CS6240

Rachna Reddy Melacheruvu

HW 2

https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment2/MapReduce/Reduce-Side-Join/MR-Demo

https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment2/MapReduce/Replicated-Join/MR-Demo

URLs:

Google drive URLs belong to Northeastern. Please access by logging in with your husky ID**.**

Reduce Side Join-

1. **Project:**
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment2/MapReduce/Reduce-Side-Join/MR-Demo

2. **Logs:**
   Small Cluster - https://drive.google.com/open?id=1-AQc8bRUB89sOf-N1D61eZzBn7ziPsMd
   Large Cluster - https://drive.google.com/open?id=1KCZ_V08KrpOi9WakPiQNvzXUpqPxtlKY

3. **Output:**
   Small Cluster –
   Job1 output – https://drive.google.com/open?id=1_nvFHPBAuen5Dl2ZtOWjWw-lmVCZ1Yz1
   Job 2 output - https://drive.google.com/open?id=1BYvz3XhkXt52gY4sgKnnRUZp8wTe42RC

   Large Cluster –
   Job1 output - https://drive.google.com/open?id=1vMP1ewZCu4ZoXJX9HfkM6OW7LtmFSWEI
   Job 2 output - https://drive.google.com/open?id=1Z5HAlAhxS3ldQMRHyQFsXb-PU21hCJQN

Replicated Join:

1. **Project:**
   https://github.ccs.neu.edu/melacheruvur/cs6240/tree/master/Assignment2/MapReduce/Replicated-Join/MR-Demo

2. **Logs:**
   Small Cluster - https://drive.google.com/open?id=1KIO8wu-nKJffhH33i7YXU0043ZRPcVIQ
   Large Cluster – https://drive.google.com/open?id=1s4WDGsQmLzTwZgHxjwQeqG6-AoTt6X8X

3. **Output:**
   Small Cluster - https://drive.google.com/open?id=1_UD_TR6d5rUANQMl1Bl4NLbYTRlJTrIJ
   Large Cluster – https://drive.google.com/open?id=1QjVFXml_O8im_5se7NKYZVjlDpz0p-8Q

Citation:

1. **For implementing counters in the program to keep track of the cardinality and triangle count.**
   Reference – https://diveintodata.org/2011/03/15/an-example-of-hadoop-mapreduce-counter/

2. **For Implementing file cache since Distributed Cache as given in example program is deprecated.**
   Reference 1 - https://stackoverflow.com/questions/21239722/hadoop-distributedcache-is-deprecated-what-is-the-preferred-api
   Reference 2- http://hadoop.apache.org/docs/stable2/api/org/apache/hadoop/mapreduce/Job.html

PROBLEM ANALYSIS

Pesudo Code to determine Cardinality

This code is part of the main program itself.

```
int MAX = 50000
enum Counter
{
   Max_cardinality
   No_of_Triangles
}
Mapper1 {
        map()
        {
            // reads from input file
            String s = covert value to string
            String[] followers = s.split(“,”)      // split string on delimited as “,”
            String user1 = followers[0]
            String user2 = followers[1]
            if (user1<=MAX && user2<=MAX)
            {
                String fromVal = user1 + “ ” + user2 + “;”+ “F”
                String toVal = user1 + “ ” + user2 + “;” + “T”
                emit(user1 , fromVal)
                emit(user2 , fromVal)
            }
        }
}

Reducer1 {
        reduce(key,values)
        {
            for(Text t : values)
           {
               String s = covert value to string
                String[] nodes = s.split(“;”)      // split string on delimited as “;”
                if (nodes[1]==”F”)
                      list_F.add(nodes[0])
                else
```

```
                list_T.add(nodes[0])
        }

        for(String s1 in list_F)
        {
           String[] user1 = s1.split(" ")
           String user11 = user1[0]
           String user12 = user1[1]

            for(String s2 in list_T)
            {
                String[] user2 = s2.split(" ")
               String user21 = user2[0]
               String user22 = user2[1]
               String key = user12 + "," + user21
               Context.write(key,"from")
                Counter.Max_cardinality++

            }
        }

    }
}
```

12 Cardinality and Volume values

RS Join – Performed 2 steps (one for cardinality and other for number of triangles)

Rep Join – Merged both the steps.

MAX Value	Steps		RS Join Input	RS Join Shuffled	RS Join output	Rep join input	Rep join file cache	Rep join output
40000	Step 1	Cardinality	85331845	1514350	574639402	85331845	85331845	4741564
		Volume (approx. bytes/GB)	1.3 GB	11654772 bytes	8.62 GB	1.3 GB	1.3 GB	71123460 bytes
	Step 2	Cardinality	85331845	575396577	4741564			
		Volume (approx. bytes/GB)	1.3 GB	3529957042 bytes	71123460 bytes			

To determine the MAX values:

For RS Join- Program has 2 jobs. Job 1(1 Map task, 1 Reduce task) is used to count the Path2 Cardinality. Job 2 (2 Map tasks, 1 Reduce task) is used to check with the Path joined by Path 2 and edges exists.

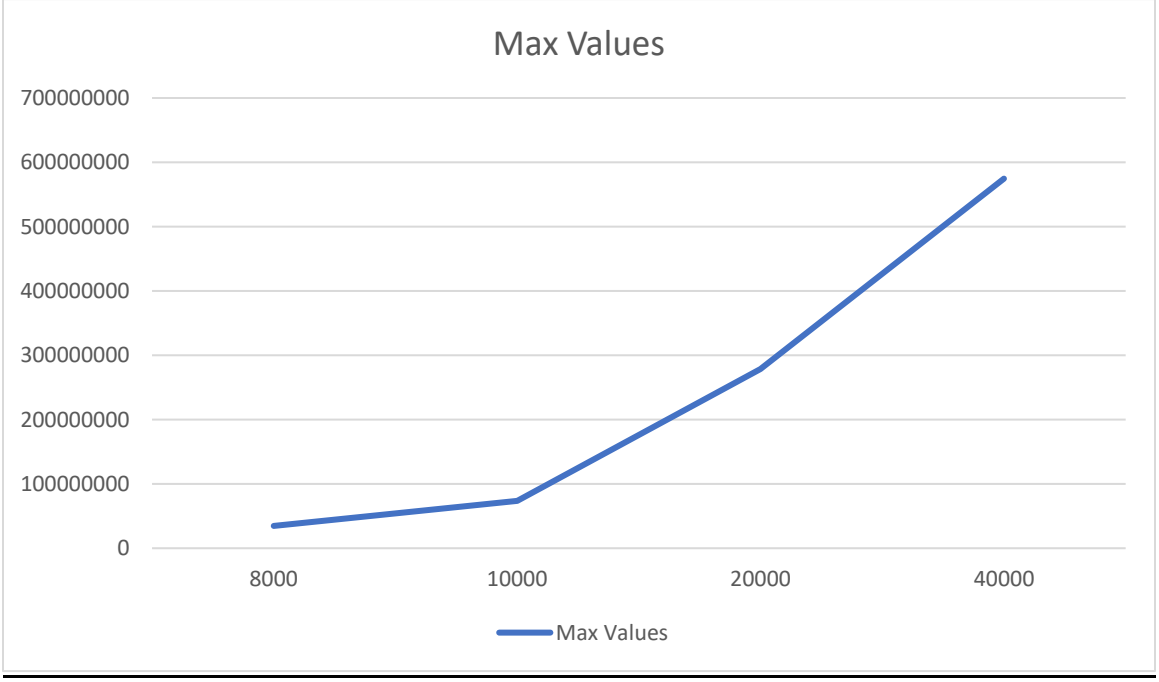
Ran only Job 1 to determine the MAX values which take 20-60 min to run. With trial and error, I could conclude that it takes approximately 60 mins for 40000 nodes on a small cluster.

Performed step 1 on various max values to determine the threshold.

MAX Value		**RS Join Input**	**RS Join Shuffled**	**RS Join output**
8000	Cardinality	85331845	175364	34685928
	Volume (approx. bytes/GB)	1.3 GB	3271958 bytes	0.5 GB
10000	Cardinality	85331845	258402	73597234
	Volume (approx. bytes/GB)	1.3 GB	4843569 bytes	1.1 GB
20000	Cardinality	85331845	724126	278536415
	Volume (approx. bytes/GB)	1.3 GB	5352121 bytes	4.17 GB
40000	Cardinality	85331845	1514350	574639402
	Volume (approx. bytes/GB)	1.3 GB	11654772 bytes	8.62 GB

For Rep Join – Since program is Map only, I merged both the steps into. For Rep join, I have used trial and error considered 40000 nodes as the MAX.

Graph to estimate full cardinality depending on the various values obtained by running the program:



JOIN IMPLEMENTATION

Reduce Side Join Implementation

Pseudo-code

```
int MAX = 50000
enum Counter
{
   Max_cardinality
   No_of_Triangles
}
Mapper1 {
        map()
        {
            // reads from input file
            String s = covert value to string
            String[] followers = s.split(“,”)      // split string on delimited as “,”
            String user1 = followers[0]
            String user2 = followers[1]
            if (user1<=MAX && user2<=MAX)
            {
                String fromVal = user1 + “ ” + user2 + “;”+ “F”
                String toVal = user1 + “ ” + user2 + “;” + “T”
                emit(user1 , fromVal)
                emit(user2 , fromVal)
            }
        }
}

Reducer1 {
        reduce(key,values)
        {
            for(Text t : values)
            {
                String s = covert value to string
                String[] nodes = s.split(“;”)      // split string on delimited as “;”
                if (nodes[1]==”F”)
                      list_F.add(nodes[0])
                else
                      list_T.add(nodes[0])
            }

            for(String s1 in list_F)
            {
                String[] user1 = s1.split(“ “)
                String user11 = user1[0]
                String user12 = user1[1]

                for(String s2 in list_T)
```

```
                {
                    String[] user2 = s2.split(“ “)
                   String user21 = user2[0]
                   String user22 = user2[1]
                   String key = user12 + “,“ + user21
                   Context.write(key,”from”)
                    Counter.Max_cardinality++

                }
            }

        }
}

Mapper2
{
         map()
         {
             // reads input from file written by Reducer1 taks

              StringTokenizer str = split input based on /t
              String key = itr.nextToken()
              String val = itr.nextToken()
               emit(key,val)
         }

}

Mapper3
{
       map()
         {
             // reads from input file
             String s = covert value to string
             String[] followers = s.split(“,”)      // split string on delimited as “,”
             String user1 = followers[0]
             String user2 = followers[1]
             if (user1<=MAX && user2<=MAX)
             {
                 String key = user1 + “,” + user2
                 String val = “to”
                 emit(key , val)
             }
         }

}

Reducer2
{
```

```
    reducer(key,values)
    {
        for (Text t : values)
        {
            if( t==”from”)
               from++
            if(t==”to”)
                to++
        }

        if (from>=1 && to>=1)
            Counter.No_of_Triangles = Counter.No_of_Triangles.getValue()+ from

    }

}
```

Replicated Join Implementation

Pseudo-code

```
Public static enum TRIANGLE_COUNTER
{
    MAX_CARDINALITY,
    NO_OF_TRIANGLES
};

Public static int maxId = 44000;

Mapper{

        HashMap userFollower;

        Setup ()
        {
            files = Get files from cache
            if (files != null)
            {
                reader = BufferReader (each file)
                // read each record in the file
                While ((line = reader.readLine())!=null)
                {
                    String[ ] users = line.split(“,”)
                    String user1 = users[0]
                    String user2 = users[1]
                    if (Integer.parseInt(user1)<=maxId && Integer.parseInt(user2)<=maxId)
                    {
                        if (!userFollower.containsKey(user1) )
                        {
```

```
                        Set<String> nodes = new HashSet<>();
                        nodes.add(user2)
                       userFollower.put(user1, nodes)
                    }

                    else {
                            Set<String> nodes = userFollower.get(user1);
                            nodes.add(user2)
                            userFollower.put(user1, nodes)
                    }

                }
            }
        }
    }

    map ()
    {

        // reads from input file
        StringTokenizer str = split input based on /t
          String user1 = itr.nextToken()
          String user2 = itr.nextToken()
        if (user1<=MAX && user2<=MAX)
        {
            If (user2.lookup())
            {
                Values = user2.getValues()
                Counter.MAX_CARDINALITY.increment(Values.size())
                for each val in value
                {
                      If (val.lookup())
                      {
                          Dest_nodes = val.getValues()
                          for each node in Dest_nodes
                          {
                               if ( node.equals(user1))
                                Counter.NO_OF_TRIANGLES++;

                          }
                      }
                }
            }

        }
    }
}
```

Running Time Measurements

Configuration	*Measurements*	*Small Cluster Result (6 m4.large)*	*Large Cluster Result (11 m4.large)*
RS Join ***MAX = 40000***	Running Time	35 min 56 sec	21 min 09 sec
	Triangle Count	4741564	4741564
REP Join ***MAX = 40000***	Running Time	32 min 24 sec	30 min 40 sec
	Triangle Count	4741564	4741564