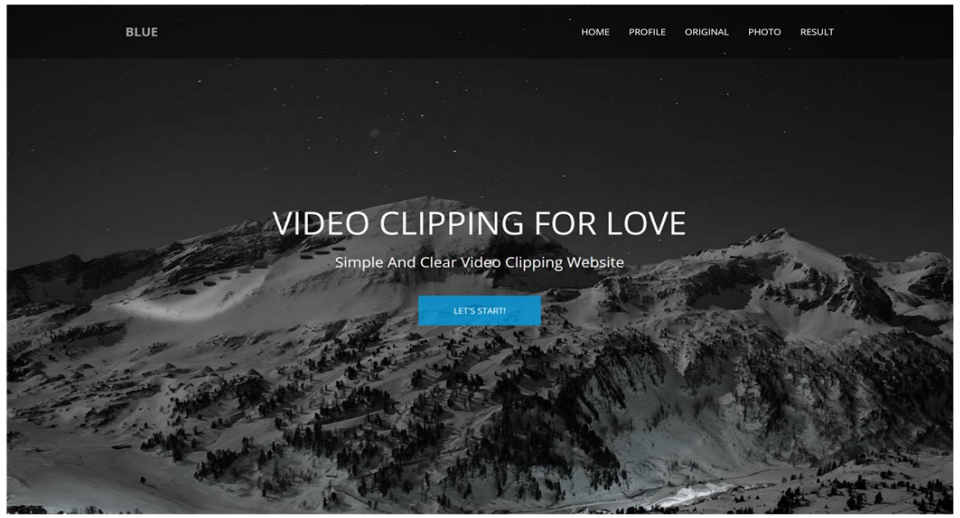
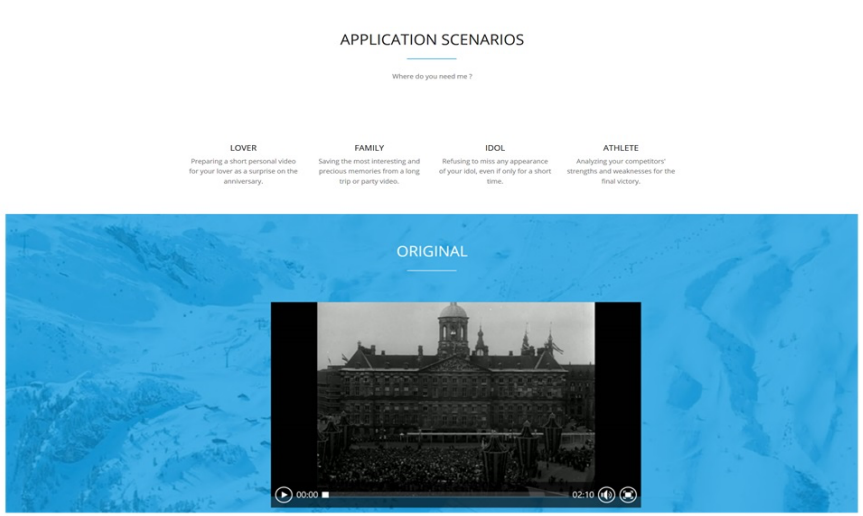
VIDEO CLIPPING FOR LOVE ❤️

Run Demo Video Demo Result Demo on Web	https://drive.google.com/file/d/1kzjw7i9d_IqJz0m4ivGBbPAqdMlWGDS-/view?usp=sharing https://drive.google.com/file/d/1Nn70liQBIiITLZZBWzSkZ8MRn5IxYqeC/view?usp=sharing http://202.45.128.135:10889/view/tomcat/website_1/index.html (Password: student)	
How to Run Project Demo	1. Login into hduser@gpu20 (Password: student) 2. Check all images we could use, which is stored in HDFS: `hdfs dfs -ls /David/img2/` (Around 21, 000 images here) 3. At the home path: /home/gpu20,`run /opt/spark-2.4.0-bin-hadoop2.7/bin/spark-submit --master yarn test1.py` 4. Check records on http://202.45.128.135:20220 and http://202.45.128.135:20620 5. Check our web for result representation (open with Safari or Microsoft Edge) http://202.45.128.135:10889/view/tomcat/website_1/index.html	
System Usage	Data Storage	More than 2.5 GB data are used
	Resource of Cluster	more than 52GB virtual memory and 9 virtual CPU cores are used
	Spark	6 executors are parallel during demo on spark. (We updated our code and make it as 7 executors finally)
Job Records Webs	Ganglia	http://202.45.128.135:10089/ganglia/
	Hadoop Overview	http://202.45.128.135:20120
	Hadoop History	http://202.45.128.135:20220 (During project running)
	Spark History	http://202.45.128.135:20620

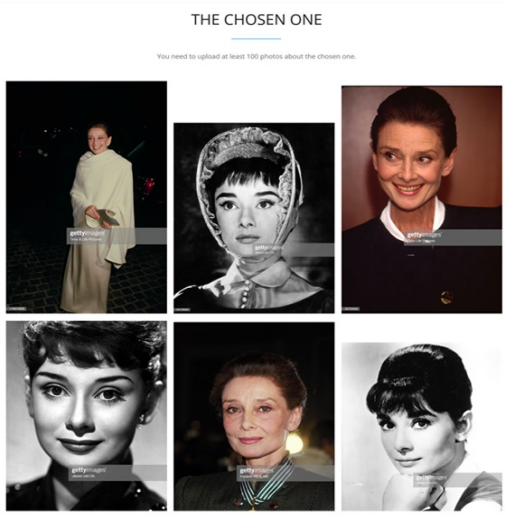
Web UI Design



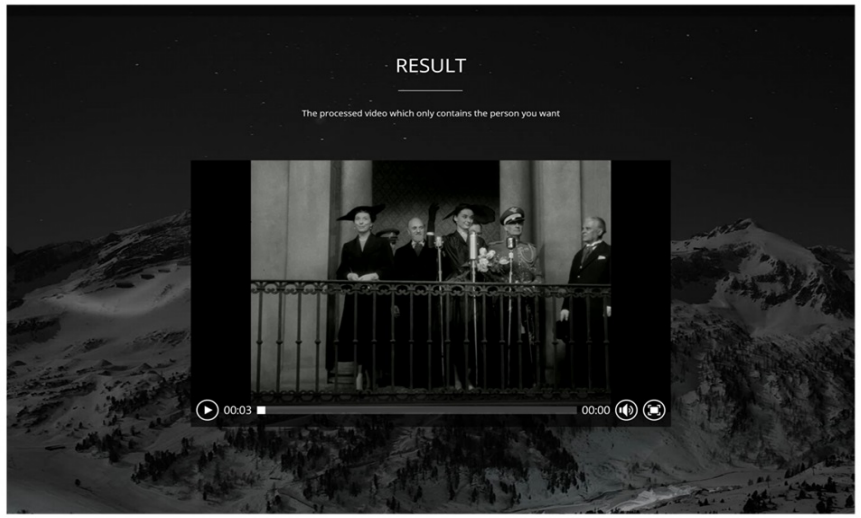
Front Page



The original video to process and suggested application scenarios

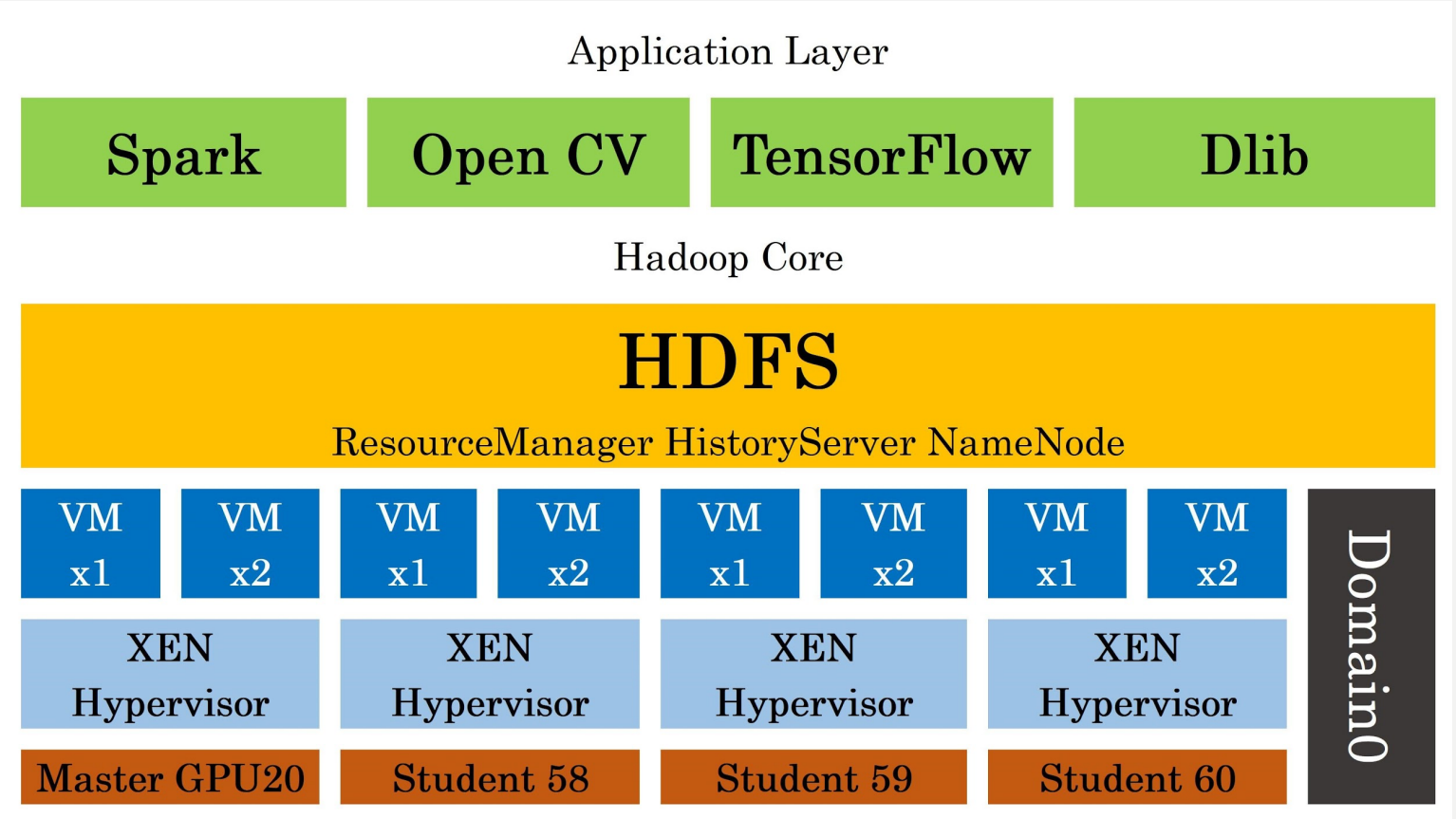


Several photos to train the Face Recognition



The clipped videos

Overall System Configuration

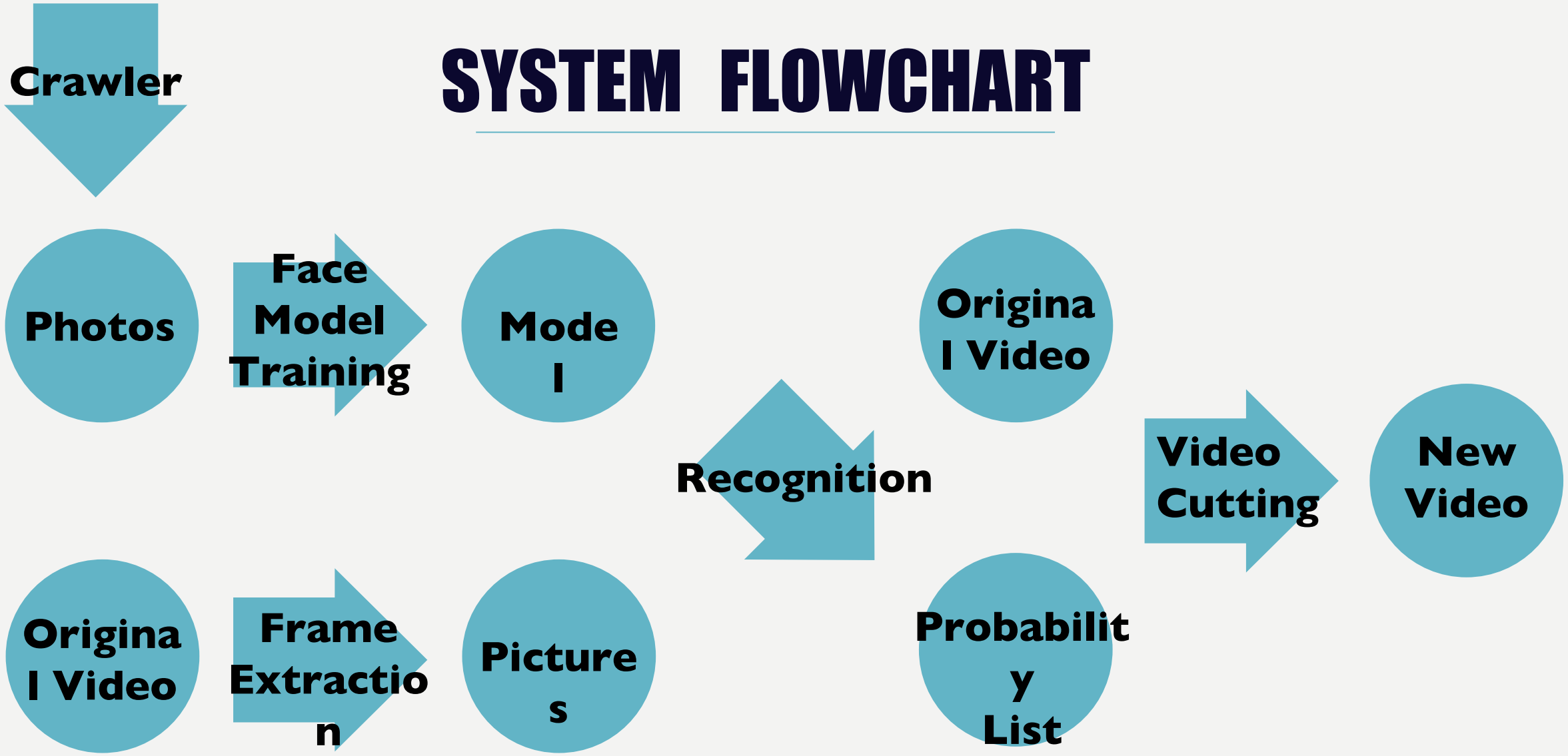


System components: Spark + OpenCV + TensorFlow
OpenCV-Python is a library of Python designed to solve computer vision problems.
Dlib is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software

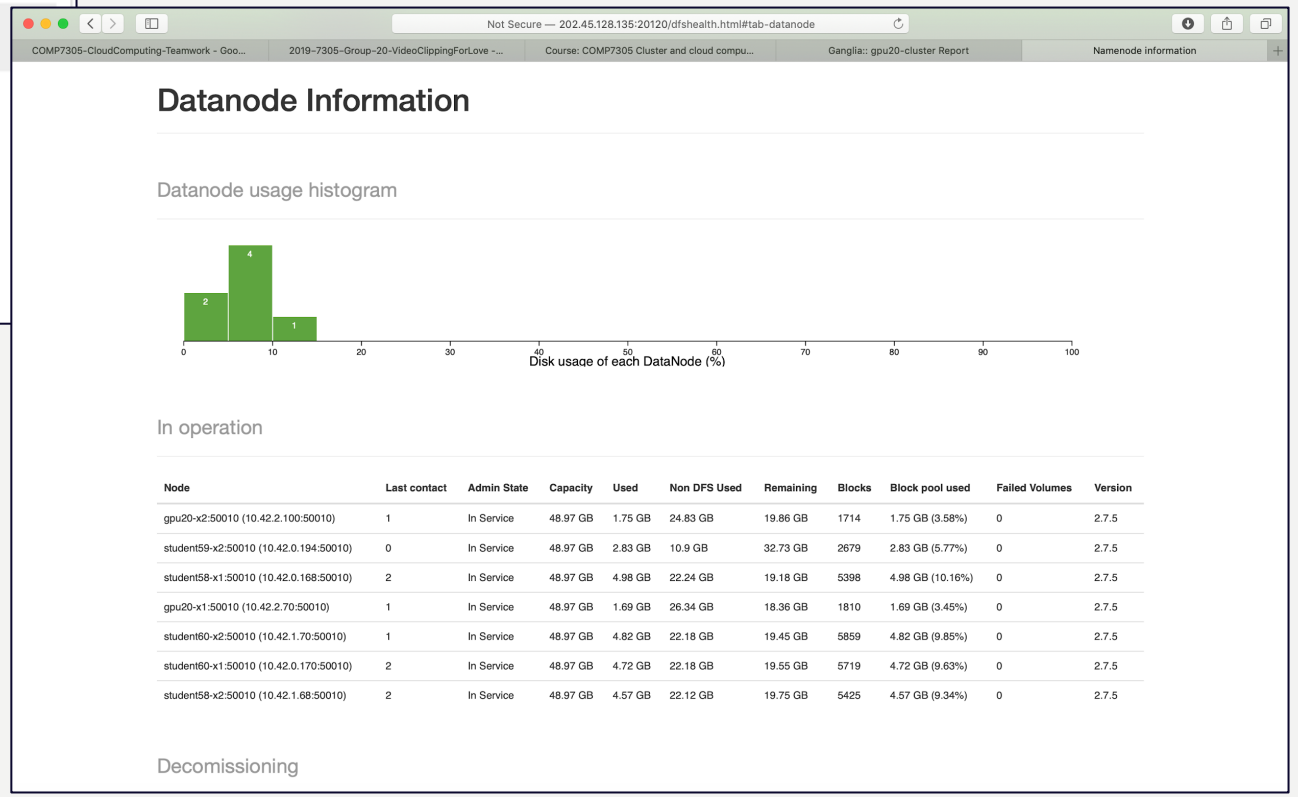
Cluster Resources

Phys. Machines		GPU20			Student58			Student59			Student60		
CPU model		Intel(R) Core(TM) i5-8400 CPU @ 2.80GHz			Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz			Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz			Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz		
# of cores		2			2			2			2		
RAM(GB)		16			16			16			16		
Dist storage(GB)		250			250			250			250		
Virtual Machine	Name	Dom0	X1	X2	Dom0	X1	X2	Dom0	X1	X2	Dom0	X1	X2
	vCPU	2	1	1	2	1	1	2	1	1	2	1	1
	Mem(GB)	16	4	4	16	4	4	16	4	4	16	4	4
	Swap(GB)	0	8	8	2	8	8	2	8	8	2	8	8
	MAC	04:92:26:4c:9f:b5	52:54:01:1f:43:5b	52:54:01:76:be:e2	54:BF:64:8B:3F:38	52:54:00:5f:b6:02	52:54:00:54:60:1a	54:BF:64:8B:3D:DE	52:54:00:6c:90:91	52:54:00:64:11:f0	54:BF:64:8A:D0:C5	52:54:00:1c:3d:e1	52:54:00:6c:55:6c
	IP	10.42.2.40	10.42.2.70	10.42.2.100	10.42.0.68	10.42.0.168	10.42.1.68	10.42.0.69	10.42.0.169	10.42.1.69	10.42.0.70	10.42.0.170	10.42.1.70
Role	ganglia	master	slave	slave	none	slave	slave	none	slave	slave	none	slave	slave
	HDFS	namenode	datanode	datanode	none	datanode	datanode	none	datanode	datanode	none	datanode	datanode
	Yarn	resource manager	NM	NM	none	NM	NM	none	NM	NM	none	NM	NM

SYSTEM FLOWCHART



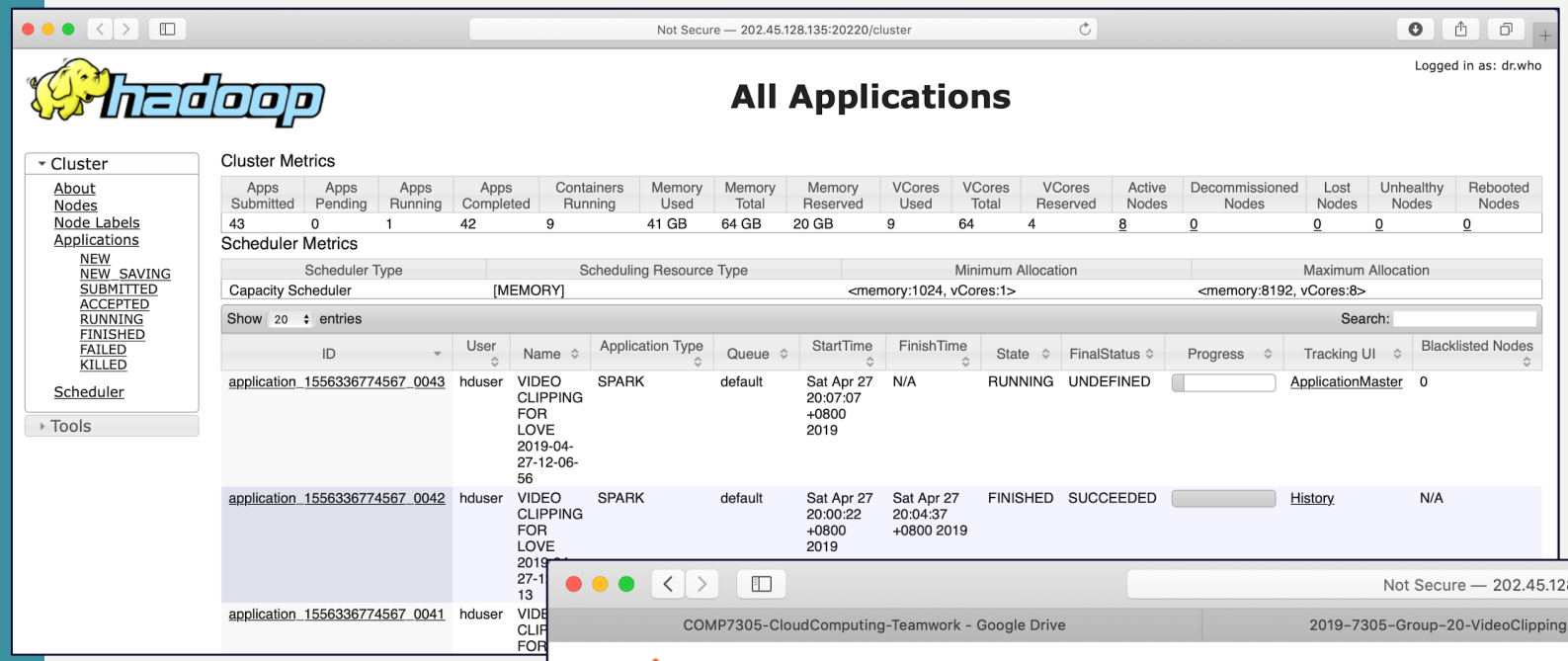
Software installation

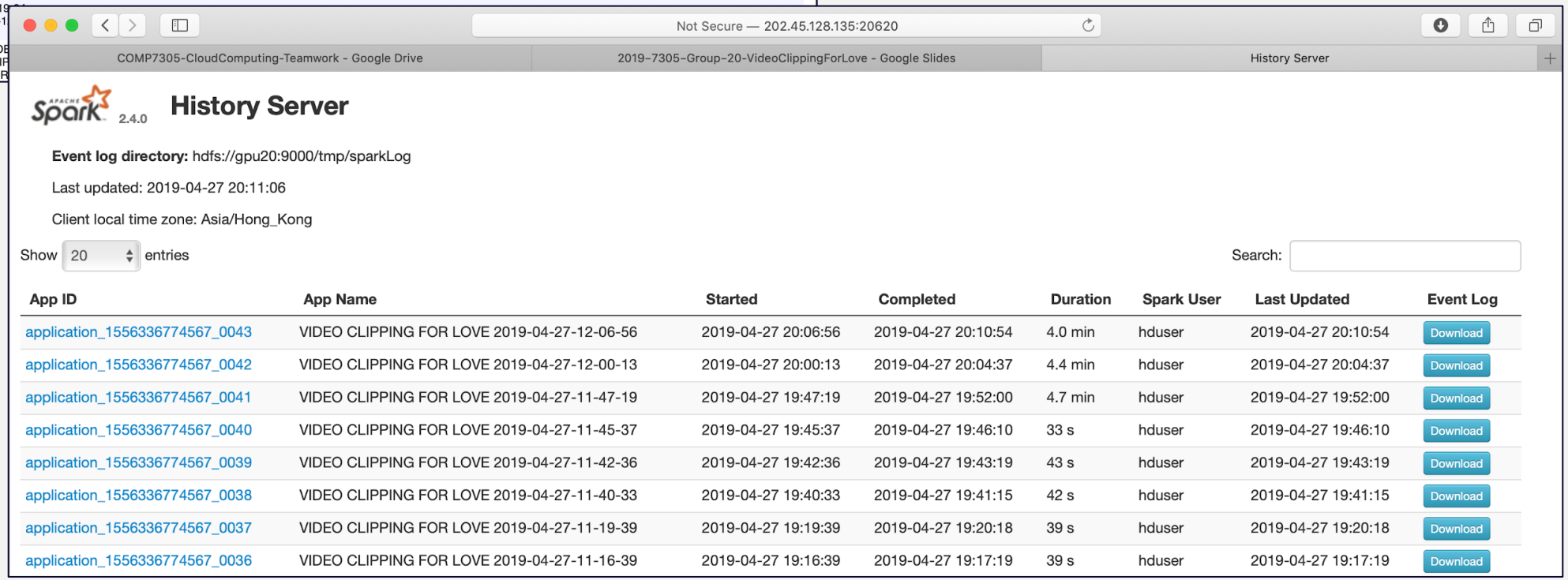


http://202.45.128.135:10089/ganglia/
http://202.45.128.135:20120/

Software installation

http://202.45.128.135:20220
http://202.45.128.135:20620





Crawl Images Data Set

```
#-*- coding:utf-8    -*-
import urllib
from bs4 import BeautifulSoup
import os
import re
import datetime

url_head = 'https://www.gettyimages.com'

def get_page_url(page_num):
    # page_url = 'https://www.gettyimages.com/photos/audrey-hepburn?
mediatype=photography&page=1&phrase=audrey%20hepburn&sort=mostpopular&family=editorial'
    left = 'https://www.gettyimages.com/photos/audrey-hepburn?mediatype=photography&page='
    num = str(page_num)
    right = '&phrase=audrey%20hepburn&sort=mostpopular&family=editorial'
    return left+num+right

def get_detail_urls(page_url, url_head):
    soup = BeautifulSoup(urllib.request.urlopen(page_url), features='lxml')
    items = soup.find_all('a',class_='asset-link draggable')
    hrefs = [url_head+item['href'] for item in items]
    return hrefs
def download_img(detail_url, dir='/Users/david/Downloads/HB'):
    soup = BeautifulSoup(urllib.request.urlopen(detail_url), features='lxml')
    item = soup.find('div', class_='zoom-wrapper')
    soup = BeautifulSoup(urllib.request.urlopen(detail_url), features='lxml')
    item = soup.find('div', class_='zoom-wrapper')
    img_url = item.select('img')[0]['src']
    tmp = item.select('img')[0]['title'].strip()
    img_title = re.split(':|,|',tmp)[1].strip()
    # urllib.request.urlretrieve(img_url, os.path.join(dir,img_title+'.jpg'))
    name = img_title+datetime.datetime.now().strftime("%Y-%m-%d-%H-%M-%S")+'.jpg'
    f = open(os.path.join(dir,name),'wb')
    f.write(urllib.request.urlopen(img_url).read())
    f.close()
    return name


if __name__ == "__main__":
    dir = '/Users/david/Downloads/HB'
    if not os.path.isdir(dir):
        os.mkdir(dir)
    # for each page
    for i in range(4,60):
        page_num = i+1
```

We use Python to crawl Miss Herben's images from www.gettyimage.com and store the files in GPU20
~/FaceRecognition-tensorflow-master/HB/

There are 3, 000 images we totally got, and more than 300 images are used for training our face-recognition model.

Test Video Data Set

-rwxr-xr-x	hduser	supergroup	236.34 KB	4/18/2019, 10:33:32 PM	2	64 MB	000056.png
-rwxr-xr-x	hduser	supergroup	266.29 KB	4/18/2019, 10:33:33 PM	2	64 MB	000057.png
-rwxr-xr-x	hduser	supergroup	301.78 KB	4/18/2019, 10:33:33 PM	2	64 MB	000058.png
-rwxr-xr-x	hduser	supergroup	328.86 KB	4/18/2019, 10:33:33 PM	2	64 MB	000059.png
-rwxr-xr-x	hduser	supergroup	362.52 KB	4/18/2019, 10:33:33 PM	2	64 MB	000060.png
-rwxr-xr-x	hduser	supergroup	701.66 KB	4/18/2019, 11:01:33 PM	2	64 MB	000100.png
-rwxr-xr-x	hduser	supergroup	697.81 KB	4/18/2019, 11:01:33 PM	2	64 MB	000101.png
-rwxr-xr-x	hduser	supergroup	706.92 KB	4/18/2019, 11:01:34 PM	2	64 MB	000102.png
-rwxr-xr-x	hduser	supergroup	708.53 KB	4/18/2019, 11:01:35 PM	2	64 MB	000103.png
-rwxr-xr-x	hduser	supergroup	713.39 KB	4/18/2019, 11:01:35 PM	2	64 MB	000104.png
-rwxr-xr-x	hduser	supergroup	710.07 KB	4/18/2019, 11:01:36 PM	2	64 MB	000105.png
-rwxr-xr-x	hduser	supergroup	711.24 KB	4/18/2019, 11:01:37 PM	2	64 MB	000106.png
-rwxr-xr-x	hduser	supergroup	703.06 KB	4/18/2019, 11:01:37 PM	2	64 MB	000107.png
-rwxr-xr-x	hduser	supergroup	702.07 KB	4/18/2019, 11:01:38 PM	2	64 MB	000108.png
-rwxr-xr-x	hduser	supergroup	702.53 KB	4/18/2019, 11:01:38 PM	2	64 MB	000109.png
-rwxr-xr-x	hduser	supergroup	705.81 KB	4/18/2019, 11:01:38 PM	2	64 MB	000110.png
-rwxr-xr-x	hduser	supergroup	706.24 KB	4/18/2019, 11:01:40 PM	2	64 MB	000111.png

-rwxr-xr-x	hduser	supergroup	428.89 KB	4/18/2019, 11:18:09 PM	2	64 MB	022985.png
-rwxr-xr-x	hduser	supergroup	449.84 KB	4/18/2019, 11:18:09 PM	2	64 MB	022986.png
-rwxr-xr-x	hduser	supergroup	468.73 KB	4/18/2019, 11:18:10 PM	2	64 MB	022987.png
-rwxr-xr-x	hduser	supergroup	483.17 KB	4/18/2019, 11:18:10 PM	2	64 MB	022988.png
-rwxr-xr-x	hduser	supergroup	492.25 KB	4/18/2019, 11:18:10 PM	2	64 MB	022989.png
-rwxr-xr-x	hduser	supergroup	495.53 KB	4/18/2019, 11:18:10 PM	2	64 MB	022990.png
-rwxr-xr-x	hduser	supergroup	494.43 KB	4/18/2019, 11:18:11 PM	2	64 MB	022991.png
-rwxr-xr-x	hduser	supergroup	468.81 KB	4/18/2019, 11:18:12 PM	2	64 MB	022992.png
-rwxr-xr-x	hduser	supergroup	454.08 KB	4/18/2019, 11:18:12 PM	2	64 MB	022993.png
-rwxr-xr-x	hduser	supergroup	463.18 KB	4/18/2019, 11:18:12 PM	2	64 MB	022994.png
-rwxr-xr-x	hduser	supergroup	480.62 KB	4/18/2019, 11:18:12 PM	2	64 MB	022995.png
-rwxr-xr-x	hduser	supergroup	495.81 KB	4/18/2019, 11:18:13 PM	2	64 MB	022996.png
-rwxr-xr-x	hduser	supergroup	506.54 KB	4/18/2019, 11:18:13 PM	2	64 MB	022997.png
-rwxr-xr-x	hduser	supergroup	513.53 KB	4/18/2019, 11:18:13 PM	2	64 MB	022998.png
-rwxr-xr-x	hduser	supergroup	513.9 KB	4/18/2019, 11:18:13 PM	2	64 MB	022999.png
-rwxr-xr-x	hduser	supergroup	512.16 KB	4/18/2019, 11:18:14 PM	2	64 MB	023000.png

The second dataset is the video, which is directly downloaded from a documentary movie website. We split the video into single frame and the sample video here is around 16 minutes and the frame per second is 24, therefore around 500KB * 21, 000 = 10.5 GB store in HDFS, where 2.5 GB is used for further testing in a single, 2 minutes video job.

Critical setting

```
<configuration>
        <property>
                <name>mapreduce.framework.name</name>
                <value>yarn</value>
        </property>
        <property>
                <name>mapreduce.map.memory.mb</name>
                <value>4096</value>
        </property>
        <property>
                <name>mapreduce.reduce.memory.mb</name>
                <value>4096</value>
        </property>

</configuration>
```

mapred-site.xml

```
spark.master                    spark://gpu20:7077
spark.serializer                org.apache.spark.serializer.KryoSerializer
spark.executor.instances        10
spark.eventLog.enabled          true
spark.eventLog.dir              hdfs://gpu20:9000/tmp/sparkLog
spark.history.fs.logDirectory hdfs://gpu20:9000/tmp/sparkLog
spark.driver.memory             2g
spark.executor.memory           4g
```

spark-defaults.conf

Key Features Summary

/home/gpu20/test1.py

```
main():
sc = SparkContext(appName="VIDEO CLIPPING FOR LOVE "+timestring)
sc = SparkContext(appName="VIDEO CLIPPING FOR LOVE")
image1 = sc.binaryFiles("hdfs://gpu20:9000/David/img2/003[0-3]*")
image2 = sc.binaryFiles("hdfs://gpu20:9000/David/img2/003[4-6]*")
image3 = sc.binaryFiles("hdfs://gpu20:9000/David/img2/003[7-9]*")
image4 = sc.binaryFiles("hdfs://gpu20:9000/David/img2/004[0-3]*")
image5 = sc.binaryFiles("hdfs://gpu20:9000/David/img2/004[4-6]*")
image6 = sc.binaryFiles("hdfs://gpu20:9000/David/img2/004[7-9]*")
image7 = sc.binaryFiles("hdfs://gpu20:9000/David/img2/005[0-3]*")
image8 = sc.binaryFiles("hdfs://gpu20:9000/David/img2/005[4-6]*")
image9 = sc.binaryFiles("hdfs://gpu20:9000/David/img2/005[7-9]*")

#read the frames to 9 rdd. The reason of using so many rdd is that we found that when read too many pictures, the order in the rdd is disordered.
```

Key Features Summary

/home/gpu20/test1.py

```
main():
images =
(((((((image1.union(image2)).union(image3)).union(image4)).union(image5)).union(ima
ge6)).union(image7)).union(image8)).union(image9)
#add all images in rdds to one rdd

image_to_array = lambda rawdata: np.asarray(Image.open(io.BytesIO(rawdata)))
#this function transform the data in rdd from binary file to array

imagerdd = images.values().map(image_to_array)
#transform the data in rdd from binary file to array
```

Key Features Summary

/home/gpu20/test1.py

```
main():
lists = imagerdd.map(get_prob).collect()
#this sentence apply “get_prob” function on the data first. Then it gets the data in rdd.
get_prob(image_df):
#this function uses the already built cnn model based on tensorflow, dlib and opencv.
The input is picture array and the output is the probability of if one frame has Hepburn
```

Key Features Summary

```
"""
This function is aim to cut the long video into a new video
according to the probability of object appearing in per frame
"""
def cutvideo(akk,enterpath,outpath,thr=0.7,nstart=3000):
#arr: The probability of object appearing in per frame
#enterpath: the original video path
#outpath: the result video path
#thr: the threshold value to determine whether an object appears
#nstart:the start frame of the video
    videoCapture = cv2.VideoCapture(enterpath)
    fps = int(videoCapture.get(cv2.CAP_PROP_FPS))
    nfps=videoCapture.get(cv2.CAP_PROP_FRAME_COUNT)
    ak=[]
    n=nstart
    for i in akk:
        if float(i)>thr:
            ak.append(n)
        n+=1
    m=nstart
    ak=buqiarr(ak,5)#Fill in the gaps between frames
    size = (int(videoCapture.get(cv2.CAP_PROP_FRAME_WIDTH)),
        int(videoCapture.get(cv2.CAP_PROP_FRAME_HEIGHT)))
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    videoWriter = cv2.VideoWriter(outpath, fourcc, fps, size)
    for i in ak:
        if i > nfps:
            continue
        videoCapture.set(cv2.CAP_PROP_POS_FRAMES,i)
        ret,frame = videoCapture.read()
        videoWriter.write(frame)
    videoCapture.release()
    videoWriter.release()
```

Reference (Open Source)

- Face recognition using Tensorflow:
  https://github.com/davidsandberg/facenet
  https://www.cnblogs.com/mu---mu/p/FaceRecognition-tensorflow.html
  https://github.com/TuXiaokang/pyseeta
- Machine Learning Library (MLlib) Programming Guide:
  http://spark.apache.org/docs/1.2.1/mllib-guide.html
- Spark Streaming Programming Guide:
  http://spark.apache.org/docs/1.2.1/streaming-programming-guide.html
- Find Face in Video:
  https://github.com/chengstone/FindFaceInVideo
- Locate Face in Pic:
  https://github.com/TuXiaokang/pyseeta
- Training a neural network allows her to recognize me:
  https://www.cnblogs.com/mu---mu/p/FaceRecognition-tensorflow.html

Dataset Used

- Audrey Hepburn Dataset:

We collect data from image database https://www.gettyimages.com

60 images in each page and we get 60 pages. We also clean date manually, removing noise date and resizing all of them. Finally we get 3414 Hepburn's images.

We Also use two movies -- *My Fair Lady* and *Roman Holiday* as our test data.

URL:https://drive.google.com/drive/u/0/folders/1iWEWWp6dRq4rZkRIUWXCVVfbJZpZYtkW

URL:https://pan.baidu.com/s/1fXxhrtSqEbx2ViEx-xwB2A

- Video Captures

We make video captures from video Roman Holiday, storing these images in HDFS, which has 21k images in total.

HDFS Direction: hdfs://gpu20:9000/David/img2/

- Input-Output:

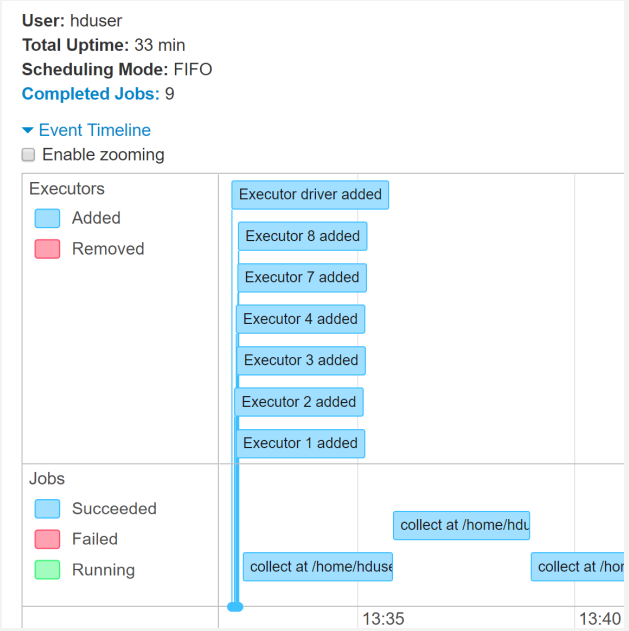
Hepburn Images -(train)-> recognition model -(input video captures)-> edited video(output)

Execution Time Optimization

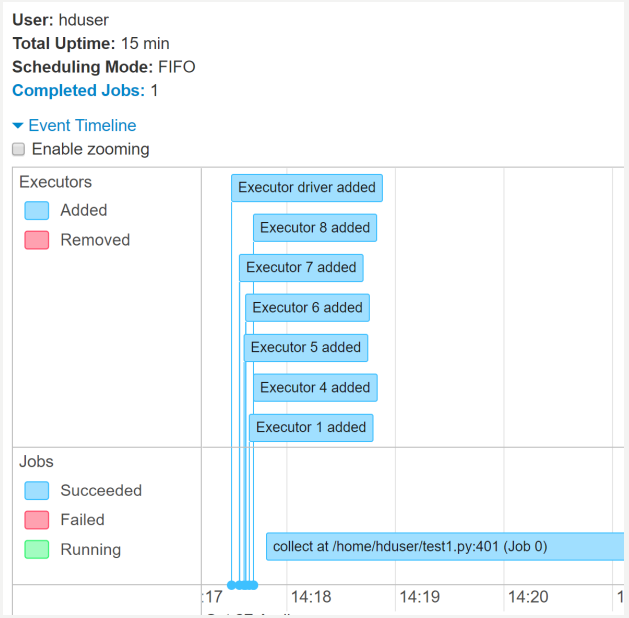
Data Size	Execution Time
2.3 GB (3000 Pics)	33 min

After Optimization →

Data Size	Execution Time
2.3 GB (3000 Pics)	15 min
390 MB (500 Pics)	3.4 min



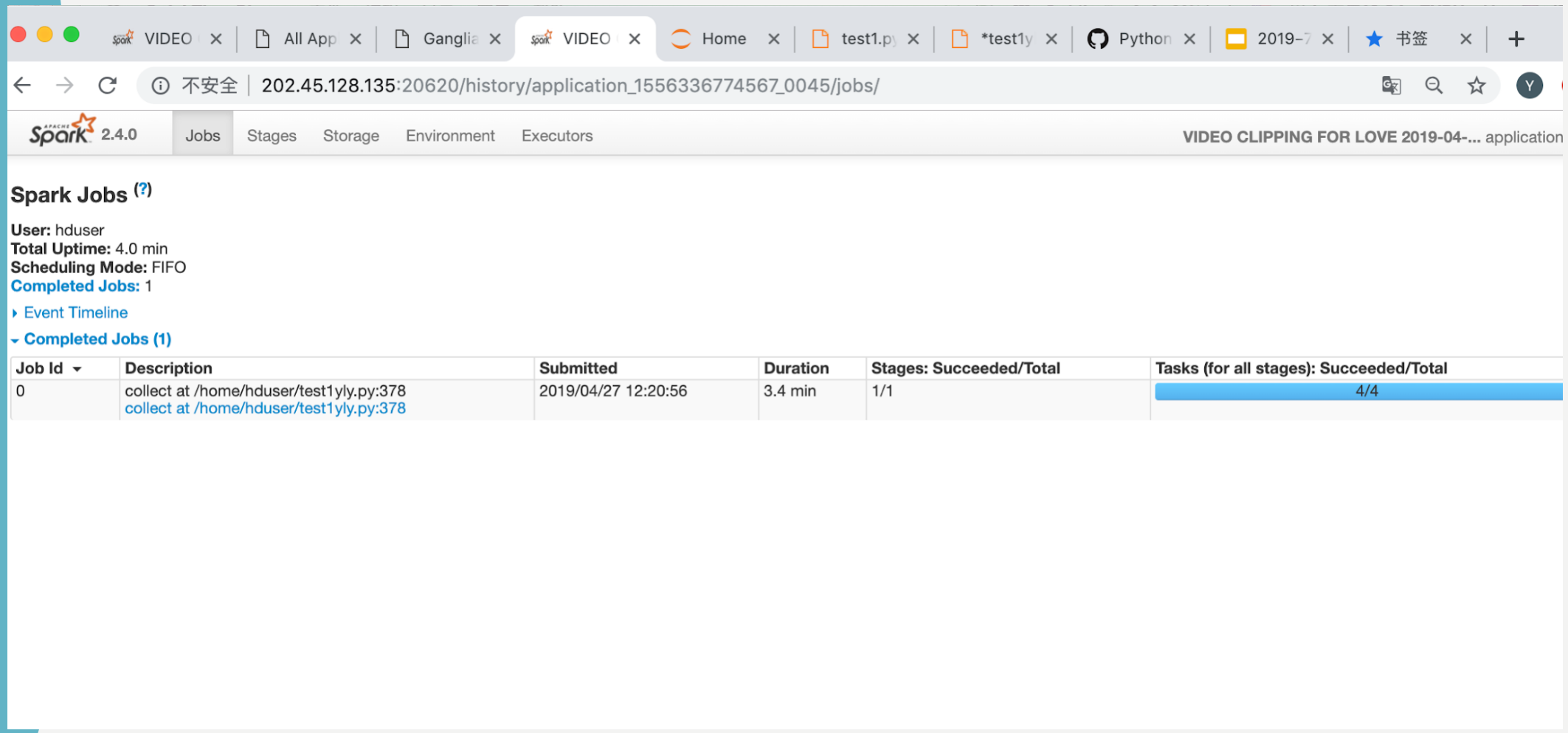
After Optimization →



Configuration settings See P26

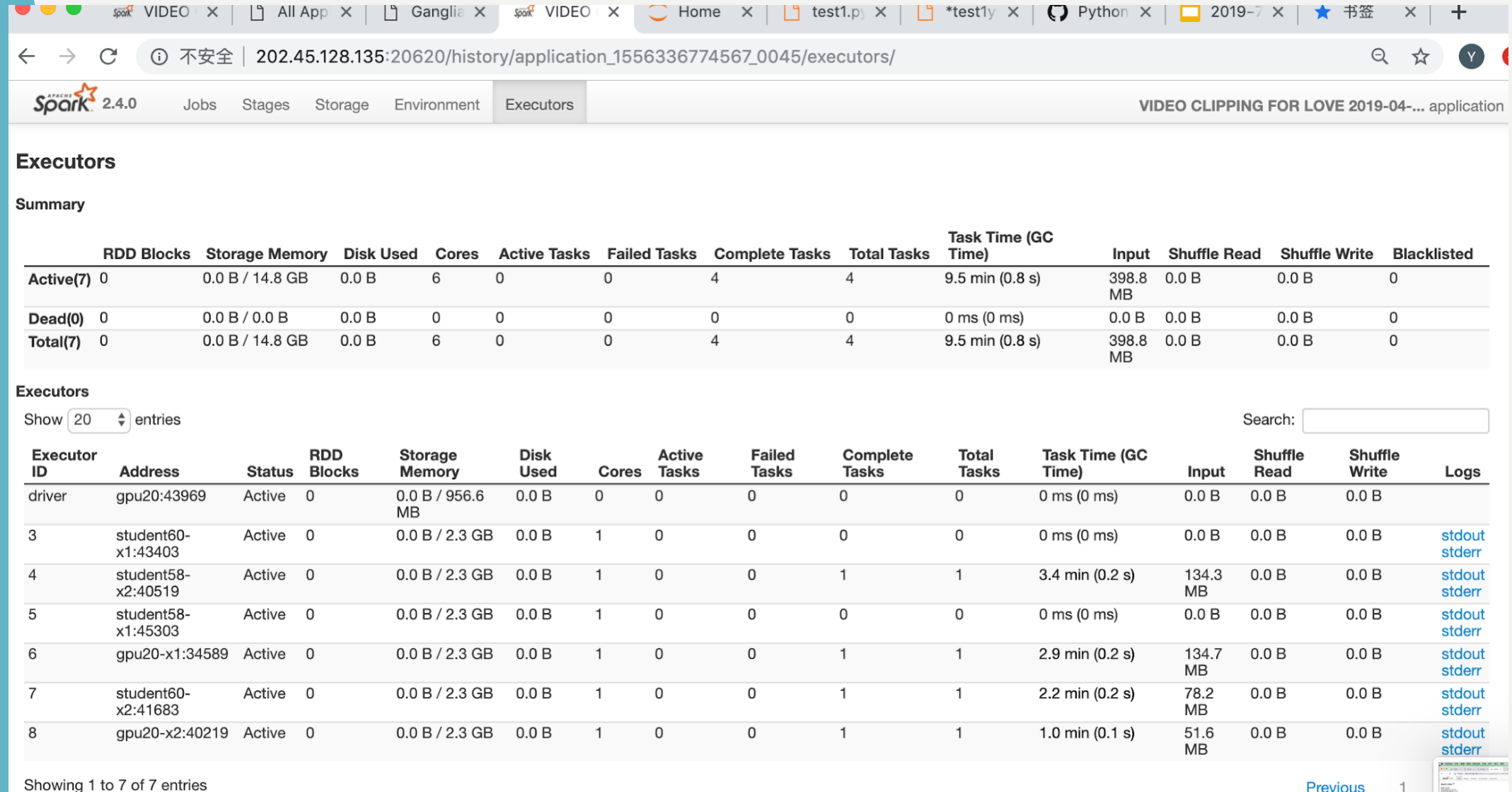
Execution Time

execute 500 images:3.4 min



Execution Time

execute 500 images:398.9 MB



Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Blacklisted
Active(7)	0	0.0 B / 14.8 GB	0.0 B	6	0	0	4	4	9.5 min (0.8 s)	398.8 MB	0.0 B	0.0 B	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0
Total(7)	0	0.0 B / 14.8 GB	0.0 B	6	0	0	4	4	9.5 min (0.8 s)	398.8 MB	0.0 B	0.0 B	0

Executors

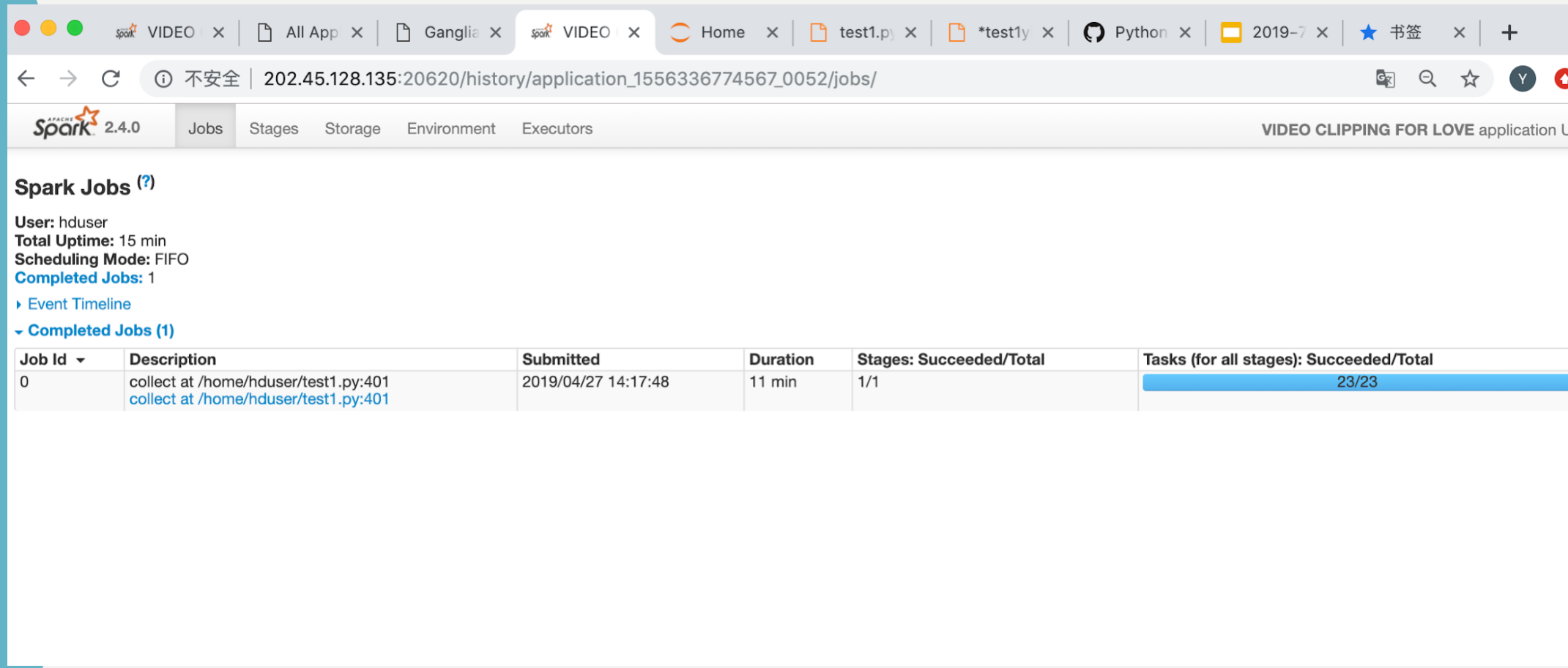
Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs
driver	gpu20:43969	Active	0	0.0 B / 956.6 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	
3	student60-x1:43403	Active	0	0.0 B / 2.3 GB	0.0 B	1	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr
4	student58-x2:40519	Active	0	0.0 B / 2.3 GB	0.0 B	1	0	0	1	1	3.4 min (0.2 s)	134.3 MB	0.0 B	0.0 B	stdout stderr
5	student58-x1:45303	Active	0	0.0 B / 2.3 GB	0.0 B	1	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	stdout stderr
6	gpu20-x1:34589	Active	0	0.0 B / 2.3 GB	0.0 B	1	0	0	1	1	2.9 min (0.2 s)	134.7 MB	0.0 B	0.0 B	stdout stderr
7	student60-x2:41683	Active	0	0.0 B / 2.3 GB	0.0 B	1	0	0	1	1	2.2 min (0.2 s)	78.2 MB	0.0 B	0.0 B	stdout stderr
8	gpu20-x2:40219	Active	0	0.0 B / 2.3 GB	0.0 B	1	0	0	1	1	1.0 min (0.1 s)	51.6 MB	0.0 B	0.0 B	stdout stderr

Showing 1 to 7 of 7 entries Previous 1

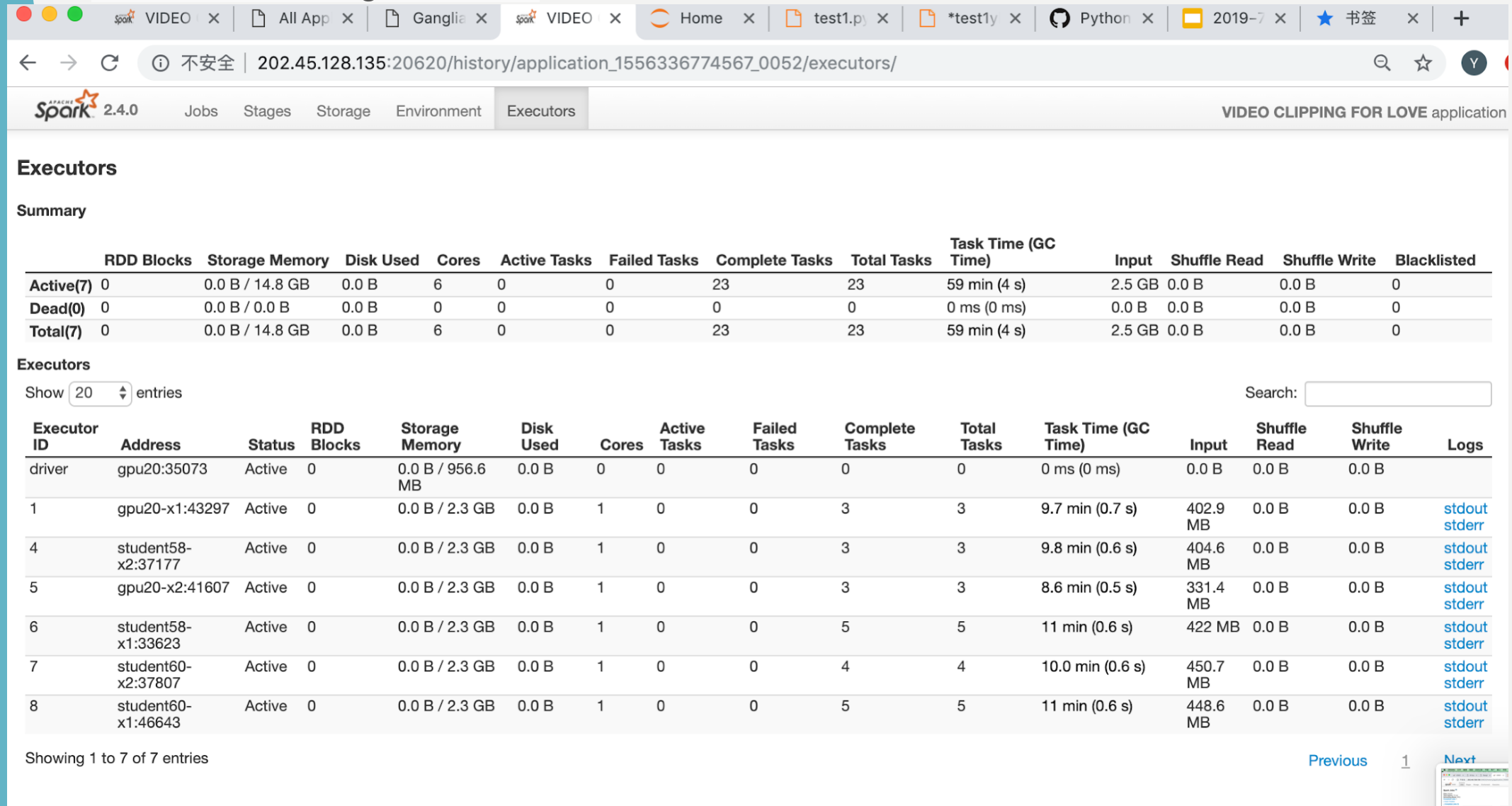
Execution Time

execute 3000 images: 11min



Execution Time

execute 3000 images: 2.5 GB



Executors

Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Blacklisted
Active(7)	0	0.0 B / 14.8 GB	0.0 B	6	0	0	23	23	59 min (4 s)	2.5 GB	0.0 B	0.0 B	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	0
Total(7)	0	0.0 B / 14.8 GB	0.0 B	6	0	0	23	23	59 min (4 s)	2.5 GB	0.0 B	0.0 B	0

Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs
driver	gpu20:35073	Active	0	0.0 B / 956.6 MB	0.0 B	0	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B	
1	gpu20-x1:43297	Active	0	0.0 B / 2.3 GB	0.0 B	1	0	0	3	3	9.7 min (0.7 s)	402.9 MB	0.0 B	0.0 B	stdout stderr
4	student58-x2:37177	Active	0	0.0 B / 2.3 GB	0.0 B	1	0	0	3	3	9.8 min (0.6 s)	404.6 MB	0.0 B	0.0 B	stdout stderr
5	gpu20-x2:41607	Active	0	0.0 B / 2.3 GB	0.0 B	1	0	0	3	3	8.6 min (0.5 s)	331.4 MB	0.0 B	0.0 B	stdout stderr
6	student58-x1:33623	Active	0	0.0 B / 2.3 GB	0.0 B	1	0	0	5	5	11 min (0.6 s)	422 MB	0.0 B	0.0 B	stdout stderr
7	student60-x2:37807	Active	0	0.0 B / 2.3 GB	0.0 B	1	0	0	4	4	10.0 min (0.6 s)	450.7 MB	0.0 B	0.0 B	stdout stderr
8	student60-x1:46643	Active	0	0.0 B / 2.3 GB	0.0 B	1	0	0	5	5	11 min (0.6 s)	448.6 MB	0.0 B	0.0 B	stdout stderr

Showing 1 to 7 of 7 entries Previous 1 Next

Creativity and Business Potential

What the model does

1. Upload sample photos and videos of target face
2. Train photos to extract the features
3. Divide the sample video into frame
4. Select video clips only contain target face

Business Potential and Application

For Couple

- Preparing a short personal video for your lover as a surprise on the anniversary.

For Families

- Saving the most interesting and precious memories from a long trip or party video.

For Fans

- Never miss any appearance of your idol, even if only for a short time.
- Analyzing your competitors' strengths and weaknesses for the final victory.

Key Findings

- Bottleneck:

  VideoCapture by Open-CV, which is not implement in cluster.

- Improvement:

  Use Spark binary read function to read video file in, and then try to read a capture from binary data, which could be executed in parallel process.

Key Findings

The first code is below:

```
#imagerdd1 = imagerdd.map(get_prob).collect()
#list1 = imagerdd1.map(get_prob).collect()
#list2 = imagerdd2.map(get_prob).collect()
#list3 = imagerdd3.map(get_prob).collect()
#list4 = imagerdd4.map(get_prob).collect()
#list5 = imagerdd5.map(get_prob).collect()
#list6 = imagerdd6.map(get_prob).collect()
#list7 = imagerdd7.map(get_prob).collect()
#list8 = imagerdd8.map(get_prob).collect()
#list9 = imagerdd9.map(get_prob).collect()
#lists = list1+list2+list3+list4+list5+list6+list7+list8+list9
```

We found that when read to many pictures the order will be wrong. Thus, we read no more than 400 pictures one time. Then, to execute 3000 pictures, we created 9 rdd and applied function “get_prob” at each rdd and applied collect() at each rdd. However, the execution time was too long, over 30 min.

Key Findings

We considered that when we executed our original code, there was lots of cleaned accumulator like below. Thus, we considered creating lots of rdd and apply map() and collect() on them are time consuming. We try to combine the small rdd to a big one and executed on it. We used “images = (((((((image1.union(image2)).union(image3)).union(image4)).union(image5)).union(image6)).union(image7)).union(image8)).union(image9)” to combine the small rdd.

Finally, the time become 11 min

```
00
2019-04-27 13:58:08 INFO  ContextCleaner:54 - Cleaned accumulator 167
2019-04-27 13:58:08 INFO  ContextCleaner:54 - Cleaned accumulator 185
2019-04-27 13:58:08 INFO  ContextCleaner:54 - Cleaned accumulator 187
2019-04-27 13:58:08 INFO  ContextCleaner:54 - Cleaned accumulator 194
2019-04-27 13:58:08 INFO  ContextCleaner:54 - Cleaned accumulator 173
2019-04-27 13:58:08 INFO  ContextCleaner:54 - Cleaned accumulator 195
2019-04-27 13:58:08 INFO  ContextCleaner:54 - Cleaned accumulator 155
```

Technical Problems Encountered

- COC Server Crashed
  - TA restart coc server
  - Copy host file from coc and re-configure the ip address for all single server
  - Reboot our server; Restart VMs, ganglia and Hadoop node managers
  - Run ssh -Nf in coc server to specific port in our cluster
- Python Environment Configuration
  - Every server need same python env, so that the cluster could run the same python application corporately
  - Copy Anaconda env folder to every node (both manager and VMs); Set anaconda as default python env by modifying .bashrc/.profile file
- After python environment configuration, there is still error “ python2.7 conflict with python 3.6”

  we wrote two sentences at the top of file:
  os.environ['PYSPARK_PYTHON']='/home/hduser/anaconda3/bin/python'
  os.environ['SPARK_HOME']='/opt/spark-2.4.0-bin-hadoop2.7'

Technical Problems Encountered

- Order problem in reading files to RDD:
  - when reading 100 files, they can be input in right order, but when reading more than 500 files, these files will be input out of order, which will have a bad influence on the final result.

  Solution:
  - Reading small files to RDD, and then using ‘union()’ to combine them, so that a large number of files can be input in right order.

Technical Problems Encountered

- Can Not read caption on spark directly
  - # Common Way:

```
import cv2
video = cv2.VideoCapture(video_path)
while(true):
    success, capture = video.read()
    pass
```

Solution:

  - Run the script separately in 8 VMs, holding different start and end num

```
start_num = 100
end_num = 200
video = cv2.VideoCapture(video_path)
video.set(propId=cv2.CAP_PROP_POS_FRAMES, value=start_num)
for i in range(start_num, end_num):
    success, capture = video.read()
    pass
```

Technical Problems Encountered

- Order problem in reading files to RDD:
  - when reading 100 files, they can be input in right order, but when reading more than 500 files, these files will be input out of order, which will have a bad influence on the final result.

  Solution:
  - Reading small files to RDD, and then using ‘union()’ to combine them, so that a large number of files can be input in right order.

Parallel Crawler

As spark's common tools such as mllib, streaming, sparkSQL have no specialized packages for web crawler, we just optimize this part with multiple threads tech in python.

Although some third party tools such as "Sparkler" created by DataBricks, the given examples have some bugs sometimes which are time killer for us. As the train dataset is not that much, we could figure it out only by one single server with multi-threaded optimization.