Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

KATEDRA INFORMATYKI STOSOWANEJ

PRACA MAGISTERSKA

KONRAD MALAWSKI

PRZETWARZANIE I ANALIZA DANYCH MULTIMEDIALNYCH W ŚRODOWISKU ROZPROSZONYM

PROMOTOR:
dr inż. Sebastian Ernst

Kraków 2014

OŚWIADCZENIE AUTORA PRACY

OŚWIADCZAM, ŚWIADOMY ODPOWIEDZIALNOŚCI KARNEJ ZA POŚWIADCZENIE NIEPRAWDY, ŻE NINIEJSZĄ PRACĘ DYPLOMOWĄ WYKONAŁEM OSOBIŚCIE I SAMODZIELNIE, I NIE KORZYSTAŁEM ZE ŹRÓDEŁ INNYCH NIŻ WYMIENIONE W PRACY.

. . . . . .

PODPIS

AGH
University of Science and Technology in Krakow

Faculty of Electrical Engineering, Automatics, Computer Science and Electronics

DEPARTMENT OF APPLIED COMPUTER SCIENCE

MASTER OF SCIENCE THESIS

KONRAD MALAWSKI

PROCESSING AND ANALISYS OF MULTIMEDIA IN DISTRIBUTED SYSTEMS

SUPERVISOR:
Sebastian Ernst Ph.D

Krakow 2014

Acknowledgements...

Contents

Todo list

1. Introduction

1.1. Goals of this work

The primary goal of this work is to research how to efficiently work with humongous amounts of multimedia data in a distributed setting, and weather this approach is the tight one.

In order to guarantee that recommendations and measurements made during this research are applicable in the „real world", outside of laboratory or „experiment" environments, I have defined a series of problems (described in the next section) and implemented a system which is able to solve those problems as well as easily adapt to any new requirements benefiting from the use of parallel access to hundreds of gigabytes of reference video material.

1.2. Targeted use cases

As stated in the introduction section, in order to be able reliably verify criteria such as responsiveness, cost-efficiency, performance and of course scalability of distributed systems, there must be some reference problem and solution the measurements will be made on.

For the sake of this paper I propose a „video material analysis platform" from here on referred to as the „*Oculus*" system. In the following two sub sections I will explain the use cases (thus - requirements) this system will aim to solve, which while being a very interesting topic of research by itself already, will provide me a platform to measure the usefulness of the selected distributed system building blocks used for it.

1.2.1. Video similarity calculation

One of the simplest use cases in which this system might be used is trivial *near–duplicate detection* of video files. One might think that a simple checksum of the video files would suffice, but imagine a system like *YouTube* **??** where many hours of video content are uploaded *each second*. It is easy to imagine two (or more) people uploading the same trailer for an upcoming movie during the same day. In the easiest case, a simple checksum check would suffice, but in practice the uploaded materials may be in different resolutions or qualities. Such system

must also account for intended malice in uploaded materials - such as a practice of „mirroring" the video material, or slightly brightening every frame. These modifications are often seen on content on services such as YouTube, Dailymotion and others.

The goal of our system is to reliably determine similarity between movies, as well as subsections of movies, Obviously the users will try to trick our algorithm into not recognising that the uploaded content is the same as our reference material. One of the tricks often seen on YouTube.com is that the users upload the material „mirrored" which further complicates our matching algorithm – we must now also be resilient against data modified especially for the goal of us not being able to detect it.

Another, less copyright focused, goal of the presented system is to be able to extensively mine data from the uploaded video content. Here the canonical example would be a „*TOP 10 Movies of All Time*" video, which obviously contains video material from at least 10 movies, usualy in the order of 10th, 9th ... until the 1st (best) movie of all time. If we would be able to match parts of each video to their corresponding reference materials, we would be able to get meta data about the now recognised movies and even mine out the data what is the best / worst movie of all time, even without it being written per se - only by looking at the frames in the video. This idea only scratches the surface of what the system implemented during this thesis work might do, but it will be our test case that we'll be working towards during this paper.

The system implemented during this work represents a basic effort to tackle the above problems as well as these goa

1.3. Paper structure

In section 1 I will describe the architecture of the system, and briefly go over how this architecture benefits future extension.

In section 2 I will focus in the technical challenges encountered and solved during implementing the reference system.

In the last section I will summarise the findings.

This is not done until I'm done with the paper ;-)Oculus

2. System and design overview

The system, from here to be referred to by the name „*Oculus*", is designed with an asynchronous as well as distributed aproach in mind. In order to achieve high asynchronousity between obtaining new reference data, and running jobs such as „compare video1 with the reference database", the system was split into two primary components:

– **loader** – which is responsible for obtaining more and more reference material. It persists and initially processes the videos, as well as any related metadata. In a real system this reference data would be provided by partnering content providers, yet for this

– and the **job runner** – which is responsible for running computations on top of the Hadoop cluster and reference databases.

In this chapter I will discuss the high-level design of these subsystems, and how they interact with each other. In the next chapter I will describe each of the systems technical challanges and how they were solved.

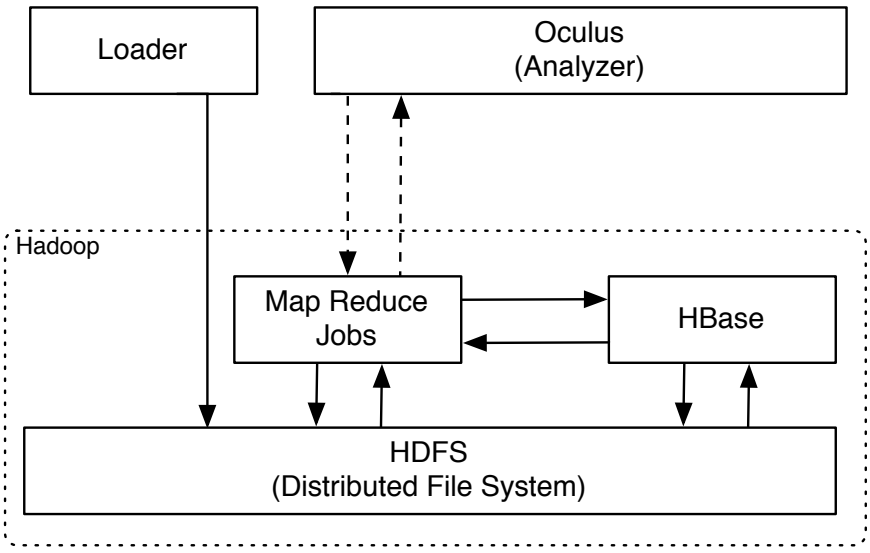

Figure 2.1: High level overview of the system's architecture

2.1. Loader

The downloader is responsible for obtaining as much as possible „reference data”, by which I mean video material – from sites such as *youtube.com* or video hosting sites.

It should be also noted that while I refer to this system as „the” downloader, in fact it is composed out of many instances of the same application, sharing the common workload of downloading and converting the input data. The deployment diagram on drawing [12] explains the interactions between the nodes.

2.1.1. Work sharing in an actor-based cluster

The Loader system is implemented following the style of „Actor model” style of concurrency **??**.

„message pasing” style, which means that the work requests to the cluster come in as *messages*, and then are *routed* to an *actor*, who performs the work, and then responds with another message – in other words, all comunication between Actor instances is performed via messages and there is no shared state between them. This also applies to Actor instances residing in the same JVM - for the user of an „Actor System”, the location (on which phisical node the actor is actually executing) of an actor remains fully transparent - this has huge gains in terms of load balancing the message execution on the cluster - the router decides who will be notified on which message, the listing 2.1 shows a typical workflow using the so-called „smallest inbox” routing strategy **??**.

Listing 2.1: smallest-inbox routing algorithm

```
                                              ---> [inbox1, size = 3]
                                                 =>
                                                  YoutubeDownloadActor(1)
YoutubeCrawlerActor -- Msg(url=http://...) --> router ---/
                                             \    [inbox2, size = 5]
                                                =>
                                                YoutubeDownloadActor(2)


                          /* because inbox1.size < inbox2.size */
```

The underlying Actor System is implemented by a project called Akka **??**, and can be easiest explained as „porting Erlang concepts of the Actor Model to the JVM”. I selected the „Smallest Inbox” routing strategy instead of the other widely used „Round Robin” approach in order to guarantee not overloading any Actor with too many requests to download movies (which is a relatively slow process). Thanks to the smallest inbox routing, I can guarantee that if some of

the nodes have a faster connection to the Internet, they will get more movie download requests, than nodes located on a slower network.

As mentioned before, the system is fully distributed and *any node can perform any task* submited to the cluster. For example let's take the first step in the processing pipeline in Oculus, which is `Download video from http://www.youtube.com/watch?v=-X9bcrJ3TjY` – such message will be emited by the `YoutubeCrawlerActor` and sent via a `Router` instance to an `YoutubeDownloadActor` which has the „smallest inbox".

2.2. Job Runner

The job runners responsibility is at the core of the systems... finish me

2.2.1. Tackling the „small-files problem" on HDFS

In this section I will explain the „small-files problem" which would lead to major performance degradation of the Hadoop cluster is left undressed, and how I solved it in the reference system – Oculus.

The so called „*small files problem*" is a known problem in the Hadoop world relating to how HDFS stores files. The problem arises whenever we try to store files smaller than the block size used by HDFS to allocate files. One can think of it as wasting space – because Hadoop allocates at least one block per file we store in it, and all operations (such as migrating data between nodes, reading files into memory etc.) are optimised towards larger block-sizes (usually around 32MB or 64MB).

To illustrate the problem with an example: if we try storing a 2MB file in HDFS we'll in fact produce 62MB of „wasted space", reducing the cluster's capacity way faster than we ought to. Another p

3. Performance and scalability analysis

In this section I will analyse the devised system on such aspects as general performance, ability (and ease) of scaling the system to support larger amounts of data or to speed-up the processing times by adding more servers to the cluster.

These measurements will be made in the previously described system and backed by measurements

3.1. Scaling Hadoop

In this section I will investigate the impact of scaling the Hadoop cluster vertically (by adding more nodes) on processing times of movies.

The task used to measure will be the longest pipeline available in the oculus system - comparing a movie, percentage wise with all other movies in the database. This process touches a tremendous amount of data, and is also very parallelizable.

At first I measured the time to process the input movie ....

3.2. Scaling the Loader (actor system)

4. Results and processing times

5. Conclusions

The applied technologies have indeed been very helpful, and proved to be very elastic for different kinds of jobs related to processing large amounts of data. I was also positively surprised with the ease of Scaling Hadoop infrastructure.

A. Preparing the cluster

Hadoop's filesystem must be formated before put into use. This is achieved by issuing the `-format` command to the namenode:

```
kmalawski@oculus-master > hadoop namenode -format
```

It is worth pointing out that a "format" takes place only on the namenode, it does not actually touch the datab stored on the datanodes, but instead it deleted the data stored on the namenode. The Namenode, as explained previously, stores all metadata about where a file is located, thus, cleaning it's data makes the files stores in HDFS un-usable, since we don't know "where a file's chunks are stored".

Bibliography