BIG-DATA PROJECT DOCUMENT

Lu Li

1 PROJECT INTRODUCTION

This project deploys MNIST application into the container. Users submit digital pictures with handwritten characters through curl-XPOST command. This program first recognizes the pictures, and then returns the recognized numbers to users. Every time a user submits an image, identifies a text and timestamp information in MNIST, it is recorded in Cassandra for storage.

The web service is created using Flask, and the prediction model is the Softmax model from Tensorflow tutorial. The prediction model is exported using tensorflow.builder, and loaded when building Docker Container under app.py. Next we have the online database. Online database is achieved using Cassandra through Docker. First we launch as Cassandra image, configure it using cqlsh, then link the container we built earlier with this Cassandra image.

2 TERMINOLOGY EXPLANATION

2.1 MNIST

MNIST is a classic introductory demo for deep learning. It consists of 60,000 training pictures and 10,000 test pictures, each of which is 28*28 in size (as shown below), and is made of black and white (where the black is a floating point number of 0-1, the deeper the black is, the closer the value is to 1). These pictures are different handwritten numbers from 0 to 9.

2.2 Tensorflow

TensorFlow is an open source software library using data flow graphs for numerical computation. Nodes represent mathematical operations in graphs, while edges represent multidimensional arrays of data that are interconnected among nodes, i.e. tensors. Its flexible architecture allows you to deploy computing on a variety of platforms, such as one or more

CPUs (or GPUs), servers, mobile devices, and so on. TensorFlow was originally developed by researchers and engineers from the Google Brain Group (affiliated to the Google Machine Intelligence Research Institute) for machine learning and deep neural network research, but its versatility makes it widely applicable to other computing fields.

2.3 Deep Learning

Deep learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. Learning can be supervised semi-supervised or unsupervised.

2.4 Flask

Flask is a lightweight Web application framework written in Python. Its WSGI toolbox uses Werkzeug and its template engine uses Jinja2. Flask uses BSD authorization. Flask is also known as the "microframework" because it uses a simple core and extensions to add other functions. Flask does not have the default database, window experience tool.

2.5 Docker

Docker is an open source application container engine that allows developers to package their applications and dependencies into a portable container and then publish them to any popular Linux machine, as well as virtualization. Containers are completely sandboxed and do not have any interfaces with each other.

2.6 Cassandra

Apache Cassandra is an open source distributed database management system designed to handle large amounts of data on many commercial servers, providing high availability without a single point of failure. Cassandra provides powerful support for clusters spanning multiple data centers with asynchronous, unowned replication, allowing all clients to perform low-latency operations.

2.7 CQL

Cassandra Query Language or CQL is a declarative language that enables users to query Cassandra using a language similar to SQL. CQL was introduced in Cassandra version 0.8 and is now the preferred way of retrieving data from Cassandra. Prior to the introduction of CQL, Thrift an RPC based API, was the preferred way of retrieving data from Cassandra. A major benefits of CQL is its similarity to SQL and thus helps lower the Cassandra learning curve. CQL is SQL minus the complicated bits. Think of CQL as a simple API over Cassandra's internal storage structures.

2.8 RESTful API

A RESTful API is an application program interface API that uses HTTP requests to GET, PUT, POST and DELETE data. A RESTful API -- also referred to as a RESTful web service -- is based on representational state transfer REST technology, an architectural style and approach to communications often used in web services development.

2.9 Big Data

Big data refers to data sets that are too large or complex for traditional data-processing application software to adequately deal with. Data with many cases (rows) offer greater statistical power, while data with higher complexity (more attributes or columns) may lead to a higher false discovery rate. Big data challenges include capturing data, data storage, data analysis, search, sharing, transfer, visualization, querying, updating, information privacy and data source. Big data was originally associated with three key concepts: volume, variety, and velocity. Other concepts later attributed with big data are veracity (i.e., how much noise is in the data) and value.

3 IMPLEMENTATION

3.1 Export MNIST Deep Learning Model

The mnist deep learning model used in this project uses softmax Regression, referring to the official code of tensorflow on github, its final correct rate is around 91%. We need to save the model after training. First we should define a saver, then use the function saver.save to save the model to a predefined path. The relevant code is as follows:

```
saver = tf.train.Saver()
sess = tf.InteractiveSession()
tf.global_variables_initializer().run()
for _ in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys}
     saver.save(sess, 'model/model.ckpt')
```

The following four files are saved using the above code:

```
1. checkpoint: saves the path of all models. The first line represents
the current state, which model is used by the first line when loading
the model.
2. model.ckpt.data-00000-of-00001
3. model.ckpt.index
4. model.ckpt.meta
```

After saving the model, you also need to load the model into use when using the model for prediction. The relevant code for loading the model is as follows:

```
def restore(self):
    saver = tf.train.Saver()
    ckpt = tf.train.get_checkpoint_state(CKPT_DIR)
    if ckpt and ckpt.model_checkpoint_path:
        saver.restore(self.sess, ckpt.model_checkpoint_path)
```

3.2 Connected to Cassandra

We need to connect to Cassandra first, and then use it as a database to store the user's submitted images, recognized text and timestamp information. We need to start the Cassandra container and CQLSH, and build namespaces and tables through the command line.

First, start a Cassandra server instance with the following command, I name it ll-cassandra. After running, it will run a cassandra container named ll-cassandra, and the version is latest.

```
$ docker run --name ll-cassandra -d cassandra:latest
```

Enter the command `$ docker ps` to get the Cassandra container instance that is now running. As shown below, you can see that the ll-cassandra container is running, and the port mapping has been successful `'0.0.0.0:9042->9042/tcp'`

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
 PORTS                                                NAMES
e890498e529b        8dd9c5333d40        "docker-entrypoint.s…"   4 days ago          Up 4 days
 7000-7001/tcp, 7199/tcp, 9160/tcp, 0.0.0.0:9042->9042/tcp   ll-cassandra
```

Then, we use the following command entry Docker adds for linked containers ll-cassandra and runs cqlsh (Cassandra Query Language Shell) , allowing you to execute CQL statements against your database instance:

```
$ docker run -it --link ll-cassandra:cassandra --rm cassandra cqlsh
cassandra
```

After the connection is successful, the following will occur:

```
Connected to Test Cluster at cassandra:9042.
[cqlsh 5.0.1 | Cassandra 3.11.3 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh>
```

Create the following KEYSPACE called demandkeyspace by entering the following code:

```
CREATE KEYSPACE demandkeyspace
WITH REPLICATION = { 'class' : 'SimpleStrategy' ,
 'replication_factor' : 1 };
```

Finally, enter the following command in turn to create a table called demand, which is used to store the user-uploaded image file information `filename`, prediction result `guess` and timestamp `time`, where `time` is used as the primary key.

```
use demandkeyspace;
CREATE TABLE demandtable (
  time text PRIMARY KEY,
  filename text,
  result text);
```

3.3 Web Deployment Using Flask

First we have to write a page that submits the image, named index, and the routing address is '/'. A

tag is marked with `enctype='multipart/form-data'` and contains an 选取文件 未选择文件 tag. The server application accesses the file through the files dictionary on the request object and jumps to the route `'/predict'` when the file is submitted. The specific code is as follows:

```
@app.route('/')
def index():
    return '''
    <!doctype html>
    <html>
    <body>
    <form action='/predict' method='post' enctype='multipart/form-
data'>
        <input type='file' name='file'>
    <input type='submit' value='提交'>
    </form>
    '''
```

Then receive the image file uploaded by the user and perform a security check on the name of the image file. We specify that the image format can only be 'pdf' , 'png', 'jpg', and 'jpeg', the file needs to exist, and the file name cannot be empty. Call the secure_filename() function in werkzeug to make sure the file name is safe. Finally, the image file is read in.

```
ALLOWED_EXTENSIONS = set([ 'pdf', 'png', 'jpg', 'jpeg'])
def allowed_file(filename):
    return '.' in filename and \
```

```
            filename.rsplit('.', 1)[1] in ALLOWED_EXTENSIONS

@app.route('/predict', methods=['POST'])
def predict():
if request.method == 'POST':
        #In case no file was uploaded
        if 'file' not in request.files:
            flash('No file part')
            return redirect(request.url)
        file = request.files['file']
        #In case file has an empty name
        if file.filename == '':
            flash('No selected file')
            return redirect(request.url)

        #Everything is correct and we can run the prediction
        if file and allowed_file(file.filename):
            #save and read uploaded image
            filename = secure_filename(file.filename)
            file.save(secure_filename(file.filename))
```

Then create a link to cassandra, connect to the consumerkeyspace we created earlier, and we will manipulate it after.

```
  cluster = Cluster(contact_points=['127.0.0.1'],port=9042)
  session = cluster.connect(KEYSPACE)
```

Then, We predict the user's image through the previously saved deep learning model and return the predicted value. Finally, insert the obtained current time, file name and prediction result into the database, and return the recognition result.

```
app = Predict()
x=app.predict(image)
current_time = str (datetime.datetime.now())
session.execute(
"""
INSERT INTO demandtable (time, filename, result)
VALUES (%s, %s, %s)
""",
(current_time,filename,x)
)
return '识别结果: '  + x
```

3.4 Build Docker Container

To build a docker container, we need python files, requirements.txt and Dockerfile. The python file is a web application like we mentioned in 3.3. Requirements.txt is used to record all dependencies and their exact version numbers for new environment deployment. In this project, the contents of requirements.txt are as follows:

```
Flask
numpy
pillow
datetime
cassandra-driver
tensorflow
```

Dockerfile is a text file that contains a strip of instructions. Each instruction builds a layer. Therefore, the content of each instruction is to describe how the layer should be constructed, and customize the configuration and files added to each layer. The contents of the Dockerfile in this project are as follows:

```
#Use an official Python runtime as a parent image
FROM python:3.6

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80
EXPOSE 9042

# Define environment variable
ENV NAME World

# Run app.py when the container launches
CMD ["python", "app.py"]
```

Then run the build command to create a Docker image, which we're going to name using the `--tag` option.

```
docker build --tag=big-data .
```

Then run the app in the background, in detached mode:

```
docker run -d -p 4000:80 big-data
```

After that, we need to upload our built image and run it somewhere else. Now we push the container to registries as I want to deploy containers to production. First, we should log in with my Docker ID

```
$ docker login
```

The notation for associating a local image with a repository on a registry is `username/repository:tag`. Now, put it all together to tag the image. Run docker tag image with your username, repository, and tag names so that the image uploads to your desired destination. The syntax of the command is:

```
docker tag big-data lululu016/project:bigdata
```

Then, upload my tagged image to the repository:

```
docker push lululu016/project:bigdata
```

From now on, you can use `docker run` and run my app on any machine with this command:

```
docker run -p 4000:80 lululu016/project:bigdata
```

4 PROJECT DISPLAY

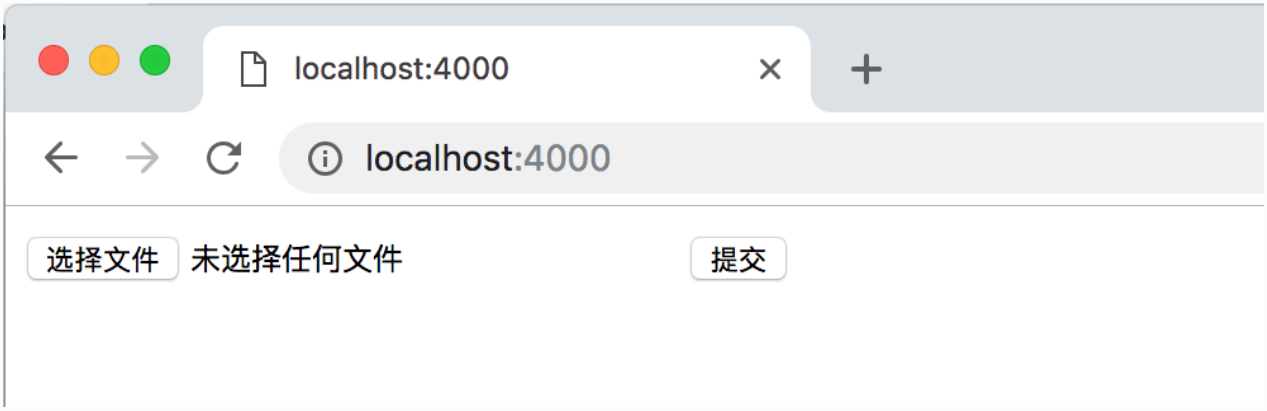
First, start our cassandra container:

```
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
     PORTS                                                NAMES
e890498e529b        8dd9c5333d40        "docker-entrypoint.s…"   8 days ago          Up 4 days
     7000-7001/tcp, 7199/tcp, 9160/tcp, 0.0.0.0:9042->9042/tcp   ll-cassandra
```

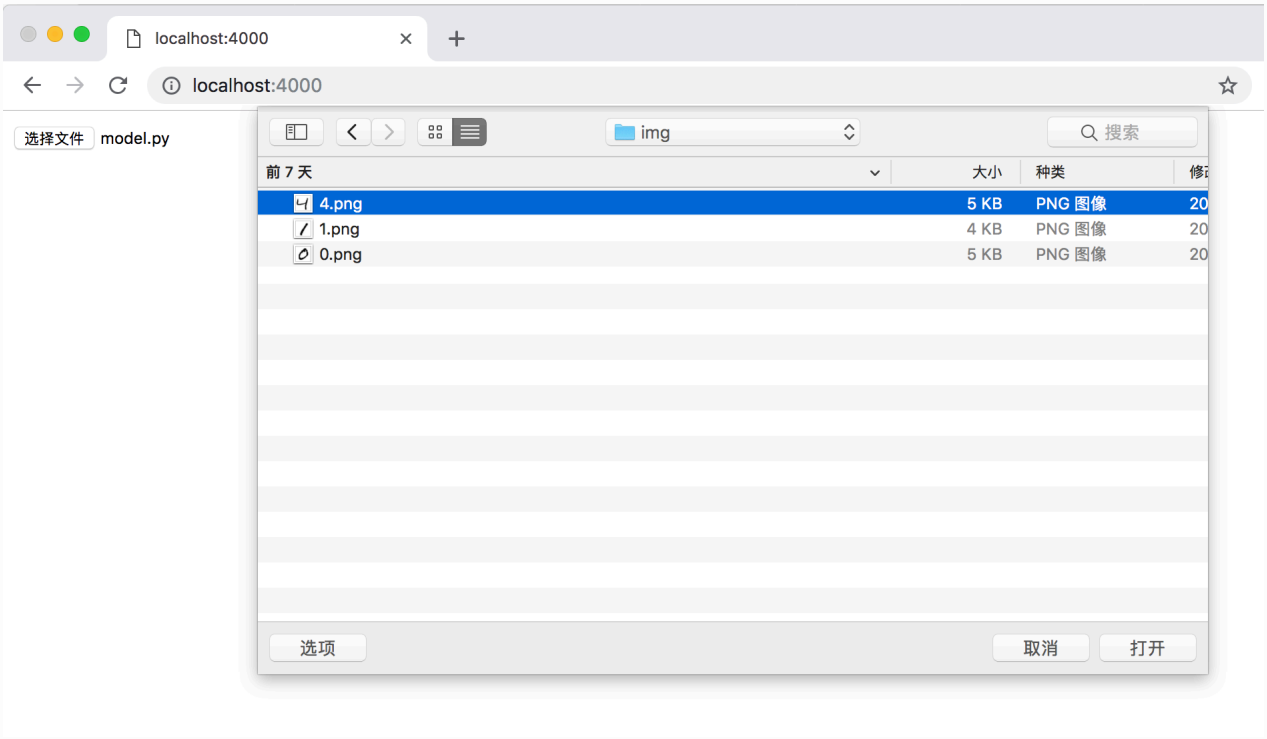
Then, this image exposes the standard Cassandra ports, so container linking makes the Cassandra instance available to other application containers. Start my application container like this in order to link it to the Cassandra container:

```
docker run --name ll-mnistproject --link ll-cassandra:cassandra -p
4000:80 ll-bigdata
```

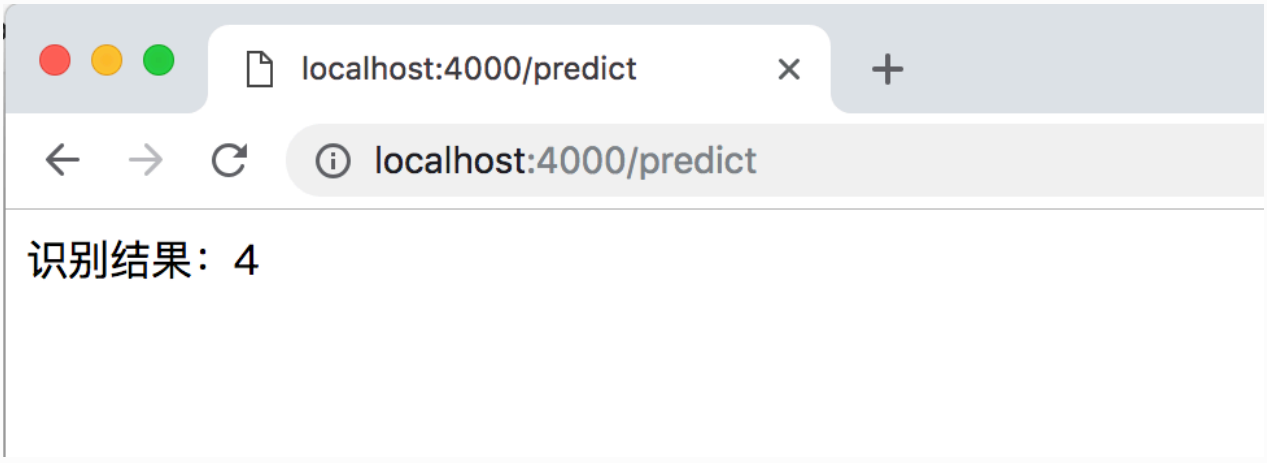
The app is running on localhost:4000. Click the button of "选择文件".



Select some pictures of number.



Then using the model of mnist, it will return the predict reult.



The following command starts another Cassandra container instance and runs cql against your original Cassandra container, allowing you to execute CQL statements against your database instance:

```
docker run -it --link ll-cassandra:cassandra --rm cassandra cqlsh
cassandra
```

Using cql to check the update of cassandra database. First using the keyspace we defined before.

```
cqlsh> use demandkeyspace;
```

Then select all information from the table we defined before.

```
cqlsh:demandkeyspace> select * from demandtable;
```

We can see all the information of time, filename and result which is stored every time users operate on this app.

time	filename	result
2019-01-29 16:55:06.393990	4.png	4
2019-01-29 16:52:43.468591	4.png	4
2019-01-29 17:00:56.744410	0.png	0
2019-01-29 17:24:30.083995	0.png	0
2019-01-29 22:26:27.220744	4.png	4
2019-01-29 17:21:51.489677	4.png	4
2019-01-29 17:07:07.545905	0.png	0
2019-01-29 17:00:51.787634	1.png	1
2019-01-29 17:28:24.469449	4.png	0
2019-02-10 14:28:55.254396	4.png	4
2019-01-29 17:25:40.487908	1.png	1
2019-02-02 07:52:56.474453	4.png	4
2019-02-10 14:29:30.843461	0.png	0
2019-01-29 17:26:33.197332	1.png	1
2019-02-02 07:35:46.944728	4.png	4

5 CONCLUSION

In this study, I think the most rewarding thing is that I learned relevant knowledge of Docker and Cassandra and applied them into practice. Because these two things are new to me, and they are lightweight substitutes for some basic tools I have learned in normal study, which makes me feel that they are very convenient and efficient to use, which will be of great help to my future study. The following part is my conclusion of these two tools.

5.1 Docker

As an emerging virtualization approach, Docker has many advantages over traditional virtualization.Docker has a higher utilization of system resources because the container does not require additional hardware virtualization and running a full operating system. Whether it's application execution speed, memory loss or file storage speed, it is more efficient than traditional virtual machine technology. Therefore, compared to virtual machine technology, a host with the same configuration can often run a larger number of applications.

Traditional virtual machine technology often takes several minutes to start an application service, while Docker container applications can run at the second or even millisecond level because they run directly on the host kernel without having to start a full operating system. Greatly saves development, testing, and deployment time.

A common problem in the development process is the issue of environmental consistency. Due to the inconsistency of the development environment, test environment, and production environment, some bugs were not discovered during the development process. The Docker image provides a complete runtime environment in addition to the kernel, ensuring consistency in the application's runtime environment, so that there is no such problem as "this code is fine on my machine."

5.2 Cassandra

Cassandra is a distributed system, only need to add nodes to expand storage space ; As we all know, mysql single-table data volume has a bottleneck, when the amount of data reaches a certain level, you need to consider sub-library table or partition, etc. Wait. And mysql is not a distributed database (although there is master-slave, this is not really distributed). When using cassandra, you don't need to consider this problem. When the amount of data increases, you only need to increase the cassandra machine (operation and maintenance level expansion), for development, there is almost no impact.

The design mechanism of cassandra determines that the cost of changes to its data schema (column increase or decrease) is very low. In mysql, the cost of schema changes (column additions and deletions) to a large data table is very, very high, and accidentally leads to lock tables, resulting in business anomalies. And some business data is large and because the instability of the demand will often require the change of the data schema definition, then you can consider cassandra, Cassandra is very suitable for doing such a thing;

Cassandra write performance is very high, Netflix once reached more than 1 million writes per second in a test; very suitable for high-write applications, such as ad click records, user browsing records, etc.

REFERENCE

http://docs.jinkan.org/docs/flask/patterns/fileuploads.html

http://abiasforaction.net/a-practical-introduction-to-cassandra-query-language/

https://hub.docker.com/_/cassandra/

https://docs.docker.com/get-started/

http://flask.pocoo.org/docs/0.12/quickstart/#a-minimal-application