Assignment 1: Basics and Map Reduce

Group 1 Jin Yin a1761810. Yingyao Lu a1784870

Exercise1, Suspected Pairs (10 points)

1, The number of days of observation is 5000; and

2, The number of people observed was raised to 5 billion (and there were therefore 500, 000 hotels); and

3, We only reported a pair as suspect if they were at the same hotel at the same time on four different days.

In this case, the chance that they will visit the same hotel on one given day is:

Four days the, the possibility is

The number of pairs in the people is

The number of pairs of days is

So, the amount of the people is

Exercise 2, Hadoop (20 points)

Hadoop output please find in attached files.

Exercise 3, Friend Recommendation System (Stanford) (35 points)

Source code and Hadoop output please find in attached files.

Include in your writeup a short paragraph describing your algorithm to tackle this problem.

924 439,2409,6995,11860,15416,43748,45881

8941 8943,8944,8940

8942 8939,8940,8943,8944

9019 9022,317,9023

9020 9021,9016,9017,9022,317,9023

9021 9020,9016,9017,9022,317,9023

9022 9019,9020,9021,317,9016,9017,9023

9990 13134,13478,13877,34299,34485,34642,37941

9992 9987,9989,35667,9991

9993 9991,13134,13478,13877,34299,34485,34642,37941

Exercise 4, MapReduce (15 points)

Q7: How many words are there with length 10 in FirstInputFile?

• Q8: How many words are there with length 4 in FirstInputFile?

• Q9: What is the most frequent length and what is its frequency in FirstIn- putFile?

• Q10: How many words are there with length 5 in SecondInputFile?

• Q11: How many words are there with length 2 in SecondInputFile?

• Q12: What is the second-most frequent length and what is its frequency in SecondInputFile?

Exercise 5, Summary of 2.4 and 2.5 (10 +10 points) (Postgraduate Students (COMP SCI 7306) only)

# Summary Section 2.4 Extensions to MapReduce

This section introduces extensions and modifications of Hadoop MapReduce system. The most popular systems include UC Berkeley’s Spark., Google’s TensorFlow, and a graph model of data, Google’s Pregel.

MapReduce paradigm consists of a simple two step structure, Map and Reduce. It can solve most massive data processing problems. However, MapReduce model has a few limitations when deal with complicated tasks. MapReduce save intermediate results on local file system of Map and Reduce workers. And in complicate cases, one output is often input to another MapReduce task. This will require repeated read from and write to disks. It will then require more job completions time for run through multiple steps and multiple jobs.

Spark, TensorFlow, Pregel and other MapReduce extensions are all use a “Workflow” architecture. They share three major characteristics with MapReduce.

1. Build on a distributed file system.
2. Manage tremendous tasks, whereas only need to write small number of functions.
3. Handle failures occur during execution without restart job all over.

Workflow systems improved MapReduce by using an acyclic graph to deal with any collection of functions. Workflow systems use a master controller for dividing the works among the tasks by hashing inputs. The output of function f will be passed as inputs of f’s successors g and i. Workflow systems use effectively cascades of MapReduce jobs, that can significantly reduce communication cost that read and save to local files between chaining job tasks.

Workflow inherits MapReduce’s blocking property by only deliver completed output. If a task fails, its master control can easily kill the failed task at that node and restart that task.

Diagram

Description automatically generated

## Spark

Spark improves many MapReduce drawbacks, while keeps many benefits.

Spark Implementation is different from MapReduce in many of aspects:

* Performance

Spark could drastically speed up large scale of big data tasks, because it utilises RAM to process data in memory, while MapReduce persists data back to the disk after each Map-Reduce task. This allows Spark save communication time between tasks.

Same as MapReduce, with workflow architecture, Spark breaks down large dataset and process them in parallel. However, Spark works well for smaller data sets that can all fit into a server's RAM.

* Ease of use

Spark has a faster learning curve than MapReduce. Spark provide pre-built APIs for Java, Scala, Python, and R, etc. It is easy to program user-defined functions for different developers. Whereas MapReduce is written in Java.

Spark includes a core data processing engine, as well as libraries for SQL, machine learning, and stream processing.

* Compatibility

Hadoop focus on process key-value pairs as inputs and outputs. Whereas Spark is more flexible. Spark is compatible with all of Hadoop’s data sources and file formats. In addition, Spark use a Resilient Distributed Dataset (RDD), that is distributed and fault-tolerant and not restricted only for key-value pairs in the MapReduce. Spark use transformation and action operations that apply one RDD to produce another RDD such as Map, Flatmap, and Filter operations.

* Data processing

Hadoop MapReduce is great for batch processing. Whereas Spark can do much more. Spark can do real-time processing due to its high performance. Spark is capable to process graphs and deal with machine learning tasks. Spark offers a "one size fits all" platform that you can use rather than splitting tasks across different platforms.

TensorFlow

TensorFlow is another workflow system and use a multidimensional matrix instead of RDD in Spark model. It supports machine-learning with an easy-to-use built-in operation.

Recursive Extensions to MapReduce

Another main stream of extensions to MapReduce adopt recursion approach. It recursively use MapReduce job for a unknown steps until the result of two consecutive iterations are close enough. A few classic uses of recursive algorithm include PageRank and gradient descent.

However, recursions approach has limitations for failure recovery. Three different approaches have been used: Iterated MapReduce, Spark Approach and Bulk Synchronous Systems.

Pregel Bulk Synchronous Systems

Google’s Pregel system is a graph-based, bulk-synchronous system that consider its data as a graph. Each node is viewed as a task, and generate outputs as the inputs for other graph nodes. Computations are grouping as supersteps, where all messages received by any nodes at previous supersteps are processed and then generate new messages to destination nodes. This grouping message will make communication great but very short.

Pregel failure management is designed as checkpoints at certain supersteps, so that will not restart failed tasks.

# Summary of Chapter 2.5 The Communication Cost Model

This chapter mentioned some methods to evaluate the performance of algorithms implemented on a computing cluster which is acyclic workflow. the bottleneck of this method is moving data among tasks, such as transporting the outputs of Map tasks to their proper Reduce tasks.

If an algorithm is implemented in the acyclic network, the output of these tasks could be input of the Map tasks. Such as standard MapReduce algorithm, the output in MapReduce jobs cascaded, and other general algorithms.

The communication cost of an algorithm is defined as the sum of all the communication cost produced during computing basing on this algorithm. This cost is considered as an important cost to measure the quality of the algorithm. we do not consider the amount of time it takes each task to execute when estimating the running time of an algorithm.

There are some reasons why communication cost is important. Firstly, the task in each node is simple. The complexity is linear in the scale of data input. Secondly, communication speed is lower than CPU speed. There is a competition in many cluster architectures, which also would enhance the communication cost. As a result, the compute node can run the tasks on a received input element after it takes to deliver that element.

Only input data size is considered because the output data size of one task should be as same as the input data size of following task, unless this output is the result. If the output is so larger that it is more than the input size, it is necessary to implement aggregation to reduce the output size, and normally, it is executed in reducer. In this case, the result will be sent to another collection to implement this aggregation. So that the communication cost is always proportional to the computation.

An example is used to explain how to calculate the communication cost in , and the side of is , and the size of is . The sum of the communication costs for all the map tasks is .

On the other side, wall-clock time which is the time that parallel algorithms to finish tasks. We can optimize communication cost by adjusting task distribution on different computing nodes. However, this algorithm is takes high wall-clock time.

Meanwhile, In the cluster-computing environment, some methods are mentioned to analyse the communication cost. A general theory is introduced.

Firstly, certain attributes need to be selected in the join with more than two relations and their values would be hashed and assigned to some number of buckets.

Secondly, select the number of buckets for each attribute, and use the product of these numbers k, as the number of reducers that will be used.

And then identify each of the k reducers with a vector of bucket numbers. These vectors have one component for each of the attributes selected in the first step.

At last, send tuples of each relation to all those reducers where it might find tuples to join with. Other components of the vector are unknown, so it must be sent to reducers for all vectors having any value in these unknown components.

Another example is provided in this section. and R, S and T have sizes , respectively. And is the probability that an R-tuple and S-tuple agree on B, and also the probability that An S-tuple and a T-tuple agree on C.

If we join and first, Using MapReduce algorithm of above sample, the communication is . And the probability of is . Before get the input, the communication cost is . In sum, the total communication cost of the algorithm is .

Similarly, if we join and first, the communication cost is .

There is another way, which is use a single MapReduce job that joins the three relations together. In this case, we need more than one reducer to finish aggregation tasks.