TARTU UNIVERSITY

Faculty of Mathematics and Computer Science

Institute of Computer Science

Computer Science

Karl Potisepp

# Large-scale image processing using MapReduce

M.Sc. Thesis (30 ECTS)

Supervisor: Pelle Jakovits M.Sc., Satish Narayana Srirama Ph.D.

Author: .................................. " ..... "  april 2013

Supervisor: ............................... " ..... "  april 2013

Approved for defence

Professor: .................................. " ..... "  april 2013

Tartu 2013

# Contents

# Chapter 1

# Introduction

Along with the development of information technology, a constant stream of new applications for solving humanity's problems has also appeared. As we possess more computing power, we can tackle more and more resource-intensive problems such as DNA sequencing, seismic imaging and weather simulations. When looking at these subjects, a common theme emerges: all of these involve either analysis or generation of large amounts of data. While personal computers have gone through a staggering increase in power during the last 20 years, and the processing power even within everyday accessories - such as smartphones - is very capable of solving problems that were unfeasible for supercomputers only a couple of decades ago, analysing the amount of data generated by newest generation scientific equipment is still out of reach in some areas. Moreover, as processor architectures are reaching their physical limitations with regard to how small individual logic gates and components can get , using distributed computing technologies has become a popular way to solve problems which do not fit the confines of a single computer. Supercomputers, GRID-based systems and computing clouds are an example of this approach. Since the fields of distributed computing and image processing are too broad to fully cover in this thesis, this work will focus on the latter of the three with regard to image processing.

Due to the increasing prevalence of personal computers, smart televisions, smartphones, tablets and other devices carrying within themselves a full-fledged operating system such as Android, iOS or Windows 8, and due to the capability of these devices to act as producers of many kinds of content instead of being passive receivers (like radio and television, for example), there is a need to be able to process that content: photos need to be resized, cropped and cleaned up, and all the recorded sound and video shaped into a coherent whole through montage. These procedures however may not be

something that is best achieved on the same device that was used for recording, because of limiting factors in processing power and storage space. However with the widespread availability of wireless internet or high-throughput cell phone networks, any of the aforementioned devices can simply upload their data to a more capable computer in order to do necessary processing. Because in many cases the recorded media will be consumed using a different device (for example, viewing holiday photos taken with your smartphone on your computer or smart TV), it can be argued that both the steps of transferring media from the recording device and processing it are inevitable anyway. Facebook and YouTube both provide a good example of this scenario: the user can upload their media in more or less in an unprocessed format and the frameworks take care of resizing and re-encoding the media so that it can be consumed by users. In this thesis I will explore whether using a computing cluster made out of virtual machines running open source software is a solution for doing this.

The rest of the thesis is structured as follows. In the next sections I will describe in more detail the terminology and the problem at hand, give a short overview of previous work in this area. Chapter 2 will focus on describing the MapReduce model and the specifics of it's implementation in Apache Hadoop. In chapter 4 I will look at two different practical use cases for applying MapReduce to large scale image data. The fifth and final chapter provides an overview of the results and outputs of this thesis.

Possible subtopics:

- Microscope photo processing with bilateral filter - Datexim SAS (ENSICAEN)

- de-noising satellite images (maybe)

- Classification and metadata extraction from archaeological excavation data (TO BE ELABORATED)- Graeme Earl and Hembo Pagi (U of Southampton)

## 1.1   Problem statement

Before going too much into specifics, I will first specify the size of data I consider to be large-scale with with regard to this work. This requires some grossly simplified description of the architecture of shared amongst all modern computers. It is common knowledge that a computer consists of a processor, memory and a hard drive. The processor performs calculations on the data stored in memory, which has previously been read from a hard

drive. It is important to note here that since very many computers are also connected to the Internet, the hard drive in question may reside in a different physical location than the processing unit and memory. Now, it is also known that the data transfer speed between the processor and memory is generally orders of magnitude faster than between memory and hard drive. Similarly, reading from a local hard drive is faster than accessing data from storage in a different computer, due to overhead added by having to communicate over a network.

Therefore, as the size of the data to be processed by one algorithm increases so that the computer no longer can hold all the information in memory, there is a significant decrease in processing speed. Similarly, if the data does not fit on the local hard drive, the processing speed drops due to having to wait for it to be sent in from another computer. While this can be alleviated somewhat by using buffering techniques, the general rule remains the same: it's best if the problem fits within memory, worse if it fits on the local hard drive and worst if the data has to be read across the network. Processing a large image is an example of such a problem.

In this case we are dealing with a microscope image with a resolution of 86273 by 81025 pixels (roughly 6.99 gigapixels), where each pixel is made up of 3 values - red, green and blue. Assuming that each of these values is stored as a 32-bit precision floating point number, the total memory consumption of storing this data in an uncompressed way can easily be calculated:

$$86273 * 81025 * 3 * 32 \text{ bits} = 78.12 \text{ gigabytes.}$$

At the time of writing this document, most commodity computers do not have the required memory to even store this amount of data, and certainly not to perform any sort of processing with an overhead dependent on the input size, and even though there do exist specialised computers with enough memory for solving this issue, they are significantly more expensive to acquire and maintain. However, our aim is to find out whether it is possible to process this kind of images using commodity computers in such a way that all the necessary data is stored within memory.

The second case involves a data set of 48469 images totaling 308 GiB (the average image here is a JPEG2000 file around 6.5MiB in size). While the size of the data set is small enough to fit on regular hard drives, and processing the images individually is not a problem, because the average size remains around 13 megapixels, thus requiring roughly only 40 MiB of memory, which is orders of magnitude less than was the case with the large image. In this case, the issue is not so much being able to fit the problem within memory, but rather about being able to process the data quickly enough. Here we

depend on the processor - it does not matter how many more images you can fit inside the memory since generally the processor can only work on one image at a time. In reality this depends on how many cores the processor has and how well the algorithm can take advantage of that, but even with many cores, going through all of the data is too time-consuming. Therefore the problem to solve in this case is how to process this data set in an efficient way.

In this section I have established that processing of large images or large data sets of regular images can not be done on a single personal computer, because in the first case, they do not fit into memory, and in the second case one computer can not process them fast enough. Neither of these issues can be expected to be solved by advances in computing power, because CPUs are already reaching their theoretical physical limitations and the scale of data is increasing faster than the processing capabilities of single commodity computers.

A solution for these problems is turning towards distributed computing, where limitations of a single computer are dealt with by combining the resources of many computers to perform one large task. While this approach is not new - supercomputers and computing clusters have existed for many years already - it is only recently that techniques of using commodity computers for distributed processing have gained popularity. Combining this with the ability to rent virtual machines from cloud providers such as Amazon and Microsoft it is easy to see how solving large scale processing tasks is now feasible for those who do not have access to a supercomputer nor the resources to build or maintain one.

### 1.1.1   Image processing

Lots of images on the internet, youtube, produced by high definition cameras. Realtime processing and feature extraction, denoising. Focus on 2-dimensional color images, subfocus on one large image and a dataset of many small images with regard to MapReduce.

### 1.1.2   Why MapReduce?

Problem statement. What is the problem, why does it need to be solved, why solve it this way?

What are the alternatives? batch processing on the PC is feasible for only small amounts of data; maintaining a purpose-built computer cluster is only cost-effective for very large and important problems; Hadoop is affordable

and easy to use, yet can be used even if the data scales to huge sizes making it ideal for this (however we need to find out whether that is the case)

# Chapter 2

# Background

In this chapter I will describe and summarise relevant work that has been done in the field of distributed image processing, then describe the MapReduce computing model with regard to Apache Hadoop and Hadoop Distributed Filesystem.

## 2.1  Relevant work

The following is a list of previous work that has been done in solving issues with distributed image processing:

- Web-Scale Computer Vision using MapReduce for Multimedia Data Mining [**?**]: A case study of classifying and clustering billions of regular images using MapReduce. No mention is made of average image dimensions or any issues with not being able to process certain images because of memory limitations. However, a way of pre-processing images for use in a sliding-window approach for object recognition is described. Therefore one can assume that in this approach, the size of images is not an issue, because the pre-processing phase cuts everything into a manageable size. The question still remains whether a sliding window approach is capable of recognizing any objects present in the image that do not easily fit into one analysis window, and whether the resource requirements for image classification and image processing are significantly different or not.

- An Architecture for Distributed High Performance Video Processing in the Cloud [**?**]: This article outlines some of the limitations of the MapReduce model when dealing with high-speed video encoding, namely it's dependence on the NameNode as a single point of failure

(however a fix is claimed at [**?**]), and lack of possibility for generalization in order to suit the issue at hand. An alternative - optimized - implementation is proposed for providing a cloud-based IaaS (Infrastructure as a Service) solution. However, considering the advances of distributed computation technology within the past two years (the article was published in 2010) and the fact that the processing of large images was not touched upon, the problem posed in this work still remains.

- Clustering Billions of Images with Large Scale Nearest Neighbor Search [**?**]: Here, a MapReduce-based approach for parallelizing nearest-neighbor clustering is described. The report focuses more on the technicalities of adapting a spill-tree based approach for use on multiple machines. Also, a way for compressing image information into smaller feature vectors is described. With regards to this thesis, again the focus is not so much on processing the images to attain some other result than something intermediate to be used in search and clustering.

- Parallel K-Means Clustering of Remote Sensing Images Based on MapReduce [**?**]: A description of sing the k-means algorithm in conjunction with MapReduce and satellite/aerophoto images in order to find different elements based on their color (i.e. separate trees from buildings). Not much is told about encountering and overcoming the issues of analyzing large images besides mentioning that a non-parallel approach was unable to process images larger than 1000x1000 pixels, and that the use of a MapReduce-based parallel processor required the conversion of TIFF files into a plaintext format.

- Case Study of Scientific Data Processing on a Cloud Using Hadoop [**?**]: This article describes the methods used for processing sequences of microscope images of live cells. The images and data units in question are relatively small - 512x512 16-bit pixels, stored in folders measuring 90MB - there were no issues with regard to having to split data for fitting into Hadoop DFS chunks besides improving upon a couple of Java classes. No mention was made about the algorithm used to extract data from the images besides that it was written in MATLAB, which meant that the processing chain had much more complexity than just a MapReduce job.

- Using Transaction Based Parallel Computing to Solve Image Processing and Computational Physics Problems [**?**]: This article describes the use of distributed computing with two examples - video processing/analysis

and subsurface transport. The main focus is put on the specifications of the technology used (Apache Hadoop, PNNL MeDICI), whereas there is no information presented on how the image processing parts of the examples given were implemented.

- Distributed frameworks and parallel algorithms for processing large-scale geographic data [?]: Here, many issues with processing large sets of geographic information systems (commonly known as GIS) data are described and solved in order to be able to extract useful knowledge from it. This article was published in 2003, so while some of the issues have disappeared due to the increase in computing power available to scientists, problems stemming from the ever-increasing amount of data generated by different types of monitoring technologies (such as ensuring distribution of data to computation nodes and storing big chunks of data in memory) still remain. Also, considering that the Amazon EC2 [?] web service came online just in 2006, it is obvious that one can not make an apt comparison whether or not a MapReduce-based solution in 2012 is better or not for large-scale image processing than what was possible using grid technology in 2003.

- A Scalable Image Processing Framework for gigapixel Mars and other celestial body images [?]: This article describes the way NASA handles processing of celestial images captured by the Mars orbiter and rovers. Clear and concise descriptions are provided for the segmentation of gigapixel images into tiles, how these tiles are processed, and how the image processing framework handles scaling and works with distributed processing. The authors used the Kakadu JPEG2000 encoder and decoder along with the Kakadu Java Native Interface to develop their own processing suite. The software is proprietary and requires the purchase of a license to use.

- Ultra-fast processing of gigapixel Tissue MicroArray images using high performance computing [?]: This article talks about speeding up the analysis of Tissue MicroArray images by substituting human expert analysis for automated processing algorithms. While the images sizes processed were measured in gigapixels, the content of the image (scans of tissue microarrays) was easily segmented and there was no need to focus on being able to analyse all of the image at once. Furthermore, the work was all done on a specially built grid high performance computing platform with shared memory and storage, whereas this thesis is focused on performing processing on a Apache Hadoop cluster.

As evidenced by the summaries listed above, a lot of work has already been done, however the question whether (and how well) Hadoop is suited for large scale image processing tasks still remains, because no known results exist.

## 2.2   MapReduce

MapReduce is a programming model developed by Google for processing and generating large datasets used in practice for many real-world tasks [?]. In this section, I will focus on describing the general philosophy and methodology behind this model, whereas the following part will describe in more detail one of the more popular implementations of the model - Hadoop - which is also used for all the practical applications featured in this work.

The basic idea behind MapReduce is based on the observation that a lot of processing tasks involving terabytes of data all need to deal with the issues of distributing the data across a network of computers to ensure that the available storage and CPU resources are maximally utilised, and it would be easier if programmers could focus on writing the processing part that is actually different per task.

Description of the general philosophy of MapReduce, the logic behind Map/Reduce/etc. parts of the processing chain, and how this approach suits/is cumbersome when dealing with image processing.

### 2.2.1   Apache Hadoop

Hadoop is an open-source framework for distributed computing developed by the Apache Foundation, inspired by Google's MapReduce (TODO cite Hadoop and MapReduce papers). It has been in development since 2005 and - at the time of writing this work - is one of the most popular freely available applications of it's kind. Due to this and also a large user base ranging from home users to large companies makes Hadoop a good candidate for attempting to use it for image processing.

#### Hadoop Distributed File System (HDFS)

HDFS is the underlying file system that Hadoop uses in order to store data meant to be processed by MapReduce. It is inspired by the Google File System (GFS) (TODO cite). There are two reasons why the file system comes into play when discussing image processing using Hadoop. Firstly the distributed nature of HDFS ensures that all input data is spread across

the cluster's storage so that at least initially every worker can process data stored on it's local hard drive. Secondly, the configured block size somewhat restricts the storage of files - if a file is larger than the block size and can not easily be split, then all it's blocks will have to be stored in the same location, which in essence means that the advantages from having a small block size are lost.

TODO what is a MapReduce job?

## 2.3  Image processing

In this thesis I will restrict my focus with regard to the field of image processing to two-dimensional color images. The following will not be a description of how images are acquired through the use of scanning or digital cameras, therefore it is assumed here that the reader is familiar with the notions of pixels, 2-dimensional coordinate notation and representing color using red, green and blue values. Therefore I will start with the following formal definition: I consider the image $I$ with width $x$ and height $y$ as a collection of pixels $p$, such that

$$I = \{\, p_{i,j} \,|\, i \in [1,x],\, j \in [1,y] \,\}, \text{ and}$$
$$p_{i,j} = (r_{i,j}, g_{i,j}, b_{i,j}),$$

where $r_{i,j}$, $g_{i,j}$ and $b_{i,j}$ are respectively the red, green and blue values of the pixel at $x$-coordinate $i$ and $y$-coordinate $j$. From this definition it is more or less straightforward to estimate the minimal memory requirements for storing an image with known dimensions when the programming language and data type for storing individual color values is also chosen. Since this thesis deals with Java, and image processing algorithms tend to prefer floating point values (which are 32 bits in Java) in the interests of precision, we can estimate the memory consumption $M$ of an image with dimensions $x$ and $y$ as follows:

$$M = x * y * 3 * 32 \text{ bits.}$$

As image processing algorithms tend to operate on uncompressed images, using this sort of calculations provides a way of estimating the memory requirements from the time complexity of the processing tasks. Therefore the size of compressed the image file (for example JPEG or PNG) is only very loosely correlated with the time it takes to process that image, as the efficiency of compression algorithms depends on the information content of the image itself, whereas the estimation method described above only takes into account the spatial dimensions of the image.

### 2.3.1 ImageJ

For processing images while working on the practical part of this thesis, I used the ImageJ library. It is a freely available image processing application written in Java and developed by the National Institutes of Health [?]. It supports the processing and analysis of a wide variety of image types (8-, 16-, 32-bit RGB/black & white images) and many commonly used image file formats such as PNG, TIFF and JPEG.

## 2.4 Image processing and MapReduce

It is known that the MapReduce method places some restrictions on iterative parallel processing algorithms that require communication between computing nodes. Since the only step in a MapReduce job that contains any communication takes place when the mappers send data to reducers, all the algorithms whose parallelised implementations require more than one communication phase, require more than one MR job to complete. Due to the overhead involved in starting jobs, this can lead to a huge increase in computation time, particularly for algorithms that require many iterations to complete, and therefore makes MapReduce less than ideal for performing this sort of processing.

Fortunately most image processing tasks involve doing many independent small-scale calculations in parallel. A good example of this is Gaussian blurring (smoothing), which transforms an input image by calculating a new value for every pixel based on it's neighbouring pixels. This procedure can be trivially parallelised since each calculation only requires the values of a small group of pixels in order to calculate the new value of the pixel in focus. In this case, the procedure requires only one step, so there is no need to worry about accommodating to iterations.

Things get a bit more complicated once we attempt to use some sort edge-detection heuristic in order to smooth only those regions in the image which are homogenous in colour, and preserve sharpness in the contours. A common approach here is to iteratively smooth the image, re-calculating the heuristic at every step.

TODO explain parallelisation by way of splitting the image into blocks, how to figure out the best block size analytically (take into account processing time, HDFS block size), overlap size etc.

Description of different ways the MapReduce method has been used in conjunction with image processing libraries. Also some use cases.

# Chapter 3

# Details of methodology

Descriptions on what was implemented, how it was implemented, problems encountered, and what has been done to solve these problems. If there were unsolvable problems, then how do these limit the applicability of the methodology in the real world and how to work around these limitations.

# Chapter 4

# Use case

Description of how the methodology established in this thesis was applied to real-world problems.

## 4.1 Analysis and processing of a large image

In this section I will describe a practical scenario of performing distributed processing on a large-resolution image.

### 4.1.1 Description of the data and use case

What sort of image, why does it need processing?

### 4.1.2 Bilateral Filter

In this section I will focus on describing the general principle behind the bilateral filter and some of it's uses. In order to somehow gauge the feasibility of using Hadoop MapReduce as a platform for image processing, I used the implementation of a bilateral filter by Chaudhury et al. (TODO cite).

#### Naive bilateral filter algorithm

Description of the bilateral filter algorithm, it's applications and details about the implementation used for benchmarking.

### 4.1.3 Performance

One of the main issues with regard to measuring the performance of the FBF algorithm was comparing the results of the sequential version to the

distributed MapReduce one.

## 4.2 Analysis of many small images

### 4.2.1 Description of the data and possible use case

The data set consists of 40326 JPEG encoded images (totalling at about 308GB) taken across the span of 9 years at the Portus archaeological excavation site near Rome, Italy. It is somewhat structured and some images are also tagged with meta-data using folder structures. However the structuring is incomplete, evidenced by duplicates and missing meta-data (which currently has to be manually added). The motivation behind applying distributed image processing on this data set is to find out whether it is possible to speed up the structuring process and extract some meta-data with minimal human interference.

The downside of using this data set to illustrate the analysis of many small images using MapReduce is that since there is no ideally structured version of it to compare to, all efficiency statistics will be purely subjective. However the intent of this work is not to describe an effective meta-data extraction method, but rather analyse the suitability of MapReduce for this sort of task and discuss the issues (both solved and unsolved) encountered.

### 4.2.2 Problem statement

Extracting meaningful data from images is a non-trivial task. OCR, object recognition, identifying similar images, measuring image quality.

Problem - how to feed images into MapReduce when the folder structure is complex? Hadoop does not provide a way for recursively going through the input folder and starts a new map task for every folder which creates unnecessary overhead. Creating a large SequenceFile to store all file data along with original full paths is not an option because it is too resource-intensive and the resulting file may be difficult to store. Flattening the structure and writing original paths into image meta-data also requires too much processing (reading and writing all files in the data set). Ignoring the path info altogether is not an option, because it provides a structuring/grouping of the data not encoded into meta-data.

Problem - since images come from many different makes and models of cameras, and there is no easy way in distinguishing between meta-data that is useful in the context of this task and meta-data that encodes some cryptic information that is only used by some proprietary software (for example

serial numbers and firmware version numbers) or simply the parameters of the camera at the time of taking the photo (such as Base ISO, Auto Focus settings etc.). While some of the latter meta-data may be useful when searching for similar images (for instance, one can group together photos taken with flash), there is no way of specifying which meta-data key refers to the correct value without first looking through a representative sampling (or even all) of the images and manually filtering out the meta-data keys to group together. (TODO maybe explain this better)

TODO explain hadoop small file problem http://amilaparanawithana.blogspot.com/2012/06/small-file-problem-in-hadoop.html http://blog.cloudera.com/blog/2009/02/the-small-files-problem/

## 4.2.3   Performance

# Chapter 5

# Overview and discussion

This thesis does blabla and provides blabla with regard to MapReduce.