Pattern definition of MapReduce

Joonas Salo

Department of Software Systems
Tampere Univeristy of Technology
PL 553, 33101 Tampere, Finland
joonas.salo@tut.fi

1 Introduction

MapReduce is a programming model that was introduced in 2003 by Google [1]. The idea is to split the input data so that it can be processed in small items or groups of items in parallel. At Google the model is was first used to build the search index and later it has also been found useful in many other contexts.

This paper was motivated by the success of MapReduce in different distributed environments. There was interest to analyze the reasons for applying the pattern and also differentiating it from similar approaches. This paper first describes the MapReduce pattern and then three different use cases. Each use case applies the pattern in different environments. The two first cases, Hadoop and CouchDB, are production ready open-source solutions for efficient data processing, while the last one, MashReduce, is a more experimental research application for building web mashups.

2 MapReduce pattern

2.1 Context

The amount of data is increasing each year. Data formats vary from text to multimedia files such as pictures and sounds. This data may be part of the process, or it may have been collected as a side product of the process, such as server logs.

2.2 Problem

The data is not presentable as such, but needs to be processed in some way to view it. The view usually requires to transform the data in addition to simply filtering out some of it.

2.3 Forces

– Ad-hoc solutions and spaghetti code in software world work in small programs, but require much maintenance and stress the developers if the programs grow.

– For processing huge amounts of data a delay of days or weeks can be completely acceptable. For small amounts of data the timeframe can be counted in seconds. In any case, there is not enough patience or money to wait for too long.
– Data has a tendency to grow
– Performance can be enhanced in the SW by optimizing the algorithms, and then in the HW by upgrading components such as cpu and memory. Unfortunately, these measures are not always enough, or they can not be taken, so some different approach needs to be taken to reach the desired level of performance.

2.4 Solution

MapReduce is a programming model that works by splitting the data into small items and groups of items that can be processed in isolation, and thus, in parallel. MapReduce is best described in two levels.

First there is the programming model that defines how the data flows trough the map and reduce phases to the output. This idea takes form in the development of the map and reduce functions. They can be very simple, but require some thinking when combined together.

Secondly there is the framework that runs the map and reduce functions. Different frameworks have different charasteristics that make them useful for either big data processing in batches, or realtime processing for smaller amounts of data.

The map and reduce functions are the paramount of MapReduce, but the underlying framework is equally important and has surely taken more effort to develop. It does a lot of work shuffling the data around between different processes and different computers, so that the map and reduce functions do not have to worry about the details of distribution. Each MapReduce framework manages the map and reduce functions in different ways. Some of the different details are explained later in the case studies, but in the following is given a general pattern description.

The first thing to understand is, that the data is always split into items. Items are constituted of a key and a value. In the beginning, before running any functions, the split is done automatically by the framework. For example, if there are many text documents, the split may be such that each document is one item where the name of the document is the key and the content of the document is the value.

The splitted data is then processed either by a map, or a reduce function. The difference is in the granularity, a map function processes each item, while a reduce function processes each group of items. The groups are discriminated by keys. The reduce operation is heavier, as the framework needs to gather all items with the same key together.

Each map and reduce step filters or transforms the data and changes somehow. The data split is also changed at each step and at some intermediate steps there may be many times more data than at the beginning or at the end of the

task. The framework takes care of all this data management in a distributed environment.

Wordcount example

The flow of data from the input documents and trough map and reduce to the output is explained in the following wordcount example:

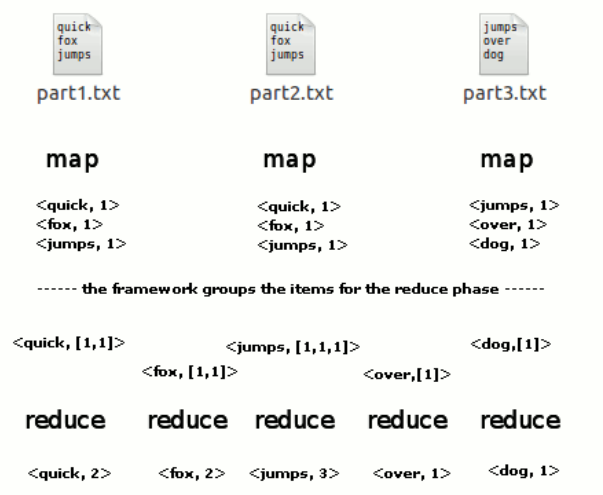

Fig. 1. Wordcount example

In this example, the data is constituted of three text documents that each contain a few words. The data is split in three, naturally by the number of documents, and a map function is run for each split. The map function creates an item for each word it encounters in the document. The key of the item is the word, and the value is 1, meaning that it occured once. The occurences of each word could also be counted together at this point, but this is omitted for simplicity. When all maps have run, the framework sorts the items by key for the reduce functions. Then, one reduce function is started for each set of items with the same key. The reduce function simply sums up the values (1+1+..) to produce the final count of that word in all the documents.

Example functions for the wordcount example

```
map( filename, words ):
    for word in words.split():
        emit( word, 1 )

reduce( word, counts ):
    wordcount = 0
    for c in counts:
        wordcount += c
    emit(word, wordcount)
```

Note that in this, and most other documents, map and reduce may be reffered to as functions, programs, agents, or simply as mappers and reducers. usually these terms simply mean the same thing from slightly different point of views, but sometimes there is a distinction that a mapper or reducer means the computer that does the mapping or reducing.

2.5 Positive Consequences

- Complex data processing can be split into multiple steps
- The framework does a lot of work in the background so that it is easy to develop the business logic in map and reduce functions
- Map and reduce functions may be reused for different views
- Data processing time can theoretically be devided by the number of nodes and nodes can be added and removed as needed

2.6 Negative Consequences

- The framework is a dependency and it may require some work to implement the solution to another framework

2.7 Related Patterns

A simple choice for distributed computing is to use a queue. The queue is filled with tasks that worker computers fetch whenever they have capacity. This kind of solution may be faster to setup and can accomodate any kind of programs. On the other hand as the data and processing programs are loosely coupled, a lot of automated data handling that a MapReduce framework can provide is not available.

2.8 Resulting Context

Big amounts of the data can be viewed in reasonable amounts of time. But if the input data keeps growing, the processing cluster needs to scale also to cope with that. On the other hand, it might be possible to think about what data really is important and if some of it could be discarded.

3 Case 1: Crunching Big Data with Hadoop MapReduce

Hadoop MapReduce is a mature and advanced MapReduce framework. It is the closest open-source equivalent to Googles proprietary MapReduce framework for dealing with really big data.

In Hadoop, the map and reduce functions are run in pairs, first a map and then a reduce function. Either one can also be omitted or replaced with an identity function. To accomplish more complex tasks, the workflow of running multiple mapreduce steps needs to be managed outside of Hadoop. Basically the output of the first task is used as the input for the second task, and so on. There exists many tools managing task chaining automatically.

For the best possible integration with all the Hadoop features, the map and reduce functions should be written in Java, as that is also Hadoops programming language. There is also an alternative, which is called streaming. It means that the map and reduce functions interact with the framework with standard input and output streams. Streaming makes it possible to write the functions in any programming language.

Hadoop can read input from a local or network filesystem, but also its own distributed filesystem, HDFS. HDFS can store multiple copies of very big files on multiple nodes in a cluster. Compression can be used to save some space and time in transferring the files. Hadoop can automatically decompress the files when reading items for the map and reduce functions.

The Reduce step is heavy, as the framework first needs to group together all the values that have the same key into one node. Hadoop can optimize this step by using a special combine function, which is basically the same function as the reduce function, but it runs on each node with those items that are locally available. The combiner runs before the reduce function, acting as a kind of a pre-reduce and makes the reduce step lighter. The combiner, however, can not be used in cases where the reduce function expects to have all the items at it's disposal at the same time.

One of Hadoops strengts is its strong focused ecosystem. There are a lot of additional supporting software and also commercial products that are based on Hadoop. Amazon Web Services, for example, have released an easy to use service called Elastic MapReduce. It allows to provision a cluster of the desired size and start a new MapReduce job on the cluster with a few clicks.

4 Case 2: Viewing Documents with CouchDB

CouchDB is a document database designed for the web. It builds on the RESTful[2] design principles and uses technologies such as JSON and JavaScirpt that are familiar to web developers. Documents are stored in JSON format and single documents can be requested as they are with an ID, but custom views are generated in the CouchDB server with map and reduce functions that are written in JavaScript.

CouchDB implements a highly optimized version of MapReduce. It allows to run a map function and then optionnally a reduce function. The input data is split by documents, so each document gets trough the map function separately. The idea with the map function is to index the documents by some key. The key may contain many values that then pass trough the optional reduce function.

This process is optimized so that after the whole index is generated once, new documents do not require to map the old documents. The optional reduce function is also designed so, that it can "rereduce" new data together with the old one. Indexes are based on the keys, and once generated, the client can ask for the keys in certain order, or a certain range of the keys.

Example of a JSON document:

```
{
 "name":"Joonas",
 # month and amount of payment
 "payments":{"2012-01":10, "2012-02":10}
}
```

With this example document, we can think about a few useful map functions. The map may for example emit the document as it is, but with an added total payments field,which can easily be calculated with JavaScript. Another map could emit items where the key is payment month and the value is the amount of payment in that month. This would be done for all documents and then reduce function could count all the payments together, resulting in an index where the client can quickly look up the total sum of payments for each month.

Document databases are being developed enthusiastically, and one side-effect of that is fragmented ecosystems. There is one important fork of CouchDB, BigCouch, that scales CouchDB to multiple nodes. It provides the same API but the underlying software is a distributed system. This way the map and reduce functions can be run on parallel at different computers to optimize big views. In reality, however, the scaling by adding nodes does not work fully yet and in some cases may even slow down the view generation. There is a plan to merge BigCouch back to CouchDB repository, and hopefully more of that potential will be worked out.

5 Case 3: Image Processing with MashReduce

MashReduce is a project to adapt MapReduce for building web mashups in short time. The rational is, that web data is distributed, and can thus be processed in parallel. If we want to scale up the mashups by using more content, or heavier algorithms, then finally the only alternative is to paralellize the computation.

In MashReduce, the map and reduce functions are usually used to process image data. In practice, the key-value items are such, that the value contains a jpeg-picture. Map functions can for example transform the image somehow or extract some information out of it. Reduce functions can for example create a collage of images in the same location. MashReduce also extends the MapReduce pattern by adding another type of function, called mashup. Whereas a map function is processed for each item and a reduce function is processed for each group of items, the mashup function is processed for all items together.

To run some task, the functions are first organized into a pipeline, which shows clearly all the steps of the task. The pipeline is then distributed to the available worker nodes and fed some input. The framework supports a few popular web content sources, such as Flickr and Picasa, where public albums can be automatically downloaded. The following picture illustrates the creation of a processing pipeline (right) from the available map, reduce and mashup agents (left).

Fig. 2. Pipeline with map, reduce and mashup agents

This particular pipeline can download and process 22 pictures from three content sources using four worker nodes in less than 7 seconds. The result is a HTML-page that shows collages of faces taken in various locations.

One clear advantage of using the MapReduce pattern here is that each map, reduce or mashup agent exists as a self-contained module. They can be later combined in different ways for different application needs. The agents can take parameters, making them even more reusable. New applications can be created

faster, if it is possible to rely on already existing agents and combine them into new pipelines with new agents.

There is, however, some hidden information behind each agent, which limits the way they can be combined. The designer needs to get familiar with the existing agents to understand what kind of input they exept. For example, the GroupByLocation agent expects that the input images contain EXIF headers that tell the location where the picture has been taken.

References

1. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. OSDI p. 13 (2004)
2. Fielding, R.T.: REST: Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine (2000), http://www.ics.uci.edu/ fielding/pubs/dissertation/top.htm