

Monash University Information Technology FIT3162 - Computer Science Project 2

Code Report

Stream Processing Architecture using Apache Kafka for Analytics and
Visualisation On A Dashboard

Khai Fung, Lim

29297311

klim0022@student.monash.edu

Yi Ping, Ho

29352258

yhoo0007@student.monash.edu

Chiu Gin, Chong

28842022

ccho0024@student.monash.edu

Fernando Ng

28737377

fern0002@student.monash.edu

Semester 1, 2020

Team 1C

Supervisor: Dr. Vishnu Monn Baskaran

WORD COUNT: 4533

Contents

1	Code Introduction	3
2	Process and Project Management	3
2.1	Version Control	3
2.2	Maven	4
3	Source Code	5
3.1	Code Readability	5
3.2	Code Comments	5
3.3	Code Structure, Maintainability and Modularity	6
4	Code Aspects	7
4.1	Robustness	7
4.2	Scalability	8
4.3	Platform and OS Independence (Portability)	9
4.4	Overall Summary	10

5	Limitations	10
6	Security	10
7	Usability and User Experience	11
8	Future Work	12
8.1	DevOp Tools	12
8.2	Mac OS Compatibility	12
8.3	Decouple Analytical Engine	12
9	Technical User Guide	12
9.1	Basic Setup and Installation	12
9.2	Running The Software	23
9.3	Accessing Database	25
10	End User Guide	26
10.1	Camera Properties	26
10.2	Archiver Properties	26
10.3	Collector Properties	27
10.4	Dashboard Properties	27
10.5	Processor Properties	28
10.6	OpenCV and Maven Setup	28
10.7	Starting the System	30
10.8	Operating the System	30
10.9	Shutting Down the System	30
	References	30

1 Code Introduction

The code base of this project comprises of a single Java project which contains all required Java files. The Java files roughly correspond to the various components shown in figure 1. There are also a number of miscellaneous Java file for components such as the custom message partitioner etc. Each major component in our system is designed to be ran separately from each other, i.e. the producer application can be ran without the consumer application running and so on.

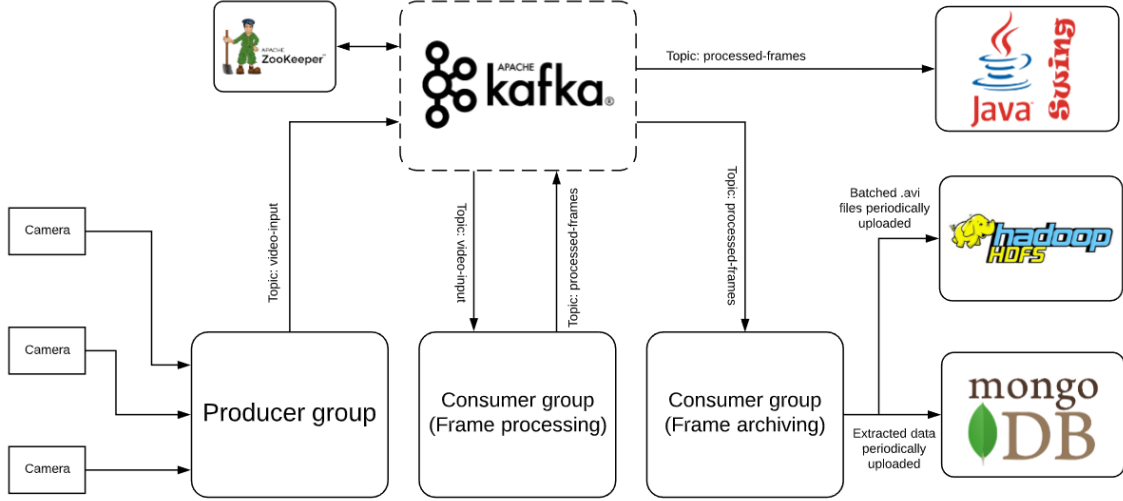


Figure 1: System Architecture

Additionally, the system also relies on a number of third party systems to function. Namely Apache Maven, Apache Kafka, Apache Zookeeper, Apache Hadoop HDFS, Apache YARN, MongoDB and OpenCV. These applications must be setup and running beforehand as they form the building blocks of the system. Refer to the technical user guide below for instructions on how to set up and run the system as a whole.

2 Process and Project Management

2.1 Version Control

GitHub is our project's revision control system. All code files of the project are pushed onto GitHub when there are major changes to the code. This ensures that the team will be working on the latest codes without having to retrieve a soft copy from other teammates. Instead, they can just pull our repository, update and merge the codes from the repository and their code accordingly.

GitHub is very useful in tracking the changes made, reverting the code back to previous versions and track all files and codes that are changed by mistake. Therefore, the code can be worked on independently without needing to asks other members if there are any changes on the code.

Link to GitHub repository: <https://github.com/yhoo0007/FIT3161-Team-1C>

Most importantly, GitHub allows the team to continuously work on the project separately without meeting face to face. A clear history of the commits can be found easily through GitHub along with all of the updated documentations of our code. This allows the tracking of all codes to help meet deadlines with completed code deliverables.

2.2 Maven

Apache Maven is a build automation tool specifically for Java. It helps in project management by ensuring the project has an optimal file structure to help navigating through large project directories. Furthermore, all dependencies are downloaded through a central repository of Maven that will can be managed, updated and downloaded according to the dependencies information in the POM.xml file at the root of the project. Therefore, members or users who download and runs the project does not need to set anything or download any external dependencies of the project code itself and just follow the technical user guide listed below.

Our code structure is shown is Figures 2a and 2b below. The codes are split up nicely to properties directory, source code directory and test code directory to manage the project. All sources codes will be under the src directory and most of the coding will be under that directory. A sudo maven clean package or maven clean package command can build the whole project to its target folder with just one command line.



(a) Tree structure of Project Part 1



(b) Tree Structure of Project Part 2

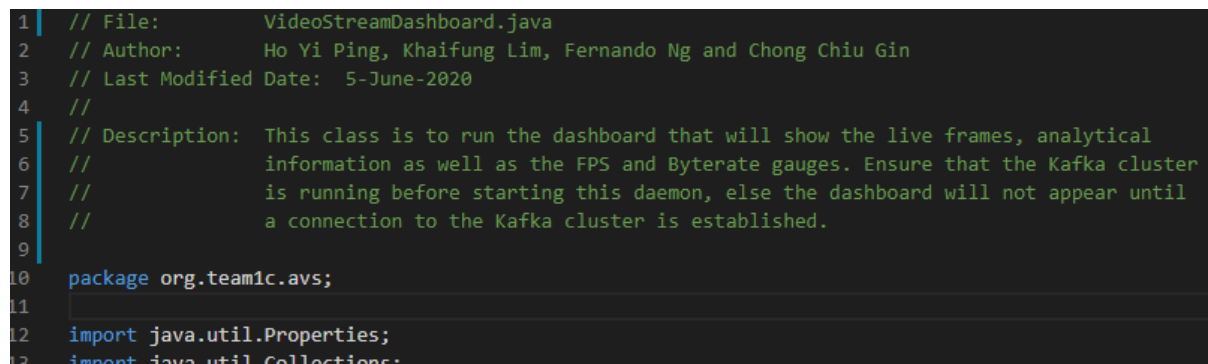
3 Source Code

3.1 Code Readability

All Java code files follow the Google Java Style (Google, 2020) to ensure good readability and maintainability which is especially important when working on a large code base involving multiple developers. The style guide for Java also shares many similarities with those for other languages which greatly improves the accessibility of the code base to developers coming from other languages. Following the guide, we emphasised on giving identifiers meaningful names depending on their function and purpose. This makes our code much more easily understood and also allows for fewer redundant comments.

3.2 Code Comments

Each code file includes a comment header containing a brief description of the file along with extra information following the example shown in Figure 3. It is a synopsis of the file in a quick and concise manner which will save the reader from having to read any code to find the purpose of any particular source code file.



```
1 // File:      VideoStreamDashboard.java
2 // Author:    Ho Yi Ping, Khaifung Lim, Fernando Ng and Chong Chiu Gin
3 // Last Modified Date: 5-June-2020
4 //
5 // Description: This class is to run the dashboard that will show the live frames, analytical
6 //              information as well as the FPS and Byterate gauges. Ensure that the Kafka cluster
7 //              is running before starting this daemon, else the dashboard will not appear until
8 //              a connection to the Kafka cluster is established.
9
10 package org.team1c.avs;
11
12 import java.util.Properties;
13 import java.util.Collections;
```

Figure 3: Example comment header

Every function also contains a doc-string which provides a brief description of the function, its parameters, its return value, and exceptions that it may raise as shown in Figure 4. The format of the doc-string is also such that Java docs can be generated from them which not only saves the time and effort of having to write them, but also results in documentation which is in a familiar format to Java developers.

Comments are written where necessary based on the combined judgement of all the developers. They can be found explaining chunks of code that work together to perform a certain task, or parts that are particularly difficult to read and understand. Great care was also taken to ensure the code base is not flooded with meaningless and unnecessary comments. A JavaDoc is generated to help in documentation.

```

55  /**
56   * Queries the given consumer's Kafka cluster to get the number of cameras/partitions there
57   * are.
58   * Note: Camera IDs are equivalent to partition numbers.
59   * @param consumer Kafka consumer to query with
60   * @param topic Kafka topic to query for
61   * @return list of integers representing the list of camera ids/partitions
62   */
63  public static List<Integer> getCameraIds(Consumer<?, ?> consumer, String topic) {
64      List<Integer> cameraIds = consumer.partitionsFor(
65          topic
66      ).stream().map(p -> p.partition()).collect(Collectors.toList());
67      return cameraIds;
68  }

```

Figure 4: Example function declaration

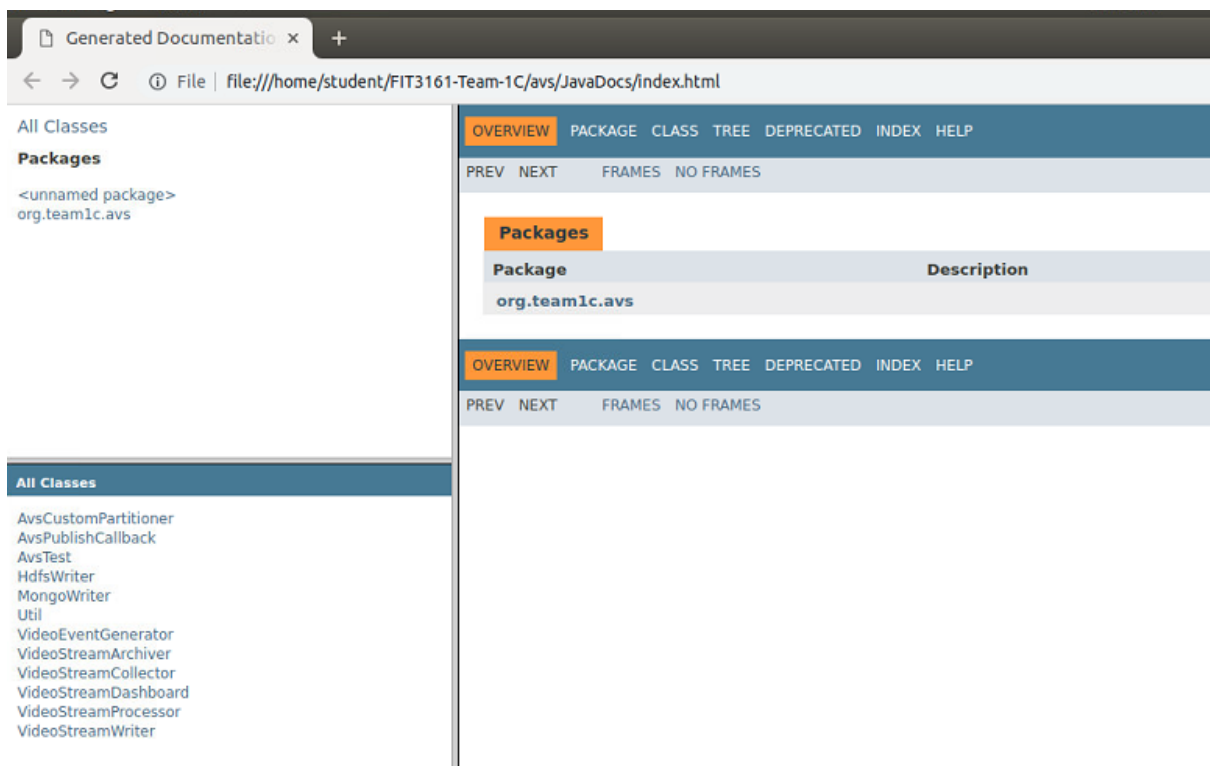


Figure 5: JavaDoc Main Page

3.3 Code Structure, Maintainability and Modularity

Java provides excellent support for good code structure and maintainability in the form of packages, classes, inheritance, access modifiers, etc. These tools allowed the code base to be organised in a highly modular fashion where each component of the system can be isolated from each other as much as possible to reduce dependencies. On the other hand, code duplication is nearly non-existent due to the generalised approach taken when writing functions. This allows for extensive reuse of code whenever possible which not only reduces the size of the code base, but also drastically improves reliability and maintainability as stated in Microsoft's code metric values (Microsoft, 2020).

There are four major components to the system (not including aforementioned third party applications). These components are modular, can be added and removed according to the requirements of the framework and capabilities of the hardware:

- Collector - connects to video sources and introduces frames into the system
- Processor - performs real time video processing
- Dashboard - displays video and other relevant information
- Archiver - stores the video and other information into archive

Each of these components have their own driver code and their functionalities are wrapped up in individual files named according to their functions. As mentioned previously, each of these components do not rely on any other component running at the same time in order to function. This means that every one of them can be ran independently from each other as separate tasks on an individual machine or across multiple machines connected via local network.

The high degree of modularity is largely due to the use of an Apache Kafka cluster as a central message broker which allows for nearly complete decoupling of the individual components of the system. This means that not only can the entire system be run all together at once, but each component may be individually started, restarted, or terminated without much effect on the rest of the system. As a result, debugging of the system is made much easier as the bugs can be easily isolated and reproduced by simply running the relevant components.

4 Code Aspects

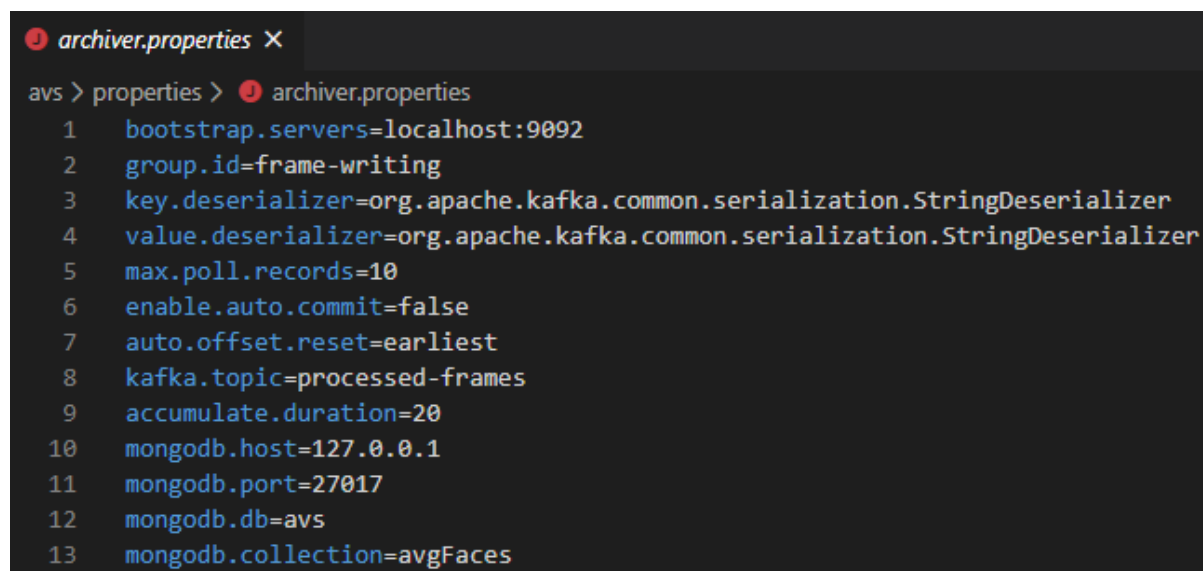
4.1 Robustness

Besides Java being a strongly typed language which naturally results in more robust code, several steps have also been taken from the developers point of view. As the main end-user inputs to the system are done via various configuration files, the system can use a file parser to quickly detect invalid or missing inputs before any crucial sections of the code is executed. This helps prevent run-time errors which may result in more serious problems such as data corruption.

The code is also written with the fail-fast principle in mind, i.e. functions do not attempt to resolve exceptions unless they are the direct cause of it and they have the entire autonomy to do so. This prevents errors from propagating deeper into the system and causing further errors and help pinpoint the source of bugs. When a function encounters something unexpected, it should promptly throw an appropriate exception instead of trying to continue. All exceptions raised by the system are also given meaningful error messages to help the user identify and resolve the problem.

Every component of the system interacts with the central Kafka cluster either as a producer or as a consumer. Thus, the code utilises Kafka's highly robust and reliable library functions for consuming and publishing messages to and from the cluster. Apache Kafka's library provides a plethora of configuration options for varying levels of robustness and consistency depending on the use case. This system is configured to slightly favour

throughput and latency over consistency as occasional frame drops are not a major issue. The Kafka library functions will automatically handle any cluster related issues such as leader reelection and metadata communication internally. Additionally, the library functions also handle retry attempts automatically. These Kafka guarantees allow our code to offload much of these error handling work to Kafka's internal functions which greatly improves the robustness of the system as a whole.

A screenshot of a terminal window with a dark background. The title bar at the top reads "archiver.properties" with a red icon on the left and a close button (X) on the right. The terminal shows a command prompt "avs > properties > archiver.properties" followed by a list of 13 configuration properties, each on a new line and numbered from 1 to 13. The properties are: 1 bootstrap.servers=localhost:9092, 2 group.id=frame-writing, 3 key.deserializer=org.apache.kafka.common.serialization.StringDeserializer, 4 value.deserializer=org.apache.kafka.common.serialization.StringDeserializer, 5 max.poll.records=10, 6 enable.auto.commit=false, 7 auto.offset.reset=earliest, 8 kafka.topic=processed-frames, 9 accumulate.duration=20, 10 mongodb.host=127.0.0.1, 11 mongodb.port=27017, 12 mongodb.db=avs, and 13 mongodb.collection=avgFaces.

```
archiver.properties X
avs > properties > archiver.properties
1 bootstrap.servers=localhost:9092
2 group.id=frame-writing
3 key.deserializer=org.apache.kafka.common.serialization.StringDeserializer
4 value.deserializer=org.apache.kafka.common.serialization.StringDeserializer
5 max.poll.records=10
6 enable.auto.commit=false
7 auto.offset.reset=earliest
8 kafka.topic=processed-frames
9 accumulate.duration=20
10 mongodb.host=127.0.0.1
11 mongodb.port=27017
12 mongodb.db=avs
13 mongodb.collection=avgFaces
```

Figure 6: Example configuration file

4.2 Scalability

Scalability is one of the main focuses of this system. Therefore each framework chosen for the system features high horizontal scaling abilities. With a highly scalable Kafka cluster at the heart of the system, and every other component built as either a Kafka consumer or producer, the entire system can be scaled horizontally with relative ease. These frameworks are usually scaled by adding more machines to their cluster and tweaking the configuration files. Hence a similar design pattern is employed by our system.

As previously mentioned, our system reads its configuration from the various configuration files and also from the Kafka cluster itself. Increasing the number of producers and consumers can simply be done by running additional daemons. Due to the use of Kafka as the central cluster, these new daemons can simply establish a connection to the Kafka cluster and the system will automatically adjust to accommodate them (see technical user guide section for more information on scaling the system). Kafka will automatically perform load balancing according to the number of partitions as well as manage all groups and topics according to their ids to ensure that messages will be passed based on their own groups and topics maintaining ordering.


```

C:\Windows\System32\cmd.exe - bat\windows\kafka-server-start.bat config\zookeeper.properties
(2020-06-09 18:23:49,213) INFO Got user-level KeeperException when processing sessionid:0x100015093c40004 type:create
xid:0x0 zxid:0x0 tntype:-1 repath:n/a Error Path:/config/changes Error:KeeperErrorCode = NodeExists for /config/
changes (org.apache.zookeeper.server.PrepRequestProcessor)
(2020-06-09 18:23:49,240) INFO Got user-level KeeperException when processing sessionid:0x100015093c40004 type:create
xid:0x0 zxid:0x0 tntype:-1 repath:n/a Error Path:/admin/delete_topics Error:KeeperErrorCode = NodeExists for /ad
min/delete_topics (org.apache.zookeeper.server.PrepRequestProcessor)
(2020-06-09 18:23:49,273) INFO Got user-level KeeperException when processing sessionid:0x100015093c40004 type:create
xid:0x0 zxid:0x0 tntype:-1 repath:n/a Error Path:/brokers/seqid Error:KeeperErrorCode = NodeExists for /brokers/
seqid (org.apache.zookeeper.server.PrepRequestProcessor)
(2020-06-09 18:23:49,300) INFO Got user-level KeeperException when processing sessionid:0x100015093c40004 type:create
xid:0x0 zxid:0x0 tntype:-1 repath:n/a Error Path:/isr_change_notification Error:KeeperErrorCode = NodeExists for
/isr_change_notification (org.apache.zookeeper.server.PrepRequestProcessor)
(2020-06-09 18:23:49,333) INFO Got user-level KeeperException when processing sessionid:0x100015093c40004 type:create
xid:0x0 zxid:0x0 tntype:-1 repath:n/a Error Path:/latest_producer_id_block Error:KeeperErrorCode = NodeExists fo
r /latest_producer_id_block (org.apache.zookeeper.server.PrepRequestProcessor)
(2020-06-09 18:23:49,372) INFO Got user-level KeeperException when processing sessionid:0x100015093c40004 type:create
xid:0x0 zxid:0x0 tntype:-1 repath:n/a Error Path:/log_dir_event_notification Error:KeeperErrorCode = NodeExists
for /log_dir_event_notification (org.apache.zookeeper.server.PrepRequestProcessor)
(2020-06-09 18:23:49,405) INFO Got user-level KeeperException when processing sessionid:0x100015093c40004 type:create
xid:0x0 zxid:0x0 tntype:-1 repath:n/a Error Path:/config/topics Error:KeeperErrorCode = NodeExists for /config/t
opics (org.apache.zookeeper.server.PrepRequestProcessor)
(2020-06-09 18:23:49,432) INFO Got user-level KeeperException when processing sessionid:0x100015093c40004 type:create
xid:0x0 zxid:0x0 tntype:-1 repath:n/a Error Path:/config/clients Error:KeeperErrorCode = NodeExists for /config/
clients (org.apache.zookeeper.server.PrepRequestProcessor)
(2020-06-09 18:23:49,465) INFO Got user-level KeeperException when processing sessionid:0x100015093c40004 type:create
xid:0x0 zxid:0x0 tntype:-1 repath:n/a Error Path:/config/users Error:KeeperErrorCode = NodeExists for /config/us
ers (org.apache.zookeeper.server.PrepRequestProcessor)
(2020-06-09 18:23:49,492) INFO Got user-level KeeperException when processing sessionid:0x100015093c40004 type:create
xid:0x0 zxid:0x0 tntype:-1 repath:n/a Error Path:/config/brokers Error:KeeperErrorCode = NodeExists for /config/
brokers (org.apache.zookeeper.server.PrepRequestProcessor)

C:\Windows\System32\cmd.exe - bat\windows\kafka-server-start.bat config\server.properties
p-BRGSDM,9004,1,listenerName(PLAINTEXT,PLAINTEXT), czxid (broker epoch): 243 (kafka.zk.KafkaZkClient)
(2020-06-09 18:23:54,069) WARN No meta.properties file under dir E:\tmp\kafka-logs\meta.properties (kafka.server.Bro
kerMetadataAccessPoint)
(2020-06-09 18:23:54,753) INFO [ExpirationReaper-2-topic]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOp
erationReaper)
(2020-06-09 18:23:54,756) INFO [ExpirationReaper-2-Heartbeat]: Starting (kafka.server.DelayedOperationPurgatory$Expir
edOperationReaper)
(2020-06-09 18:23:54,757) INFO [ExpirationReaper-2-Rebalance]: Starting (kafka.server.DelayedOperationPurgatory$Expir
edOperationReaper)
(2020-06-09 18:23:54,769) INFO [GroupCoordinator 2]: Starting up. (kafka.coordinator.group.GroupCoordinator)
(2020-06-09 18:23:54,770) INFO [GroupCoordinator 2]: Startup complete. (kafka.coordinator.group.GroupCoordinator)
(2020-06-09 18:23:54,771) INFO [GroupMetadataManager brokerId=2] Removed 0 expired offsets in 1 milliseconds. (kafka.
coordinator.group.GroupMetadataManager)
(2020-06-09 18:23:54,771) INFO [ProducerIdManager 2]: Acquired new producerId block (brokerId=2,blockStartProducerId
=4000,blockEndProducerId=4999) by writing to zk with path version 5 (kafka.coordinator.transaction.ProducerIdManager)
(2020-06-09 18:23:54,848) INFO [TransactionCoordinator Id=2] Starting up. (kafka.coordinator.transaction.TransactionCo
ordinator)
(2020-06-09 18:23:54,860) INFO [TransactionMarker Channel Manager 2]: Starting (kafka.coordinator.transaction.Transac
tionMarkerChannelManager)
(2020-06-09 18:23:54,860) INFO [TransactionCoordinator Id=2] Startup complete. (kafka.coordinator.transaction.Transac
tionCoordinator)
(2020-06-09 18:23:54,883) INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotification
ListenerChangeEventProcessThread)
(2020-06-09 18:23:54,893) INFO [SocketServer brokerId=2] Started data-plane processors for 1 acceptors (kafka.network
.SocketServer)
(2020-06-09 18:23:54,895) INFO Kafka version: 2.3.1 (org.apache.kafka.common.utils.AppInfoParser)
(2020-06-09 18:23:54,896) INFO Kafka commitId: 18a913733fb71c01 (org.apache.kafka.common.utils.AppInfoParser)
(2020-06-09 18:23:54,896) INFO Kafka startTimestamp: 15916981234893 (org.apache.kafka.common.utils.AppInfoParser)
(2020-06-09 18:23:54,898) INFO [KafkaServer Id=2] started (kafka.server.KafkaServer)

C:\Windows\System32\cmd.exe - bat\windows\kafka-server-start.bat config\server.properties
p-BRGSDM,9003,1,listenerName(PLAINTEXT,PLAINTEXT), czxid (broker epoch): 197 (kafka.zk.KafkaZkClient)
(2020-06-09 18:21:52,776) WARN No meta.properties file under dir E:\tmp\kafka-logs\meta.properties (kafka.server.Bro
kerMetadataAccessPoint)
(2020-06-09 18:21:52,853) INFO [ExpirationReaper-1-topic]: Starting (kafka.server.DelayedOperationPurgatory$ExpiredOp
erationReaper)
(2020-06-09 18:21:52,855) INFO [ExpirationReaper-1-Heartbeat]: Starting (kafka.server.DelayedOperationPurgatory$Expir
edOperationReaper)
(2020-06-09 18:21:52,856) INFO [ExpirationReaper-1-Rebalance]: Starting (kafka.server.DelayedOperationPurgatory$Expir
edOperationReaper)
(2020-06-09 18:21:52,868) INFO [GroupCoordinator 1]: Starting up. (kafka.coordinator.group.GroupCoordinator)
(2020-06-09 18:21:52,869) INFO [GroupCoordinator 1]: Startup complete. (kafka.coordinator.group.GroupCoordinator)
(2020-06-09 18:21:52,870) INFO [GroupMetadataManager brokerId=1] Removed 0 expired offsets in 1 milliseconds. (kafka.
coordinator.group.GroupMetadataManager)
(2020-06-09 18:21:52,868) INFO [ProducerIdManager 1]: Acquired new producerId block (brokerId=1,blockStartProducerId
=3000,blockEndProducerId=3999) by writing to zk with path version 4 (kafka.coordinator.transaction.ProducerIdManager)
(2020-06-09 18:21:52,915) INFO [TransactionCoordinator Id=1] Starting up. (kafka.coordinator.transaction.TransactionCo
ordinator)
(2020-06-09 18:21:52,927) INFO [TransactionMarker Channel Manager 1]: Starting (kafka.coordinator.transaction.Transac
tionMarkerChannelManager)
(2020-06-09 18:21:52,927) INFO [TransactionCoordinator Id=1] Startup complete. (kafka.coordinator.transaction.Transac
tionCoordinator)
(2020-06-09 18:21:52,950) INFO [/config/changes-event-process-thread]: Starting (kafka.common.ZkNodeChangeNotification
ListenerChangeEventProcessThread)
(2020-06-09 18:21:52,960) INFO [SocketServer brokerId=1] Started data-plane processors for 1 acceptors (kafka.network
.SocketServer)
(2020-06-09 18:21:52,962) INFO Kafka version: 2.3.1 (org.apache.kafka.common.utils.AppInfoParser)
(2020-06-09 18:21:52,962) INFO Kafka commitId: 18a913733fb71c01 (org.apache.kafka.common.utils.AppInfoParser)
(2020-06-09 18:21:52,960) INFO Kafka startTimestamp: 1591698112960 (org.apache.kafka.common.utils.AppInfoParser)
(2020-06-09 18:21:52,964) INFO [KafkaServer Id=1] started (kafka.server.KafkaServer)

```

Figure 7: Three Kafka daemons and Zookeeper running in terminals

Apache Hadoop HDFS is also chosen as the archiving solution for the system due its ability to scale. Similar to the Kafka, the HDFS cluster can also be easily scaled by connecting new machines to the network and registering them with the HDFS name-node. The Hadoop Distributed File System automatically handles load balancing, replication, and scaling. As a result, this allows the archiving subsystem to be loosely coupled to the rest of the system such that the HDFS cluster can be managed and scaled independently from the rest of the system.

In a similar vein, MongoDB also supports horizontal scaling in the form of sharding. The system uses a MongoDB database for storing analytical data produced from the real time analytics in JSON documents. MongoDB will be able to store a large number of documents based on collections which will store all data in order. Even though we did not specify a proper database model, we can scale the database with a capped upper bound based on the time buckets where a set of time duration is stored in a collection for ease of querying. As with HDFS, the database subsystem can also be managed and scaled independently due to the loose coupling between the different subsystems.

4.3 Platform and OS Independence (Portability)

The system is built atop a number of different technologies, namely Java, OpenCV, Apache Kafka, HDFS, and MongoDB. Each of them support the three major operating systems: Linux, Windows, Mac OS. However, due to hardware limitations, the system has only ever been installed and tested on Linux and Windows 10. Therefore we can only infer that the system is compatible with Mac OS due to every component being available for Mac OS individually. **At the time of writing, we have not confirmed that the system can run on Mac OS.**

4.4 Overall Summary

The software is achieving its goal in having features that fulfill the criteria set by the product requirements. The system has the required robustness, readability, maintainability, modularity, scalability and usability to ensure that the code can be easily understood, used as well as further developed by other users.

5 Limitations

As much as Java is a highly popular language for systems development (and the native language for many of the frameworks used), it is an old language which does not support many of the newer programming features and syntax sugars. As a result, the code base exhibits a slightly bloated and convoluted structure.

Another limitation is the complicated Java importing system and Maven dependencies. In this system, installation of the image processing library OpenCV for Java is a complicated, error-prone, and OS dependent process. This is because certain OpenCV libraries for video handling must be loaded during run-time. The highly verbose importing syntax of Java also makes it difficult to track imports especially when many different modules need to be imported especially when JAR files and Java dependencies must be set correctly beforehand.

Another major limitation stems from Apache Kafka's inability to change partitions during run-time. Apache Kafka only guarantees ordering within partitions. Therefore, in order for the system to keep its video streams in order, each video stream must be allocated its own partition within the Kafka cluster. Since Kafka does not allow changing the number of partitions of a topic during run-time, it follows that the number of video streams in the system also cannot be changed during run-time and a restart is required whenever video inputs are to be added or removed.

6 Security

Besides the built in security credentials in most IP cameras, this system does not have any added security functions, i.e. authentication, authorisation, etc, besides those included in the third party frameworks used. This system is solely meant to be run within a secure local network environment, and by authorised personnel, therefore the built in security functions provided by the involved frameworks should suffice.

Despite Kafka and HDFS having TLS network encryption, SSL encryption and authentication useful for a clustered network where nodes are connected through the Internet, our project focuses on the performance of the system in performing big data analytics in a local network. Therefore, we did not specifically encrypt our messages and set up user authentication although having such possibilities.

7 Usability and User Experience

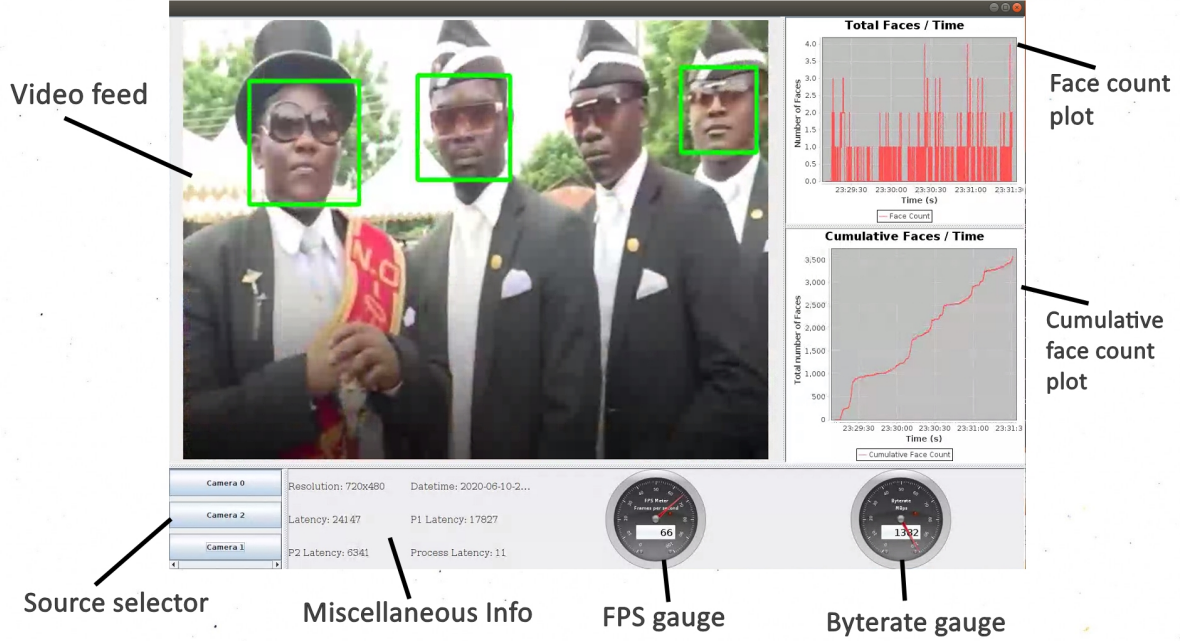


Figure 8: System dashboard

Usability and User Experience mainly on our dashboard is very subjective, but we will still be designing a dashboard that conveys the most important data that the user requires which adheres to the project requirements. All video streams are resized to a predefined resolution to ensure a clear view of the video. The size of the video is not too big to take away space for displaying information but not too small to be hard to view.

The functionality of our dashboard is designed to be as user friendly as possible. The dashboard is designed with a large panel showing the video feed clearly and easily comprehensible graph plot of face count and cumulative face count on the right. All of these charts are clearly labelled and titled to update the user on the latest information. The data analytics displayed on the bottom panel shows clear annotations and are straight to the point. Gauge systems are used to display byte rate and frame per second (FPS) as an indicator to make the dashboard look more interesting and aesthetic.

Besides, users can drag and adjust the panels to their liking and resize them to view the data clearly. The camera buttons on the bottom left section can be scrolled and user can navigate to other video feed by clicking on the buttons. Furthermore, the dashboard features a simple design that focuses on displaying information with minimal clickable items to ensure that the user will not be able to crash the dashboard through misclicks that might affect the entire project system.

The features of the dashboard are user friendly, however, running the whole system can be complicated. Our system requires a number of terminals with specific command to run

the main components of our framework, namely Kafka, Zookeeper, archiver, dashboard, collector and processor that are describe under the technical user guide. Aside from that, as of right now, our current system does not allow user to query data stored in database from the dashboard itself. Data and processed videos can be retrieved from the database shell terminal. Users with no prior knowledge in IT may find it challenging and complicated to run our system.

8 Future Work

8.1 DevOp Tools

Starting the system takes a rather tedious procedure which involves starting a large number of daemons on different terminals which introduces clutter. It would be very useful for developers and technical end users alike to develop a collection of tools and scripts such as bash scripts for Ubuntu Linux and batch files for Windows to assist in these menial tasks.

8.2 Mac OS Compatibility

As previously mentioned, the system has not yet been installed and tested to run on Mac OS. This was due to the hardware limitations at the time of development. Although it is unlikely that the system will be deployed in a Mac OS environment, it would still be appreciated if Mac OS compatibility can be verified for the sake of completeness.

8.3 Decouple Analytical Engine

At the time of writing, the system exhibits a relatively high degree of coupling between the streaming system and the analytical processing engine. Decoupling of the analytical engine from the rest of the system would be useful as it would make it much easier for different analytical engines to be integrated into the system. Our system should ideally also support run-time plug-and-play of different analytical engines. These features make good objectives for future work on the system.

9 Technical User Guide

9.1 Basic Setup and Installation

Setup Software

1. Step 1: Update packages using terminal

```
1  sudo apt-get update
2  sudo apt-get upgrade
```

2. Step 2: Install required softwares:

(a) JDK 8	(c) cmake-gt-gui	(e) git	(g) libgtk2.0-dev
(b) cmake	(d) ant	(f) maven	(h) pkg-config

- (i) libavcodec-dev (j) libavformat-dev (k) libswscale-dev

```
1 sudo apt install openjdk-8-jdk cmake cmake-qt-gui ant git
2 maven libgtk2.0-dev pkg-config libavcodec-dev
3 libavformat-dev libswscale-dev -y
```

3. Step 3: Set Java Home path

- (a) If several java versions have been installed, select "java-8-openjdk" using command below:

```
1 sudo update-alternatives --config javac
```

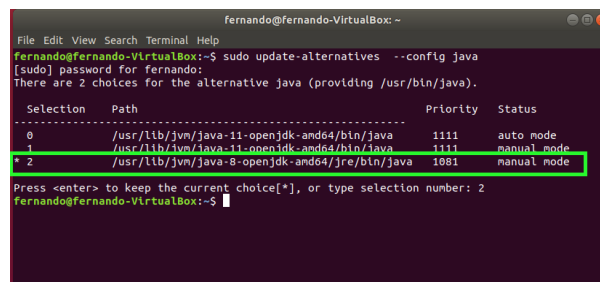


Figure 9: Java version

- (b) Use gedit to edit environment file on "/etc/environment"

```
1 sudo gedit /etc/environment
```

- (c) Append code below to the end of the file

```
1 JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64"
```

4. Step 4: Install Kafka Version 2.4.1

- (a) Create "kafka" folder on "/home/<user>"

```
1 sudo mkdir /home/<user>/kafka
```

- (b) Download and decompress Kafka to "/home/kafka" from:

https://www.apache.org/dyn/closer.cgi?path=/kafka/2.4.1/kafka_2.12-2.4.1.tgz

5. Step 5: Install Hadoop Version 3.2.1

- (a) Create "hadoop" folder on "/home/<user>"

```
1 sudo mkdir /home/<user>/hadoop
```

- (b) Download and decompress hadoop to "/home/hadoop" from:

<https://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-3.2.1/hadoop-3.2.1.tar.gz>

(c) Configure Hadoop environment variables

```
1 sudo nano ~/.bashrc
```

append following content into the end of the file

```
1 #Hadoop Related Options
2 export HADOOP_HOME=/home/<user>/hadoop/hadoop-3.2.1
3 export HADOOP_INSTALL=$HADOOP_HOME
4 export HADOOP_MAPRED_HOME=$HADOOP_HOME
5 export HADOOP_COMMON_HOME=$HADOOP_HOME
6 export HADOOP_HDFS_HOME=$HADOOP_HOME
7 export YARN_HOME=$HADOOP_HOME
8 export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
9 export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
10 export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
```

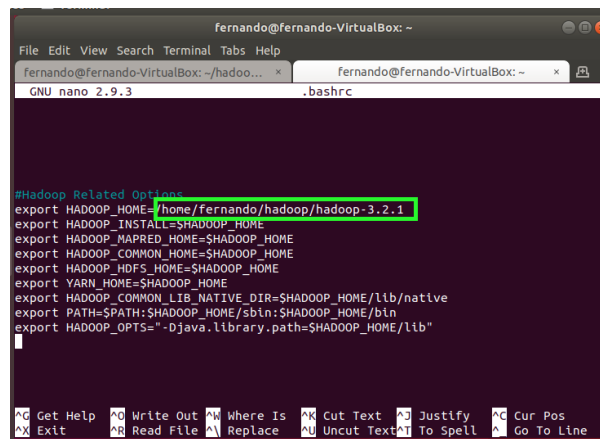


Figure 10: .bashrc

press ctrl+x and enter to save the file

(d) Apply changes to current running environment by executing following command:

```
1 source ~/.bashrc
```

(e) Configure hadoop-env.sh file using following command:

```
1 sudo nano $HADOOP_HOME/etc/hadoop/hadoop-env.sh
```

append following command into the file:

```
1 export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

```

fernando@fernando-VirtualBox: ~
File Edit View Search Terminal Tabs Help
fernando@fernando-VirtualBox: ~/hadoo... x fernando@fernando-VirtualBox: ~
/home/fernando/hadoop/hadoop-3.2.1/etc/hadoop/hadoop-env.sh Modified

#
# To prevent accidents, shell commands be (superficially) locked
# to only allow certain users to execute certain subcommands.
# It uses the format of (command)_(subcommand)_USER.
#
# For example, to limit who can execute the namenode command,
# export HDFS_NAMENODE_USER=hdfs

export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64

```

Figure 11: hadoop-env.sh configuration

press ctrl+x and enter to save the file

(f) Configure core-site.xml file using following command:

```
1 sudo nano $HADOOP_HOME/etc/hadoop/core-site.xml
```

append following configuration into the end of the file:

```

1 <configuration>
2 <property>
3   <name>hadoop.tmp.dir</name>
4   <value>/home/<user>/tmpdata</value>
5 </property>
6 <property>
7   <name>fs.default.name</name>
8   <value>hdfs://127.0.0.1:9000</value>
9 </property>
10 </configuration>

```

```

fernando@fernando-VirtualBox: ~
File Edit View Search Terminal Tabs Help
fernando@fernando-VirtualBox: ~/hadoo... x fernando@fernando-VirtualBox: ~
/home/fernando/hadoop/hadoop-3.2.1/etc/hadoop/core-site.xml

<property>
  <name>hadoop.tmp.dir</name>
  <value>/home/fernando/tmpdata</value>
</property>
<property>
  <name>fs.default.name</name>
  <value>hdfs://127.0.0.1:9000</value>
</property>
</configuration>

```

Figure 12: core-site.xml configuration

press ctrl+x and enter to save the file

(g) Configure hdfs-site.xml file using following command:

```
1 sudo nano $HADOOP_HOME/etc/hadoop/hdfs-site.xml
```

append following configuration into the end of the file:

```
1 <configuration>
2 <property>
3   <name>dfs.data.dir</name>
4   <value>/home/<user>/dfsdata/namenode</value>
5 </property>
6 <property>
7   <name>dfs.data.dir</name>
8   <value>/home/<user>/dfsdata/datanode</value>
9 </property>
10 <property>
11   <name>dfs.replication</name>
12   <value>1</value>
13 </property>
14 </configuration>
```

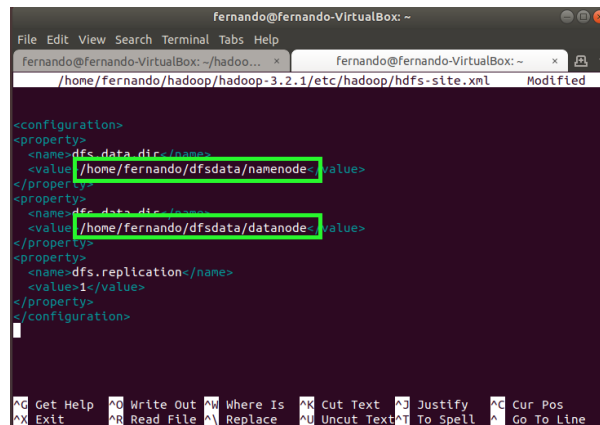


Figure 13: hdfs-site.xml configuration

press ctrl+x and enter to save the file

(h) Configure mapred-site.xml file using following command

```
1 sudo nano $HADOOP_HOME/etc/hadoop/mapred-site.xml
```

append following configuration into the end of the file:

```
1 <configuration>
2 <property>
3   <name>mapreduce.framework.name</name>
4   <value>yarn</value>
5 </property>
6 </configuration>
```

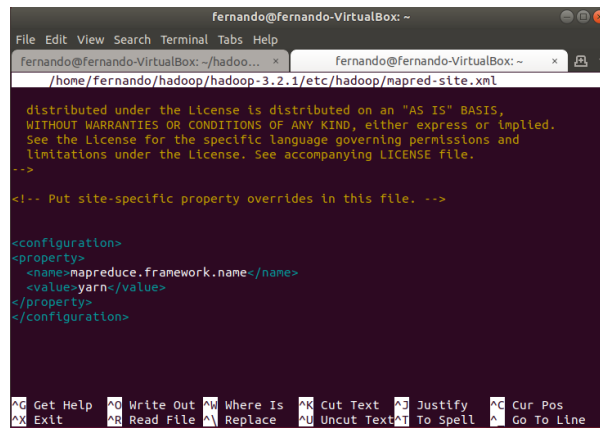



Figure 14: mapred-site.xml configuration

press ctrl+x and enter to save the file

- (i) Configure yarn-site.xml file using following command

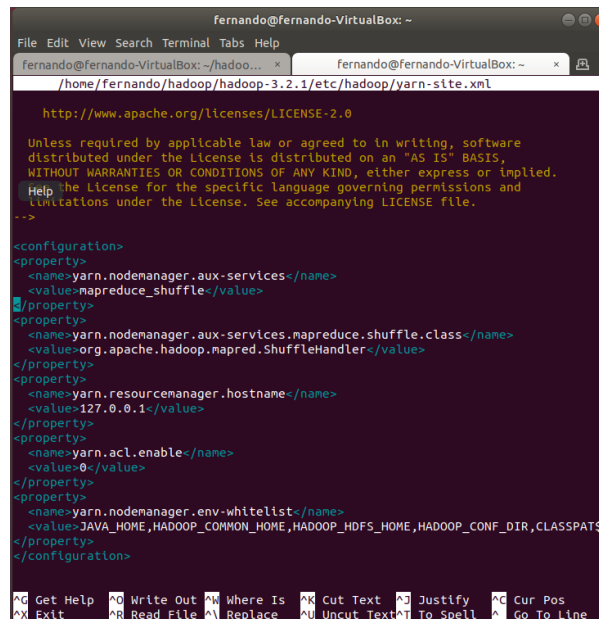
```
1 sudo nano $HADOOP_HOME/etc/hadoop/yarn-site.xml
```

append following configuration into the end of the file:

```

1 <configuration>
2 <property>
3   <name>yarn.nodemanager.aux-services</name>
4   <value>mapreduce_shuffle</value>
5 </property>
6 <property>
7   <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
8   <value>org.apache.hadoop.mapred.ShuffleHandler</value>
9 </property>
10 <property>
11   <name>yarn.resourcemanager.hostname</name>
12   <value>127.0.0.1</value>
13 </property>
14 <property>
15   <name>yarn.acl.enable</name>
16   <value>0</value>
17 </property>
18 <property>
19   <name>yarn.nodemanager.env-whitelist</name>
20   <value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,
21   HADOOP_CONF_DIR,CLASSPATH_PERPEND_DISTCACHE,
22   HADOOP_YARN_HOME,HADOOP_MAPRED_HOME</value>
23 </property>
24 </configuration>

```



```
fernando@fernando-VirtualBox: ~  
File Edit View Search Terminal Tabs Help  
fernando@fernando-VirtualBox: ~/hadoo... x fernando@fernando-VirtualBox: ~  
/home/fernando/hadoop/hadoop-3.2.1/etc/hadoop/yarn-site.xml  
  
http://www.apache.org/licenses/LICENSE-2.0  
  
Unless required by applicable law or agreed to in writing, software  
distributed under the License is distributed on an "AS IS" BASIS,  
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
the License for the specific language governing permissions and  
limitations under the License. See accompanying LICENSE file.  
-->  
  
<configuration>  
<property>  
<name>yarn.nodemanager.aux-services</name>  
<value>mapreduce_shuffle</value>  
</property>  
<property>  
<name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>  
<value>org.apache.hadoop.mapred.ShuffleHandler</value>  
</property>  
<property>  
<name>yarn.resourcemanager.hostname</name>  
<value>127.0.0.1</value>  
</property>  
<property>  
<name>yarn.acl.enable</name>  
<value>0</value>  
</property>  
<property>  
<name>yarn.nodemanager.env-whitelist</name>  
<value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,HADOOP_CONF_DIR,CLASSPATHS  
</property>  
</configuration>  
  
Get Help Write Out Where Is Cut Text Justify Cur Pos  
Exit Read File Replace Uncut Text To Spell Go To Line
```

Figure 15: yarn-site.xml configuration

press ctrl+x and enter to save the file

- (j) Format HDFS namenode using command below:

```
1 hdfs namenode --format
```

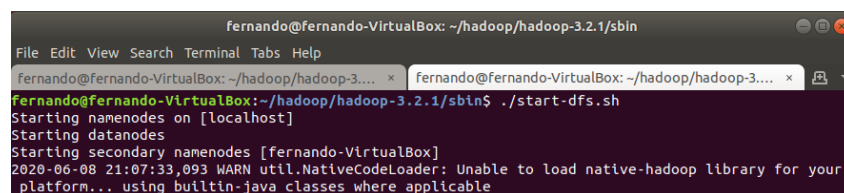
- (k) Start hadoop cluster

Open 2 terminals inside:

```
1 /home/<user>/hadoop/hadoop-3.2.1/sbin
```

- i. Start NameNode and DataNode using command below:

```
1 ./start-dfs.sh
```

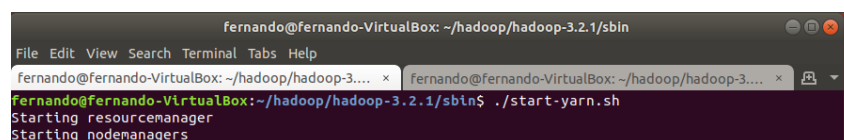


```
fernando@fernando-VirtualBox: ~/hadoop/hadoop-3.2.1/sbin  
File Edit View Search Terminal Tabs Help  
fernando@fernando-VirtualBox: ~/hadoop/hadoop-3.2.1/sbin$ ./start-dfs.sh  
Starting namenodes on [localhost]  
Starting datanodes  
Starting secondary namenodes [fernando-VirtualBox]  
2020-06-08 21:07:33,093 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your  
platform... using builtin-java classes where applicable
```

Figure 16: start namenode and datanode

- ii. start YARN resource and node managers by typing:

```
1 ./start-yarn.sh
```



```
fernando@fernando-VirtualBox: ~/hadoop/hadoop-3.2.1/sbin  
File Edit View Search Terminal Tabs Help  
fernando@fernando-VirtualBox: ~/hadoop/hadoop-3.2.1/sbin$ ./start-yarn.sh  
Starting resourcemanager  
Starting nodemanagers
```

Figure 17: start node managers

- iii. Finally, ensure that all daemons are running using a simple command below:

```
1 jps
```

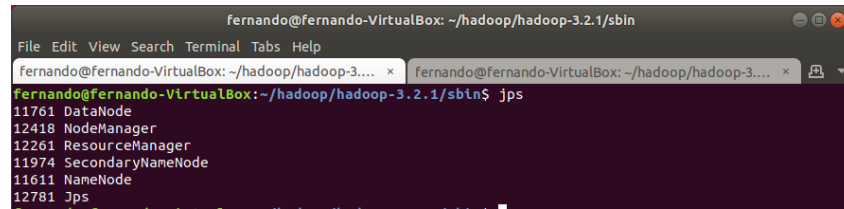


Figure 18: hadoop is working as intended

6. Step 6: Install OpenCV Version 3.4

- (a) Create "opencv" folder on "/home/<user>"

```
1 sudo mkdir /home/<user>/opencv
```

- (b) Download and decompress OpenCV 3.4 to "/home/opencv" from:
<https://github.com/opencv/opencv/tree/3.4>

- (c) Create a folder named "build" inside decompressed OpenCV 3.4

```
1 sudo mkdir /home/<user>/opencv/opencv-3.4/build
```

- (d) Open Terminal inside "build" directory

```
1 cmake-gui
```

- (e) Set the source into "/home/<user>/opencv/opencv-3.4"

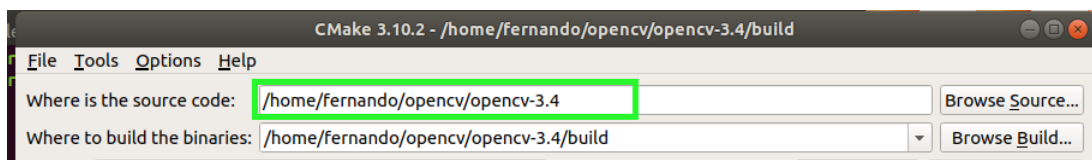


Figure 19: cmake-gui source path

- (f) Set the destination into "/home/<user>/opencv/opencv-3.4/build"

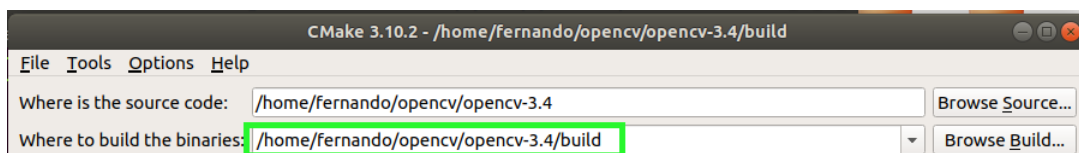


Figure 20: cmake-gui destination path

- (g) Check "Grouped" and "Advanced"

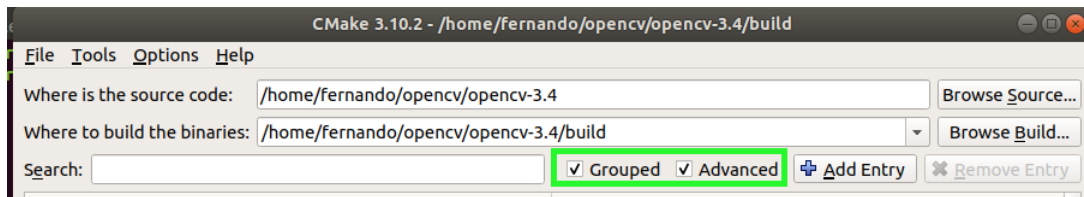


Figure 21: cmake-gui checked "Grouped" and "Advanced" box

(h) Press "Configure"

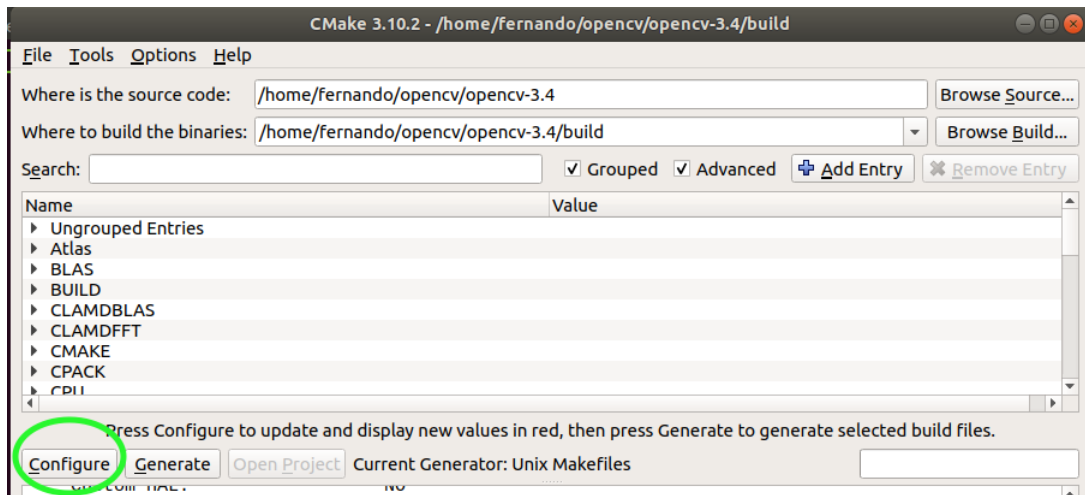


Figure 22: Configure button location

(i) Under "JAVA", ensure that the values are the same as figure below:

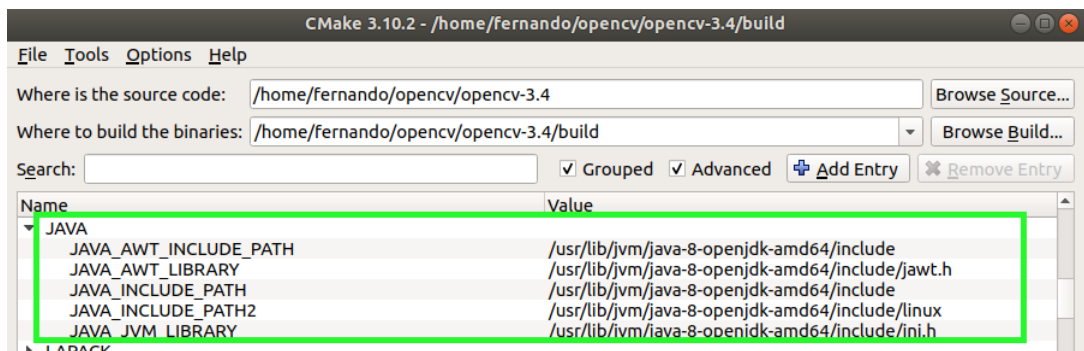


Figure 23: Valid "JAVA" values

- (j) Under "BUILD" ensure that names below are unchecked
- BUILD_PERF_TEST
 - BUILD_SHARED_LIBS
 - BUILD_opencv_python_bindings_generator
 - BUILD_opencv_python_tests

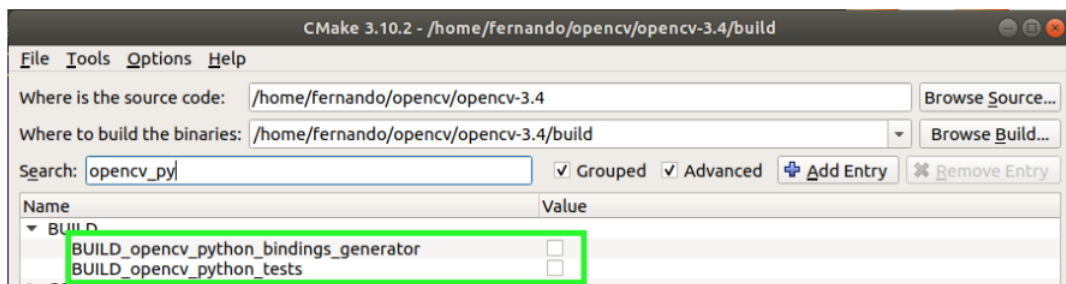
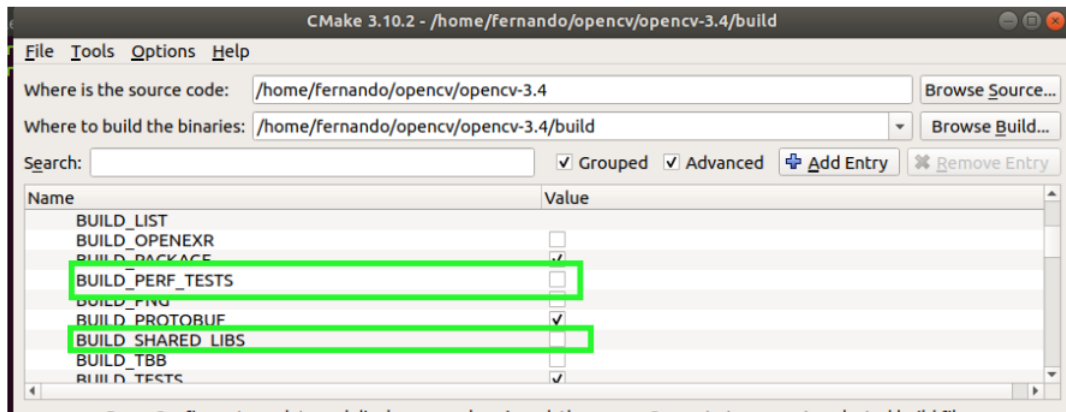


Figure 24: Unchecked "BUILD" values

(k) Under "Ungrouped Entries" ensure that values is `/usr/bin/ant`

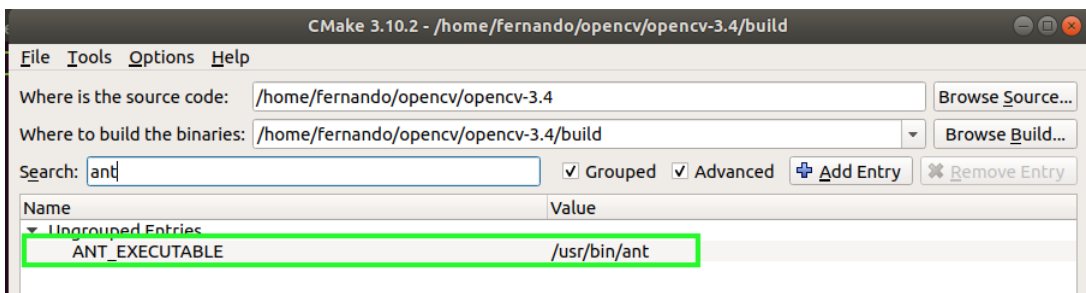


Figure 25: Valid "Ungrouped entries" value

(l) Press "Configure" button Twice and ensure that no entries are red

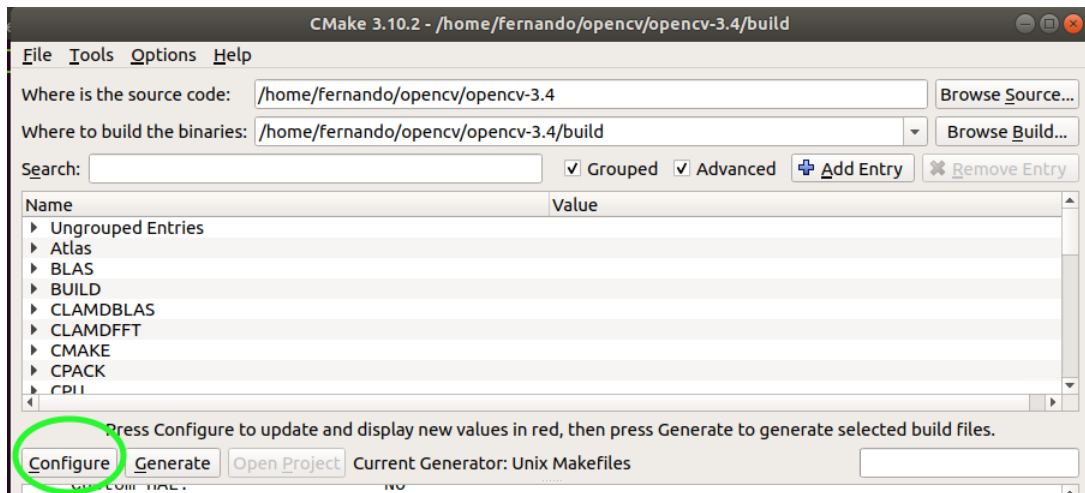


Figure 26: "Configure" button

- (m) Press "Generate" button and close cmake-gui

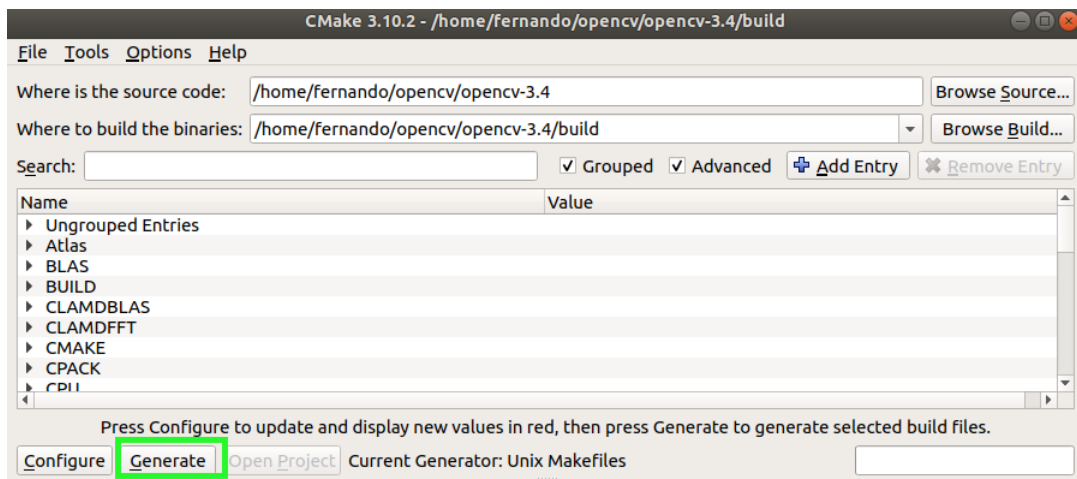


Figure 27: "Generate" button

- (n) Open terminal inside

```
1 /home/<user>/opencv/opencv-3.4/build
```

- (o) Execute following command (-j# where # is the number of processes to make)

```
1 make -j8
```

7. Step 7: Install MongoDB

- (a) Import public key using command below:

```
1 wget -qO - https://www.mongodb.org/static/pgp/server-4.2.asc |
2 sudo apt-key add -
```

- (b) Open a terminal and execute following command

- i. If Ubuntu 18.0.4 (Bionic)

```
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/
apt/ubuntu bionic/mongodb-org/4.2 multiverse" |
sudo tee /etc/apt/sources.list.d/mongodb-org-4.2.list
```

ii. If Ubuntu 16.04 (Xenial)

```
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/
apt/ubuntu xenial/mongodb-org/4.2 multiverse" |
sudo tee /etc/apt/sources.list.d/mongodb-org-4.2.list
```

(c) Reload local package database

```
1 sudo apt-get update
```

(d) Install the latest version by executing following command:

```
1 sudo apt-get install -y mongodb-org
```

8. Step 8: Git Clone AVS directory into /home/<user>/Documents

<https://github.com/yhoo0007/FIT3161-Team-1C>

```
1 git clone https://github.com/yhoo0007/FIT3161-Team-1C
```

9.2 Running The Software

1. Step 1: Start Zookeeper

(a) Open terminal inside

```
1 /home/<user>/kafka/kafka_2.12-2.4.1
```

(b) Start Zookeeper using following command:

```
1 bin/zookeeper-server-start.sh config/zookeeper.properties
```

2. Step 2: Start Kafka

(a) (Optional) Change the number of partitions to match the number of inputs in config/server.properties file. Under Log Basics change num.partitions to the number of partitions desired.

(b) Open another terminal inside

```
1 /home/<user>/kafka/kafka_2.12-2.4.1
```

(c) Start Kafka using following command:

```
1 bin/kafka-server-start.sh config/server.properties
```

3. Step 3: Start Hadoop (HDFS)

(a) Open two terminals inside

```
1 /home/<user>/hadoop/hadoop-3.2.1
```

(b) Start Hadoop using following commands:

```
1 terminal 1 : $HADOOP_HOME/sbin/start-dfs.sh
2 terminal 2 : $HADOOP_HOME/sbin/start-yarn.sh
```

4. Step 4: Start MongoDB

Open a terminal and start MongoDB using following command:

```
1 sudo systemctl start mongod
```

5. Step 5: Start Archiver

(a) Open terminal inside

```
1 /home/<user>/Documents/avs
```

(b) Start Archiver by executing following command:

```
1 mvn exec:java -Dexec.mainClass=org.team1c.avs.VideoStreamArchiver
```

6. Step 6: Start Dashboard

(a) Open terminal inside

```
1 /home/<user>/Documents/avs
```

(b) Start Dashboard by executing following command:

```
1 mvn exec:java -Dexec.mainClass=org.team1c.avs.VideoStreamDashboard
```

7. Step 7: Start Processor

(a) Open terminal inside

```
1 /home/<user>/Documents/avs
```

(b) Start Processor by executing following command:

```
1 mvn exec:java -Dexec.mainClass=org.team1c.avs.VideoStreamProcessor
```

8. Step 8: Start Collector

(a) Open terminal inside

```
1 /home/<user>/Documents/avs
```

(b) Start Collector by executing following command:

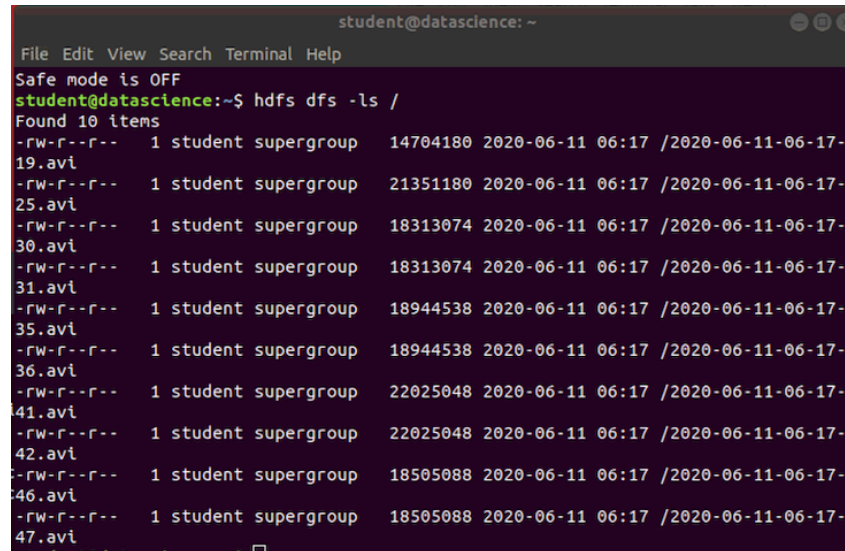
```
1 mvn exec:java -Dexec.mainClass=org.team1c.avs.VideoStreamCollector
```


9.3 Accessing Database

1. Accessing batched avi file (Hadoop)

(a) Open terminal and check the file inside hadoop using comand below:

```
1 hdfs dfs -ls /
```



```
student@datascience: ~  
File Edit View Search Terminal Help  
Safe mode is OFF  
student@datascience:~$ hdfs dfs -ls /  
Found 10 items  
-rw-r--r-- 1 student supergroup 14704180 2020-06-11 06:17 /2020-06-11-06-17-19.avi  
-rw-r--r-- 1 student supergroup 21351180 2020-06-11 06:17 /2020-06-11-06-17-25.avi  
-rw-r--r-- 1 student supergroup 18313074 2020-06-11 06:17 /2020-06-11-06-17-30.avi  
-rw-r--r-- 1 student supergroup 18313074 2020-06-11 06:17 /2020-06-11-06-17-31.avi  
-rw-r--r-- 1 student supergroup 18944538 2020-06-11 06:17 /2020-06-11-06-17-35.avi  
-rw-r--r-- 1 student supergroup 18944538 2020-06-11 06:17 /2020-06-11-06-17-36.avi  
-rw-r--r-- 1 student supergroup 22025048 2020-06-11 06:17 /2020-06-11-06-17-41.avi  
-rw-r--r-- 1 student supergroup 22025048 2020-06-11 06:17 /2020-06-11-06-17-42.avi  
-rw-r--r-- 1 student supergroup 18505088 2020-06-11 06:17 /2020-06-11-06-17-46.avi  
-rw-r--r-- 1 student supergroup 18505088 2020-06-11 06:17 /2020-06-11-06-17-47.avi
```

Figure 28: Batched avi list files

(b) Batched avi files are located inside:

```
1 /home/<user>/dfsdata/datanode/current/BP-1752091259-118.139.135.57  
2 -1591725016986/current/finalized/subdir0/subdir0
```

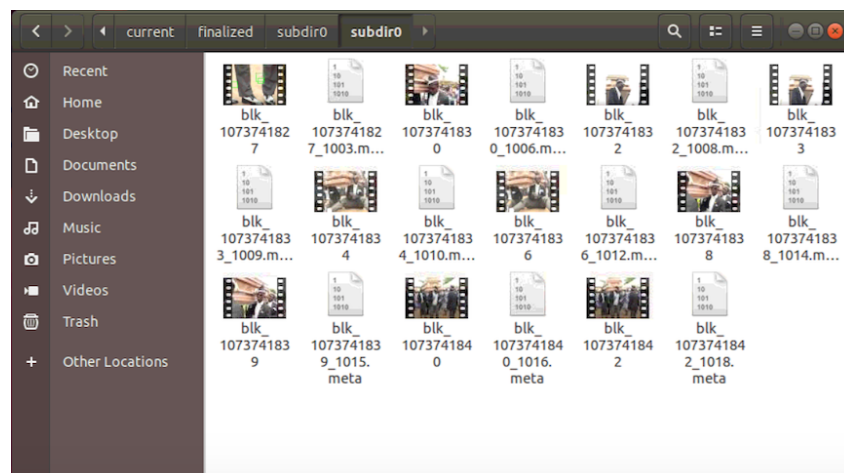


Figure 29: Batched ".avi" list files

2. Accessing Video Information and Data (MongoDB)

(a) Open terminal and access MongoDB shell using following command:

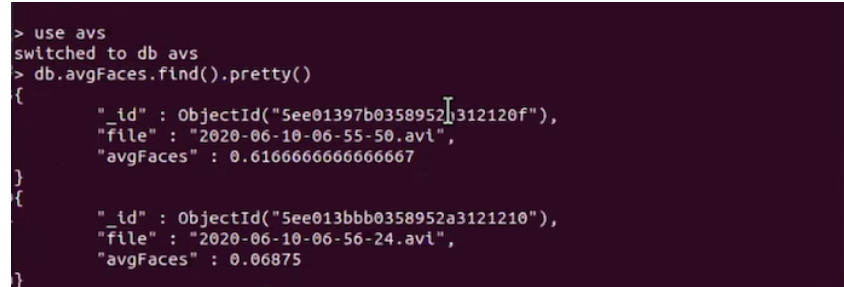
```
1 mongo
```

(b) Select "avs" database using following shell command:

```
1 use avs
```

(c) to query data inside avs database, type shell command below:

```
1 db.avgFaces.find().pretty()
```



```
> use avs
switched to db avs
> db.avgFaces.find().pretty()
{
  "_id" : ObjectId("5ee01397b0358952a312120f"),
  "file" : "2020-06-10-06-55-50.avi",
  "avgFaces" : 0.6166666666666667
}
{
  "_id" : ObjectId("5ee013bbb0358952a3121210"),
  "file" : "2020-06-10-06-56-24.avi",
  "avgFaces" : 0.06875
}
```

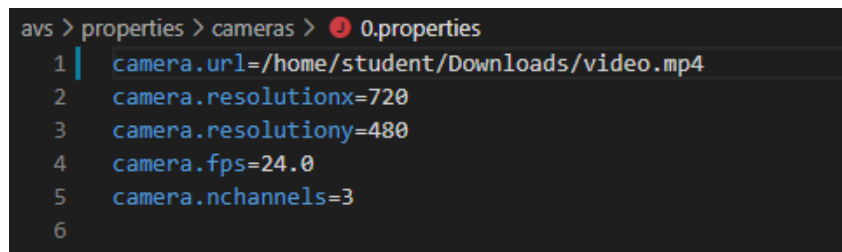
Figure 30: MongoDB shell commands

10 End User Guide

After setting up all required frameworks and software to run the project, it is recommended to change all the settings such as file paths in order to play the correct video inputs due to the different installation paths for every user. We will look at how to set the different settings for all configuration files provided by the project.

10.1 Camera Properties

Based on Figure 31, we can see that the property file for a video input is very simple. First and foremost, the filename of the property file should match the id of the camera id to ensure the system finds the video input property file via the id. Then, just change the camera.url to the correct file path of the video or camera the user is expected to stream. The other settings represent the resolution, fps and nchannels to be passed through the framework.



```
avs > properties > cameras > 0.properties
1 camera.url=/home/student/Downloads/video.mp4
2 camera.resolutionx=720
3 camera.resolutiony=480
4 camera.fps=24.0
5 camera.nchannels=3
6
```

Figure 31: Camera Properties

10.2 Archiver Properties

Bootstrap.servers represents the Kafka servers that the archiver consumer will connect to. Group Ids, serializers and Kafka topics are already set by default of our project which is not recommended to change in order to maintain consistency throughout the

architecture. Finally, MongoDB settings can be changed accordingly such as database and collection used to suit the user.

```
avs > properties > archiver.properties
1 bootstrap.servers=localhost:9092
2 group.id=frame-writing
3 key.deserializer=org.apache.kafka.common.serialization.StringDeserializer
4 value.deserializer=org.apache.kafka.common.serialization.StringDeserializer
5 max.poll.records=10
6 enable.auto.commit=false
7 auto.offset.reset=earliest
8 kafka.topic=processed-frames
9 accumulate.duration=20
10 mongodb.host=127.0.0.1
11 mongodb.port=27017
12 mongodb.db=avs
13 mongodb.collection=avgFaces
14
15
```

Figure 32: Archiver Properties

10.3 Collector Properties

Collectors are Kafka producers which will create a Kafka producer for each video input and send data through the Kafka cluster accordingly. Kafka producer settings are not recommended to be changed to ensure the optimal settings for latency and throughput unless the user understands what they are doing. Partitioner classes can be changed to encode ids to different partitions while the most important setting, camera.id, will be the number of inputs of videos and their respective ids. Eg. if there is one input video of id 0, set camera.id=0, if there are tree input videos of ids 0, 1 and 2, set camera.id=0,1,2 (separated by commas).

```
avs > properties > collector.properties
1 bootstrap.servers=localhost:9092
2 acks=1
3 retries=1
4 batch.size=30000000
5 linger.ms=100
6 max.request.size=50000000
7 kafka.topic=video-input
8 compression.type=none
9 camera.id=0,1,2
10 camera.retries=1
11 key.serializer=org.apache.kafka.common.serialization.StringSerializer
12 value.serializer=org.apache.kafka.common.serialization.StringSerializer
13 partitioner.class=org.team1c.avs.AvsCustomPartitioner
14
```

Figure 33: Collector Properties

10.4 Dashboard Properties

Since dashbaords are a Kafka consumer by itself, it will have its own property file to set the consumer properties to ensure that the consumer is consumer in the correct Kafka topic. It is not recommended to change these settings to ensure topics and groups are correctly defined.

```

avs > properties > dashboard.properties
1 bootstrap.servers=localhost:9092
2 group.id=dashboard
3 key.deserializer=org.apache.kafka.common.serialization.StringDeserializer
4 value.deserializer=org.apache.kafka.common.serialization.StringDeserializer
5 max.poll.records=50
6 enable.auto.commit=false
7 auto.offset.reset=earliest
8 kafka.topic=processed-frames
9
10

```

Figure 34: Dashboard Properties

10.5 Processor Properties

Processors are Kafka consumer and Kafka producer combined into one node. Therefore, there will be two property files one for consumer properties and the other for producer properties. This is the same as with the consumers and producers of collector and dashboard. However, to help in scalability, consistency and modularity, Kafka processors can be reused for other Kafka topics and other Kafka groups when it is required to further develop and expand the project.

```

avs > properties > processor-consumer.properties
1 bootstrap.servers=localhost:9092
2 group.id=frame-processing
3 key.deserializer=org.apache.kafka.common.serialization.StringDeserializer
4 value.deserializer=org.apache.kafka.common.serialization.StringDeserializer
5 max.poll.records=50
6 enable.auto.commit=false
7 auto.offset.reset=earliest
8 kafka.topic=video-input
9
10

```

```

avs > properties > processor-producer.properties
1 bootstrap.servers=localhost:9092
2 acks=1
3 retries=1
4 batch.size=30000000
5 linger.ms=100
6 max.request.size=50000000
7 kafka.topic=processed-frames
8 compression.type=None
9 key.serializer=org.apache.kafka.common.serialization.StringSerializer
10 value.serializer=org.apache.kafka.common.serialization.StringSerializer
11 partitioner.class=org.team1c.avs.AvsCustomPartitioner
12
13

```

(a) Processor Consumer Properties

(b) Processor Producer Properties

10.6 OpenCV and Maven Setup

If you have installed OpenCV locally following the Technical User Guide, there will be two ways tell the system to locate OpenCV.

Firstly, you can add local OpenCV to Local Maven Repository. Open a terminal and enter the code below:

```

1 mvn install:install-file -Dfile=<path-to-opencvJAR> -DgroupId=<org.opencv> \
2 -DartifactId=<opencv> -Dversion=<version_number> -Dpackaging=<jar> \
3 -DgeneratePom=true

```

Then, update the dependency accordingly to the parameters you have set in mvn install in the pom.xml file under FIT3161-Team-1C/avs as shown in Figure 36 and then run mvn clean package.

```
<dependency>
  <groupId>org.opencv</groupId>
  <artifactId>opencv</artifactId>
  <version>4.2.0</version>
</dependency>
```

Figure 36: OpenCV Maven POM Dependency

If Maven does not detect the local OpenCV, change the dependency shown in Figure 36 to the following as shown in Figure 37:

```
<dependency>
  <groupId>org.opennpn</groupId>
  <artifactId>opencv</artifactId>
  <version>3.4.2-2</version>
</dependency>
```

Figure 37: OpenCV Central Maven POM Dependency

However, this version of OpenCV that is available in central Maven repository does not support FFMPEG which is crucial in reading and writing video files. Therefore, to ensure the OpenCV that is loaded is the one you have installed locally, apply the following changes to the following files.

- VideoEventGenerator.java
- VideoStreamDashboard.java
- VideoStreamProcessor.java
- VideoStreamWriter.java

Comment the following code in the class files

```
static { System.loadLibrary(Core.NATIVE_LIBRARY_NAME); }
```

The resulting code should look like this:

```
// static { System.loadLibrary(Core.NATIVE_LIBRARY_NAME); }
static{System.load("/OpenCV/build/lib/libopencv_java#.so");}
```

For VideoStreamProcessor.java class files only, apply the above change along with changing the filepath to the installed OpenCV haarcascade_frontalface_alt.xml. Eg:

```
public static final String HAAR_CASCADE_FP =
  "OpenCV/data/haarcascades/haarcascade_frontalface_alt.xml";
```

10.7 Starting the System

Assuming that the system has been set up correctly beforehand, running the system is a matter of starting all the required daemons in a compatible order. Ideally, all third party applications (Zookeeper, Kafka, HDFS, MongoDB) should be started first in any order. Next, the video stream processors, dashboard, and archivers should be started and be ready for incoming video frames. Lastly the producers should be started which will start introducing data into the system. This particular order ensures that each component can be put into their ready state immediately upon startup and be ready for streaming. The producers should ideally be started up last such that the rest of the system does not need to do any 'catching up'. This results in the best real-time performance of the system.

10.8 Operating the System

Non-technical operation of the system is fairly straightforward as all the interaction is done via the system dashboard as shown in Figure 8.

The video feed of the selected source on the top left takes up most of the dashboard. To its right are two live plots of the number of faces in the current frame (top right) and the cumulative face count (bottom right). On the bottom left there is a scroll panel with buttons for selecting different video sources available. The number of buttons in this scroll panel with correspond to the number of partitions for video frames in the Kafka cluster. Finally on the bottom right there are displays for the frame metadata and other miscellaneous information. The panels can also be resized by clicking and dragging on the borders.

10.9 Shutting Down the System

Shutting down of the system is simply a matter of terminating each daemon in the reverse order as which they should be started. The producers should be terminated first to stop the inflow of new data into the system. Next, the video stream processors, dashboard, and archivers can be terminated. Finally, each of the third party applications can be terminated accordingly. It is generally better to allow the rest of the system to finish processing frames after the producers have been terminated before terminating them. Although Kafka does maintain consumer offsets, doing this will ensure that the system is on the same page before terminating.

References

- Google. (2020). *Google java style guide*. Retrieved from <https://google.github.io/styleguide/javaguide.html>
- Microsoft. (2020). *Code metric values*. Retrieved from <https://docs.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values?view=vs-2019>