Week 5 lecture notes

# Part 1: Spark & RDD

This week we move to a new big data processing platform, Spark.

In week 3 & 4 we learned MapReduce, and the basic operations are Map and Reduce. Mapper aims to transform your input to some key value pairs as intermediate data. Next, intermediate data is forwarded to the reducer for aggregation, to generate the final output. For each mapper there will be optionally a combiner, if you would like to reduce the size of the mapper output to improve the performance.

We can see that the flow of data from input to mapper, to combiner, to reducer, to output, they are all based on disks. You read data from HDFS and next in the mapper you generate some key value pairs, which are written to mapper nodes local file system. The data is then copied from the mapper computer to the reducer computer, using the network. Then in the reducer we do the aggregation and attain the final result. The result will be written back to the the disk, to HDFS.

So everything related to the data flow is based on disk.

MapReduce is very scalable, and we can use it to do very large scale data management. However a problem arises when your mapper requires multiple MapReduce jobs, like a chain of jobs.

However MapReduce is not always so suitable.

* For some applications like machine learning or graph data management, we always need to have multiple round jobs to complete the entire task. Between different rounds of jobs, you need to also use the HDFS, to share the data between different MapReduce jobs.
* And also for some interactive queries, like in RDBMS, we can use queries to search for and access data. Map reduce is not very suitable for these interactive queries.
* Some other applications require real-time streaming, and the data is dynamic, like twitter. Again in this case it is assumed that the data is already stored in HDFS, so it does not support dynamic data.

So for these 3 applications we need to think about some other ways for managing big data. This is the reason for the development of Spark.

MapReduce has some other limitations. Ref slide 1.5 & 1.6.

This week we start to learn Spark which is more efficient and easier to code, compared with MapReduce. In MapReduce we used a lot of disks and lots of CPUs. What we did not use in MapReduce is memory (RAM).

## Goals of spark

The objective of Spark is

* to try to keep as much data as possible in memory. Memory is inherently more efficient as there is no I/O cost.
* In MapReduce you can only use map and reduce. In spark you have much more functionality to do tasks. This allows spark to support more data processing applications.
* Enhanced programmability. Scala is the default programming language for spark but you can also use other languages like Java or Python. Scala is not a very difficult language if you already know python. Should be easy to move from Python to Scala.
* These days, MapReduce is not used that much unless your data is really really huge. In most cases these days Spark is more efficient.

## What is Spark?

* Opens source engine for large-scale distributed data processing. Supports generalised data flows and has bindings in Java, Python and R.
* Developed in Berkley in 2009
* Open sourced in 2010
* Became top level apache project in Feb 2014
* Commercial support provided by DataBricks

Spark is the basis of a wide set of projects in the Berkeley Data Analytics Stack (BDAS), including spark SQL, Spark streaming, GraphX, Mllib. In this course we only use Spark Core, so there is a lot of other stuff to learn

**see slide 1.19 for information about sparkshell**

* sc is used for creating RDD’s. This is the focus of week 5
* spark is used for dataframes. This is the focus of week 6.

## What is RDD?

A common misconception is that you cannot distinguish between RDD and normal python object. This is not correct.

RDD is a spark API, stands for Resilient Distributed Dataset. Resilient means that it is fault tolerant. It is able to recover the data if some piece of the data is lost.

Emphasise again: our course is Big Dat Management. It means that the data is not stored on a single machine, it is distributed between many machines (nodes) that make up a cluster. We need to think about how the data is split into different partitions, and based on this distributed data, how we can do distributed computation.

RDD is the primary data abstraction in Spark and the core of spark, but since spark v2 we begin to use the dataframes API. This is based on RDD but spark will be able ot know more detail, and can apply more operations to a task.

What is the difference between RDD and normal python object? The difference is that for RDD, it means that this piece of data is stored in different computers. So you are not able to process the data based on a single computer as you normally would in Python. If you want to see the data you have to collect them from all the computers in the cluster to the driver node. Some network cost will be incurred.

For a normal Python object, it is always stored in the driver node. If you want this to be part of the computation you need to send the python object to the nodes.

Looks like we use a lot of lambda functions in spark!

## Setting the level of parallelism

By default, Spark will check your resources and check the size of the data and then decide how many partitions Spark will use store your data in the RDD. But you can also manually tell Spark how many partitions you want to have for your RDD.

In MapReduce the data is already split for you. Its stored in HDFS as data blocks. But in Spark, we don’t have this kind of data block, but we have partitions for our data. If you have more computers in the cluster you can always increase the number of partitions to achieve the scalability, so more jobs can be run in the same time.

## Spark Workflow & components (Spark 3.x)

See slide 1.25, - 1.34

## How Spark works: Example

Slide 1.35 – 1.42

insert image

reduceByKey: we are going to do the reduce operation by keys. Word is the key, count is the value. reduceByKey will apply the function you define here over the values for the same key. So in the word count example, it will count all the values for a particular word. You always need to have 2 variables for the reduceByKey operation.

The try it: per key average in week 5 provides a great example.

1:06:48 the top 3 task will help you to do part A of the assessment.

Execution plan: the stages will form a DAG

1:12:09 mapValues is useful to do the assessment

in the average computation example you will also see some examples of mapValues. After you complete that task you will have a better understanding of RDD transformations.

NB: slides 1.45 – 1.46 use scala syntax. We would need to write lambda functions.

1:13:50 shows a link to a useful website

## Part B: Assessment

Marking criteria:

* can be compiled and run on Spark
* correct number of books ordered each day
* correct order
* correct output format

## Part A: Assessment

Marking criteria

* can be compiled and run on spark
* correct top-3 terms for each year
* correct order of the terms and years
* correct output format
* achieve good efficiency – how to find the top 3 result

top k most frequent pairs example: this activity aims to do two things. The first is to perform self defined operations/functions. Normally we use short functions like lambda functions, but in this case we need to write a more complicated function. Then this function can be passed to flatmap as an argument.

More importantly is how to do the top-k computation. The solution presents two ways. The first uses sortBy. Another way is to takeOrdered. **The difference between the two is that sortBy is not as efficient as takeOrdered.**

The reason is that sortBy requires you to first sort all the elements, it means that the run time required is the size of the list times the log of the size of the list.

However takeOrdered will not do the sorting for all the elements. We only keep the top-k. If its not in the top-k we just throw it away. If it is in the top-k then we insert this new element into the top-k then remove the least frequent one. In this way you only need to sort the top-k to achieve the final output. This means you have a lot less terms to sort and its a lot more efficient.

The efficiency is mainly about this part. Maybe not exactly follow the takeOrdered operation, but think about what is the more efficient way of doing it. 2 marks for efficiency.

**Best way of debugging**: you have to collect the information from RDD into a python object and then print out the python object.

You can have more self defined functions for Part A and not affect efficiency, depending on how you use the self defined functions. Case by case.