MapReduce

Authors: Jordi Toda and Meritxell Gomez
Subject: Distributed Systems
Date:26/06/2018

Introduction

MapReduce is a programming model and implementation to enable the parallel processing of huge amounts of data. In a nutshell, it breaks a large dataset into smaller chunks to be processed separately on different worker nodes and automatically gathers the results across the multiple nodes to return a single result.

As it name suggests, it allows for distributed processing of the **map()** and **reduce()** functional operations, which carry out most of the programming logic.

Here you can see the code: https://github.com/jorditoda/MapReduce

Goals:

- A WordCount function that count how many times a word appears in a file.
- A CountWord function that count the total number of words of a file.

The main objectives of this project are:

- To learn the implementation of basic Distributed Systems.
- See the speed up between a sequential WordCount/CountWord with its distributed implementation.

The steps used for the distributed implementation are the following ones:

- "Map" step: Each worker node applies the "map()" function to the local data, and writes the output to a temporary storage. A master node ensures that only one copy of redundant input data is processed.
- "Reduce" step: Worker nodes now process each group of output data, per key, in parallel.

To evaluate:

- Take the time of the sequential part.
- Take the time of the distributed part.
- See the speed up.

Implementation

CountWordSequencial:

It counts how many times appears a word in a text.

WordCounterSequencial:

It counts all the words in a text.

Mapper:

It creates a host and gets registered in the registry class. And starts serving.

Registry:

It creates a host and waits until the mappers and the reducer get registered. It have some functions.

- bind: Register a host in the dictionary.
- unbind: Deletes a host from the dictionary.
- lookup: Returns the proxy associated to the name entered by parameter.
- get_all: Return all the values of the dictionary.

Reduce:

It creates a host and gets registered in the registry class. And starts serving.

Master:

It creates a host and checks that the file we want to read is already downloaded or not. It counts all the lines to divide them to all the mappers. Catch and delete the reduce proxy from the registry list and catch the rest of the proxys to count how many mappers we have. We calculate the number of lines of the text for each mapper. We make a spawn of the reducer and another for the master. After that for all the mappers we do a spawn and send the mapper the information needed for their work (the text, the line_start, the line_finish, the reducer's spawn, the master's spawn, the number of mappers, the time of the starting execution.) only for the WordCount. After waiting the until the user taps the enter to start the CountWord part and gives to the mappers the same information as before for counting. It serves forever because it waits for the reducer's respond to print it.

WordCount:

- wordCount:It counts all the words in a text.
- puntuation: Changes all the punctuation symbols into spaces and the capitals letters to lower letters.

CountWord:

- countWord: It counts how many times appears a word in a text.
- add: It adds a word into a dictionary if it doesn't exist. If it exists this add one value to the word.
- puntuation: Changes all the punctuation symbols into spaces and the capitals letters to lower letters.

Echo:

- echo: It prints a string.

ReduceMapper:

- addR: It adds a word to the dictionary if it doesn't exists. If the word exists it adds one value to the times it has appeared.
- reduceW: It collects all the words of each mapper in the same dictionary and also returns the time that has used in calculate all of them.
- reduceC: It collects all the word counted of each mapper and returns the time used to do it and the total number of words.

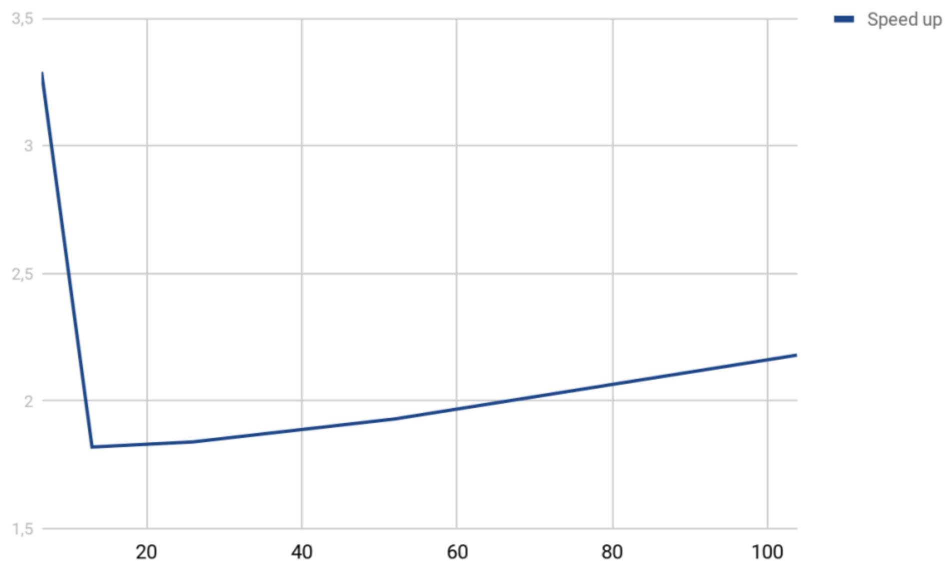
Results

For each function we’ve made a graphic that shows the speedup of a distributed in front of a sequential implementation for different size of files.

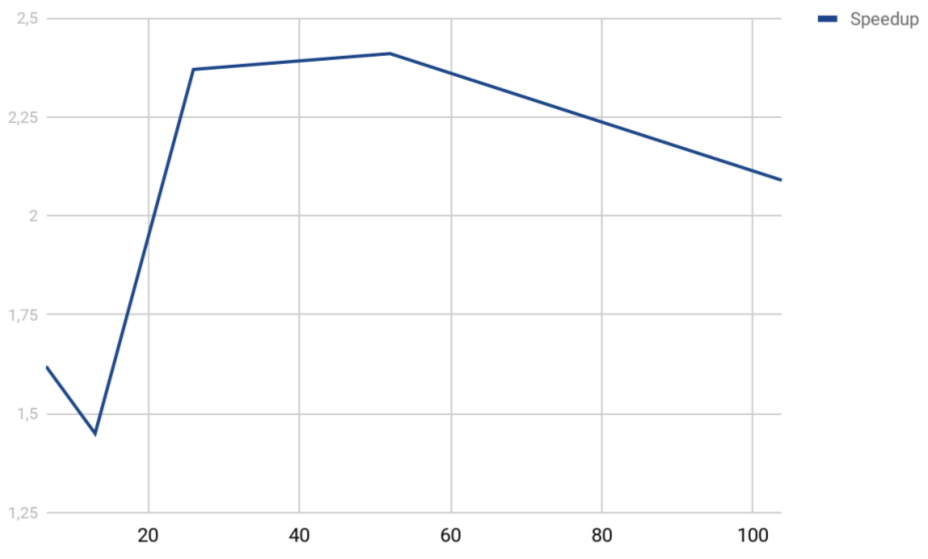
We’ve used the file of sherlock.txt and we`ve multiplied it every time until we``e arribet to 103,8MB.

The graphics are:

WordCount:

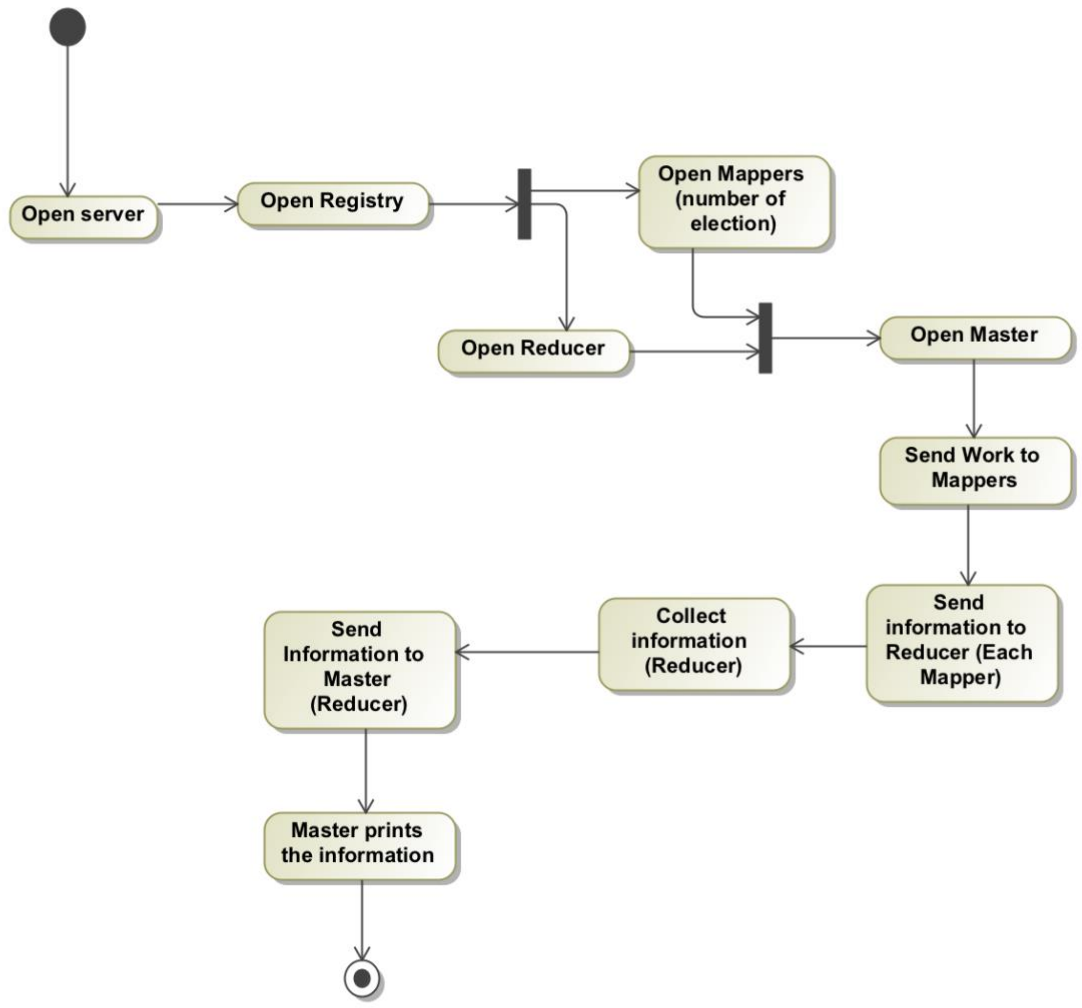


CountWord:



In this graphics we have used one computer for all the work. This implementation was made to work in more computer for a better speed up, so if we use a computer for each Mapper we increment the speed up.

UML


Open server
Open Registry
Open Mappers (number of election)
Open Reducer
Open Master
Send Work to Mappers
Send information to Reducer (Each Mapper)
Collect information (Reducer)
Send Information to Master (Reducer)
Master prints the information