The Summary Report of the Big Data Project

by

Yi Nianzhang

Under the Supervision of

Zhang Fan

Department of Physics

University of Science and Technology of China

June 2019

Acknowledgements

First of all, I would like to express my heartfelt gratitude to my supervisor Mr. Zhang Fan, for his valuable guidance, enlightening instruction and careful modification throughout the process. Without his patience and prudence, I could not have finished the project to its present form.

Besides, I shall extend my thanks to New Oriental, for providing me with the valuable experience which have benefited me a lot.

Last but not the least, I'd like to thank all my friends, for their encouragement and support.

Y. N.

CONTENTS

Introduction

Pattern recognition is an important aspect of artificial intelligence research, which studies how to use machines to realize the ability of learning, recognizing and judging. As a classic example of pattern recognition, Handwritten Digit Recognition is usually the first AI recognition program that beginners start to learn, which uses simple programs to achieve high precision prediction results. In the project, I will use it as an instance.

Docker container technology is the mainstream virtualization technology in recent two years. Docker enables developers to publish applications to any popular Linux and Windows machines in a way that packages code and dependencies together. A Docker container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another. It is standard, lightweight (by sharing the guest operation system) and secure.

NoSQL means 'not only SQL'. When dealing with big data, relational databases often expose many problems that are difficult to overcome. NoSQL is created to solve the challenges brought by multiple types in big data aggregation, especially meeting the problems of big data application. By abandoning the requirement of high data consistency, Cassandra has the advantages of fast query speed, strong scalability and easy distributed extension.

The final realization of this project is:

Put the MNIST applications in the container. Users can submit handwritten digital pictures by 'curl' command. The program will identify the picture and return back the predicted numbers, upload time and the file name. At last, it will also record the information in the Cassandra for storage.

General

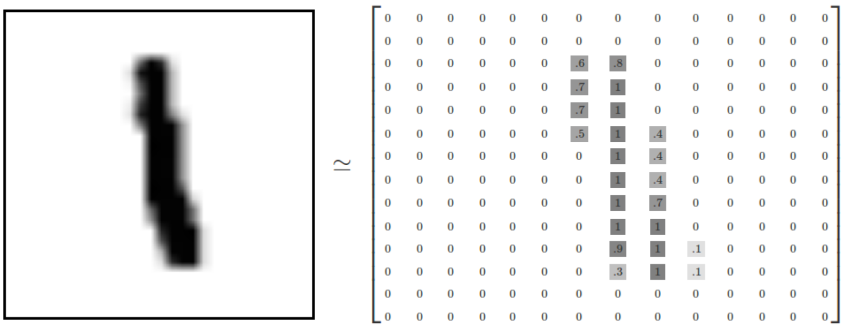
I. MNIST

The MNIST database is a large database of handwritten digits that is commonly use for training various image processing systems.

The database contains a variety of handwritten digital pictures:

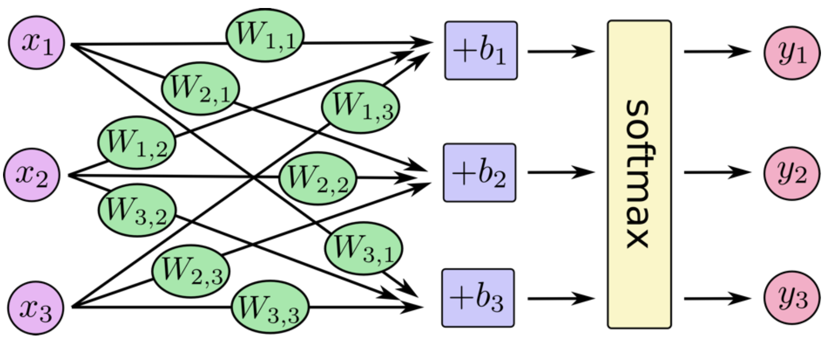
And tags 5,0,4,1 for each picture, to tell us the real number of the image.

Each picture contains 28*28 pixels, which can be represented by an array:



We can expand this array into a vector 784 in length, so the picture will be points in the 784-dimensional vector space.

The following graph can be used to explain the soft-max model. The weighted sum of the inputs is added with a constant, and then input into the soft-max function:



It can be represented as:

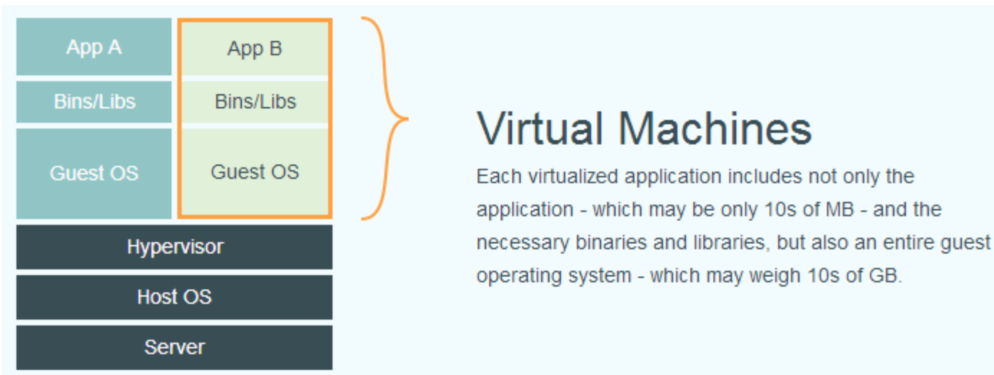
$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

The soft-max function will give us the result of the possibilities of every number. Through training, we can change the weight W to achieve recognition and improve accuracy.

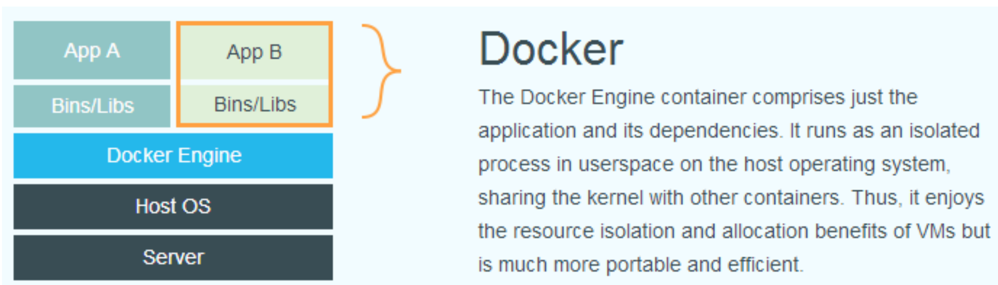
II. Docker

Docker is a lighter technology than virtual machine to run applications in any popular systems which are designed for one special type operation system originally.

Traditional virtualization is implemented at the hardware level, and the operating system part needs to be packaged together:

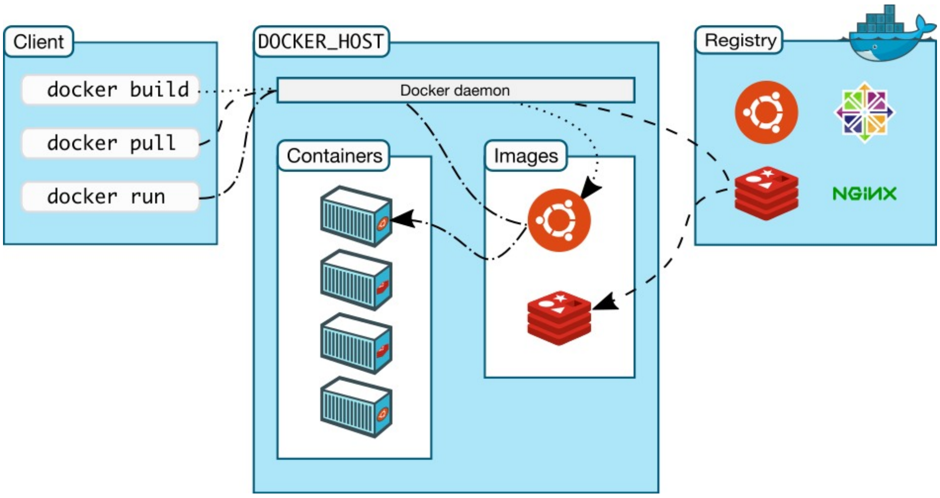


However, Docker realizes virtualization at the operating system level, and will directly reuse the local host OS:



Users can create containers by downloading images, which are centrally stored in

Repository, and Docker can use containers to run applications.



III. Cassandra

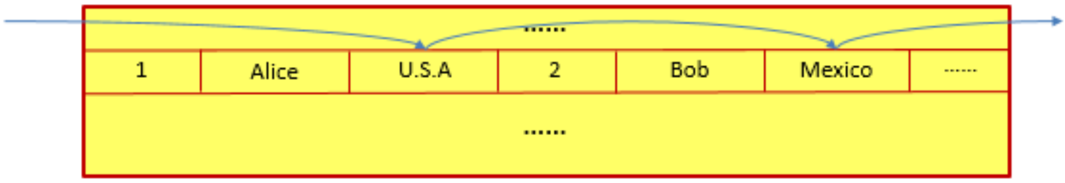
In a traditional relational database, the table that operates on may be as follows:

id(PK)	name	Country
1	Alice	U.S.A
2	Bob	Mexico
...	...	

And in the database file corresponding to the table, the values in each row will be recorded sequentially, thus forming a data file as shown in the following figure:

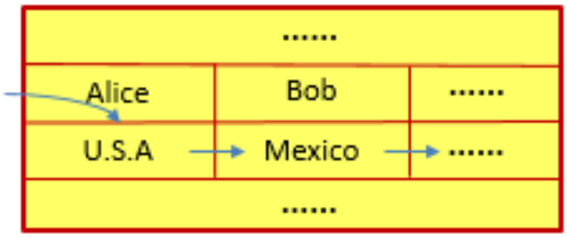
......						
1	Alice	U.S.A	2	Bob	Mexico	......
......						

Therefore, when executing the SQL statement of searching one column, the relational database cannot continuously manipulate the data recorded in the file:



This greatly reduces the performance of relational databases: in order to run the SQL statement, relational databases need to read the ID and name fields in each row. This will lead to a significant increase in the amount of data to be read by relational databases, and also requires a series of offset calculations when accessing the required data. Moreover, the example given above is just the simplest table. If the table contains dozens of columns, the amount of data read will increase by tens of times, and the calculation of offset will become more complex.

To solve this problem, we can save the data in a column together:



And this is the core idea of Cassandra, which is a column-based database: record data in data files according to columns, in order to obtain better request and traversal efficiency.

Chapter One

Core Code and Interpretation

I. Save the Prediction Model

create_model_1.py

```
#import modules
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

#import data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = -tf.reduce_sum(y_*tf.log(y))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)

init_op = tf.initialize_all_variables()
saver = tf.train.Saver()


# Train the model and save the model to disk as a model.ckpt file
# file is stored in the same directory as this python script is started
"""
The use of 'with tf.Session() as sess:' is taken from the Tensor flow documentation
on on saving and restoring variables.
https://www.tensorflow.org/versions/master/how_tos/variables/index.html
"""
with tf.Session() as sess:
```

```
    sess.run(init_op)
    for i in range(1000):
        batch_xs, batch_ys = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

    save_path = saver.save(sess, "model.ckpt")
    print ("Model saved in file: ", save_path)
```

After the code is run, you will get the “model.ckpt” , and put them in the same directory with the “predict_1.py” :

II. Prepare Standard Pictures

imageprepare(argv)

```
def imageprepare(argv):
    """
    This function returns the pixel values.
    The imput is a png file location.
    """
    im = Image.open(argv).convert('L')
    width = float(im.size[0])
    height = float(im.size[1])
    newImage = Image.new('L', (28, 28), (255)) #creates white canvas of 28x28 pixels

    if width > height: #check which dimension is bigger
        #Width is bigger. Width becomes 20 pixels.
        nheight = int(round((20.0/width*height),0)) #resize height according to ratio width
        if (nheight == 0): #rare case but minimum is 1 pixel
            nheight = 1
        # resize and sharpen
```

```
            img = im.resize((20,nheight), Image.ANTIALIAS).filter(ImageFilter.SHARP
EN)
            wtop = int(round(((28 - nheight)/2),0)) #caculate horizontal pozition
            newImage.paste(img, (4, wtop)) #paste resized image on white canvas
        else:
            #Height is bigger. Heigth becomes 20 pixels.
            nwidth = int(round((20.0/height*width),0)) #resize width according to r
atio height
            if (nwidth == 0): #rare case but minimum is 1 pixel
                nwidth = 1
             # resize and sharpen
            img = im.resize((nwidth,20), Image.ANTIALIAS).filter(ImageFilter.SHARPE
N)
            wleft = int(round(((28 - nwidth)/2),0)) #caculate vertical pozition
            newImage.paste(img, (wleft, 4)) #paste resized image on white canvas

        #newImage.save("sample.png")

        tv = list(newImage.getdata()) #get pixel values

        #normalize pixels to 0 and 1. 0 is pure white, 1 is pure black.
        tva = [ (255-x)*1.0/255.0 for x in tv]
        return tva
        #print(tva)
```

The image is transformed into a standardized matrix and returned as a parameter.

III. Predict the Matrix

predictint(imvalue)

```
    def predictint(imvalue):
        """
        This function returns the predicted integer.
        The input is the pixel values from the imageprepare() function.
        """

        # Define the model (same as when creating the model file)
        x = tf.placeholder(tf.float32, [None, 784])
        W = tf.Variable(tf.zeros([784, 10]))
        b = tf.Variable(tf.zeros([10]))
        y = tf.nn.softmax(tf.matmul(x, W) + b)
```

```

    init_op = tf.global_variables_initializer()
    saver = tf.train.Saver()

    """
    Load the model.ckpt file
    file is stored in the same directory as this python script is started
    Use the model to predict the integer. Integer is returend as list.

    Based on the documentatoin at
    https://www.tensorflow.org/versions/master/how_tos/variables/index.html
    """
    with tf.Session() as sess:
        sess.run(init_op)
        saver.restore(sess, app.root_path + "/model.ckpt")
        #print ("Model restored.")

        prediction=tf.argmax(y,1)
        return prediction.eval(feed_dict={x: [imvalue]}, session=sess)
```

The input matrix is predicted, and return a list, in which the predicted values are ranked from high to low probability are stored.

IV. Predict the Picture

predict(argv)

```
def predict(argv):
    """
    Main function.
    """
    imvalue = imageprepare(argv)
    predint = predictint(imvalue)
    return predint[0] #first value in list
```

Call the previous two functions and return the most likely predictions.

V. Create KEYSPACE mnist and TABLE mnist_predict

creatKeySpace()

```
KEYSPACE = "mnist"

def createKeySpace():
    """
    The container of Cassandra is located at '172.17.0.2:9042'.
    the program acesses the Cassandra container and records information in the Cassandra.
    """
    # connect to the container
    cluster = Cluster(contact_points=['172.17.0.2'],port=9042)
    session = cluster.connect()

    log.info("Creating keyspace...")
    try:
        # create keyspace mnist
        session.execute("""
           CREATE KEYSPACE %s
           WITH replication = { 'class': 'SimpleStrategy', 'replication_factor' : '1' }
           """ % KEYSPACE)

        log.info("setting keyspace...")
        session.set_keyspace(KEYSPACE)

        # create table mnist_predict
        log.info("creating table...")
        session.execute("""
           CREATE TABLE mnist_predict (
               file_name text,
               upload_time timestamp,
               prediction int,
               PRIMARY KEY (file_name)
           )
           """)

    except Exception as e:
       log.error("Unable to create keyspace")
       log.error(e)
```

```

createKeySpace();
```

VI. Insert Data

insertData(name, time, prediction)

```
def insertData(name, time, prediction):
    """
    Insert data to the table.
    """
    # access to the keyspace mnist
    cluster = Cluster(contact_points=['172.17.0.2'],port=9042)
    session = cluster.connect("mnist")

    try:
        # insert data
        log.info("insertData...")
        session.execute("""
                INSERT INTO mnist_predict (file_name, upload_time, prediction)
                VALUES (%s, %s, %s);
            """ % (name, time, prediction))

    except Exception as e:
        log.error("Unable to insert data")
        log.error(e)
```

VII. Secure Standard Format

allowed_file(filename)

```
def allowed_file(filename):
    # judge whether the user submits the image format
    return '.' in filename and filename.rsplit('.', 1)[1] in ALLOWED_EXTENSIONS
```

VIII. The Main Function

```
@app.route("/mnist", methods=["POST"])
def mnist():
    """
    when users submit pictures to '0.0.0.0:8000/mnist',
    the program retruns users predictions and records them on Cassandra
    """
    req_time = datetime.datetime.now()

    if request.method == "POST":
        file = request.files['file']
        if file and allowed_file(file.filename):

            # get the upload time, the filename, the filepath and the prediction
            upload_filename = secure_filename((file).filename)
            save_filename = str(req_time).rsplit('.',1)[0] + ' ' + upload_filename
            save_filepath = os.path.join(app.root_path, save_filename)
            file.save(save_filepath)
            mnist_result = str(predict(save_filepath))

            insert_filename = '\'' + upload_filename + '\''
            insert_time = '\'' + str(req_time).rsplit('.',1)[0] + '\''

            # insert data to the Cassandra
            insertData(insert_filename, insert_time, mnist_result)

    # return the user with the information
    return ("%s%s%s%s%s%s%s%s%s" % ("Upload File Name: ", upload_filename, "\n",
                                    "Upload Time: ", req_time, "\n",
                                    "Prediction: ", mnist_result, "\n"))

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=80)
```

Chapter Two

Actual Operation

I have already uploaded the image to the public repository, you can pull the image:

```
$ docker pull nickynz/mnist
```

Meantime, you need to pull the Cassandra image:

```
$ docker pull cassandra
```

You can find them by:

```
$ docker ps
```

```
(base) 192:~ yinianzhang$ docker images
REPOSITORY          TAG          IMAGE ID          CREATED          SIZE

nickynz/mnist       latest       c2fd506eb36f      10 hours ago     782MB

cassandra           latest       a05e8a072b59      4 weeks ago      323MB
```

Run the Cassandra:

```
$ docker run --name some-cassandra -p 9042:9042 -d cassandra:latest
```

```
(base) 192:~ yinianzhang$ docker run --name ynz-cassandra -p 9042:9042 -d cassandra:latest
31bca1b301b4305c6bbcc692585ec64c7e6e1b2ac40026a242713f184555e80d
```

Run the nickynz/mnist:

```
$ docker run -p 4000:80 nickynz/mnist
```

```
2019-06-11 10:36:56,071 [WARNING] cassandra.cluster: Cluster.__init__ called with contact_points
specified, but no load_balancing_policy. In the next major version, this will raise an error; ple
ase specify a load-balancing policy. (contact_points = ['172.17.0.2'], lbp = None)
2019-06-11 10:36:56,194 [INFO] cassandra.policies: Using datacenter 'datacenter1' for DCAwareRoun
dRobinPolicy (via host '172.17.0.2:9042'); if incorrect, please specify a local_dc to the constru
ctor, or limit contact points to local cluster nodes
2019-06-11 10:36:56,249 [INFO] root: Creating keyspace...
2019-06-11 10:36:56,317 [INFO] root: setting keyspace...
2019-06-11 10:36:56,320 [INFO] root: creating table...
 * Serving Flask app "predict_1" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
2019-06-11 10:36:56,485 [INFO] werkzeug:  * Running on http://0.0.0.0:80/ (Press CTRL+C to quit)
```

If you encounter an error, try typing this statement again (Maybe it’s because the input interval between the above two statements is too short).

Now you can post the picture which you want to predict by this(you need to open another terminal to enter this code):

```
$ curl -F "file=@/path/to/your/image/1.jpg" http://0.0.0.0:4000/mnist
```

```
(base) 192:~ yinianzhang$ curl -F "file=@/Users/yinianzhang/Desktop/MIT/1.jpeg" http://0.0.0.0:4000/mnist
Upload File Name: 1.jpeg
Upload Time: 2019-06-11 12:29:06.714920
Prediction: 7

2019-06-11 12:29:06,926 [INFO] root: insertData...
2019-06-11 12:29:06,935 [INFO] werkzeug: 172.17.0.1 - - [11/Jun/2019 12:29:06] "POST /mnist HTTP/
1.1" 200 -
```

And the program will upload the information to Cassandra. You can enter the Cassandra container by this:

```
$ docker ps
```

```
(base) 192:~ yinianzhang$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
bc39c13c98c1        nickynz/mnist       "python predict_1.py"    2 hours ago
31bca1b301b4        cassandra:latest    "docker-entrypoint.s…"   2 hours ago
```

```
$ docker exec -it <container id> bash
/# cqlsh
```

```
(base) 192:~ yinianzhang$ docker exec -it 31bca1b301b4 bash
root@31bca1b301b4:/# cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.4 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh>
```

```
cqlsh> use mnist;
cqlsh:mnist> Select * from mnist_predict;
```

```
cqlsh> use mnist;
cqlsh:mnist> Select * from mnist_predict;

 file_name | prediction | upload_time
-----------+------------+---------------------------------
    1.jpeg |          7 | 2019-06-11 12:29:06.000000+0000

(1 rows)
```

The insert data will be recorded in the table mnist_predict.

Conclusion

This project helps me understand the process of Engineering Research better. In terms of technology, I learned about big data science, in-depth learning, virtualization technology, container technology, NoSQL and other specific research knowledge. The main goal of the project is to let me familiar with knowledge and technology such as MNIST, Docker, Cassandra and teach us how to communicate between code and container through Flask. In this process, I basically understand the significance of big data. By making an image, users can download the image and run it locally, and finally store the relevant information, so as to realize the collection and application of big data.