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 2013

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 2013

Acknowledgements...

Contents

Todo list

1. Problem definition

The primary goal of this work is to research how to efficiently work with large amounts of multimedia data in distributed systems. In order to guarantee the devised aproach would make sense

1.0.1. Use cases

One of the simplest use cases in which this system might be used is trivial *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 hours of video content are uploaded each second. It's easy to imagine two people uploading the same trailer for an upcoming movie during the same day. In the easiest case, a simple checksum check would suffice, but what if the videos where encoded using different codecs or even were uploaded in different resolutions?

To further represent how difficult it may be to answer the question of „Are those two files the same material?" let's go back to the example of users uploading video content to our YouTube-like service. What if the users knew the uploaded material is copy righted and should not be uploaded to the public internet? As one might imagine, this might happen quite often on a service like YouTube. Let's assume our users know about our „duplicate" detection algorithm and that we might be partnering with movie companies, so that we have reference material on the servers we can compare the uploaded videos to. 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 resiliant aganist data modified especialy 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

1.1. Architecture overview

In this chapter I will present a broad overview on the system's design and various components. It should also provide some background to why a distributed file system was required in order to enable this system, and why only a distributed system is suited to handle the kind of jobs that Oculus is designed for.

1.2. HDFS - Distributed File System

TODO explain why HDFS is the reasonable choice here

1.3. Akka - Distributed Actor System

TODO why do we need actors here? why does it make sense to distribute the work

2. System design

The system, from here to be refered 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:

- **downloader** – which is responsible for obtaining more and more reference material. It persists and initially processes the videos, as well as any related metadata,
- 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.

2.1. Downloader

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 downloader system is implemented in an „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 responsility is at the core of the systems

3. Results and processing times

4. Conclusions

Overall it seems that

Bibliography