CPT_S 415

Big Data, Fall 2020

Homework # 3

Tayyab Munir

11716089

1. [Parallel Data Models] (30)

a. What is speedup and scaleup? Give three reasons why we cannot do better than linear speedup.

Solution:

Speed-up:

Speed-up is the effect of applying an increasing number of resources to a fixed amount of work achieved. In this we have more processors which tends to increase the speed. Having more processors makes it easy to run individual queries faster and more transactions per second can be achieved. It results in resource availability for other tasks.

It is also defined as the ratio between the serial runtime of the best sequential algorithm and time taken by parallel algorithm to solve the same problem on p processors.

$$S = \frac{T_s}{T_p}$$

Scale-up:

Scale-up us the ability to keep the same response time when the workload/ transactions and resources increases. In this more data can be processed using a greater number of processors. It also expresses that how much work can be done in the same time period by a system n times larger than that.

Following are a list of reasons that shows that why we cannot do better than linear speed-up.

1. Startup cost: Cost of sharing an operation on many processors.
2. Interference: contention between processor (network, disk, etc).
3. Skew: Slowest processors become the bottleneck and it is not easy to divide a task into exactly equal sizes.

b. Assume a program P running on a single-processor system takes time T to complete. 40% of P can only be executed sequentially on a single processor, and the rest is “embarrassingly parallel” in that it can be easily divided into smaller tasks executing concurrently across multiple processors. What are the best time costs to execute P using 2, 4, 8 machines (expressed by T)? What are the speed-ups respectively? What are the optimal speed-ups given an infinite number of machines?

Solution:

To calculate the best time costs to execute P using 2, 4, and 8 machines, Expressed by T. Let say the time is represented T_2, T_4, T_8 respectively.

Whereas, we have
f = 40/100 => 0.4
p = 2, 4, 8

$$T_p = f\,T + (\frac{1-f}{p})T$$

$T_2 = 0.4T + (\frac{1-0.4}{2})T$ => 0.7T

$T_4 = 0.4T + (\frac{1-0.4}{4})T$ => 0.55T

$T_8 = 0.4T + (\frac{1-0.4}{8})T$ => 0.475T

To calculate the speed-up we have

Speed-up = $\frac{1}{f+\frac{(1-f)}{p}}$

Now, for p = 2
Speed-up = $\frac{1}{0.4+\frac{(1-0.4)}{2}}$ => 1.42

Now, for p = 4
Speed-up = $\frac{1}{0.4+\frac{(1-0.4)}{4}}$ => 1.81

Now, for p = 8
Speed-up = $\frac{1}{0.4+\frac{(1-0.4)}{8}}$ => 2.105

$T_\infty = 0.4T + (\frac{0.6}{\infty})T$ => 0.4T

Speed-up = $\frac{T}{T_\infty} = \frac{T}{0.4T}$ = 2.5

c. Describe and compare the pros and cons of the three architecture for parallel systems.

Solution:

Pros and cons for shared memory

Pros	Cons
Shared memory system is easy to program and processed.	Shared memory architecture is expensive to build.
	It is difficult to scale-up because shared memory and network becomes bottleneck. It is also not scalable beyond the limit of 32- or 64-bits processors.

Pros and cons for shared disk

Pros	Cons
It has better fault tolerance.	It is scalable in shared memory but not scalable enough in terms of memory disk subsystem and it is using a single network which is not scalable for after a couple of hundred processors.
Shared disk architecture has better scalability than shared memory. In this architecture memory is not any longer considered as a bottleneck.	

Pros and cons for shared nothing

Pros	Cons
Shared nothing system is cheaper to build.	It is hard to program.
There is no single point of failure if shared nothing architecture is used.	

Similarities between shared memory, shared disk and shared nothing

In shared memory architecture the memory is shared among all processes (for example if any two processors are working on same memory data and if one processor saves the updated data which will create problems for the other processor because the other one was using the old data and processing it.) Shared disk architecture have separate blocks of memory for each processor which wil not create the problem that was caused using shared memory, but in shared disk the network that is used is shared which causes problem for I/o and the processors may have to wait and be idle. Whereas, shared nothing architecture seems to be the good one and it share nothing with other processors.

2. MapReduce] (40)

This set of questions test the understanding and application of MapReduce framework.
a. (20) Facebook updates the "common friends" of you and response to hundreds of millions of requests every day. The friendship information is stored as a pair (Person, [List of Friends]) for every user in the social network. Write a MapReduce program to return a dictionary of common friends of the form ((User i, User j), [List of Common Friends of User i and User j]) for all pairs of i and j who are friends. The order of i and j you returned should be the same as the lexicographical order of their names. You need to give the pseudo-code of 1 main function, and 1 Map() and 1 Reduce() function. Specify the key/value pair and their semantics (what are they referring to?).

Solution:

```
Map(key, value){

    user = (<friend_1><friend_2> …. <friend_n>);
    Foreach friend in user{
      Pair = SortedKey(person, friend);
      Emit(pair, user);
    }
}
```

In this function key is the person and value is the list of friends for this person such as value = (<friend_1><friend_2>….). This SortedKey is a function used to sort the output keys. Let us take an example over her that (A is a friend of B on Facebook whereas, B is also a friend of A). So, to sort this alphabetically we have

```
SortedKey(user1, user2){
    If (user1 < user2){
      Return (user1, user2);
    }
    else{
    Return (user2, user1);
    }
}
```

The reduce function finds the common friends of (user_i, user_j) by intersecting all associated friends in between.

In the following Reduce function user is the key that was emitted by the mapper and pair is the list of friends of that user.

```
Reduce(user, pair){
  While(pair.hasNext()){
    text[index++] = new Text(pair.next());
  }
  String[] list1 = text[0].toString();
  String[] list2 = text[1].toString();
  for(String useri : list1){
    for(String userj : list2){
      if(useri.equals(userj)){
        Return(user);
      }
    }
  }
}


Public void main(){

  JobConf conf = new JobConf(Facebook.class);
  Conf.setJobName("Friend");
  Conf.setMapperClass(Map.class);
  Conf.setReducerClass(Reduce.class);

  conf.setMapOutputKeyClass(Text.class);
  conf.setMapOutputValueClass(Text.class);

  conf.setOutputKeyClass(Text.class);
  conf.setOutputValueClass(Text.class);

  FileInputFormat.setInputPaths(conf, new Path(args[0]));
  FileOutputFormat.setOutputPath(conf, new Path(args[1]));

  JobClient.runJob(conf);
}
```

b. (20) Top-10 Keywords. Search engine companies like Google maintains hot webpages in a set R for keyword search. Each record $r \in R$ is an article, stored as a sequence of keywords. Write a MapReduce program to report the top 10 most frequent keywords appeared in the webpages in R. Give the pseudo-code of your MR program. Hit: You may need two rounds of MR processes for (b)

Solution:

Following is the pseudo code of my MapReduce program

Pseudo code:

Round 1:

Map function

```
  Input
  Key: object id, i
  Value: Rank, j

  Output
  Key: Rank, j
  Value: Set of r words in R
      Emit ( j, i)
```

Reduce function

```
  Input
  Key: Rank, j
  Value: List of all r records, R

  Output
  Counter = 0
  K = 10
  For each key t
    If (counter < k)
          Counter = counter + sum®
    If (counter ≥)
          Emit (t, counter -1)
  Round 2:
Map function
 Input
 Key: object id, i
 Value: Rank, j

 Output
  Key: object id, i
  Value: rank, j
  Load t;
  If (j ≥ t)
      Emit (j , i)
```

```
Reduce function
 Input
 Key: Rank, j
  Value: object id, i

  Output
  The top 10 keywords
  Load t, a
  If (j == t)
      Skip the word
  Else
      Emit (i, j)
```

In the first round we identifies the maximum score or rank threshold t, such that at least k words have a rank greater j greater than or equal to t. for each value j, the reducer keeps the counter for the number of words that have rank greater than or equal to the current data, when this counter reaches or exceeds k it stops and emits the current score t. the a stress the words that have same counts. Whereas, in the second round it gives the output of exactly 10 words with a score greater than or equal to t.

3. [Apache Spark] (30) This set of questions relate to Apache Spark

a. Explain the definition of RDD and how the lineage retrieval works
Solution:

Resilient Distributed Datasets (RDD) is one of the Abstractions of Spark. We can say that RDD is the core component of Spark. So, whenever we are operating on any data, we will be using RDD. RDD is basically read-only or we can say it makes the datasets immutable, which means that the data cannot be changed or modified, once processed. So, any changes on RDD is permanent and ensures the high-level consistency. RDD is also capable of handling any kind of data loss, for example if there is any kind of bugs or loss found, RDD itself recover from the loss and recovers the lost data.

b. List the reasons why Spark can be faster than MapReduce.
Solution:

- Performance
- Ease of Use
- Costs
- Data Processing
- Fault Tolerance
- Security

c. Explain the definitions of narrow dependencies and wide dependencies. In addition, explain how Spark determines the boundary of each stage in a DAG and why put operators into stages will improve the performance.
Solution:

In narrow dependency, all the elements of datasets which are required to do calculations are in a single node partition of the RDD. for this kind of dependency, we can say that only limited subset of a partition is used to calculate or process the result and does not require all the data in the cluster. For example, the operations for narrow dependency are filter, sample, union etc.

For wide dependency, the data to be calculated or processed lies in multiple nodes of the cluster of RDD. We can also say that there is a parent RDD in which there are multiple small partitions/nodes in which the data is saved. All the data is accessed to do calculation in this kind of dependency, and it can be computationally expensive. Examples for wide dependency operations are intersection/join, distinct, cartesian, repartition, coalesce, Groupby, reduceby etc.

DAG (Directed Acyclic Graph) is a finite direct graph with no directed cycles. There are finitely many vertices and edges where each edge is directed from one vertex to another. It contains a sequence of vertices such every edge is directed from earlier to later within the sequence. DAG operations can do better global optimization than other systems like MapReduce.
Apache Spark DAG allows the user to dive into the stage and expand on detail on any stage. In stage view, the details of all RDDs belonging to that stage are expanded. The scheduler splits the valid. Spark RDD into stages based on various transformation applied in terms of Narrow vs Wide.
Each stage is comprised of tasks, supported the partitions of the RDD, which can perform same computations in parallel. The graph refers to navigation and directed and acyclic refers to how it is done. Moreover, if we talk about how it works is that the interpreter is the first layer, Sparks interprets the code with some modifications. Then is creates an operator graph when you enter your code in spark console. When we call an Action on spark RDD at a high level, spark submits the operator graph to the graph scheduler. Then the operators are divided into stages of the task in the DAG scheduler. Stage that contains task based on the partition of the input data. Then the DAG scheduler pipelines the operators together. For example, it maps operators schedule in a single stage. The stages pass on to the Task Scheduler and it launches task through cluster manager. The dependencies of stages are unknown to the task scheduler and the Workers execute the task on the slave DataNodes