FREE eBook

LEARNING mapreduce

Free unaffiliated eBook created from
Stack Overflow contributors.

#mapreduc

e

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: mapreduce

It is an unofficial and free mapreduce ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official mapreduce.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with mapreduce

Remarks

This section provides an overview of what mapreduce is, and why a developer might want to use it.

It should also mention any large subjects within mapreduce, and link out to the related topics. Since the Documentation for mapreduce is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

Mapreduce is a part of Hadoop. So when Apache Hadoop (or any distribution of Hadoop is installed) MR is automatically installed.

MapReduce is the data processing framework over HDFS(Hadoop distributed file system). MR jobs maybe written using Java, python, Scala, R, etc.

What does mapreduce do and how?

Mapreduce is a programming model to do processing on (very) large amounts of data.

Traditional 'HPC' (High Performance Computing) speeds up large calculations on relatively large amounts of data by creating a set of highly connected computers (using things like extremely quick networking, and quick access to shared storage, shared memory) to handle computing problems that usually require calculations to have access to each others data. A classic example is weather forecasting.

Mapreduce on the other hand excels at handling relatively small, independent calculations on enormous amounts of data. To make this possible, the data is spread across many computers (due to the amount of data), and the desired calculation is split into a phase that can be done on each bit of data independently (the 'map' phase). Results of these independent calculations are then gathered and a second part of calculations is done to combine all these individual results into the end result (the 'reduce' phase).

Example: Counting votes

Imagine you have a very large amount of votes to count, and there is a bit of work to count each vote (e.g. finding out from the scanned image which box was ticked).

In this case, a mapreduce implementation would:

Step 1: 'Spread'

Spread the images to process over the available computers.

Step 2: 'Map'

On each computer, for each image:

- take in 1 of the images copied to this computer as an input
- find out which box was ticked
- output the number (or code or name) of the item voted for

Note that work can start as soon as a computer gets 1 image to work on. There is no need for all these computers to interact to do their work, so there is no need for them to be interconnected quickly, have shared memory or shared diskspace.

Step 3: 'Gather'

Gather all these outputs on 1 computer.

Step 4: 'Reduce'

Count how many votes for each number (or code or name) there are.

This very basic example also highlights how further optimizations are often possible. In this case the reduce step itself can clearly be done partially on each computer, and then a final reduce can be done on a central computer. This will both reduce the amount of work on the one computer running the reduce step, and limit the amount of data that needs to be transported over the network.

Example: Counting votes - optimized(by using combiner)

Step 1: 'Spread'

Same as before: Spread the images to process over the available computers.

Step 2: 'Map'

Same as before: On each computer, for each image:

- take in 1 of the images copied to this computer as an input
- find out which box was ticked
- output the number (or code or name) of the item voted for

Step 3: 'Gather' locally

Gather all the outputs of 1 computer on the computer itself.

Step 4: 'Reduce' locally

Count how many votes of each number (or code or name) there are in the local results and output these counts.

Step 5: 'Gather' globally

Gather all the outputs of the local reduces on 1 computer.

Step 6: 'Reduce' globally

Sum up the locally made counts of votes of each number (or code or name).

Note that in step 3 it is *not necessary* to wait for **all** results in any of the below cases:

- if this becomes too much for the computers local resources like storage/memory
- if the cost of the work to be redone when a computer breaks down is deemed to big to wait for all local results
- if the network is now free to transport intermediate results

the local gathering and local reducing can be done on the results produced so far on the local computer, and this can be done at any time.

The local reduce step is called the combiner step. This is an optional step used to improve performance.

Read Getting started with mapreduce online: https://riptutorial.com/mapreduce/topic/2192/getting-started-with-mapreduce

Credits

S. No	Chapters	Contributors
1	Getting started with mapreduce	Ani Menon, Community, Legolas