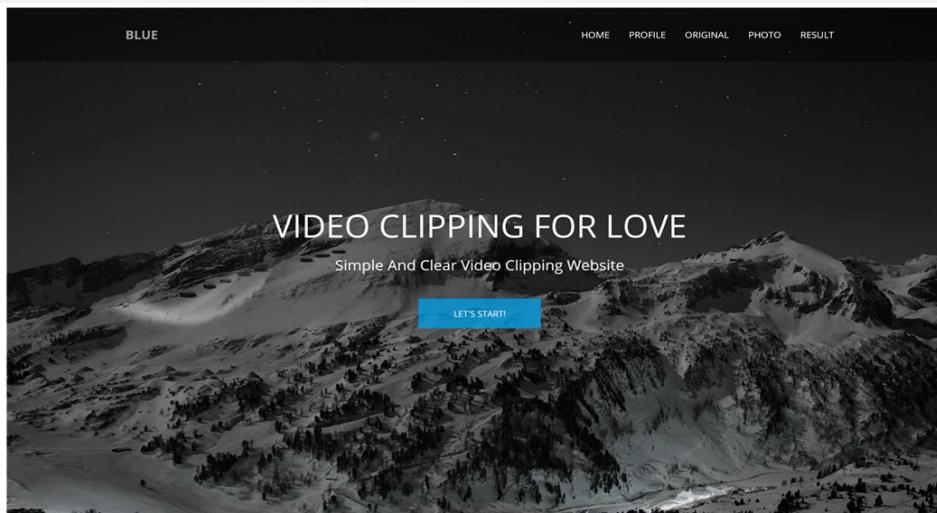


# **VIDEO CLIPPING FOR LOVE**

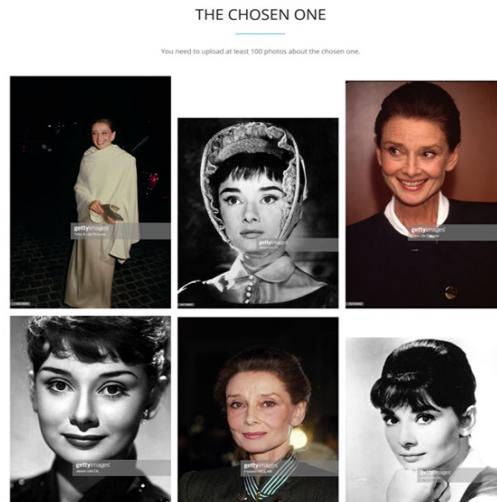


Run Demo Video	<a href="https://drive.google.com/file/d/1kzjw7i9d_lqJz0m4ivGBbPAqdMIWGDS-/view?usp=sharing">https://drive.google.com/file/d/1kzjw7i9d_lqJz0m4ivGBbPAqdMIWGDS-/view?usp=sharing</a>								
Demo Result	<a href="https://drive.google.com/file/d/1Nn70liQBlilTLZZBWzSkZ8MRn5IxYqeC/view?usp=sharing">https://drive.google.com/file/d/1Nn70liQBlilTLZZBWzSkZ8MRn5IxYqeC/view?usp=sharing</a>								
Demo on Web	<a href="http://202.45.128.135:10889/view/tomcat/website_1/index.html">http://202.45.128.135:10889/view/tomcat/website_1/index.html</a> (Password: student)								
How to Run Project Demo	<ol style="list-style-type: none"> <li>1. Login into hduser@gpu20 (Password: student)</li> <li>2. Check all images we could use, which is stored in HDFS: <code>hdfs dfs -ls /David/img2/</code> (Around 21, 000 images here)</li> <li>3. At the home path: <code>/home/gpu20, run /opt/spark-2.4.0-bin-hadoop2.7/bin/spark-submit --master yarn test1.py</code></li> <li>4. Check records on <a href="http://202.45.128.135:20220">http://202.45.128.135:20220</a> and <a href="http://202.45.128.135:20620">http://202.45.128.135:20620</a></li> <li>5. Check our web for result representation (open with Safari or Microsoft Edge) <a href="http://202.45.128.135:10889/view/tomcat/website_1/index.html">http://202.45.128.135:10889/view/tomcat/website_1/index.html</a></li> </ol>								
System Usage	<table border="1"> <tr> <td>Data Storage</td><td>More than 2.5 GB data are used</td></tr> <tr> <td>Resource of Cluster</td><td>more than 52GB virtual memory and 9 virtual CPU cores are used</td></tr> <tr> <td>Spark</td><td>6 executors are parallel during demo on spark. (We updated our code and make it as 7 executors finally)</td></tr> </table>	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)		
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	<table border="1"> <tr> <td>Ganglia</td><td><a href="http://202.45.128.135:10089/ganglia/">http://202.45.128.135:10089/ganglia/</a></td></tr> <tr> <td>Hadoop Overview</td><td><a href="http://202.45.128.135:20120">http://202.45.128.135:20120</a></td></tr> <tr> <td>Hadoop History</td><td><a href="http://202.45.128.135:20220">http://202.45.128.135:20220</a> (During project running)</td></tr> <tr> <td>Spark History</td><td><a href="http://202.45.128.135:20620">http://202.45.128.135:20620</a></td></tr> </table>	Ganglia	<a href="http://202.45.128.135:10089/ganglia/">http://202.45.128.135:10089/ganglia/</a>	Hadoop Overview	<a href="http://202.45.128.135:20120">http://202.45.128.135:20120</a>	Hadoop History	<a href="http://202.45.128.135:20220">http://202.45.128.135:20220</a> (During project running)	Spark History	<a href="http://202.45.128.135:20620">http://202.45.128.135:20620</a>
Ganglia	<a href="http://202.45.128.135:10089/ganglia/">http://202.45.128.135:10089/ganglia/</a>								
Hadoop Overview	<a href="http://202.45.128.135:20120">http://202.45.128.135:20120</a>								
Hadoop History	<a href="http://202.45.128.135:20220">http://202.45.128.135:20220</a> (During project running)								
Spark History	<a href="http://202.45.128.135:20620">http://202.45.128.135:20620</a>								

# Web UI Design



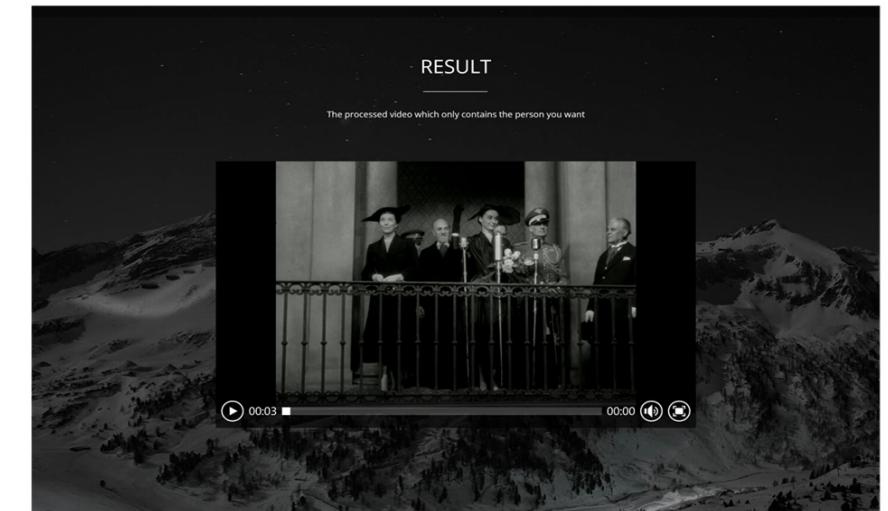
Front Page



Several photos to train the Face Recognition

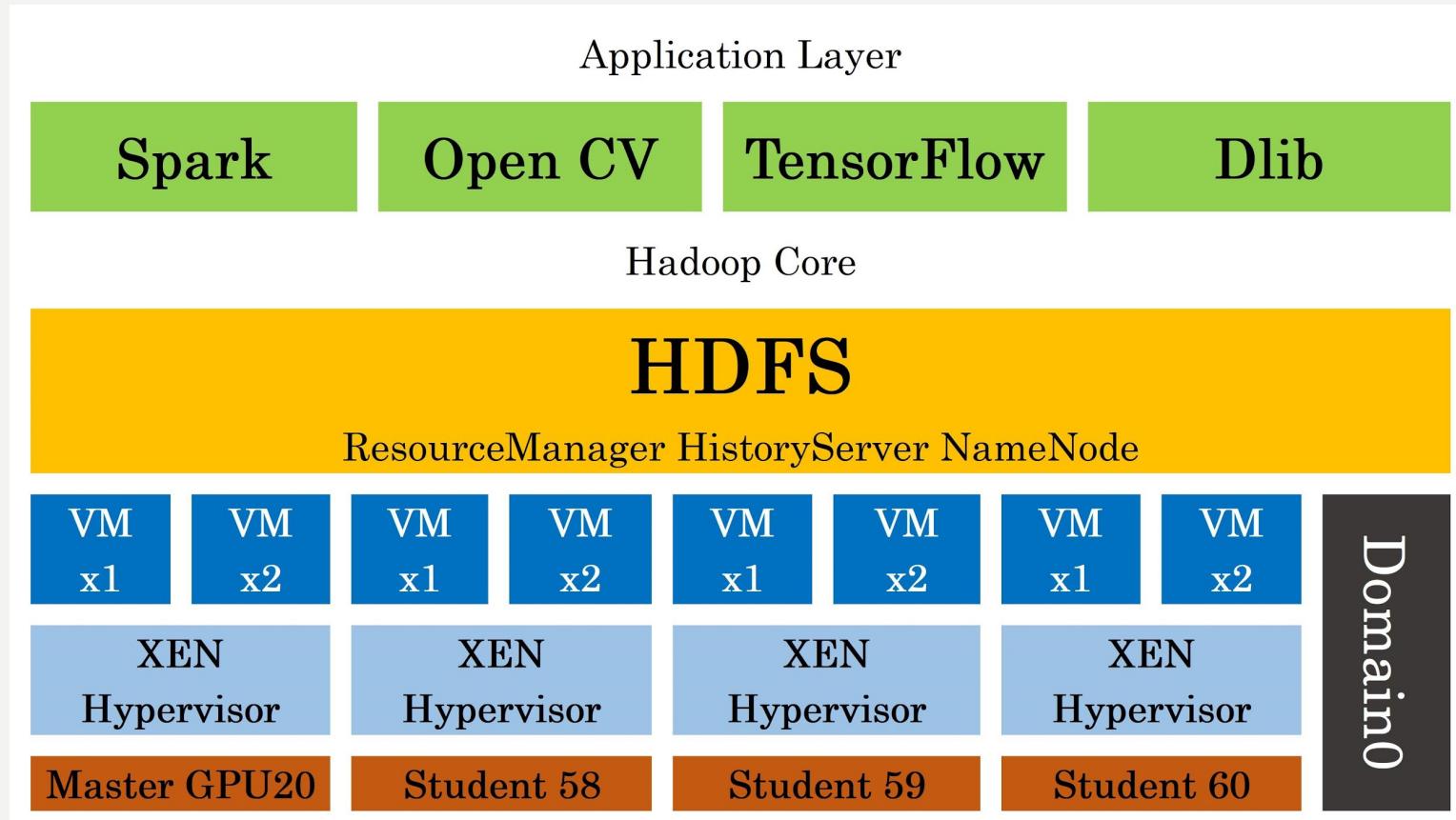
A screenshot of the website's interface showing a video player window. The video shows a historical building with a dome, likely the Royal Palace of Amsterdam. The video player includes controls for play, volume, and progress. Above the video, the word 'ORIGINAL' is displayed. To the right, there is a section titled 'APPLICATION SCENARIOS' with four categories: 'LOVER', 'FAMILY', 'IDOL', and 'ATHLETE', each with a brief description.

The original video to process and suggested application scenarios



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									
	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:3 8	52:54:00: 5f:b6:02	52:54:00: 54:60:1a	54:BF:64 :8B:3D:D E	52:54:00: 6c:90:91	52:54:00: 64:11:f0	54:BF:64 :8A:D0:C 5	52:54:00: 1c:3d:e1	52:54:00: 6c:55:6c
	IP	10.42.2.40	10.42.2.70	10.42.2.10 0	10.42.0.68	10.42.0.16 8	10.42.1.68	10.42.0.69	10.42.0.16 9	10.42.1.69	10.42.0.70	10.42.0.17 0	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

Crawler

Photos

Face  
Model  
Training

Mode  
I

Origina  
l Video

Recognition

Probabilit  
y List

Origina  
l Video

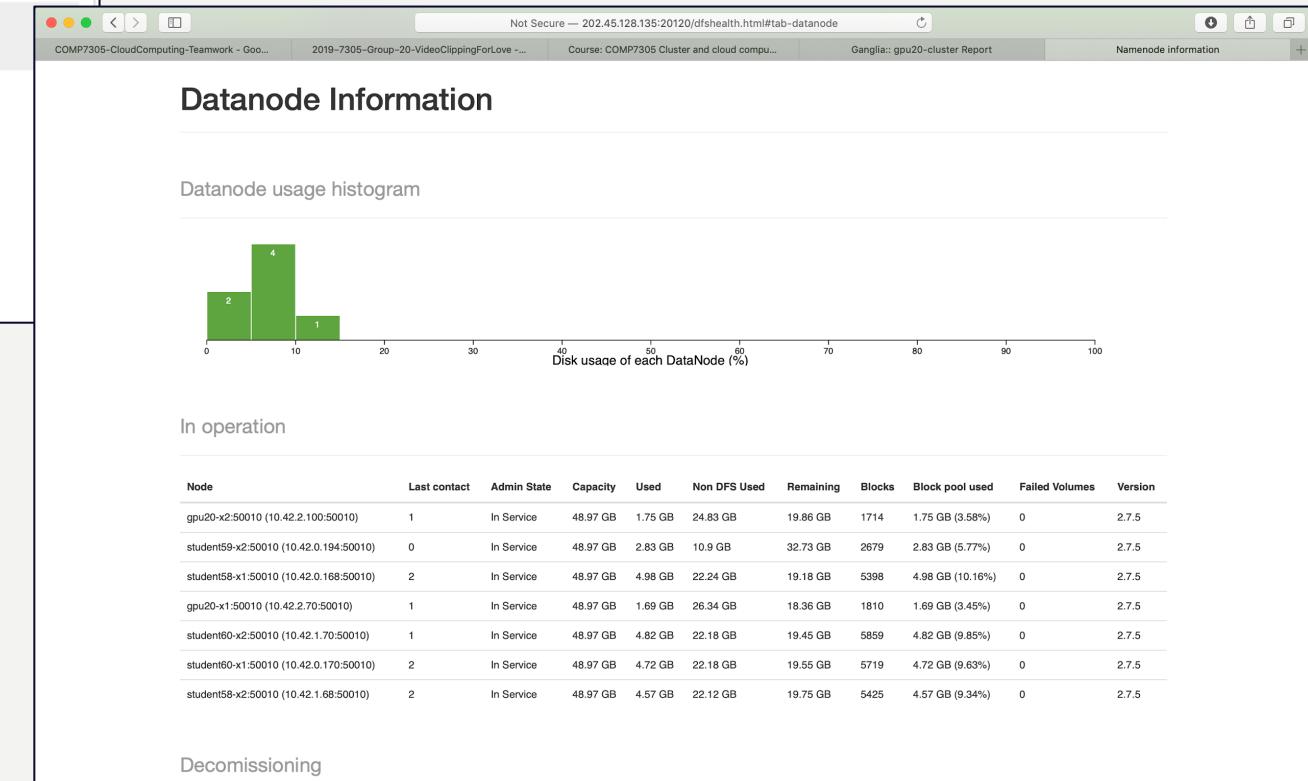
Frame  
Extractio  
n

Picture  
s

Video  
Cutting

New  
Video

# Software installation



<http://202.45.128.135:10089/ganglia/>  
<http://202.45.128.135:20120/>

# Software installation

Not Secure — 202.45.128.135:20220/cluster

**All Applications**

Logged in as: dr.who

Cluster Metrics																
About Nodes	Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
Node Labels	43	0	1	42	9	41 GB	64 GB	20 GB	9	64	4	8	0	0	0	0

**Scheduler Metrics**

Scheduler Type		Scheduling Resource Type		Minimum Allocation		Maximum Allocation	
Capacity Scheduler	[MEMORY]	<memory:1024, vCores:1>	<memory:8192, vCores:8>				

Show 20 entries

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI	Blacklisted Nodes
application_1556336774567_0043	hduser	VIDEO CLIPPING FOR LOVE 2019-04-27-12-06-56	SPARK	default	Sat Apr 27 20:07:07 +0800 2019	N/A	RUNNING	UNDEFINED		ApplicationMaster	0
application_1556336774567_0042	hduser	VIDEO CLIPPING FOR LOVE 2019-04-27-12-00-13	SPARK	default	Sat Apr 27 20:00:22 +0800 2019	Sat Apr 27 20:04:37 +0800 2019	FINISHED	SUCCEEDED		History	N/A
application_1556336774567_0041	hduser	VIDEO CLIPPING FOR LOVE 2019-04-27-11-47-19	SPARK	default	Sat Apr 27 19:47:19 +0800 2019	Sat Apr 27 19:52:00 +0800 2019	FINISHED	SUCCEEDED		History	N/A

<http://202.45.128.135:20220>  
<http://202.45.128.135:20620>

Not Secure — 202.45.128.135:20620

**History Server**

Event log directory: hdfs://gpu20:9000/tmp/sparkLog

Last updated: 2019-04-27 20:11:06

Client local time zone: Asia/Hong\_Kong

Show 20 entries

App ID	App Name	Started	Completed	Duration	Spark User	Last Updated	Event Log
application_1556336774567_0043	VIDEO CLIPPING FOR LOVE 2019-04-27-12-06-56	2019-04-27 20:06:56	2019-04-27 20:10:54	4.0 min	hduser	2019-04-27 20:10:54	<a href="#">Download</a>
application_1556336774567_0042	VIDEO CLIPPING FOR LOVE 2019-04-27-12-00-13	2019-04-27 20:00:13	2019-04-27 20:04:37	4.4 min	hduser	2019-04-27 20:04:37	<a href="#">Download</a>
application_1556336774567_0041	VIDEO CLIPPING FOR LOVE 2019-04-27-11-47-19	2019-04-27 19:47:19	2019-04-27 19:52:00	4.7 min	hduser	2019-04-27 19:52:00	<a href="#">Download</a>
application_1556336774567_0040	VIDEO CLIPPING FOR LOVE 2019-04-27-11-45-37	2019-04-27 19:45:37	2019-04-27 19:46:10	33 s	hduser	2019-04-27 19:46:10	<a href="#">Download</a>
application_1556336774567_0039	VIDEO CLIPPING FOR LOVE 2019-04-27-11-42-36	2019-04-27 19:42:36	2019-04-27 19:43:19	43 s	hduser	2019-04-27 19:43:19	<a href="#">Download</a>
application_1556336774567_0038	VIDEO CLIPPING FOR LOVE 2019-04-27-11-40-33	2019-04-27 19:40:33	2019-04-27 19:41:15	42 s	hduser	2019-04-27 19:41:15	<a href="#">Download</a>
application_1556336774567_0037	VIDEO CLIPPING FOR LOVE 2019-04-27-11-19-39	2019-04-27 19:19:39	2019-04-27 19:20:18	39 s	hduser	2019-04-27 19:20:18	<a href="#">Download</a>
application_1556336774567_0036	VIDEO CLIPPING FOR LOVE 2019-04-27-11-16-39	2019-04-27 19:16:39	2019-04-27 19:17:19	39 s	hduser	2019-04-27 19:17:19	<a href="#">Download</a>

# 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.findall('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()
    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](http://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, 10:33:22 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  $500\text{KB} * 21,000 = 10.5\text{ GB}$  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(im
        age6)).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/1fXxhrtSqEbxB2A>

- 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

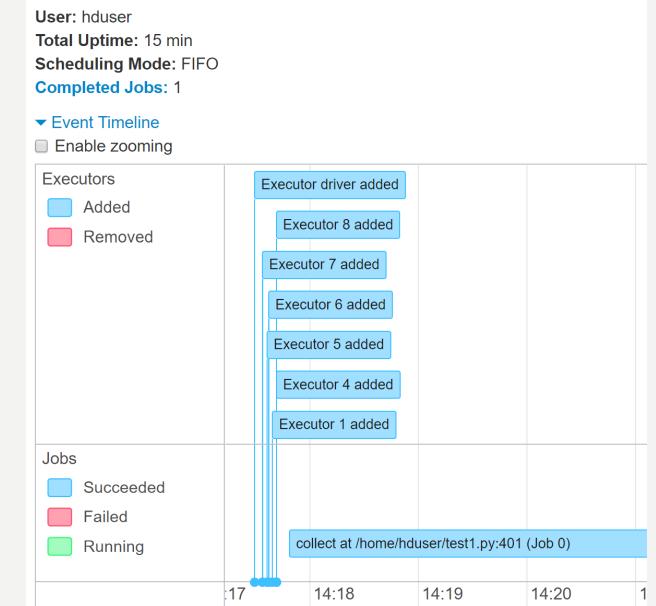
Data Size	Execution Time
2.3 GB (3000 Pics)	33 min

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

The screenshot shows a web browser window with the Apache Spark History Server interface. The URL is `202.45.128.135:20620/history/application_1556336774567_0045/jobs/`. The page title is "Spark Jobs".

**User:** hduser  
**Total Uptime:** 4.0 min  
**Scheduling Mode:** FIFO  
**Completed Jobs:** 1

**Event Timeline**

**Completed Jobs (1)**

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	collect at /home/hduser/test1ly.py:378 collect at /home/hduser/test1ly.py:378	2019/04/27 12:20:56	3.4 min	1/1	4/4

# Execution Time

execute 500 images:398.9 MB

VIDEO X | All App X | Ganglia X | VIDEO X | Home X | test1.p X | \*test1y X | Python X | 2019-7 X | 书签 X | +

← → C ⓘ 不安全 | 202.45.128.135:20620/history/application\_1556336774567\_0045/executors/

Apache Spark 2.4.0 Jobs Stages Storage Environment Executors VIDEO CLIPPING FOR LOVE 2019-04... application

## 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

# Execution Time

execute 3000 images: 11min

The screenshot shows the Apache Spark 2.4.0 UI interface. The top navigation bar includes tabs for Home, test1.py, \*test1y, Python, 2019-7, Bookmarks, and a plus sign for new tabs. Below the bar, the URL is 202.45.128.135:20620/history/application\_1556336774567\_0052/jobs/. The main header displays "VIDEO CLIPPING FOR LOVE application L". On the left, there's a sidebar with links for User: hduser, Total Uptime: 15 min, Scheduling Mode: FIFO, and Completed Jobs: 1. Under "Completed Jobs", there's a link for Event Timeline and a section for Completed Jobs (1). A table lists the details of the completed job:

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	collect at /home/hduser/test1.py:401 collect at /home/hduser/test1.py:401	2019/04/27 14:17:48	11 min	1/1	23/23

# Execution Time

execute 3000 images: 2.5 GB

Spark VIDEO | All App | Ganglia | Home | test1.py | \*test1y | Python | 2019-7 | 书签 | +

← → ⌂ ⓘ 不安全 | 202.45.128.135:20620/history/application\_1556336774567\_0052/executors/

Apache Spark 2.4.0 Jobs Stages Storage Environment Executors

VIDEO CLIPPING FOR LOVE application

## 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.