CS6240
Soumyadeep Sinha

Assignment -1Report

Pseudo-Code:

```
1: class Mapper
2:        method Map(doc edge)
3:                for all users ∈ edge e do
                          following f2 = users.split(“,”)[1]
4:                         Emit(following f2, count 1)


1: class Reducer
2:         method Reduce(following f2, counts [c1, c2, . . .])
3:                sum ← 0
4:                for all count c ∈ counts [c1, c2, . . .] do
5:                        sum ← sum + c
6:                Emit(following f2, count sum)
```

Main Idea:

My program reads the input line by line. The map function parses a line to extract the users which includes the follower and the user the follower is following: for example let’s say the data in the edge.csv is 1, 2. This means that the user 1 is following user 2. Now to determine the number of followers of a user, we need to get the number of times that particular user is being followed. For example: if the data is

1, 2
2, 3
3, 4
4, 2

Here, user 1 has 0 followers, user 2 has 2 followers, 3 has 1 follower and user 4 has 1 follower.

So in My mapper I have split the input using “,” and then assign a value/ the count of followers to 1 for each user being followed(the user after “,”)
So after the mapper the data would look something like:

User	->	Count
2	->	1
3	->	1
4	->	1
2	->	1

This intermediate output is then grouped by the user and then the reducer function takes the sum of the count for each user and calculates the number of followers for each group/ user. So the final output would be similar to :

User	->	Count
2	->	2
3	->	1
4	->	1

Running Time Measurements:

1. Repeated time measurements:
    a. First Run time:
        Total time: Step succeeded with exitCode 0 and took 60 seconds

    b. Second Run time:
        Total time: Step succeeded with exitCode 0 and took 58 seconds

2. Amount of data transferred to the Mappers:
    First Run:
        In records: 85331845
        In bytes: 1319475045

    Second Run:
        In records: 85331845
        In bytes: 1319484634

3. Amount of data transferred from Mappers to Reducers:
    First Run:
        In records: 85331845
        In bytes: 961483442

    Second Run:
        In records: 85331845
        In bytes: 961483442

4. Amount of data transferred from Reducers to Output:
    First Run:
        In records: 6626985
        In bytes: 67641452

    Second Run:
        In records: 6626985
        In bytes: 67641452

Argument regarding why or why not your MapReduce program is expected to have good speedup:

The map reduce program, I believe, has a good speed up, however, the sample data is too small to conclusively determine if its actually the case. I used 6 nodes/ machines to run my program which would be divided among the mapper workers and the reducer workers. Here, due to the parallelization of the tasks, we achieved a good speed up since a single machine is not handling the entire input file. However, master program which is the conduit through which the location of intermediate file regions is propagated from map tasks to reduce tasks (Assuming that there is only a single master program) this process is still sequential which would mean that there is still a way to parallelize and have a better speed up and fault tolerance. Other things which can slow down the speed would be the latency and time required to transfer the data to the worker computers. Hence, a way to make sure that we have a better speed up would be to use more nodes or a slightly larger cluster. Taking too much of a large cluster, however, would mean that master would spend more time communicating with the workers and the times will add up and make the process way slower. So, I believe a cluster of 10-15 machines would be ideal for a dataset like the one used for this assignment.

1. How many tasks were executed in each stage:
   Killed map tasks=1
   Killed reduce tasks=1
   Launched map tasks=20
   Launched reduce tasks=11

2. If there is a part of your program that is inherently sequential (see discussion of Amdahl's Law in the module) :

   I believe, the master program and the way it communicates with the workers is sequential since, assigning of tasks and communicating the location of the intermediate files to the reducers are still taken care of by one master. This part I believe is sequential. Another part I believe would be sequential is the way the mapper and the reducer processes the input files/ intermediate files respectively. Even Though, the input file is divided into various mappers, the portion of the input file that a single mapper has to process is still sequential and same goes for the reducer. The process of providing the intermediate output from mapper to reducer is also sequential which means that reducer would have to wait for the mapper to provide the intermediate output for it to start processing. Having said that, as discussed above, throwing unlimited processors/ workers at the problem is not the solution to speed up as well as the time it takes to communicate or rather the sequential part of the process would at some point would negate the speed up we receive from the parallel part of the process.