Cluster Computing ITNPBD7

Student ID: 2727809

HDFS

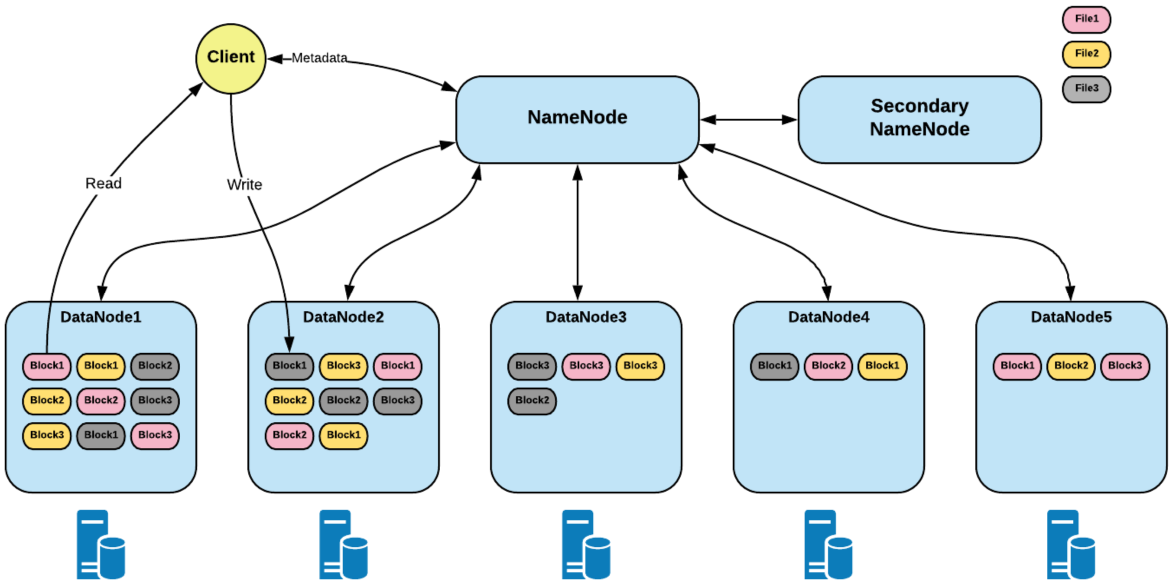
We are living in the era of datafication, which means that almost everything around us is a data source. People try to receive data from every possible resource because data itself is a new level of modern intelligence. All of this leads us to the necessity of storing and processinga large amount of data.When it comes to storing huge volumes of data, a single physical storagesometimes cannot storethe data becausemodern data volumes are much larger the modern storage's capacity. Thus, spreading the data across several physical machines become a necessity when using "Big Data".

Hadoop with HDFS provides the solution for these problems. HDFS means Hadoop Distributed File System and it was designed for storing very large fileson a network cluster. Despite the fact that Hadoop was designed as a production system, HDFS can be run on a cluster of commodity machines, which makes it more flexible and adaptable to modern needs.

The concept of HDFS is to store data on disk space in blocks. The default volume of each block is 128MB, but it is also possible to use 64MB blocks.This means that we can split a huge file into multiple blocks and store them on any of the cluster nodes. Each block by default is replicated on several machines (default number is three), and because of this we get a fault tolerant infrastructure with an opportunity to store very large files.

There are two types of Hadoop cluster nodes: name node (master node) and data node. The name node is responsible for file management in the file system and for storing metadata for all files and directories. More precisely, the name node contains information about which data node storeswhich blocks for a given file (files tree). Also, the name node is called the master node, because it is the most important node and without a master node, a filesystem cannot operate. Moreover, Hadoop provides the opportunity to run a secondary name node for a resilient reason.The data node contains the data itself and reports to the name node the information stored about the blocks.

Picture 1.1 depicts a Hadoop distributed file system.The HDFS architecture in the picturecontains a secondary name node for a resilient reason and five data nodes (arbitrary number). The client stores 3 files, which Hadoop spread across five data nodes.In the picture, the files have different colors to make recognition easier. As we can see, each file was split into 3 blocks and each block has three replications.The links in the picture, which are connecting name node and data nodes, are logical meaning that the nodes can be connected with each other indirectly, through arbitrary networks and can be spread evenly throughout the world.



Picture 1.1 Hadoop distributed file system architecture

In the real world, our goal is to process data which is already spread across and stored on the HDFS cluster, but in our academic environment, we need to have the data before processing. For this reason, we have to create a directory on the HDFS and move the file or files to the directory in order to processit using the following commands:

```
hdfsdfs -mkdir /user/akr/count
hdfsdfs -copyFromLocalratedReview.txt /user/akr/count
hdfsdfs -copyFromLocalexclude.txt /user/akr/count
```

As all students have a personal directory (in our case it is `akr`) onthe HDFS,we have to work inside it. First, we create a directory `count` in the `akr` using the `-mkdir` command. Then, we have to use the `-copyFromLocal` command and point out the full path of transfer each file takes into theaimed directory on the HDFS.

Map/Reduce design

The technique called Map/Reduce was implemented to process the audience movie reviews. All reviews are stored in a text filecalled ratedReview.txt. The file consists of lines of text (the review itself) and a rating score, where the given rating is separated from the corresponding line of text by a tab character. The aim is to find the most common word used with each possible rating number, with the rating ratio being from 1 to 9.The task can be solved by implementing two different techniques.

Technique 1:

As the Map function gets one file line of text at a time, we have to take this into account while designing a program algorithm. After getting one line of text and tab separated rating, the first thing we have to do is to split the text and corresponding ratingshown by the tab delimiter into two parts. We do not need to apply any manipulation for the key, we keep it pristine and write it into context as the main rating identification. When it comes to the review itself, we have to split the text into separate words first. Then by iterating the words, we have to write as values to the context, in order to emit them to a reducer. This means that we write into the context key as a ratingand a value, which represents a word from the text line.

If we use one reducer, all key/values pairs will be transmitted to it. As we are aiming to builda scalable design for arbitrary number of reducers, we have to clearly understand how the data will be processed across several reducers and take into account in the program algorithm.As all pairs of key/value with the same key will passto the same reducer, all values with the same key will be aggregated to process by one reducer as well. As a result, our mapper emits to the particular reducer a set of key/value pairs for each line of the rating text where the key is unique for the line andplaystherole of aggregator for further processing on the reducer node, and each value is the single word from the line of text.

Then, all data associated with a particular key is aggregated together and passes to the reducer input. By default, in the Hadoop cluster, there is one reducer, but we can change this value for our needs. In a schema with multiple reducers, several reducers will get a different set of keys and its correspondent data to process.

After getting the data, the reducer fills aHashMap for each key (rating) with its own key/value pairs, where the key is a word and the value is its quantity in the reducer input line. When the HashMap gets the word, which already exists, it will increase the value by one. Finally, we iterate through the values of the HashMap and find the maximum value and correspondent for its key. Before getting a new line, we clear the hash map to fill it with new data corresponding with the new key.

Technique 2:

The second approach is aimed to reduce the amount of Map function work and data traffic during data transfer from Map node to Reduce node across a network. For this reason, it has been decided to improve the first technique by implementinga HashMap inside the mapper method. The idea is that iterating through the initial lineof text within the map functionand filling the HashMap with the key as a word and the value as a word counter for a particular line of text.

When the same word occurs an additional time,the program increases the counter by one. As a result, we get a HashMap with all the words from the line and the corresponding word counters. As the mapper must emit to the context only key and value, and a key field is already filled witha rating value, we merge the pair from the HashMap and write it as a single value into the context, as merged word and its counter. In order to split it on the reducer side, we have to merge it with the delimiter between them. By implementing HashMap on the Map phase, we are able to count repeated words in the text and avoid passing from the mapper to the reducer repeated key/value pairs. Consequently, by implementing this in comparison with the first design, we are decreasing the amount of traffic moving across the network. The reducer phase for both techniques is the same.

In both cases, the number of reducers could be arbitrary. This is due to the fact that no matter if one or more reducer is used, all data related tothe particular key will be aggregated into the same reducer.

One more possible solution is to engage a partitioner to force the data to be processed by the specific reducer. It would likely be a more sophisticated solution for our task, but it isoutof the scope of our course.

I have chosen to implement the second technique due to its more optimized code and higher efficiency of network usage.In both designs, we intentionally have not mentioned anything about a combiner. This is due to the fact that the combiner works on the Map node and, as a result, works with the data which would be emitted by one mapper.If we speak about scalable design, for our task, the combiner could calculate the most common word just for one mapper.Thiswould be incorrect to use a combiner because we lose the rest of the words and theircounters needed to calculate them after final aggregation at the reducer node which is “responsible” for the final calculation of most common word for a particular rating.

Result

1	character
2	good
3	good
4	good
5	good
6	good
7	director
8	life
9	masterpiece