

Creating an Inverted Index using Hadoop

Phase 1

Code link: <https://github.com/yzlucas/invertedindex>

Abstract

An inverted index is a data structure storing a mapping from content, such as words or numbers, indicating its location in a document or group of documents. With the inverted index, we only have to look for a term once to retrieve a list of all documents containing the term. The advantages of inverted index are:

- Inverted index allows fast full text searches, at a cost of increased processing when a document is added to the database.
- It is easy to develop.
- It is the most popular data structure used in document retrieval systems, used on a large scale for example in search engines.

High level diagram single system prototype versus a distributed system prototype

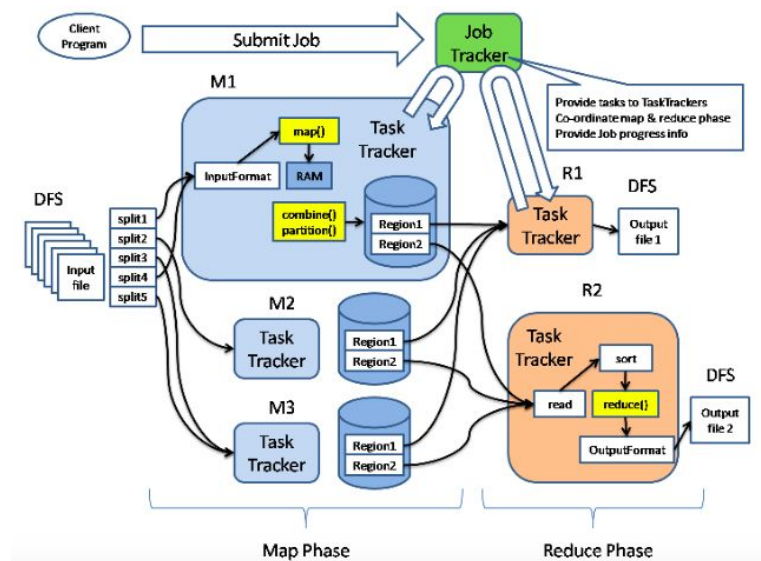
With the single system prototype, we have to repeat the procedure of tokenizing, word count and store results on all input files. It is not efficient. We apply Hadoop (Hadoop Distributed File System & MapReduce) on the same problem. All input files could be handled simultaneously. It saves us much time and space.

Hadoop/Inverted Index Background

Hadoop was released in 2005 by the Apache Software Foundation. It is an open-source software framework for storing data and running applications on clusters of commodity hardware. The flexible nature of a Hadoop system means companies can add to or modify their data system as their needs change, using cheap and readily-available parts from any IT vendor.

There are two important parts of hadoop: (1) Distributed File System, which allows data to be stored in an easily accessible format, across a large number of linked storage devices. (2) MapReduce - which provides the basic tools for poking around in the data.

The structure of Hadoop system:



We were facing some challenges when installing and setting up the Hadoop environment on Mac. We have to configure some of Hadoop's files such as core files, hdfs files and mapred files etc to connect Hadoop daemon.

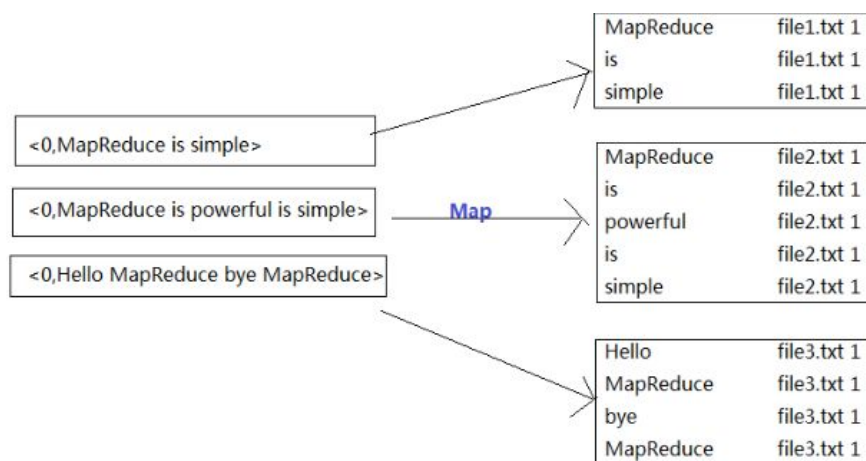
Implementation:

We used hadoop framework to help us implement inverted index:

There are three main parts: mapping, combining and reducing:

Mapping:

Map input files and get three variables: words, URI (words' location) and words' frequency (initialize to 1 for each word).



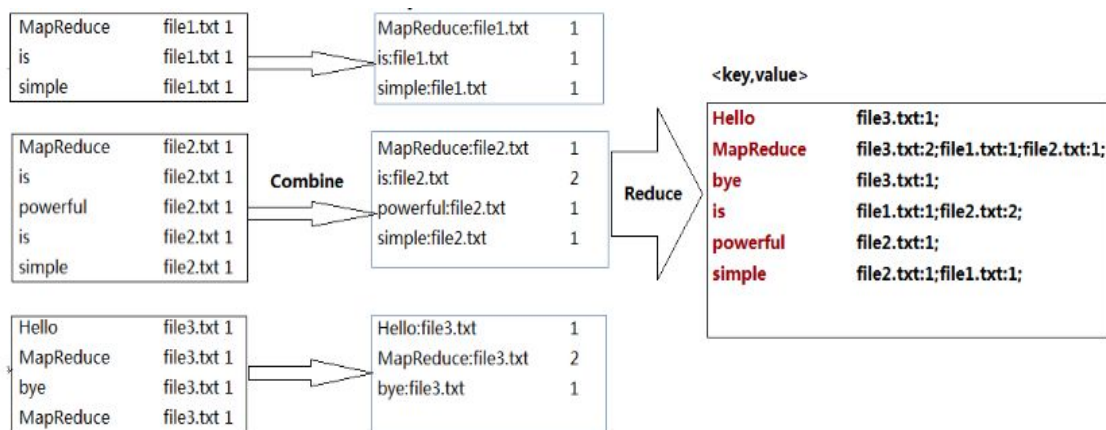
Combining:

As we only want two variables for “key-value” pairs and current values cannot be modified, we combine “words” and “URI” into the key and set “word frequency” as value.

Reducing:

We use the new “key-value” pairs as input and start reducing.

The output of Reduce Task would be the output for the whole job and be stored in HDFS.



In order to input the text files, we needed to upload them onto the HDFS daemon.

The output files will also be stored in its server.

Conclusion / Result

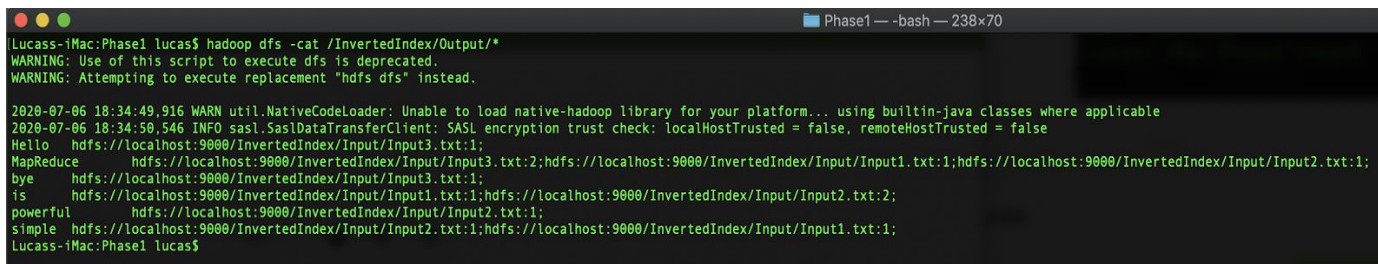
We have three input file:

input1.txt:MapReduce is simple

input2.txt:MapReduce is powerful is simple

input3.txt:Hello MapReduce bye MapReduce

Test result:

A terminal window titled 'Phase1 - bash - 238x70' showing the execution of a Hadoop MapReduce job. The user runs 'hadoop dfs -cat /InvertedIndex/Output/*'. The output shows the contents of three files: 'Hello', 'MapReduce', and 'bye' from 'input3.txt'; 'is', 'powerful', and 'simple' from 'input2.txt'; and 'bye', 'powerful', and 'simple' from 'input1.txt'. The terminal also displays several warnings and log messages from Hadoop, including a deprecation warning for 'dfs' and a warning about the native-hadoop library.

```
Lucass-iMac:Phase1 lucas$ hadoop dfs -cat /InvertedIndex/Output/*
WARNING: Use of this script to execute dfs is deprecated.
WARNING: Attempting to execute replacement "hdfs dfs" instead.

2020-07-06 18:34:49,916 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
2020-07-06 18:34:50,546 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localhostTrusted = false, remoteHostTrusted = false
Hello  hdfs://localhost:9000/InvertedIndex/Input/Input3.txt:1;
MapReduce  hdfs://localhost:9000/InvertedIndex/Input/Input3.txt:2;hdfs://localhost:9000/InvertedIndex/Input/Input1.txt:1;hdfs://localhost:9000/InvertedIndex/Input/Input2.txt:1;
bye  hdfs://localhost:9000/InvertedIndex/Input/Input3.txt:1;
is  hdfs://localhost:9000/InvertedIndex/Input/Input1.txt:1;hdfs://localhost:9000/InvertedIndex/Input/Input2.txt:2;
powerful  hdfs://localhost:9000/InvertedIndex/Input/Input2.txt:1;
simple  hdfs://localhost:9000/InvertedIndex/Input/Input2.txt:1;hdfs://localhost:9000/InvertedIndex/Input/Input1.txt:1;
Lucass-iMac:Phase1 lucas$
```

Hadoop includes MapReduce and HDFS. Using MapReduce in Hadoop is more convenient than developing MapReduce from the bottom. It handles large amount of data by breaking it into multiple sets and completing the jobs on the data over server nodes. We tried our small data samples by storing them in txt files and upload to hadoop daemon. The searching speed of the inverted index is faster than the forward index in lab1.

Plans for Phase II:

For Phase II, we plan to switch our input data from text files to more complex data such as SQL database, which is more practical and suitable for real world scenarios. We considered including SQL queries in the codebase, or saving SQL tables in XML format at this moment, but no decision has been made yet. Furthermore, we are going to implement the input files on the location machine instead of the server, because uploading files onto the server looks superfluous.