021/11/26 10:23:42	16 s	600/600			524.1 MB	
622	combineByKey at /user/viswanih/A2/a2.py:18 +details	2021/11/26 10:22:08	1.6 min	600/600			570.2 MB	399.3 MB
621	combineByKey at /user/viswanih/A2/a2.py:25 +details	2021/11/26 10:22:08	24 s	600/600			570.2 MB	124.8 MB

The combineByKey at line 18 does the mapping of the children to the parents. Hence that takes more time as the iterations increase.

I have also profiled my code in a different way by keeping the number of nodes constant and increasing the number of executors:

Nodes	Executors	Time
6	12	1252.7
6	24	1186.5
6	36	1084.4
6	48	890.1

The decrease in time is more significant is this way but as we increase the number of executors the time also becomes more constant due to the higher shuffling.

Conclusion:

The algorithm is scalable till 16 in number of nodes by keeping the number of executors constant but the change in efficiency keeps decreasing. But when we keep the number of nodes constant and change the executors there is a significant change in time. Since the last iteration taking more time is inevitable, I believe we cannot achieve better efficiency with the same algorithm at least.