CptS 415 Big Data : Assignment 3

Md Muhtasim Billah

11/15/2020

Question 1.

1(a)

Answer:

Scaleup:

Scaleup is the ability of a computer to maintain its performance accordingly when the workload (such as instructions, transactions etc.) and the available resources (such as CPU, memory etc) for executing certain instructions are increased.

Mathematically, scaleup can be expressed as below.

$$Scaleup = \frac{Small\ workload\ performed\ on\ a\ single\ machine}{Large\ workload\ performed\ on\ parallel\ machines}$$

Speedup:

Speedup is a numeric value that is used to compare performance between two identical computing systems executing the same task with the same architecture but with a difference in available resources. In other words, speedup is the proportional reduction in runtime while performing the same task with increased number of resources.

Mathematically, speedup can be expressed as below.

$$Speedup = \frac{Time\ elapsed\ on\ a\ single\ processing\ unit}{Time\ elapsed\ on\ parallel\ processing\ units}$$

Why we cannot do better than linear speedup:

If S is the speedup of a whole task and s is the speedup for a fragment of the whole task, then by definition, inear speedup is achieved when $S = s$. The efficiency for a system like this is given by,

$$\eta = \frac{S}{s}$$

where, $0 < \eta < 1$. Tasks that run with linear speedup and on a single processor have 100% efficiency ($\eta = 1$). However, machines that are hard to parallelize, they have the efficiency of $1/ln(s)$ which approaches to 0 as the number of processors, s increases. Therefore, we cannot do better than linear speedup.

1(b)

Answer:

Given that, 40% of the program P can only be executed sequentially on a single processor, and the rest can divided into smaller tasks executing concurrently across multiple processors.

Now, we can assume that a single instruction takes 1 time unit to execute. Since, 40% of program can only be executed sequentially, it indicates that minimum 40 time units will be required to execute the program.

Now, if two machines are used, processor $P1$ and $P2$ can execute the program concurrently. P1 executes 40% of program and takes 40 time units and at the same time, P2 can execute 40% of program that is independent. Finally, the remaining 20% of the program program can executed on both the processors. So, the time cost if two machines are used for the task is,

$$T_2 = 40 + max(10, 10) = 50 \ time \ units$$

And, the speedup,

$$S_2 = \frac{100}{50} = 2$$

Similarly, the time cost if two machines are used is,

$$T_4 = 40 + max(3, 3, 2, 2) = 43 \ time \ units$$

And, the speedup,

$$S_4 = \frac{100}{43} = 2.326$$

$$T_8 = 40 + max(2, 2, 1, 1, 1, 1, 1, 1) = 42 \ time \ units$$

And, the speedup,

$$S_8 = \frac{100}{42} = 2.381$$

Now, for an infinite number of machines,

$$T_\infty = 40 \ time \ units$$

Thusm the speedup,

$$S_\infty = \frac{100}{40} = 2.5$$

1(c)

Answer:

Shared Memory:

Pros: The major advantage of the shared memory communication systems is that it the fastest. Cons: The downside for such systems are that it raises concerns such as memory protection and synchronization.

Shared Disk:

Pros: The major advantage is that data might still be accessed even when one of the nodes fails. Cons: Inter-node synchronization has to be maintained which might sometimes be difficult to achieve.

Shared Nothing:

Pros: Fault tolerance is the major advantage. Also, if one worker node fails, remaining will continue and the failed one will be restarted automatically on another node. Cons: The downside is its complex, intircate nature and very poor load balancing.

Question 2.

2(a)

Answer:

MapReduce algorithm has a `map()` and a `reduce()` function. The mapper takes a key-value pair `(key1, value1)` as input where `key1` is a person and `value1` is a list of associated friends of that person. The mapper emits a set of new key-value pairs `(key2, value2)` where key is a tuple, `Tuple2` that contains `(key1, friendi)`. Here, `friendi` $\in$ `value1` and `value2` is the same as `value1` (list of all friends for `key1`). Again, the reducer's key is a pair of two users `(Useri, Userj)` and its value is a list of sets of friends. The `reduce()` function will intersect all sets of friends to find common and mutual friends for the `(Useri, Userj)` pair.

The map(), reduce() and the main() functions for finding mutual friends are given below in pseudocodes.

```
function map(key,value){
  reducerValue = (<friend_1> <friend_2>.....<friend_n>)
  for each friend in reducerValue:
    {
      reducerKey = buildSortedKey(person,friend)
      emit(reducerKey,reducerValue)
    }
}

function buildSortedKey(person1,person2){
  if(person1<person2)
    return Tuple2(person1,person2)
  else
    return Tuple2(person2,person1)
}
```

2(b)

Answer:

The pseudocode for a MapReduce program to report the top 10 most frequent keywords appeared in the webpages in Python.

```
from mrjob.job import MRjob
from mrjob.step import MRstep
regexpr = re.compile(r "[\w]+")

class WordCount(MRjob)
    def steps(say):
        return (mapper = self.mapper_getwords,
          reducer = self.reducer_countwords),
        MRstep (mapper = self.mapper_make_countskey,
          Reducer = self.reducer_outputwords)

#first round of MR process
mapper_getwords(self, _ , line):
```

```
  words = regexpr.findall(line)
  for w in words:
    yield w.lower(), 1

reducer_countwords(self, word, values):
  yield word, sum(values)

#second round of MR process
mapper_makecounts_key(say, word, count):
  yield '%4d'% int(count), word

reducer_outputwords(self, count, words):
  for word in words:
      yield count, word

if_none == '_main_':
  WordCount Count.run()
```

Question 3.

3(a)

Answer:

RDD:

Resilient distributed dataset (RDD) is the fundamental data structure of Apache Spark. It is an immutable, distributed collection of objects. A dataset in RDD format is divided into logical partitions to be computed on different nodes on the cluster. RDDs can contain R, Python, Java, or Scala objects as well as user-defined classes. It can be converted into a spark dataframe and vice versa. In simpler words, a RDD is a read-only, partitioned collection of records.

RDD and lineage retrieval:

The principal advantage of RDD is that it is fault tolerant. It is because it tracks the data lineage information so that it can rebuild the lost data automatically in case of a system failure. When an operation (map or filter etc.) is called that transofrms the data, Spark does not execut by it immediately. Instead, a lineage is created that keeps track of all the transformations applied to that RDD, including the location from where read the data.

This can be better explained by the few lines of Spark code given below.

```
val myRdd = sc.textFile("myfile.txt")
val filteredRdd = myRdd.filter(line => line.contains("myname"))
filteredRdd.count()
```

In the code above, the function calls `sc.textFile()` and `myRdd.filter()` do not get executed immediately, rather it will be executed only when an "action" is called on the RDD, in this case `filteredRdd.count()`. Example of an action is to save a result to some location or to display it.

Each RDD contains a pointer to one or more parent nodes along with the metadata about its relationship with the parent. For example, it the map() function is called on a RDD, the RDD just keeps a reference without copying to its parent and a lineage has been established. When a job is submitted, the RDD graph is serialized to the worker nodes so that each of the worker nodes can apply the the same series of transformations (such as map(), filter() and etc.) on other partitions. Most importantly, lineage can be used in future to recompute the data in case of a system failure. This is called the lineage retrieval and one of the reasons why Spark has such great fault tolerance. To display the lineage of an RDD, `toDebugString()` method can be used.

3(b)

Answer:

The reasons why Spark can be faster than MapReduce.

- In memory data processing makes Spark significantly faster than Hadoop MapReduce. It is 100 times faster on memory and 10 times faster on disk.
- For iterative data processing, Spark defeats Hadoop MapReduce because Spark's RDDs can perform multiple mapping operations in memory, while Hadoop MapReduce has to write intermediate outputs to a disk.
- For getting immediate insights such as in cases of data streaming, Spark's in-memory computations can provide near real-time processing capabilities.

- Spark's computational model is good for iterative computations that are essential in graph data processing. Also, Spark prvides GraphX which is an API for graph computation.
- Spark has a built-in machine learning library called MLlib, while Hadoop relies on third-party inregrations. MLlib algorithms also run in memory which also makes Spark faster than Hadoop specifically for ML applications.
- Although Hadoop may perform better for combining very large data sets that requires a lot of shuffling and sorting, on smaller scales, Spark outperforms Hadoop.

3(c)

Answer:

Narrow dependencies:

If all the partitions of the parent RDD is utilized by at most one partition of the child RDD, it's termed as narrow dependency. Computations for narrow dependencies are rather fast since they do not need any data shuffling over the cluster network. Moreover, optimizations such as pipelining are also fesaible for such dependencies.

Example: Transformation operations like map, filter and union.

Wide dependencies:

WhIf all the partitions of the parent RDD is utilized by at multiple partitions of the child RDD, it's called wide dependency. For such dependencies, the computation speed can be significantly affected data shuffling around different nodes would be required while creating new partitions.

Example: groupByKey and join operations.

DAG, stage boundary and performance:

DAG or Directed Acyclic Graph, is defined by Spark as a lineage graph of RDDs which represents the data distributed across different nodes. Computations in Spark are represented by DAG and the DAGScheduler in the Spark scheduling layer incorporates a stage-oriented scheduling. It converts a logical execution plan into a physical execution plan based on several stages. A stage is, in simple words, a step in the physical execution plan. In other words, each job which gets divided into smaller sets of tasks is a stage and that is how the boundary for each stage is determined.

Putting the operators into stages improves the overall performance because plans are executed in stages rather than performing them all at once. The data doesn't have to be copied along the way, neither all the operators. Taking the help of lineage retrieval and the DAG, Spark can reinitiate a task automatically from a specific point or stage of its whole execution plan rather than starting from the very beginning. This not only helps save disk space but also saves some computation which eventually leads to more available memory for computation. That is why staging and keeping the operators in stages improves the performance.