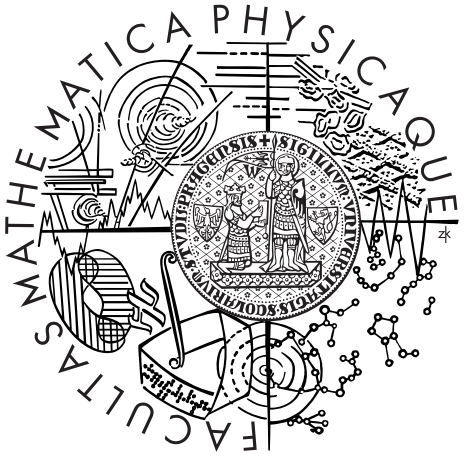
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Ondřej Klejch

Development of a cloud platform for automatic speech recognition

Institute of Formal and Applied Linguistics

Supervisor of the master thesis: Mgr. Ing. Filip Jurčíček Ph.D.
Study programme: Informatics
Specialization: Theoretical Computer Science

Prague 2015

Dedication.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In ..... date ..... signature of the author

Název práce: Development of a cloud platform for automatic speech recognition

Autor: Ondřej Klejch

Katedra: Ústav formální a aplikované lingvistiky

Vedoucí diplomové práce: Mgr. Ing. Filip Jurčíček Ph.D., Ústav formální a aplikované lingvistiky

Abstrakt:

Klíčová slova:

Title: Development of a cloud platform for automatic speech recognition

Author: Ondřej Klejch

Department: Institute of Formal and Applied Linguistics

Supervisor: Mgr. Ing. Filip Jurčíček Ph.D., Institute of Formal and Applied Linguistics

Abstract: This thesis presents a cloud platform for automatic speech recognition, CloudASR, built on top of Kaldi speech recognition toolkit. The platform supports both batch and online speech recognition mode and it has an annotation interface for transcription of the submitted recordings. The key features of the platform are scalability, customizability and easy deployment. The benchmarks of the platform show that the platform achieves comparable performance with Google Speech API in terms of latency and it can achieve better accuracy on limitted domains. Furthermore, the benchmarks shows that the platform is able to handle more than 1000 parallel requests given enough computational resources.

Keywords: cloud, Kaldi, ASR

Contents

Introduction

The most natural form of human communication is speech. In order to be able to talk with a computer, it is crucial to have a good Automatic Speech Recognition (ASR) system. On one hand, there are several open-source ASR toolkits, however deployment of such toolkits requires substantial knowledge therefore for common software developers it is not easy to use them. On the other hand, there are a few webservices that provide ASR as a service, yet these webservices do not solve all problems - either they are paid, closed-source or they are not customizable. So **the first goal of the thesis is to develop a cloud platform for ASR** that is easy to use both from user's and maintainer's point of view.

Although accuracy of ASR systems is improving, these systems are still far from perfect. One of the reasons is that accuracy of ASR systems relies heavily on the amount of the training data and there is not enough publicly available transcribed speech data. By providing free ASR webservice it is possible to collect vast amount of recordings that can be manually transcribed and used later on for further research. Consequently, **the second goal of the thesis is to create an annotation interface** so that recordings obtained by CloudASR platform can be annotated and given back to the community.

In the following text there will be described development and deployment of CloudASR platform and of its annotation interface. Chapter 1 introduces Automatic Speech Recognition theory and tools related to CloudASR. In Chapter 2 tools used for CloudASR development and deployment are presented. Implementation of CloudASR platform is described in Chapter 3. Chapter 4 contains results of conducted benchmarks. Finally, Chapter 5 concludes this thesis.

1. Automatic Speech Recognition

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.1 Acoustic Models

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.2 Language Models

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in

of the original language. There is no need for special content, but the length of words should match the language.

1.3 Decoding

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.3.1 Batch Decoding

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.3.2 Online Decoding

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.4 Transcriptions

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.5 Transcriptions via Crowdsourcing

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.5.1 Amazon Mechanical Turk

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.5.2 CrowdFlower

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.6 Open-Source ASR tools

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.6.1 HTK

HTK [18] Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get

no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.6.2 RWTH

RWTH [16] Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.6.3 Sphinx

Sphinx [12] Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.6.4 Kaldi

Kaldi [15] Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.6.5 PyKaldi

PyKaldi [14] Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the

alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.7 ASR cloud services

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.7.1 Google Speech API

[1] Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.7.2 Nuance

[2] Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.7.3 Wit.ai

[3] Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the

[1] `https://www.google.com/intl/en/chrome/demos/speech.html`
[2] `http://www.nuance.com/for-developers/dragon/index.htm`
[3] `https://wit.ai/`

alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

2. Used Technologies

In this chapter technologies that were used during development will be described. Also the motivation for the usage of these technologies will be explained.

2.1 Platform

In the following section technologies that were used to build a cloud platform will be described. These technologies made possible to build a scalable solution with an easy deployment.

Traditionally, a deployment of a such a complex system as CloudASR is consists of several steps during which necessary dependencies are installed, the application environment is set up and finally the application is started. But this approach makes the maintenance of these systems difficult, because the deployment is time consuming, error-prone and it is not replicable. The ultimate goal for the CloudASR deployment was the exact opposite: fast and replicable deployment.

The most important tool used during development was **Docker** [13] – a portable, lightweight application runtime and packaging tool. It allows to specify dependencies and environmental variables for a process and it allows to build a image from this specification called Dockerfile, see Figure 2.1 for an example. Once this image is built it can be used on any machine with Docker installed, which makes the deployment fast and replicable, because it is not neccessary to install all dependencies on every machine. Additionally, the usage of Docker images removes bugs caused by different versions of libraries used in development and production environmnent because developers use the same images in both environments.

```
FROM ubuntu
MAINTAINER Ondrej Klejch

RUN sudo apt-get update && sudo apt-get install python
ADD . /opt/app
WORKDIR /opt/app

CMD python run.py
```

Figure 2.1: *An example of Dockerfile that creates an image from the base ubuntu image, installs python, copies all files in the Dockerfile folder and sets command python run.py to be run, when the docker image is started.*

When running an application in a cloud it is neccessary to monitor all servers and handle failovers. But with the increasing number of servers the maintenance costs grows rapidly. Therefore it is not possible to do that manually. The tool that allows CloudASR to run on many servers is **Mesos** [7]. It lets users program against set of machines like it is a single machine, which means that it is possible to run and scale an application on a set of servers in a similar way as on a single

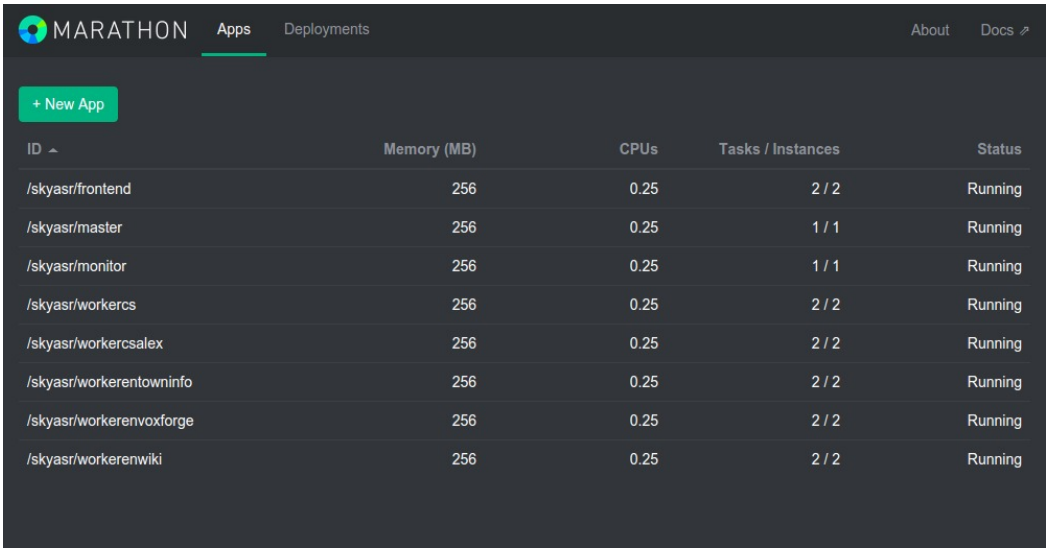


Figure 2.2: *A screenshot of a Marathon web interface with a running CloudASR platform.*

machine. Mesos takes care about scheduling and high availability of the platform. Thus, whenever some part of the CloudASR crashes Mesos will try to fix that. Finally, Mesos supports Docker so the images that are used in development can be also used on a Mesos cluster.

Marathon[1] is a framework built on top of Mesos whose main responsibility is to launch long running applications. It is an entrypoint for running and scaling the applications running on a Mesos cluster. It has a web user interface (see Figure 2.2) and REST API, through which applications can be started, scaled or stopped easily.

Since the traffic of CloudASR platform can be very large, it is not possible to process all HTTP requests by one application server. Therefore, there must be a load-balancer to distribute workload between application servers. CloudASR platform uses **HAProxy**[2] load-balancer, but any other load-balancers can also used with appropriate setup.

2.2 Continuos Integration & Delivery

During development of CloudASR several practises were obeyed, namely Continuous Integration and Continuous Delivery. In order to be able to do that a platform which consisted of **Jenkins-CI**[3] and **Docker Registry**[4] had to be deployed.

The most important tool for Continous Integration & Delivery of CloudASR is Jenkins-CI. Its task is to watch CloudASR git repository and whenever a new code is pushed into this repository it schedules a new build of the platform. During this build the most recent code is pulled from the repository and then the docker images are built. After that tests are run to check that the new code did

[1] `https://mesosphere.github.io/marathon/`
[2] `http://www.haproxy.org/`
[3] `https://jenkins-ci.org/`
[4] `https://github.com/docker/docker-registry`

not break anything. Finally, successfully built images are tagged with current build number and pushed to the Docker Registry.

Docker Registry is a repository of Docker images. Even though, there are several Docker Registry providers[5], which are free for open-source projects, CloudASR uses its own free Docker Registry, in order to be also able to use proprietary software that cannot be shared with public.

2.3 Backend

The main programming language used for development is **Python**[6]. Web and REST API are built on top of **Flask**[7] microframework and they use **Gunicorn**[8] for production deployment. As a database **MySQL**[9] is used, but any other SQL database can be used instead, because **SQLAlchemy**[10] is used to access the database.

CloudASR architecture consists of several nodes which need to communicate between each other. For this communication ClousASR uses **ZeroMQ**[11], because of its simple design, high performance and support for every modern language. With ZeroMQ it is possible to create many messaging patterns, but CloudASR uses only two: request-reply and push-pull. In request-reply messaging pattern a sender sends a message and then waits for a reply, after that another sender can send next message and wait for a reply. Whereas in push-pull messaging pattern senders just send messages and do not wait for a reply.

In order to be able to send complex messages via ZeroMQ sockets, messages have to be serialized. CloudASR uses **Google Protocol Buffers**[12], because they have support in many languages, allow specification of various message types (See Figure 2.3 for example) and serialize messages in very compact way (See Table 2.1 for a comparison of different serializations).

raw file size	56146	
bytes_protobuf	56118	0.999x
base64	74872	1.333x
json_array	158590	2.824x

Table 2.1: *The table shows comparison of different serialization used to serialize a wave file into a message for the CloudASR online mode. As can be seen from the results Google Protocol Buffers achieved the best result.*

As an ASR toolkit CloudASR uses Python wrapper for the **Kaldi speech recognition toolkit** [15] called **Pykaldi** [14]. Because CloudASR should be able to process very long recordings recordings, possibly infinite, with a limited computational resources it is necessary to split the recordings into smaller chunks.

[5] `https://hub.docker.com/`, `https://quay.io/`
[6] `https://www.python.org/`
[7] `http://flask.pocoo.org/`
[8] `http://gunicorn.org/`
[9] `https://www.mysql.com/`
[10] `http://www.sqlalchemy.org/`
[11] `http://zeromq.org/`
[12] `https://developers.google.com/protocol-buffers/`

```
message HeartbeatMessage {
  required string address = 1;
  required string model = 2;
  required Status status = 3;

  enum Status {
    STARTED = 0;
    WAITING = 1;
    WORKING = 2;
    FINISHED = 3;
  };
}
```

Figure 2.3: *An example of Google Protocol Buffer message specification with three fields. Fields address and model are just strings and the status is an enum with four possible values.*

For that purpose CloudASR uses voice activity detector implemented in **Theano** [1] to detect silences in a speech.

2.4 Frontend

Frontend uses several well-known open-source libraries, namely, **Twitter Bootstrap**[13] for CSS styling of the web, **jQuery**[14] and **Angular.js**[15] for interactive elements on the web.

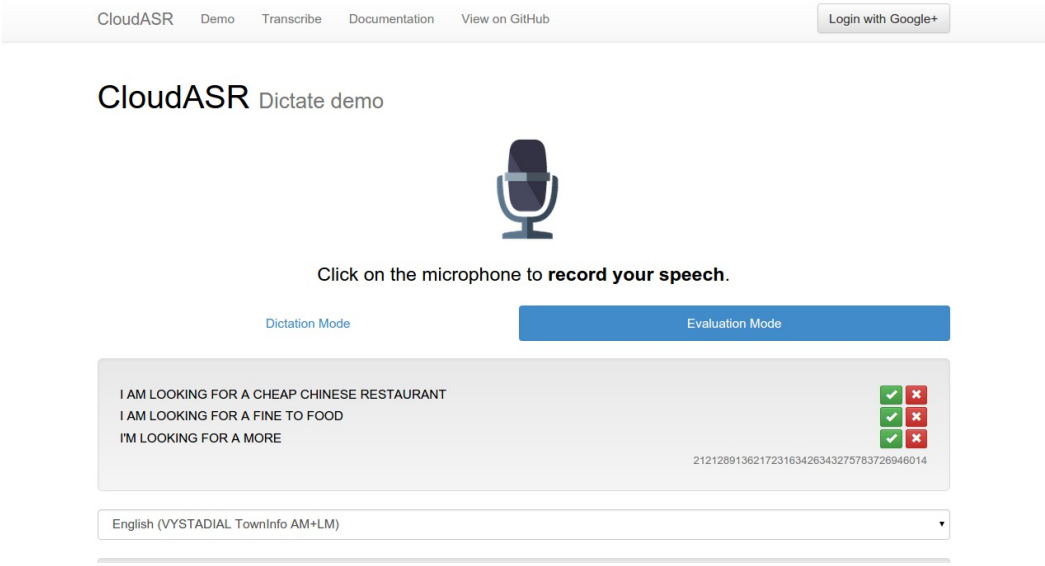


Figure 2.4: *Screen of the Web Demo*

Modern web browsers supports **WebAudio API**[16], which is a high-level

[13] http://getbootstrap.com/2.3.2/
[14] https://jquery.com/
[15] https://angularjs.org/
[16] http://webaudio.github.io/web-audio-api/

JavaScript API for processing and synthesizing audio in web applications. One of the things that can be done with this API is recording of an audio. Thus, it is possible to create a web demo for the CloudASR online mode. The demo is based on **Recorder.js**[17] library, which can record output of WebAudio API and return it as a PCM chunks.

Next step is to send these chunks to the API. Because the demo demonstrates the online speech recognition mode, it is not possible to wait for whole recording to be recorded and then send it to the API via HTTP POST request, thus, CloudASR uses **Socket.IO**[18] to send stream of chunks to the API and to receieve stream of results from the API.

[17] `https://github.com/mattdiamond/Recorderjs`
[18] `http://socket.io/`

3. Solution

In this chapter the CloudASR implementation will be described. The implementation was affected by three key requirements:

- **Scalability** - because the speech recognition is a demanding process in terms of computational resources, it is not possible to handle many parallel requests on one machine. Therefore the CloudASR architecture is designed to be able to scale across many machines.
- **Customizability** - there are already several webservices that provide an API for speech recognition, but they are not easily customizable. Thus, the second requirement was to be able to host any Kaldi model on the CloudASR platform. Moreover, the CloudASR platform is able to run any ASR system, if the users implement a wrapper for that system.
- **Easy deployment** - complex systems as CloudASR have many dependencies and a difficult deployment process, which makes their maintenance hard. CloudASR is designed to has as few dependencies as possible and an one command deployment.

IDEA: Mention somehow Batch/Online recognition mode

3.1 Architecture

In order to meet the aforementioned scalability requirement the platform had to be designed from the very beginning to be able to run on many machines. To be able to do that the architecture consits of several nodes that communicate with each other by sending messages over ZeroMQ sockets (each node acts as an actor as described in [6]). Another problem is that it is not possible start to an ASR system when the user sends a request, because the ASR systems need some time load decoding graphs in memory and this would add some unnecessary latency. Therefore the platform uses Master-Worker architecture to be able to handle many parallel requests for various languages.

The CloudASR architecture as described in Figure 3.1 consists of several types of nodes that can run on different machines. These nodes are **Master**, **Worker**, **API**, **Web**, and **Recordings Saver**. In the following section each node will be described in detail.

3.1.1 Master

The main task of Master is to keep track about running workers and scheduling tasks to them. In order to be able to handle requests for various languages Master has a queue of waiting workers for each language. Running workers send heartbeats, small messages with information about their state, to Master and when the Api asks for a worker Master can return address of an available worker. This makes it possible to dynamically add/remove new workers without need to restart the CloudASR platform.

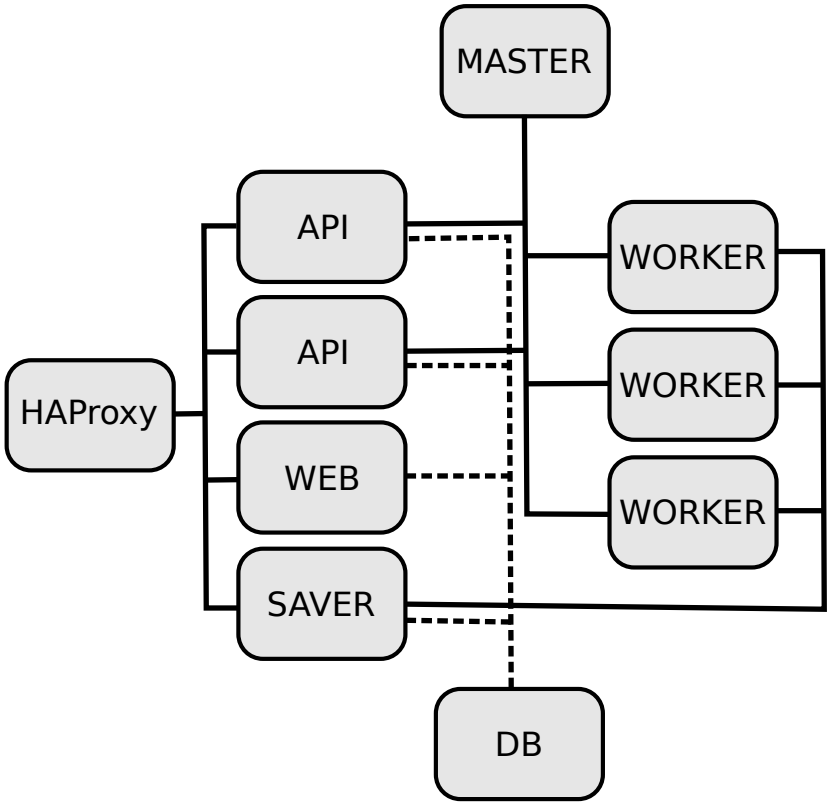


Figure 3.1: *An overview of the CloudASR architecture. The most important node is* ***Master*** *to which* ***Workers*** *sends heartbeats with their state. The client requests are handled by* ***API*** *which communicates with Master and Workers. The processed recordings are sent to* ***Saver*** *which saves them and serves them via HTTP. An annotation interface and an online demo are hosted by* ***Web****. Finally, there are also two externals nodes* ***HAProxy*** *which load-balances requests between particular application instances and* ***DB*** *which stores information about processed recordings.*

The workers can be in four different states: **started**, **waiting**, **working** and **not responding** and they send four different heartbeats: **started**, **waiting**, **working** and **finished**.

The lifecycle of the worker as described in Figure 3.2 starts in in the **started** state, after that it moves to the **waiting** state by sending the **waiting** heartbeat. The worker remains in the waiting state until **Master assings a tasks** to it, then it moves to the **working** state where it remains as long as it is working. In the working state worker sends working heartbeats periodically, to inform Master that it is working and it did not fail. At the end of the task the Worker sends **finished** heartbeat and Master changes the state of the Worker to the **waiting** state.

Additionally, when a worker crashes during the processing of the task and it gets restarted, it sends started heartbeat again, which informs Master, that the worker was restarted and it adds it to the queue again. When a worker does not send any heartbeat for 10 seconds, the master set the worker state to **not responding**. But as soon as the worker sends any heartbeat, the master will set the worker to the appropriate state.

Unfortunately, Master is a single point of failure of the CloudASR platform. When Master stops working no speech recognition requests can be processed,

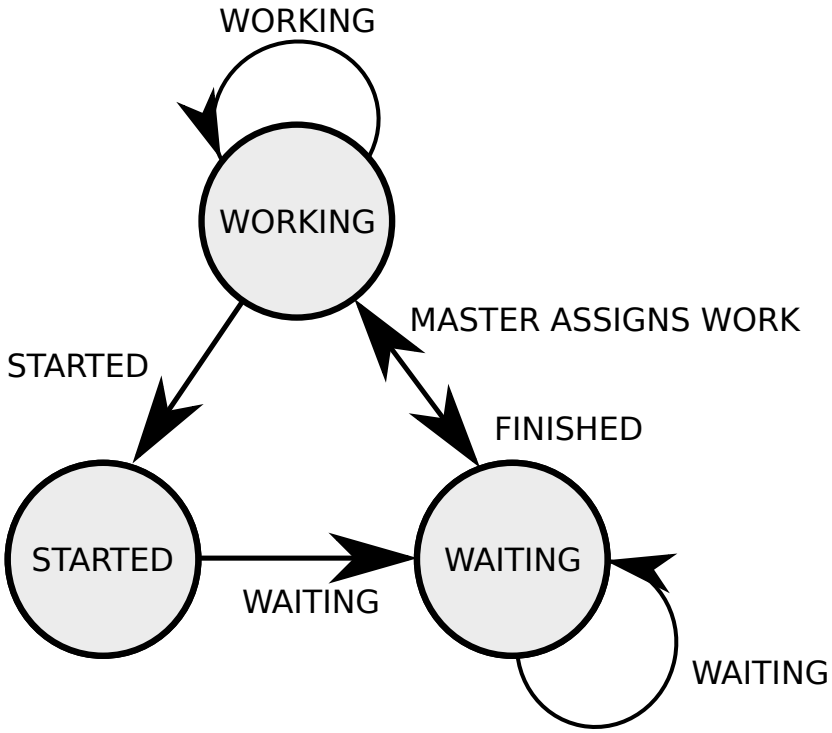


Figure 3.2: *The lifecycle of the worker starts in in the **started** state, after that it moves to the **waiting** state by sending the **waiting** heartbeat. The worker remains in the waiting state until **Master assings a tasks** to it, then it moves to the **working** state where it remains as long as it is working. In the working state worker sends working heartbeats periodically, to inform Master that it is working and it did not fail. At the end of the task the Worker sends **finished** heartbeat and Master changes the state of the Worker to the **waiting** state.*

because the API containers will not know to which worker they can forward the request. But as soon as Master starts working the platform should be available again.

3.1.2 Worker

Worker is a wrapper for an ASR system. By default Pykaldi is used but any other ASR system can be used if the users implement a python wrapper for that system. Because CloudASR should be able to process very long recordings in the online mode, it is neccessary to split the recordings into smaller chunks that can be processed with limited computational resources. For that purpose VAD component from Alex Statistical Dialogue Systems Framework [10] is used to detect silence in a speech, because the speech can be splitted at that point without any large negative effect on the accuracy of the transcriptions.

IDEA: At the end of the recognition Worker sends the whole recording together with respective n-best list to Saver. Heartbeats that Worker sends to Master are small messages with Worker address, language it processes and its state.

Deployment of New Kaldi Worker

One of the key requirements for CloudASR was customizability, primarily customizability of the workers decoding graphs. Therefore, CloudASR supports an easy way for users to add their own Kaldi models. The only thing that the users have to do is to create a worker docker image with their models.

The creation of a worker docker image consists of several steps. First, users have to create a script `download_models.sh` which will download all necessary files from their server, see Figure 3.3 for example. Second, they have to create a configuration file `config.py` with appropriate configuration for the downloaded models, see Figure 3.4. Finally, they have to copy a Dockerfile (see Figure 3.5for example) for the worker and build the docker image with the appropriate command. After that users can use the new worker in their application in the similar way as they use other models.

It is important to note that the new worker docker image will be available only on the machine where it was built. If the users want to use this model on multiple machines, they have to push to their docker registry or alternatively they can update Jenkins scripts `build_workers.sh` and `push_workers.sh` to do that for them.

```
#!/bin/bash

DOMAIN=vystadial.ms.mff.cuni.cz
PATH=/download/alex/applications/PublicTransportInfoCS/hclg/models/
URL=https://$DOMAIN/$PATH

mkdir /opt/models
wget -O /opt/models/mfcc.conf $URL/mfcc.conf
wget -O /opt/models/tri2b_bmmi.mdl $URL/url/tri2b_bmmi.mdl
wget -O /opt/models/tri2b_bmmi.mat $URL/tri2b_bmmi.mat
wget -O /opt/models/HCLG_tri2b_bmmi.fst $URL/HCLG_tri2b_bmmi.fst
wget -O /opt/models/words.txt $URL/words.txt
```

Figure 3.3: *An example of download_models.sh script.*

Deployment of Arbitrary Worker

Even though CloudASR supports only Kaldi out of the box, other ASR systems can be used too. Again, the only thing that the users has to do is to create a worker docker image with their ASR system. The only step that differs from the previous process is that the users have to implement and add to the Dockerfile a script `asr.py` with their own `create_asr` method that returns `ASR` class with these methods:

- `reset()` - this method is called after every requests and it can be used to reset the underlying ASR system.
- `recognize_chunk(pcm)` - this method is used to process small chunks of recordings. The method should accept pcm chunks with frame rate 16000

```
models_dir = '/opt/models'
wst_path = '%s/words.txt' % models_dir
kaldi_config = [
    '--config=%s/mfcc.conf' % models_dir,
    '--verbose=0', '--max-mem=10000000000',
    '--beam=12.0', '--lattice-beam=2.0',
    '--acoustic-scale=0.2', '--max-active=5000',
    '--left-context=3', '--right-context=3',
    '%s/tri2b_bmmi.mdl' % models_dir,
    '%s/HCLG_tri2b_bmmi.fst' % models_dir,
    '1:2:3:4:5:6:7:8:9:10:11:12:13:14:15:16:17:18:19:20',
    '%s/tri2b_bmmi.mat' % models_dir
]
```

Figure 3.4: *An example of config.py script.*

```
FROM ufaldsg/cloud-asr-worker
MAINTAINER Ondrej Klejch

WORKDIR /opt/app
ADD . /opt/app
RUN bash download_models.sh

ENV model cs-alex
```

Figure 3.5: *An example of worker Dockerfile.*

and it should return an interim hypothesis in form of a tuple `(confidence, transcript)`.

- `get_final_hypothesis` - this method is called at the end of every requests, it should return a list of n-best hypotheses in form of a tuple `(confidence, transcript)`.

The creation of such a script is illustrated on Figure 3.6 on the DummyASR class, which will be also used for benchmark purposes in the Chapter 4.

3.1.3 API

The main task of the API container is to forward requests from the clients to the workers. The requests are either in form of HTTP POST for the batch mode or Socket.IO for the online mode. The API is built on top of Flask framework with enabled asynchronous processing which allows single API container to process many parallel requests, because there are no blocking operations in the API container - it just receives requests from clients and forwards them via ZeroMQ to the workers.

```
import time

def create_asr():
    return DummyASR()

class DummyASR:

    def recognize_chunk(self, chunk):
        time.sleep(float(len(chunk)) / 16000 / 2)
        return (1.0, 'Dummy interim result')

    def get_final_hypothesis(self):
        time.sleep(0.2)
        return [(1.0, 'Dummy final result')]

    def reset(self):
        pass
```

Figure 3.6: *An example of alternative ASR implementation.*

3.1.4 Web

CloudASR platform has a web interface with an online demo and an annotation interface. The online demo (See Figure 3.7) allows users to try out CloudASR directly in their web browsers. It has two modes, namely, dictation mode that only shows the best transcription of the recording and evaluation mode that also allows users to confirm that the transcription of the recording is correct.

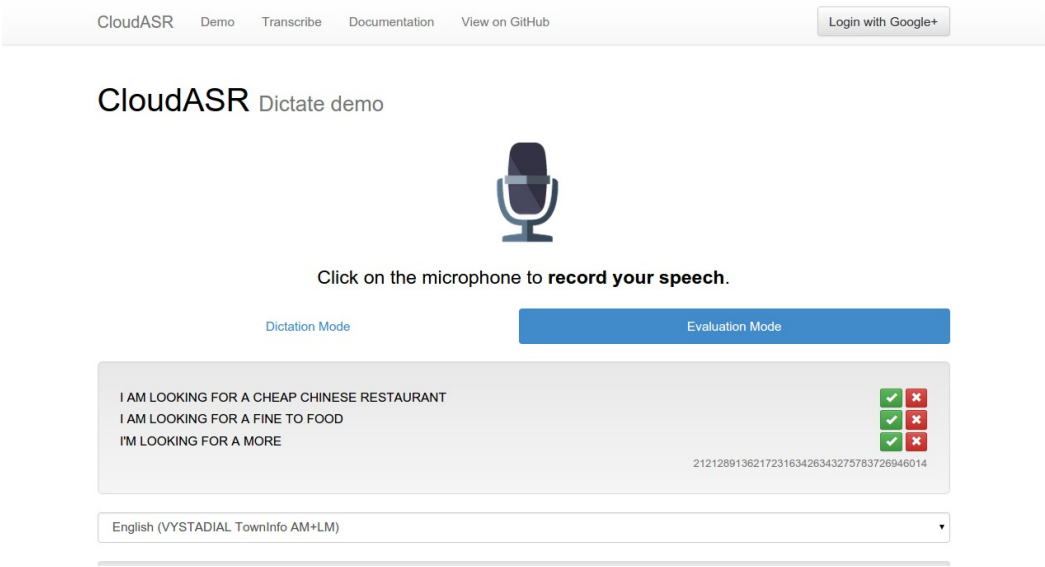


Figure 3.7: *Screen of the Web Demo*

The annotation interface distinguishes between two types of user roles: users and administrators. Normal users are only allowed to add transcriptions (See Figure 3.8) to the recordings. Administrators can also browse all recordings with

their transcriptions and manage workers descriptions.

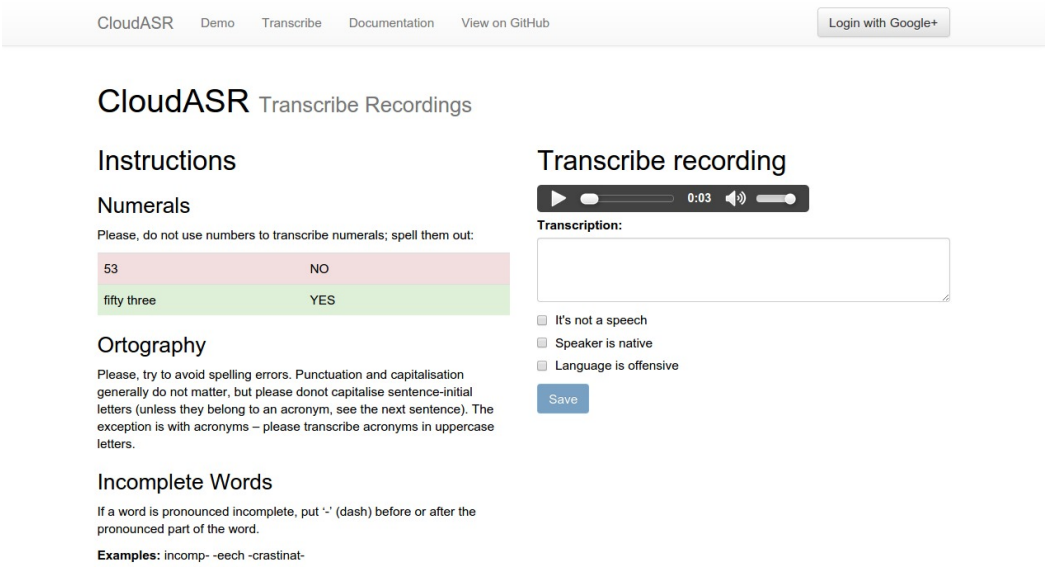


Figure 3.8: *Screen of the Annotation Interface*

IDEA: mention google+ login

IDEA: Describe process of annotation

TODO: Think of ROVER to merge user transcriptions.

3.1.5 Recordings saver

The main task of the Recordings saver is to save and serve recordings processed by workers. When the worker finishes recognition it sends the recording with its n-best hypotheses to the Recording saver via ZeroMQ socket. The saver saves the wave file to the filesystem and it save the n-best hypotheses to the database so that they can be used in the future.

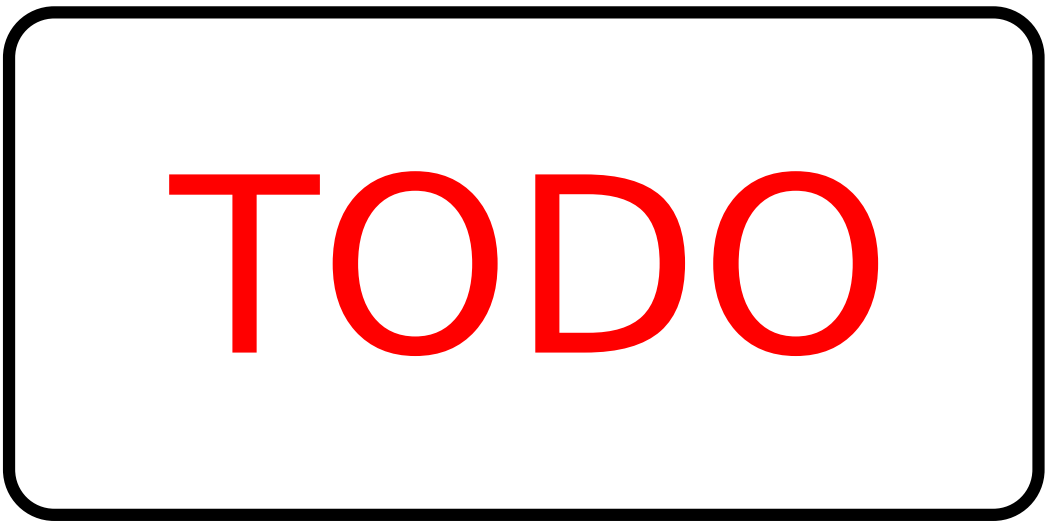


Figure 3.9: *Database schema*

3.2 Request Workflow

The CloudASR platform supports both batch and online speech recogntion. In the batch mode users sends wave files to the API using HTTP POST request and they receive a json with a n-best transcriptions. Users can specify which worker they want to use in `lang` parameter. The batch mode has a similiar interface to Google Speech API, which enables users to switch to CloudASR seamlessly. An example of batch recognition API usage is illustrated on a simple curl command in Figure 3.10, a response from the API is shown in Figure 3.11.

```
curl -X POST --data-binary @recording.wav \
     --header 'Content-Type: audio/x-wav; rate=16000;' \
     'http://api.cloudasr.com/recognize?lang=en-towninfo'
```

Figure 3.10: *An example of batch speech recognition mode request for an en-towninfo worker using curl.*

```
{
  "result": [
    {
      "alternative": [
        {
          "confidence": 0.5549500584602356,
          "transcript": "I'M LOOKING FOR A BAR"
        },
        {
          "confidence": 0.14846260845661163,
          "transcript": "I AM LOOKING FOR A BAR"
        },
      ],
      "final": true
    }
  ],
  "result_index": 0
}
```

Figure 3.11: *An example response of batch recognition mode.*

In contrast to batch mode in online mode users sends PCM chunks of a recording while the recording is recorder. Users can send these chunks as often as they want but it is advised to send a chunk four times per second to achieve a smooth experience. Chunks are sent to the server via Socket.IO technology encoded as a JSON messages. Because JSON does not support encoding of a binary data, it is neccessary to encode PCM chunks into a string. Base64 proved itself to be a sensible compromise between the message size increase and the implementation complexity. The CloudASR platform comes with a JavaScript library for online speech recognition mode. Figure 3.12 shows how this library can be used for speech recognition in Google Chrome web browser.

```
var speechRecognition = new SpeechRecognition();
speechRecognition.onStart = function() {
    console.log("Recognition started");
}

speechRecognition.onEnd = function() {
    console.log("Recognition ended");
}

speechRecognition.onError = function(error) {
    console.log("Error occured: " + error);
}

speechRecognition.onResult = function(result) {
    console.log(result);
}

var lang = "en-wiki";
$("#button_start").click(function() {
    speechRecognition.start(lang);
});

$("#button_stop").click(function() {
    speechRecognition.stop()
});
```

Figure 3.12: *JavaScript code that can be used for speech recognition in Google Chrome.*

3.3 Deployment

The CloudASR platform supports two types of deployment: single host and multi host. Single host deployment allows users to run CloudASR directly on their machines with just one dependency installed - Docker. Whereas multi host deployment allows users to run CloudASR on a set of machines with Mesos installed.

Users can specify which workers they want to run in a `deployment/cloudasr.json` configuration file, see Figure 3.13 for an example. In this file they can specify names of Docker images for the workers, number of instances of these workers and a model name with which the worker will be available for speech recognition. Also, users have to specify runtime variables such as IP address of a slave where the Master should run, mysql connection string and a domain name that HAProxy will use to route requests to the running platform. Finally, users have to specify credentials for Marathon if they want to run the platform on a Mesos cluster. After that they can run CloudASR locally with `make run_locally` command or on a Mesos Cluster with `make run_on_mesos` command.

TODO: don't use Makefile, use some python script with same interface for Multi-Host Deployment

TODO: describe Mesos installation

```
{
    "domain": "cloudasr.klejch.eu",
    "marathon_url": "marathon_url",
    "marathon_login": "marathon_login",
    "marathon_password": "marathon_password",
    "master_ip": "master_ip",
    "connection_string": "mysql connection string",
    "workers": [
        {
            "image": "ufaldsg/cloud-asr-worker-cs-alex",
            "model": "cs-alex",
            "instances": 5
        }
        {
            "image": "ufaldsg/cloud-asr-worker-en-towninfo",
            "model": "en-towninfo",
            "instances": 5
        }
    ]
}
```

Figure 3.13: *An example of the CloudASR configuration that specifies to run 5 cs-alex workers and 5 en-towninfo workers using respective Docker image. With this configuration file the CloudASR platform can be run locally with command* `make run_locally` *or on a Mesos cluster with command* `make run_on_mesos`.

3.3.1 Scalability

As mentioned before, one of the key requirements for the CloudASR platform was scalability. To successfully fulfill this requirement architecture was split into smaller nodes, which communicates which each other by sending messages over ZeroMQ sockets. This enables the platform to run on several machines. Additionally, the heartbeating makes it possible to scale workers dynamically without need to stop the platform. Finally, load-balancing allows to run several API nodes and spread the load between. Yet, this solution is limitted by the network capacity, but it is possible to deploy the CloudASR to a different data center and load balance between data centers. This makes the CloudASR platform almost infinitely scalable.

3.3.2 Contiunous Integration & Countinuous Delivery

In order to ensure quality and stability of the CloudASR platform, two practises were used during the development: Continuous Integration [3] and Continuous Delivery [8]. The goal of these practises is to built, test and deploy the CloudASR platform as often as possible, to get a feedback from real usage. To achieve that the CloudASR platform uses Jenkins-CI server, that watches the CloudASR git repository and on every push to the repository it builds Docker images, tests the code and push the built Docker images to the Docker Registry. After that it is

possible to deploy the CloudASR platform with a specific version and it is also possible to switch back to older versions when anything goes wrong. To minimize failures CloudASR is deployed first to the development environment, where the users can test it and then it can be deployed to the production environment.

One of the most important practises for Continuos Integration & Continuous delivery is testing. Thus, all crucial parts of the CloudASR platform are covered with tests, namely unit tests, integration tests and end-to-end tests.

Each node was implemented with two main design patterns in mind, namely Dependency Injection [2] and Factory Method [4]. Usage of these patterns together with message oriented architecture made it possible to unit test the whole platform easily, because it enabled to pass test doubles into the nodes and then send fake messages needed to test a correct behaviour of the node. A typical unit test structure of the CloudASR platform node looks like a test in Figure 3.14.

In addition to unit tests there are also integration tests, which test the factory methods that create production ready nodes, and end-to-end tests, which test that both batch and online recognition mode requests are handled correctly. This test suite ensures that developers do not break anything and it also gives them confidence to change the code without fear.

```
def test_worker_sends_heartbeat_after_finishing_task(self):
    messages = [
        {"frontend": self.make_frontend_request("message 1")}
    ]

    self.run_worker(messages)
    self.assertThatHeartbeatsWereSent(["STARTED", "FINISHED"])
```

Figure 3.14: *An example of unit test structure.*

TODO: add better caption

4. Evaluation

In order to show that the CloudASR platform is ready for the production usage several benchmarks were made. First, real time factor (RTF) of the batch speech recognition mode was measured and compared with Google Speech API. Second, latency of the online speech recognition mode was measured. Finally, number of parallel requests for batch speech recognition mode was measured.

4.1 RTF of Batch Speech Recognition

In the first benchmark RTF of CloudASR batch mode was compared with RTF of Google Speech API using test set from the Czech Public Transportation Information Domain [11]. WER and RTF of both APIs were measured with following results: Google Speech API had 60% WER and RTF 0.3 whereas CloudASR had 22% WER and RTF 0.22. Request times of both APIs are displayed in Figure 4.1.

This benchmark shows that RTF of CloudASR batch mode is lower than RTF of Google Speech API. Moreover, the benchmark shows that the CloudASR platform can achieve better accuracy than Google Speech API on limitted domains if the used decoding graphs are customized for that domains.

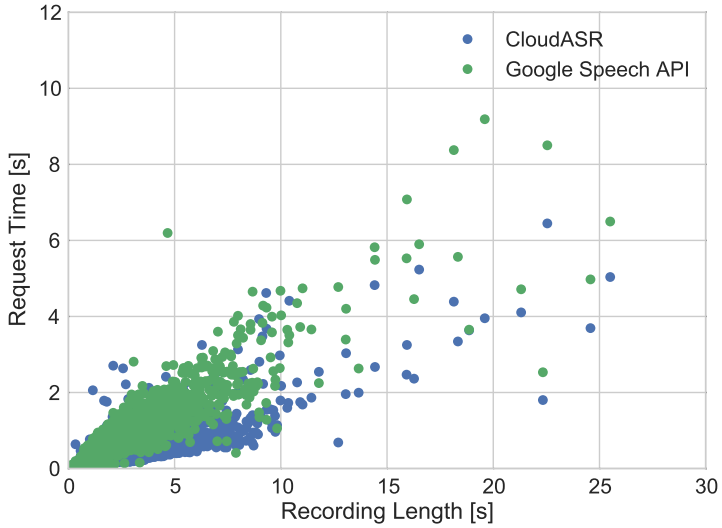


Figure 4.1: *Batch recognition benchmark.*

4.2 Latency of Online Speech Recognition

In the second benchmark latency of CloudASR online mode was measured. Since the low latency is crucial for successfull usage of speech recognition in dialogue systems and there are not so many webservice that provide online speech recognition mode support for online speech recognition mode can be seen as a key feature of the CloudASR platform. The reason why the online speech recognition mode is better suited for dialogue systems is that it is not neccessary to wait for

the whole recording to be recorded and instead it is possible to get the interim results while the speech is recorded. Thus, dialogue systems can react quickly. Results, plotted in Figure 4.2, show that the latency of the online speech recognition mode remains small even for long recordings which is in contrast to the increasing latency of the batch speech recognition mode.

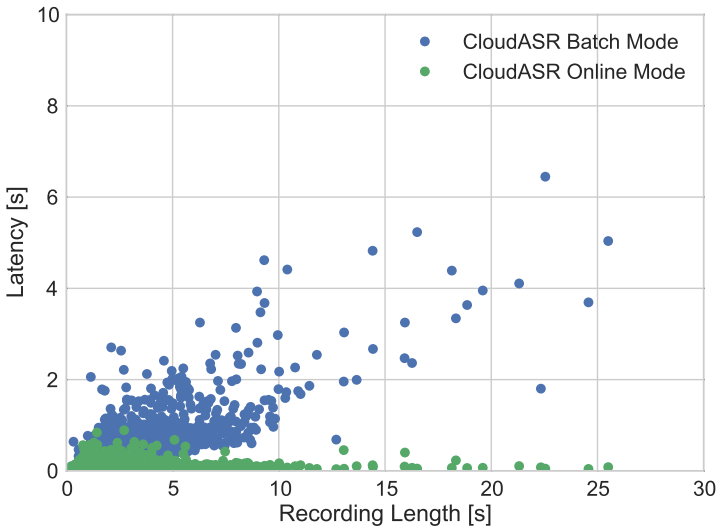


Figure 4.2: *A latency comparison for CloudASR online and batch mode.*

4.3 Parallel Requests Benchmark

Since the main bottleneck of the CloudASR platform is a number of running workers workers with dummy asr engine (described in Figure 3.6) that would enable to test how many parallel requests can CloudASR handle were used. As a result, 1000 dummy workers could run on a Mesos cluster with 5 slaves (4CPU, 16GB RAM). Also, HAProxy load-balancer, which spread workload accross 5 frontend containers, was deployed.

```
seq 1 100 | xargs -P100 -I {} \
  curl -X POST --data-binary @resources/test.wav \
    --header "Content-Type: audio/x-wav; rate=16000;" \
    -s -w "%{time_total}\n" \
    http://api.cloudasr.com/recognize?lang=dummy
```

Figure 4.3: *A simple benchmark script that sends 100 parallel requests to the CloudASR API and prints out request time for each request.*

Then several benchmarks were run to show how RTF of the batch recognition mode changes with different number of parallel requests. The platform was tested with a different number of parallel requests (50, 250, 500, 750 and 1000) and with files with different lengths (5s, 10s, 20s, 30s, 40s, 50s and 60s). In order to be able to run so many parallel request, a simple benchmark, shown in Figure 4.3,

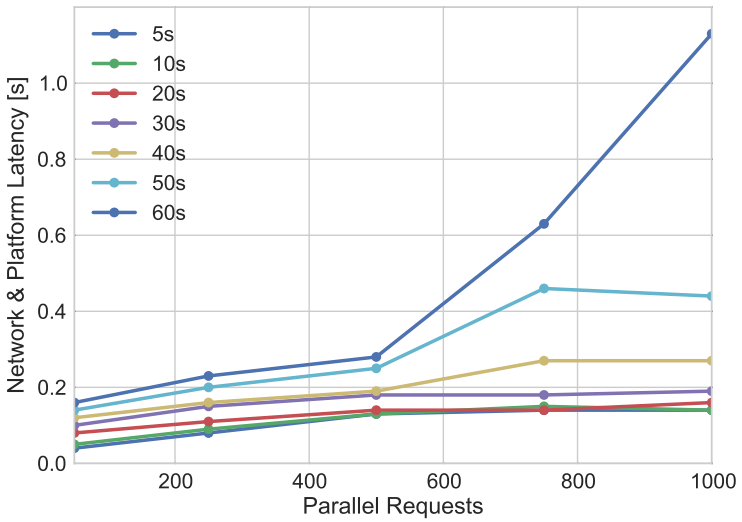


Figure 4.4: *The graph shows platform & network latency for recordings with various lengths given the number of parallel requests.*

was run on 5 machines. Each machine sent only one-fifth of the total number of parallel requests ten times and then the average latency was computed accross all machines and all batches.

Results, summarized on Figure 4.4, show that CloudASR platform adds just a very little overhead compared to the raw dummy worker for small recordings. But the latency increases rapidly for large recordings with more than 500 parallel requests. This is probably caused by network capacity of the servers that run the benchmark and it should not affect the platform with the real usage. Moreover, this will not affect the online speech recognition mode because it sends only very small messages. Therefore, the platform should able to handle much more parallel requests with appropriate number of workers.

Conclusion

Goals of this thesis were to develop a cloud platform for ASR, CloudASR, and an annotation interface for annotating speech data. These goals were successfully accomplished and in several aspects even surpassed - in addition to original requirement to create batch recognition mode, we also implemented online recognition mode. In the following sections we summarize our achievements in detail and at the end we propose ideas for future work.

Cloud platform for ASR

The first goal of this thesis was to develop a cloud platform for ASR, CloudASR, that would provide batch API for speech recognition of wave files. The platform uses Master/Worker architecture. Consequently, it is able to run both on single-machine and multi-machine setup. The platform allows us to run workers for various language models and to scale workers according to our needs. To be able to run CloudASR on several machines we chose Mesos/Marathon as an underlying technology The current implementation of the API supports two modes of speech recognition: batch and online.

Firstly, batch mode allows users to send a file with a recording to the server and then it sends transcribed text back as a json. API of this mode is similiar to Google Speech API which allows users to switch from Google Speech API to CloudASR easily.

Secondly, users can transcribe speech recordings in real-time via online mode. We have also created Python and JavaScript libraries for using our API.

Finally, we wanted CloudASR to be easily deployable. Because of that, we used Docker for creating and running application containers. As a result only dependency that users have to install is Docker for single-node setup and Mesos Cluster for multi-node setup. Moreover, installation scripts for these dependencies are included within the distribution together with deployment scripts, that can be used for CloudASR instances management.

Annotation interface

The second goal of this thesis was to create an annotation interface for annotating speech data. First responsibility of the annotation interface is to collect and store obtained recordings.

The second responsibility is to allow users to rate transcriptions of the recordings (Is the transcription correct? yes/no) or to subsequently add their own transcriptions. The annotation interface implements algorithm to choose golden transcription from several manual transcriptions that were obtained for the recording. Additionally it is also possible to add manual transcriptions via external job at CrowdFlower.

The third responsibility is to provide export of transcribed recordings. This can be done either by downloading archive from the web or by using Torrent.

Future work

- Since manual transcription of recordings is expensive it would be good to make users transcribe only parts of the recordings in which ASR system wasn't confident enough [17]. This idea could be used for both user transcription and CrowdFlower transcription.
- With manually transcribed recordings from CloudASR platform it is possible to continuously improve accuracy of the underlying ASR system by adapting the language model to the type of language that the users of the CloudASR really use. Thus CloudASR could provide an option to automatically update language model when a certain amount of new transcribed recordings was collected.
- Because running CloudASR platform is expensive in terms of costs for a server hosting, it would be good to optimize usage of individual workers so that spare workers are shut down when there is no need for them and new workers are started when the traffic arise. This can be achieved either by providing feedback control based systems [9] or by using machine learning techniques [5].
- As CloudASR platform provides API for speech recognition, it could also be used for another speech related tasks like Language Identification, Speaker Identification, Voice Activity Detection, etc.

Bibliography

[1] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio, *Theano: A cpu and gpu math compiler in python*, Proc. 9th Python in Science Conf, 2010, pp. 1–7.

[2] Martin Fowler, *Inversion of control containers and the dependency injection pattern*, 2004.

[3] Martin Fowler and Matthew Foemmel, *Continuous integration*, Thought-Works) http://www.thoughtworks.com/Continuous Integration. pdf (2006).

[4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design patterns: Abstraction and reuse of object-oriented design*, Springer, 1993.

[5] Zhenhuan Gong, Xiaohui Gu, and John Wilkes, *Press: Predictive elastic resource scaling for cloud systems*, Network and Service Management (CNSM), 2010 International Conference on, IEEE, 2010, pp. 9–16.

[6] Carl Hewitt, *Viewing control structures as patterns of passing messages*, Artificial intelligence **8** (1977), no. 3, 323–364.

[7] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy H Katz, Scott Shenker, and Ion Stoica, *Mesos: A platform for fine-grained resource sharing in the data center.*, NSDI, vol. 11, 2011, pp. 22–22.

[8] Jez Humble and David Farley, *Continuous delivery: reliable software releases through build, test, and deployment automation*, Pearson Education, 2010.

[9] Philipp K. Janert, *Feedback control for computer systems*, O'Reilly Media, 2013.

[10] Filip Jurčíček, Ondřej Dušek, Ondřej Plátek, and Lukáš Žilka, *Alex: A statistical dialogue systems framework*, Text, Speech and Dialogue, Springer, 2014, pp. 587–594.

[11] Matěj Korvas, Ondřej Plátek, Ondřej Dušek, Lukáš Žilka, and Filip Jurčíček, *Vystadial 2013–czech data*, (2014).

[12] K-F Lee, H-W Hon, and Raj Reddy, *An overview of the sphinx speech recognition system*, Acoustics, Speech and Signal Processing, IEEE Transactions on **38** (1990), no. 1, 35–45.

[13] Dirk Merkel, *Docker: lightweight linux containers for consistent development and deployment*, Linux Journal **2014** (2014), no. 239, 2.

[14] Ondrej Plátek and Filip Jurcıcek, *Free on-line speech recogniser based on kaldi asr toolkit producing word posterior lattices*, 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue, 2014, p. 108.

[15] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukáš Burget, Ondřej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlíček, Yanmin Qian, Petr Schwarz, et al., *The kaldi speech recognition toolkit*, (2011).

[16] David Rybach, Christian Gollan, Georg Heigold, Björn Hoffmeister, Jonas Lööf, Ralf Schlüter, and Hermann Ney, *The rwth aachen university open source speech recognition system.*, Interspeech, 2009, pp. 2111–2114.

[17] Matthias Sperber, Graham Neubig, Satoshi Nakamura, and Alex Waibel, *On-the-fly user modeling for cost-sensitive correction of speech transcripts*, Spoken Language Technology Workshop (SLT), 2014.

[18] Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, et al., *The htk book*, vol. 2, Entropic Cambridge Research Laboratory Cambridge, 1997.

List of Tables

List of Abbreviations

Attachments