ICS5114 Project
Big Data Processing
Speech Analytics On Live Streams

Gabriel Sammut
gabriel.sammut.13@um.edu.mt

Nicholas Frendo
nicholas.frendo.05@um.edu.mt

Academic Year 2018/2019

1 Abstract

The term 'Big Data' is often disguised under a range of shifting and evolving terms utilized to refer to the act of housing and processing of vast amounts of data. This endeavour however does not correlate only to the ever growing, large volumes of datasets, but is also used to refer to the rate of growth and the velocity at which it presents itself with. In addition, data of such vast speeds can also assume a myriad of forms, making it a challenge to effectively capture, process and produce useful information [1]. This scientific paper deals with issues presented by the velocity and variety aspects of Big Data, particularly to data of a live stream nature concerning video and text capture. This paper follows up to present possible scenarios and architectures to tackle the issues presented above and concludes with an evaluated benchmark of the proposed system.

2 Introduction

This scientific report attempts to encompass and document the work carried out to design and implement a high volume, multi-variety, distributed, data-processing system. The report will first address the presented challenges imposed by Big Data, as well as identify the two main objectives set out to achieve with the proposed system:

- Variety
- Velocity

The report then follows up with a brief literature review with particular focus on literature focussed on resolving the proposed challenges. Eventually, the proposed design and architecture will be expanded upon, with particular highlights on the type of technologies utilized, and the reasons behind such choices. Eventually, the paper addresses two particular milestones, which serve to test the proposed system under load, particularly addressing content originating from various data streams. The paper concludes with an evaluation of the proposed system, detailing and contrasting the results of each milestone run and final thoughts with respect to problems encountered and the way they were tackled.

A detailed mapping of all referenced figures throughout the report can be found in the appendix section, for the readers perusal.

3 Problem Definition

The following section addresses the many facets of the Big Data challenge, continuing with an explanation of the envisioned data processing system.

3.1 Variety

A congruent, well known challenge to the aspect of Big Data processing, results from data being produced from multiple sources, each source harnessing data of the same nature, but with different form and structure. In addition, the problem of variety extends further under varied guises, most predominantly; cases where data is being presented as varied textual, audio, video or cellular formats [2]. With data-producing giant firms serving unquantified amounts of data, the likes of which Facebook and Twitter are capable of generating on a daily basis, there is no common consensus, or agreeable format upon which dictates the transportation, consumption and storage of this data. Essentially, the problem of data variety falls upon the hands of the data consumer to tackle, requiring complex systems to cater for and build around the proposed problem.

3.2 Velocity

Another active challenge of vast quantities of data, is the rate of incoming velocity at which the data presents itself with. The problem ties directly with

the speed at which new data appears into a processing system. If the amount of data presented is not processed in a fast enough manner; that is, the incoming data generation rate is greater than the data processing and consumption rates, system backlogs and delays in data processing are inevitable. This is problematic, since such data would need to be accumulated, or otherwise is lost. In addition, this accumulation of data *traffic* can be problematic in itself, potentially overwhelming the system resources at hand if not catered to efficiently process and clear high velocity data [2].

Although the main problem lies within the remit of processing volumes of data at high speed, another potential pitfall relates to the over-provisioning of resources. It is common, that the rate of incoming data varies, depending on demand. This is difficult to plan around, requiring the data-processing system to be flexible with respect to utilized resources and incoming data. This poses a problem since any capable system must be able to plan around high intensive loads, whilst still ensuring that they operate efficiently without incurring additional costs in the case of low incoming data traffic.

3.3 Dataset Selection

One of the primary challenges of the proposed experiment was to locate a suitable source of high-traffic, rapid and varied streaming source of data. It was concluded that a pre-made dataset would be very hard, since all the following requirements needed to be met for the dataset to be valid for the proposed test scenario:

- Varied Data
    - Varied Data Platforms
    - Varied Data Content
- High Velocity Data
    - Recorded Formats
    - Streaming Formats

The decision to test and evaluate the proposed system on a number of streaming platforms was taken, enabling the system to read from a constant fresh supply of new data, never ceasing in its data output. In addition, a number of varied streaming platforms were targeted, as well as attempting to process incoming data of both audio and textual formats. In particular, the following data sources were utilized for prototype testing, and/or eventual milestone evaluation:

- Twitch.tv [3]
- YouTube [4]
- YouTube Live [5]
- Daily Motion [6]

For the proposed milestone runs, YouTube and Twitch.tv were chosen for *Milestone One* and *Milestone Two* respectively. For both cases, both video and text data was captured and processed by the proposed system.

3.4 Aims and Objectives

Therefore the aims of the proposed systems is two-fold; to tackle and present an efficient, scalable solution to the aforementioned two Vs of Big Data. The proposed architecture has been purposely modelled to handle and process rapid, incoming data streams, thus satisfying the Velocity aspect of big data. In addition, the presented solution also caters for the handling of different video and textual formats, from a number of varied platforms, thus fulfilling the Variety aspect of the presented problem.

The proposed architecture would therefore record all incoming information, both textual and video. In the case of video data streams, an additional transcription step is required in order to convert the audio channel into its textual representation. Subsequently, the resulting data is stored in a graphical network of users, representing viewers and streamers, in relation to word utterance and count. The end result of such a system is expected to behold a number of online users, coming from various platforms, in relation to their respective words uttered, in the form of a graph network for metric evaluation.

4 Background and Related Systems

The aspect of handling rapid, varied and large quantities of incoming data is already coined as a subset of Big Data. Through the widespread use of internet-based applications, huge quantities and varieties of data are churned at unprecedented rates, making it a challenge to harness information from the torrents of incoming data [17, 18]. It is well established that for any proposed architecture to tackle the challenges posed by big data, it must conform with the following standards [18, 19]:

- *Performance*, speed with which to process data as fast as possible.

- *Scalability*, requiring the architecture to be flexible enough to grow on demand.
- *Fault Tolerance*, ensuring that enough redundancy is present to maintain optimum up-time rates, therefore reducing data loss.

These standards play well into the CAP Theorem perspective [20], imposing further challenges to maintain an architecture capable of fulfilling all three subsets.

A number of Big Data oriented frameworks are well established in the field, with processing platforms such as the batch processing Hadoop toolset having achieved great success in this particular domain [18, 21]. However, the MapReduce paradigm leaves much to be desired with respect to the challenges posed by high velocity streams, due to it's batch-like nature. This limitation is further emphasized for time-sensitive data stream processing, particularly that of real-time video and audio data capture.

Particular notable mentions include Yahoo!S4, coined as 'a general-purpose, distributed, scalable, fault-tolerant, pluggable platform', created for the purpose of unbounded data stream processing [18]. Apache Kafka fits well in this bracket, allowing ceaseless interaction between distributed, communicating modules, allowing them to operate under a publish-subscribe design pattern [7]. Another component of the Apache toolset is the Spark framework [22], promising fast and stream capable performance, as well as offering seamless integration for machine learning components.

It is however Apache Storm which fully dedicates itself to the challenge presented by velocity of incoming data [8]. Compared to the well established Hadoop MapReduce architecture, Storm makes it easy to reliably integrate with unbounded streams of data, whilst still offering the capacity for large data volumes. For the variety aspect of Big Data, Storm tackles this through the use of stream repartitioning [18], making it a handy tool set in tackling the three major Vs [19, 20].

5 Architecture - Big Data Design

The following section incorporates design decisions concerning the proposed architecture. It then proceeds by taking a closer look into each component of the proposed architecture in further detail.

5.1 Overview

The presented solution is essentially a distributed system, operating over a cloud based network, composed of at least four concurrent running nodes. Each node forms an essential part to what is essentially a pipelined process, with each node being fed data, processes the data and passes it onto the subsequent processing node. The following key, architectural terms will be referenced throughout the remainder of this report:

- **Producer** – The producer script is custom made written logic utilized to acquire data from a data source, download it locally onto the machine running the process and offload it onto a Kafka [7] broker. The script is capable of handling pre-recorded data similar to that of on-demand YouTube video content, as well as capable of recording streaming, live footage feed.
- **Apache Kafka** - Kafka is a distributed streaming platform, capable of publishing and subscribing to streams of records. The prime concept behind Kafka is loosely modelled on the produce-consume methodology. The architecture offers support for the storage of records in a fault-tolerant and durable manner, capable of streaming records simultaneously depending on the demand of the incoming data streams [7].
- **Apache Storm** – For the reliable processing of unbounded streams, Apache Storm was the ideal choice [8]. Utilizing the Kafka-Spout component, the Storm topology was fed data directly from the Kafka broker. The utility of storm enabled the ease of scaling processing dependent on demand, allowing the storm topology to parallelize each spout/bolt internally for greater processing throughput, as well as offering the capability of distributing work undergoing in the topology over a distributed network of storm workers. Storm also offers a level of fault tolerance, however this configuration was not enforced heavily in the proposed architecture, since all storm components were placed on a single operating machine.
- **Neo4j Graph Platform** – The prime goal of the proposed system is to capture and model words uttered by online users, be it through audio or textual utterance. The

Neo4j graph database, being a high performance graph store whilst retaining the robustness of a mature database [9], was the perfect choice for such a data capture scenario. With heavy focus on relation modelling, insertion and retrieval of related nodes made Neo4j particularly suited for this task. In addition, the innate graph representation enhances the speed at which related data can be retrieved, especially so for larger volumes as the database grows larger. This is a stark contrast to generic relational databases, in which data relationships become complex to model and retrieve using conventional table join methods.

- **Google Cloud Platform** – For video decoding and speech to text functionality, the Cloud Speech-to-text API was utilized, allowing the proposed system to upload video segments via the API, decode them, and extract respective decoded video clippings back into the system [10]. Decoding of video to text is done in parallel to the actual submission of the streaming object through Kafka, and Storm.

The proposed architecture is capable of handling pre-recorded online data, as well as handling real-time streams. The following diagrams provide a holistic view of the proposed system, in light of both recording and streaming modes.

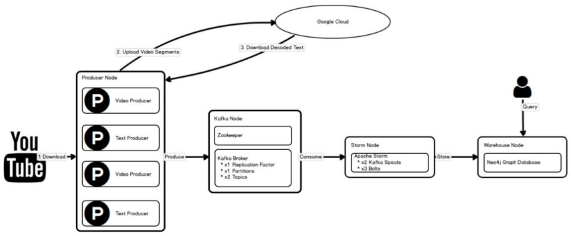

Figure 1 - Holistic Overview - Recording

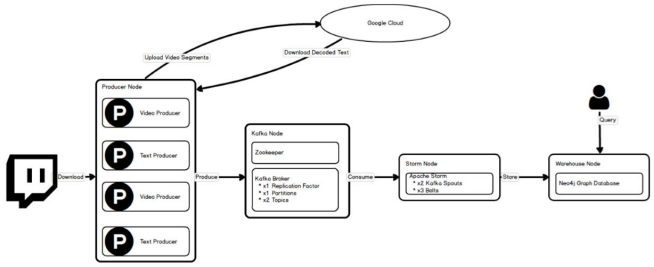

Figure 2 - Holistic Overview - Streaming

5.2 Producer

The producer is tasked to interface with the data source by and downloading the data locally and submitting it onto the pipeline by offloading data into the Kafka broker. The producer script is written in a way to handle various data formats, both textual and video formats. For text capturing, the YouTube API [11] was used to interface with and capture channel comments, by first downloading each respective channel comment thread, followed by each respective comment itself and eventually submitting all textual comments onto a Kafka-topic reserved for text handling. In addition, a second form of text capture was added to the producer, allowing the current functionality to interface with real time live chats produced from Twitch.tv streams, through the use of IRC [12].

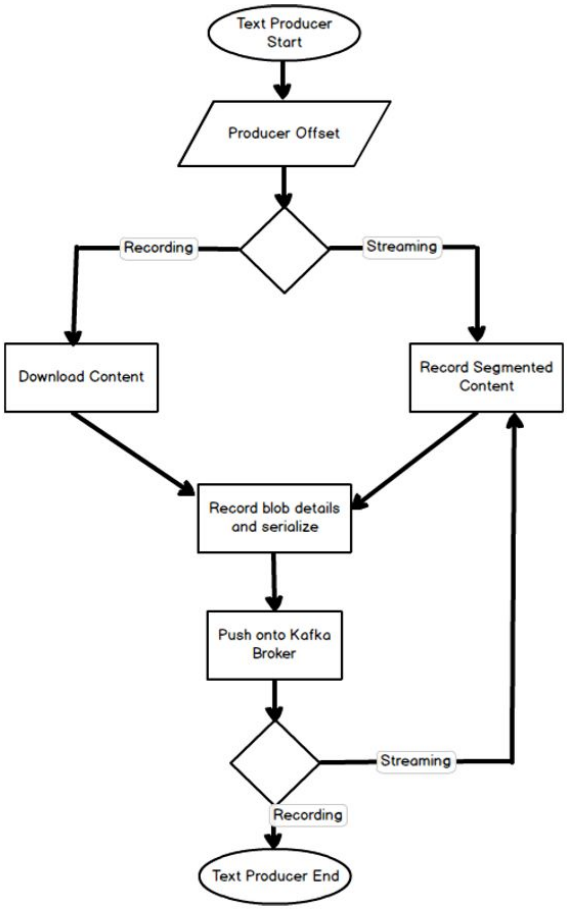

Figure 3 - Producer (Text Processing) Flow

For video capture, a set of wrapper scripts were written into the producer. For pre-recorded video extraction, the pythonic library 'pytube' [13] was used, allowing the producer to download any number of YouTube video footage, segment it locally through usage of ffmpeg [14] into snippets of preconfigured time (preconfigured to thirty seconds by default) and upload each segmented clip onto the Google Cloud Platform. The endpoint of the upload video blob storage is annotated along with several other information criteria and submitted onto the Kafka broker.

For real-time video capture, the streaming tool 'Streamlink' [15] was used through the automated use of the producer. The tool allows real-time stream capturing of video data, particularly tried and tested on Twitch.tv and YouTube-Live. Similar to the pre-recording video logic, Streamlink is used to record live-streams for thirty seconds straight. The resulting segmented clip is uploaded to Google Cloud Platform and submitting the respective uploaded video blob storage url through Kafka.

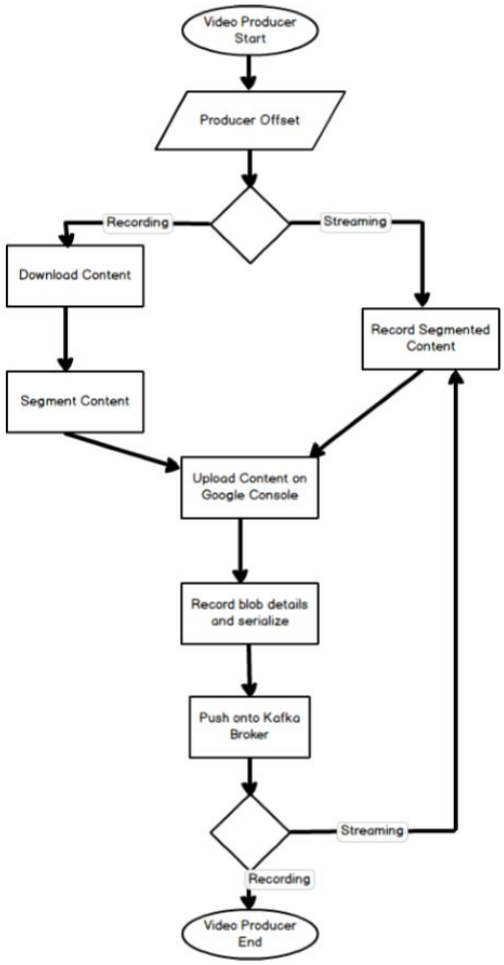

Figure 4 - Producer (Video Processing) Flow

5.3 Kafka

The usage of Apache Kafka allows the graceful interaction between Producer and Storm components, allowing both Producer and Storm to act as the 'producer' and 'consumer' respectively. For the purposes of the proposed system, Kafka was limited to a total of two partitions with the following settings:

Kafka Video Topic

- N. Partitions: 1
- Replication Factor: 1

Kafka Text Topic

- N. Partitions: 1

- Replication Factor: 1

5.4 Storm

The storm toolset was utilized through a pythonic wrapper called 'Streamparse' [16], allowing the topology itself and each Storm component to be written in native Python. The topology structure was composed of the following components:

Component Type	Component Name	Parallel Degree
Spout	video_recording_spout	1
Spout	text_recording_spout	1
Bolt	video_decoder_bolt	4
Bolt	text_decoder_bolt	1
Bolt	graph_writer	2

Table 1 - Storm Topology

Each of the two spouts utilize the Kafka-Spout Implementations, serving as a consumer to the Apache Kafka architecture, each spout subscribed to its respective kafka topic. Once subscribed, each kafka spout actively listens for incoming data on the kafka partition, at which point data is passed down the topology onto the 'video_recording_bolt' or 'text_recording_bolt', for processed video and text data respectively. It is here that both bolts carries out a number of text processing techniques onto the oncoming text, including:

- Stop Word Removal
- Stemming of Words
- Special Character/Symbol Removal
- Single Character Removal

Lemmatizing of incoming data was considered, but eventually left out, with testing results indicating that particular words were being transformed into undesirable pronunciations of the original word.

The 'video_decoder_bolt', in addition to the aforementioned text processing is required to connect to the Google Cloud Platform via usage of the Google Cloud Platform API, by which point would have already been decoded by the previously uploaded video from the producer node. The video_decoder_bolt therefore requests the blob url storage passed through the pipeline, retrieves the decoded text, processes it locally and submits it down the pipeline.

The 'graph_writer' bolt actively listens to incoming data traffic from the the 'video_decoder_bolt' and 'text_decoder_bolt', saving all textual data to the graph storage node.

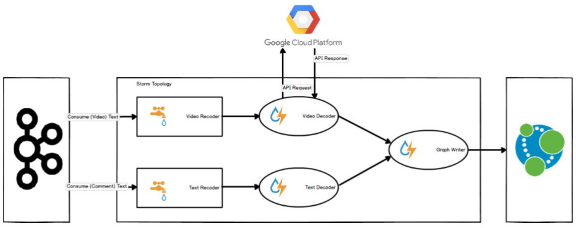

Figure 5 - Storm Topology

5.5 Neo4j

The graph database serves as a suitable endpoint for the pipeline, which receives and stores all textual data in the form of a graph. The graph serves to model received data as follows:

Nodes:

- Streamer
- Viewer
- Genre
- Word
- Platform

Relationships:

Node 1	Relationship	Node 2
Streamer	Utters	Word
Viewer	Comments	Word
Word	Features	Genre
Viewer	Follows	Genre
Streamer	Partakes	Genre
Viewer	Subscribes	Streamer
Streamer	Uses	Platform

Table 2 - Graph Relationships

For graph edges of type 'Utters' and 'Comments', each respective word utterance is counted per streamer and viewer respectively. In addition, those

nodes of type 'Word' contain an additional boolean flag variable called 'foul_flag', which denotes whether the word is registered as foul language, as denoted by a pre-recorded foul word corpus [24]. Only foul words of an English syntax are tracked.

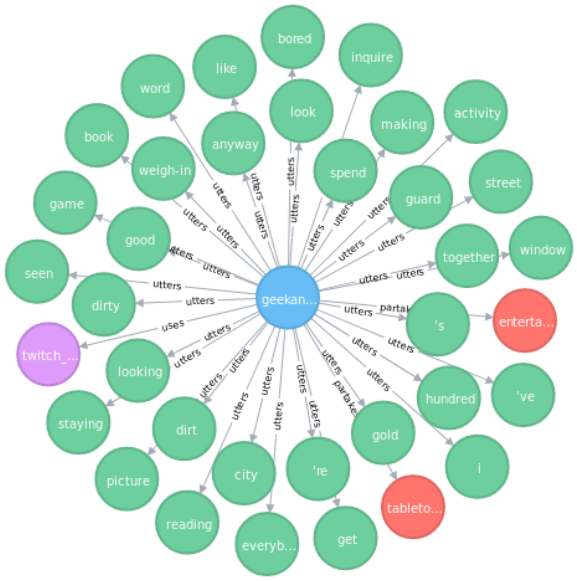

Figure 6 - Graph Topology:
Blue Node: Streaming Channel
Green Node: Word Utterance
Red Node: Attributed Genre
Purple Node: Streaming Platform

6 System Setup

The Speech Analytics system was compartmentalized into individual modules, forming the entire end-to-end system pipeline, as previously discussed. Whilst designing a strategy of bringing all the modules together, priority was given to the following aspects:

- *Costs*
  Maximizing the available cloud computing resources.
- *Test environment*
  A convenient environment that mimics the cloud setup for system debugging purposes.
- *Time*
  Minimizing the development to deployment cycle time.
  Automating the deployment procedure.

6.1 System Components

The entire pipeline was broken down into four unique modules, namely Producer, Kafka, Storm and Neo4J. All the individual modules were contained into separate Docker images, as depicted below:

<<Add Simple Image>>

All the shared software components and python packages were included into the base Docker image. On top of that, additional Docker images were built, each specific to a given Speech Analytics system module. This approach made the Docker image construction flexible, especially during the initial development stages were the software components and python packages were still not fully established. Once all Docker images were constructed, the images were appropriately tagged and uploaded to DockerHub [26].

6.2 System Deployment Automation

System deployment and execution was done on two separate environments. A test environment was set up through VirtualBox [27] such that it mimics the actual live environment. The live environment was employed on the Microsoft Azure cloud computing [28] platform.

A utility tool was developed that handles the entire system automation aspect. This was achieved through Bash scripts that make use of tools such as Docker, Docker-Machine and Nmap. This tool abstracted away the details between the local test environment and the remote Azure platform, thus commands issues were seamlessly executed on either one of the environments. The below utility features were implemented:

- *VM creation and deletion*
  VM quantities and resource specifications were specified in an external configuration file.
  As part of the VM creation process, the appropriate Docker image was pulled from the DockerHub repository into the newly created machine.

- *VM start and stop*

- *Docker container start and stop*
  The Docker container start-up followed a specific start-up sequence, based upon the inherent dependency between the respective Docker containers.

As part of the Docker container start-up, the python code was pulled from the GitHub repository. This made it convenient, since no changes were required following any changes to the source code.
Docker containers were not persisted after being unloaded from memory, thus eliminating any container management requirements.
In the specific case of the Neo4J container, a drive was mapped on the host file system in order to persist the generated data.

- *Docker container telemetry*
  Docker statistics, including CPU, memory and bandwidth usage.
  Docker logging, useful for debugging purposes.

7 Milestones

To benchmark and evaluate the proposed system, a total of two milestones were planned. Both milestones served as checkpoints which would test the system under different scenarios and conditions, allowing for a more robust evaluation of the implemented infrastructure.

7.1 Milestone One

The first of the two proposed milestones aimed at smoke testing the proposed pipeline, ensuring that all basic and detrimental functionality is working well under limited volumes. The test was aimed particularly at pre-recorded footage and pre-recorded text, particularly from YouTube sources. This makes the test more controllable in terms of data being pumped into the system. With limited, yet sufficient volumes being inserted down the pipeline, ensures that almost all aspects of the architecture has been tested.

7.2 Milestone Two

The second proposed milestone is intended to expose the proposed architecture to real-time streaming data, so as to evaluate and gauge the effectiveness of the system under high velocity streams. In addition, the objective here is also to stress-test the pipeline as follows:

- Stress-test the system with double or more data than the original milestone.
- Stress-test the system by leaving the pipeline running for double the time it took for milestone one.

8 Evaluation

System evaluation and overall performance was gauged and benchmarked on each of the proposed milestones. Each of the installed architecture components were monitored individually to achieve insight as to their granular behaviour. In addition, several graph visualizations were created to visualize the underlying data, as well as provide a series of data analytics techniques.

8.1 Milestone One - Results and Interpretation

The first of the aforementioned milestones tested the proposed architecture on pre-recorded data specifically on the following YouTube content [23]. The following approximate timings were recorded as follows:

Entity	Process Start	Process End	Total Time
Producer2 (Video)	05:37:44	10:37:00	05:00:00
Producer3 (Text)	05:37:44	13:00:00	07:23:00
Producer4 (Video)	05:37:44	10:37:00	05:00:00
Producer5 (Text)	05:37:44	13:00:00	07:23:00
Kafka	05:37:44	13:00:00	07:23:00
Storm	05:37:44	13:00:00	07:23:00
Neo4j	05:37:44	13:00:00	07:23:00

Table 3 - Milestone 1 Producer Timings

During the above mentioned time window, performance metrics were extracted and visualized. These metrics allow a clearer insight with respect to individual component performance, including CPU, Main Memory and Physical I/O Usage:

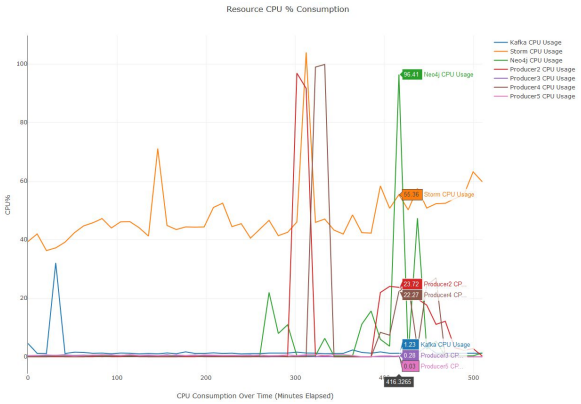

Figure 7 - Milestone 1 CPU % Usage:

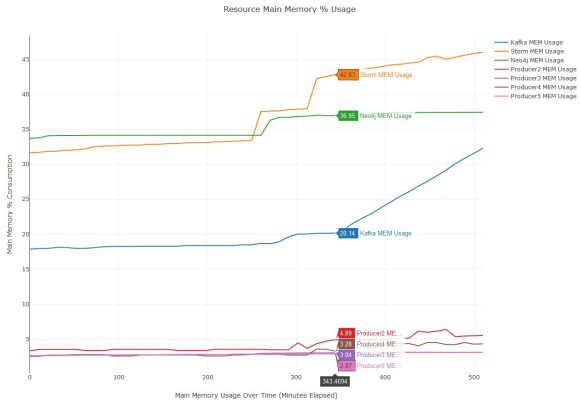

Figure 8 - Milestone 1 Main Memory % Usage:

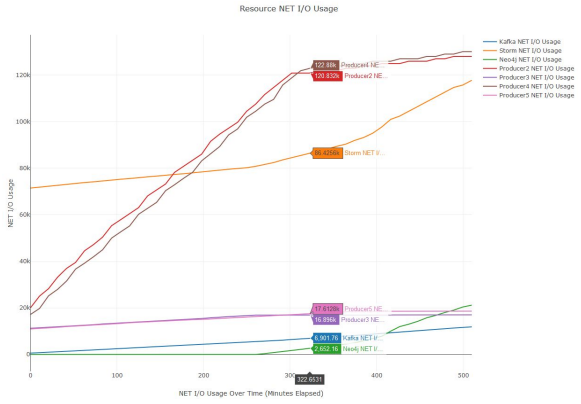

Figure 9 - Milestone 1 NET I/O Usage:

As an interactive visualization capable of monitoring data dynamics, an interactive graph model was established, based upon the generated graph structure from milestone one. The graph allows the user the possibility of navigating each respective word, streamer relationship. Each graph edge denotes the total number of word utterances from the respective viewer and/or streamer.

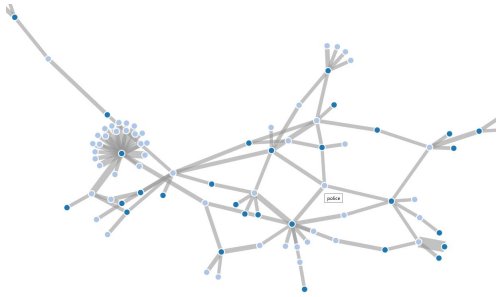

Figure 10 - Interactive Viewer/Streamer Word Utterance

Based upon the established timings, overall performance and system behaviour, it was decided that the pipeline operated as expected during the entirety of milestone one. Each of the distributed components and respective VMs were not exhausted by the volume of the input data streams, with the entire system behaving in a stable manner.

8.2 Milestone Two - Results and Interpretation

The second milestone was focussed on stress testing the system so as to ensure optimum quality during real-time input data streaming, over a longer period of time. The experiment ran for a total of nineteen hours, with eight constant input streams being channelled through the pipeline. The following performance metrics were recorded as follows, particularly for Kafka, Storm and the Neo4j environments:

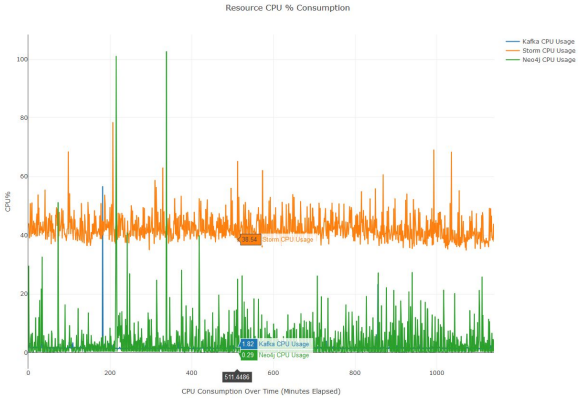

Figure 11 - Milestone 2 CPU % Usage:

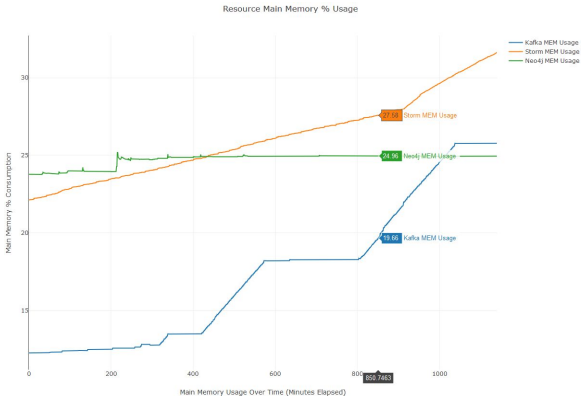

Figure 12 - Milestone 2 Main Memory % Usage:

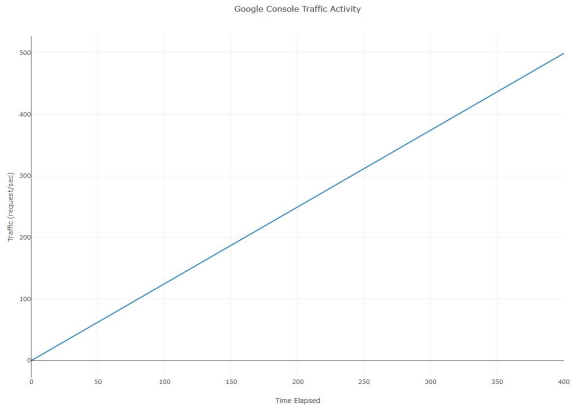

Figure 13 - Milestone 2 Google Console Traffic Activity

The above figures indicate a stable, gradual performance of the tested pipeline components. Of particular notice is the resource hungry presence of Storm, when compared to the Kafka and Neo4j entities. Statistics for each respective producer were also extracted, as follows:

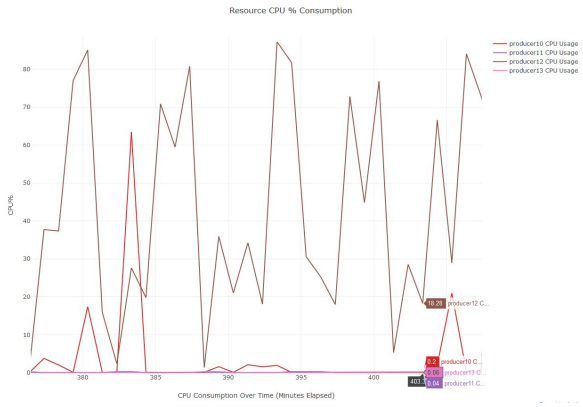

Figure 14 - Milestone 2 Producer CPU % Activity

The above metrics suggest a particular strain on the producer node, particularly so those responsible for processing video. CPU usage was visible to have been operating at over 80% at particular times, suggesting a potential hardware bottleneck, which can be tackled by scaling video processing nodes in a vertical manner.

8.3 Additional Requirements

The following section is dedicated particularly on the following three aspects, all of which were directed upon the graph store for milestone two:

- Data Analytics
- Data Dynamics
- Data Visualizations

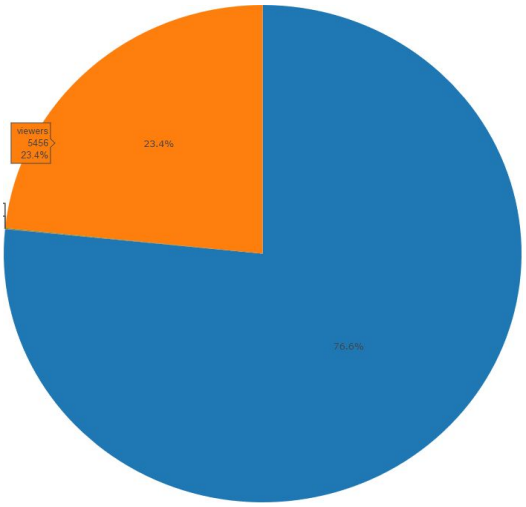

Figure 15 - Node Quantity Ratio

Node Type	Node Count	Node (%)
Word	17,880	76.600%
Viewer	5456	23.400%
Genre	9	0.038%
Streamer	4	0.017%
Platform	1	0.004%

Table 4 - Milestone 2 Node Counts

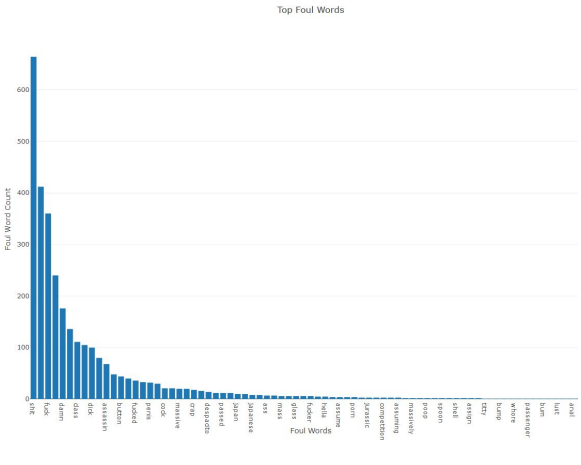

Figure 16 - Foul Word Usage, Ranked By Highest Occurance

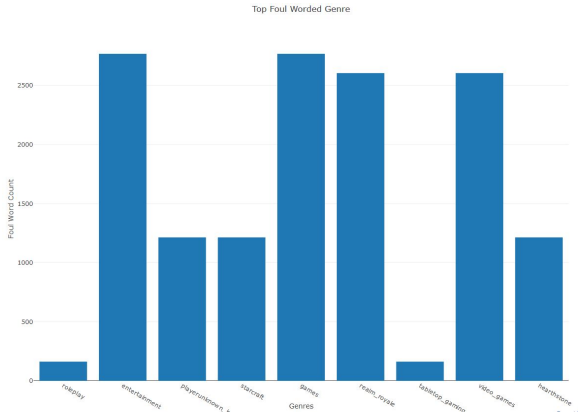

Figure 17 - Genres Foul Word Usage

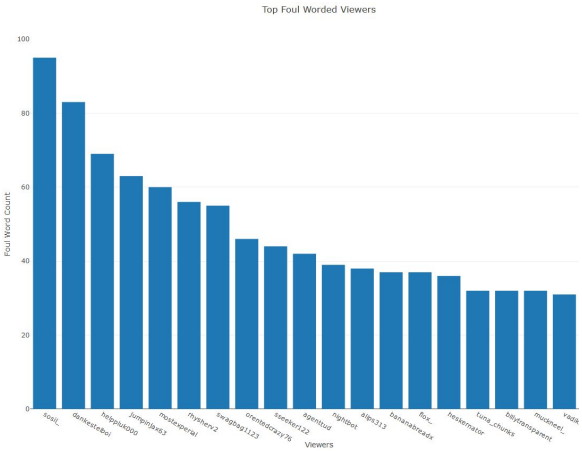

Figure 18 - Viewers Ranked by Foul Word Usage

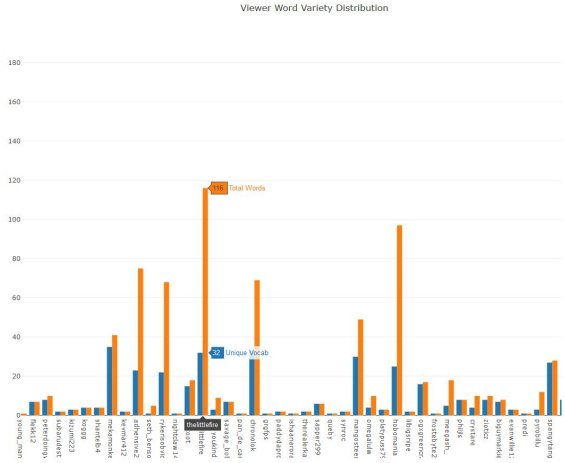

Figure 19 - Viewers Word Usage
Blue: Unique Vocab
Orange: Total Word Utterances

8.4 Limitations and Future Improvements

After both milestone runs, a number of potential improvements were noted, allowing room for future possible work on the proposed infrastructure to be carried out.

- The system security aspect was considered as beyond the scope of this project and thus was given little importance. Improvements include:
  - Authentication hardening
  - Creation of Linux users with limited privileges for the execution of all the involved OS processes.
- All producers and consumers share the same Google account. Potential distribution between several Google accounts would aid in circumventing the limitations imposed by the platform [25] and thus improve the overall video processing performance.
- Improving system robustness by adding a *watchdog* VM, capable of monitoring external nodes and re-initiating those nodes which exit in an unhandled manner.
- Bash script automation could benefit from more robust user input validation.
- Better VM specification allocation, by testing and evaluating performance on better hardware allocation, through vertical scaling of VMs.

- Storm main memory consumption was noted to be gradually increase in terms of overall usage. This could be due to a potential memory leak on the Spout/Bolt logic.

9 Conclusion

In this report, the well established challenges of big data concerning aspects of a variety and a velocity nature, were addressed by a real time video and text processing system. The proposed architecture was broken down into a number of components, for which each individual usage and performance was evaluated through extracted metrics pertaining to two separate testing scenarios. The acquired data streams were stored inside of a graph warehouse, making retrieval and representation of word usage per viewer/streamer a possibility, facilitating any data mining and data extraction attempts from the acquired data.

10 References

[1] http://lansainformatics.com/wp-content/plugins/project-mgt/file/upload/pdf/2440Data-mining-with-big-data-pdf.pdf

[2] http://tarjomefa.com/wp-content/uploads/2017/04/6539-English-TarjomeFa-1.pdf

[3] https://www.twitch.tv/directory

[4] https://www.youtube.com/

[5] https://www.youtube.com/channel/UC4R8DWoMoI7CAwX8_LjQHig?&ab_channel=Live

[6] https://www.dailymotion.com/us

[7] https://kafka.apache.org/intro

[8] http://storm.apache.org/

[9] https://github.com/neo4j/neo4j

[10] https://cloud.google.com/speech-to-text/?utm_source=google&utm_medium=cpc&utm_campaign=emea-emea-all-en-dr-bkws-all-all-trial-e-gcp-1003963&utm_content=text-ad-none-any-DEV_c-CRE_171810382929-ADGP_Hybrid+%7C+AW+SEM+%7C+BKWS+~+EXA_1%3A1_EMEA_EN_ML_Speech+API_speech+to+text+google-KWID_43700017207320787-kwd-94457331132-userloc_2470&utm_term=KW_speech%20to%20text%20google-ST_speech+to+text+google&ds_rl=1245734&gclid=Cj0KCQjwmPPYBRCgARIsALOziAMX5FrE9W-ehlpFVNgy4cWwUYpYn0MK5lCsALSWRclsLDnuFLk10i4aAtwzEALw_wcB&dclid=CPLmu-SOydsCFduUdwodC2wEhA

[11] https://developers.google.com/youtube/v3/docs/

[12] https://github.com/TwitchDev/chat-samples/tree/master/python

[13] https://github.com/nficano/pytube

[14] https://www.ffmpeg.org/

[15] https://github.com/streamlink/streamlink

[16] http://streamparse.readthedocs.io/en/stable/quickstart.html

[17] https://upcommons.upc.edu/bitstream/handle/2117/86160/Xhafa_EtAl_AINA2015.pdf

[18] https://www.researchgate.net/profile/Pengcheng_Duan3/publication/281380116_A_video_cloud_platform_combing_online_and_offline_cloud_computing_technologies/links/574ba6f508ae2e0dd301a99f/A-video-cloud-platform-combing-online-and-offline-cloud-computing-technologies.pdf

[19] https://www.semanticscholar.org/paper/A-Deep-Intelligence-Framework-for-Online-Video-Zhang-Xu/266963c903eeca6b3f79a9de93ab6fa3c4b55b98?tab=references

[20] https://ieeexplore.ieee.org/abstract/document/6122006/

[21] http://hadoop.apache.org/

[22] http://homepages.cs.ncl.ac.uk/paolo.missier/doc/p56-zaharia.pdf

[23] https://www.youtube.com/watch?v=Kye2oX-b39E&list=PLjdfbL5LrCRiKMbwY3JiPX90aWPJcr5Zh&ab_channel=LastWeekTonight

[24] https://github.com/LDNOOBW/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words

[25] https://cloud.google.com/speech-to-text/quotas

[26] https://hub.docker.com/

[27] https://www.virtualbox.org/

[28] https://azure.microsoft.com/en-us/

11 Appendix

Figure Number	Milestone	Visualization
1	N/A	Architecture_Wireframes.pdf
2	N/A	Architecture_Wireframes.pdf
3	N/A	Architecture_Wireframes.pdf
4	N/A	Architecture_Wireframes.pdf
5	N/A	Architecture_Wireframes.pdf
6	N/A	GeekandSundry Entire Pipeline Simulation.png
7	1	CPU_Resource_Percentage_Consumption.html
8	1	Main_Memory_Resource_Percentage_Consumption.html
9	1	NET_IO_Consumption.html
10	1	tree_graph.html
11	2	CPU_Resource_Percentage_Consumption.html
12	2	Main_Memory_Resource_Percentage_Consumption.html
13	2	GoogleConsole_Traffic.html
14	2	Producer2_CPU_Resource_Percentage_Consumption.html
15	2	Node_Distribution.html
16	2	Top_foul_n_words.html
17	2	Top_foul_worded_genre.html
18	2	Top_foul_worded_viewers.html
19	2	Word_Viewer_Distribution.html