GPU-accelerated Streaming Analytics

Christoforos Leventis

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science and Engineering

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor:
Assist. Prof. *Polyvios Pratikakis*
Assoc. Prof. *Sotiris Ioannidis*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

This project has received funding from the European Union's Horizon 2020 EU Research & Innovation program under Grant Agreement No 780787 (I-BiDaaS) and the European Union's Horizon 2020 EU Research & Innovation program under Grant Agreement No. 833685 (SPIDER)

University of Crete
Computer Science Department

GPU-accelerated Streaming Analytics

Thesis submitted by
Christoforos Leventis
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: .cl.
Christoforos Leventis

Committee approvals: Polyvios Pratikakis
Digitally signed by Polyvios Pratikakis
Date: 2021.03.31 10:58:45 +03'00'
Polyvios Pratikakis
Assistant Professor, Thesis Supervisor, Committee Member

SOTIRIOS IOANNIDIS
Digitally signed by SOTIRIOS IOANNIDIS
Date: 2021.03.29 15:48:18 +03'00'
Sotiris Ioannidis
Associate Professor, Thesis Advisor, Committee Member

IOANNIS TZITZIKAS
Digitally signed by IOANNIS TZITZIKAS
Date: 2021.03.29 17:20:25 +03'00'
Yannis Tzitzikas
Associate Professor, Committee Member

Departmental approval: Polyvios Pratikakis
Digitally signed by Polyvios Pratikakis
Date: 2021.03.31 10:59:08 +03'00'
Polyvios Pratikakis
Assistant Professor, Director of Graduate Studies

Heraklion, March 2021

GPU-accelerated Streaming Analytics

Abstract

Streaming analytics is the analysis of huge pools of "in-motion" data, known as streams. These streams are triggered by a specific event that happens as a direct result of an action or set of actions, like a financial transaction, a social post or a website click. Streamed data can originate from various sources such as Internet of Things (IoT) devices, bank transactions, mobile devices and sensors. By performing streaming analytics, enables the ability to work faster and stay ahead in competition.

The above analysis helps organizations to process their data and extract valuable informations in order to stay agile and identify new opportunities. Therefore, this analysis leads to smarter business moves, more efficient operations, higher profits, happier customers and responsiveness.

In this work, we present a GPU-based solution which can perform analysis on data in stream or at rest. This solution uses a sentiment-based lexicon and performs pattern matching operation over the data to extract valuable information such as sentiment score and frequent used words. This tool can also can track trends for both short and long term events by manually adjusting the desired time window intervals. Lastly, our evaluation applies the GPU-based component in a Quality of Service scenario where we examine the incoming call transcripts of different call centers and report various insights.

Ανάλυση Δεδομένων σε Ροή μέσω Κάρτας Γραφικών

Περίληψη

Streaming analytics είναι η ανάλυση τεράστιων ομάδων «σε κίνηση» δεδομένα, γνωστά και ως stream. Αυτά τα δεδομένα δημιουργούνται από ένα συγκεκριμένο γεγονός ως ένα αποτέλεσμα μιας ενέργειας ή μιας σειράς ενεργειών, όπως μια χρηματική συναλλαγή, μια κοινοποίηση από ένα μέσο κοινωνικής δικτύωσης ή με το κλικ μιας ιστοσελίδας. Αυτά τα δεδομένα προέρχονται από διάφορες πηγές όπως IoT συσκευές, τραπεζικές συναλλαγές, κινητά τηλέφωνα και αισθητήρες. Με την ανάλυση πάνω σε αυτά τα δεδομένα, παρέχεται η δυνατότητα γρήγορης και αποτελεσματικότερης δουλειάς αλλά και να διατηρήσουν το προβάδισμα στον ανταγωνισμό.

Η παραπάνω ανάλυση βοηθάει εταιρείες να επεξεργαστούν τα δεδομένα τους και να εξάγουν χρήσιμες πληροφορίες για να πάρουν νέες ή προληπτικές αποφάσεις. Αυτό οδηγεί σε εξυπνότερες επιχειρηματικές κινήσεις, πιο αποτελεσματικές δραστηριότητες, υψηλότερο κέρδος, γρήγορη ανταπόκριση και ικανοποιημένους πελάτες.

Σε αυτή την εργασία, παρουσιάζουμε μια λύση, η οποία μπορεί να αναλύσει δεδομένα που ειναι αποθηκευμενα σε δίσκο ή «σε κίνηση» χρησιμοποιώντας την κάρτα γραφικών. Αυτή η λύση χρησιμοποιεί ένα λεξικό βασισμένο στο συναίσθημα και εκτελεί μία λειτουργία ταιριάσματος προτύπων στα δεδομένα για να εξάγει πολύτιμες πληροφορίες όπως αρνητικά ή θετικά συναισθήματα και επαναλαμβανόμενες λέξεις. Το εργαλείο μας, έχει την δυνατότητα να παρακολουθεί και να συλλέγει πληροφορίες, σε διάφορα χρονικά διαστήματα, είτε μικρά είτε μεγάλα, σχετικά με τις διάφορες τάσεις που υπάρχουν μέσα σε αυτά τα δεδομένα. Τέλος, σε αυτήν την εργασία αξιολογούμε την απόδοση του συστήματός μας πάνω σε ένα σενάριο που έχει να κάνει με την ποιότητα εξυπηρέτησης όπου επεξεργαζόμαστε τα δεδομένα σε ροή και εξάγουμε τα διάφορα αποτελέσματα.

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον επιβλέπων μου, καθηγητή Πολύβιο Πρατικάκη για την πολύτιμη συνεργασία στα πλαίσια της μεταπτυχιακής μου πορείας. Επίσης, θα ήθελα να εκφράσω την βαθιά μου ευγνωμοσύνη στον σύμβουλο μου Δρ. Σωτήρη Ιωαννίδη που μου έδωσε την ευκαιρία να δουλέψω στα πλαίσια του ευρωπαϊκού έργου όπου και εκπλήρωσα την μεταπτυχιακή μου εργασία. Αυτή η ευκαιρία με βοήθησε να εμπλουτίσω τις γνώσεις μου και να εξελιχθώ. Η μεταπτυχιακή μου εργασία έλαβε χρηματοδότηση από δύο ερευνητικά και καινοτόμα πρόγραμμα της Ευρωπαϊκής Ένωσης Horizon's 2020, στο πλαίσιο συμφωνίας υποτροφίας με αριθμό 780787 (I-BiDaaS) και αριθμό 833685 (SPIDER).

Επιπλέον, θα ήθελα να ευχαριστήσω τον Δρ. Γιώργο Βασιλειάδη, την Εύα Παπαδογιαννάκη και τον Δημήτρη Ντεγιάννη για την καθοδήγηση που μου έδωσαν κατά την διάρκεια αυτής της ακαδημαϊκής μου πορείας και την ευχάριστη συνεργασία που είχαμε.

Θα ήθελα επίσης να ευχαριστήσω τον Αλέξανδρο Σαββόπουλο, Σεραφείμ Μουστάκα, Δημήτρη Καρνίκη, Γιώργο Αναγνώπουλο αλλά και όλα τα μέλη του εργαστηρίου Κατανεμημένων και Παραλλήλων συστημάτων για την φιλία τους και την συμβουλή τους.

Τέλος, θέλω να ευχαριστήσω την οικογένεια μου και τους φίλους μου πίσω στην Κύπρο για την υποστήριξη και την φροντίδα τους.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Currently we are living in the era of Data driven technologies, multiple sources such as Iot-devices, sensors, banking transactions and social media are producing an uncontrollable amount of data. These data are referred as Big Data, which is a combination of structured, semi-structured and unstructured data. There is no numerical standard to define Big, but is often characterized by the three Vs: Variety, Volume and Velocity.

Business domains use these data for all manners of analysis [55, 36, 28], when are used correctly it helps companies to improve operations, enhance costumer service, create personalized marketing campaigns and increase profitability in real life. More specifically, these data analytics is the often complex process of examining data in motion or at rest to uncover information such as hidden patterns, correlations, market trends and customer preferences that can help organizations make informed business decisions.

Although, for businesses to remain competitive, they need technology that delivers performance at scale by supporting a wide number of concurrent applications while consuming large volumes of changing data in order to analyze big data in motion in real time. To achieve this, they need to implement their analyzing methods in special streaming processing engines such as Apama[54], Apache Spark[30], Apache Storm[31].

Unfortunately, there is no standard method for analyzing these data, various programming applications using state of the art techniques and tools are being developed over the time in order to extract useful information. These applications are often using a combination of techniques such as data integration from multiple sources [11, 17], data mining methods [35, 37, 58], machine learning models [14], Natural Language Processing (NLP) tools & methods [8, 32, 39] and other statistical techniques [68, 66] in order to provide more accurate insights.

However, Sentiment Analysis has become a hot-trend topic of scientific and market research in the field of Natural Language Processing (NLP) and Machine Learning. This technique can be performed via Lexicon Based or Machine Learning approach[53]. Since Machine Learning models require a set of well known skills,

this works focuses in the Lexicon Based approach.

Another reason we are choosing this approach is because several works such as SocialSent [33] are offering open-source tools to offline generate such sentiment lexicons from a set of unlabeled data. Thus, by using this lexicon enables the capability to extract the sentiment value by performing a simple pattern matching operation.

In this work, we present a GPU-base tool which can perform Sentiment Analysis both streaming and at rest data. Our solution can analyze the input data and extract various insights using a GPU-accelerated pattern matching algorithm and a sentiment based lexicon. Moreover, our development can track trends for both short and long term events in order to help a company have a better understanding of the costumers' tendency.

To test the effectiveness of our work, we applied it in a realistic streaming scenario. In this scenario we are called to process the incoming call transcripts from different call centers in order to predict their performance by accumulating the sentiment score accordingly to their streamed transcripts and provide insights such as most frequent words and sentiment score in different time windows.

1.1 Motivation

Nowadays, organizations are looking to mitigate risk while making their businesses more agile and responsive are using big data to transform how problems are viewed and strategic policies are formed. By embracing the capabilities of big data, they are able to make more informed decisions that help them gain a competitive edge, improve overall performance and boost their bottom line. Although, developing or using state of the art data analytics or fast analytics methods often require a set of special skills. From our end, we propose a sentiment analysis tool which uses a massive parallel pattern matching operation performed by the Graphics Processing Card (GPU). We believe our work is sufficient to help many company domains in way to process their data either real-time or offline.

1.2 Contributions

The contributions of this work are the following:

- This tool can be used in streaming analytics scenarios and provide accurate real-time insights which eventually will help decision making.
- Our implementation can track trends for both short and long term events by manually adjusting the desired time window intervals.
- Offline analysis is another option offered by the current solution.

- Since the accelerated pattern matching operation is implemented in OpenCL framework, it is easily executed on the vast majority of data parallel platforms such as CPUs and high-end discrete or integrated GPUs.

1.3 Publications

Parts of the work for this thesis have been used and published in I-BiDaaS European project under grant agreement No. 780787 and also in an open access book. Specifically, parts of this work are included in the following:

- I-BiDaaS Project. Deliverable D4.2: Distributed event-processing engine,2020[5].
- I-BiDaaS Project. Deliverable D4.3: Streaming analytics and predictions,2020[6].
- Big Data Value Association Open Access Book,2020 [13].

1.4 Outline

The rest of this dissertation is organized as follows. Chapter 2 presents a brief background on the implement framework. In this chapter we also provide details about the sentiment analysis method and the pattern matching operation. Moving on, Chapter 3 we present our design overview of our component over the Java Native Interface (JNI) Framework. In addition, in this section we describe the extra functionalities we needed to implement in order to provide real-time insights.

Chapter 4 describes in detail the development and functionality of our GPU-based solution along with their challenges. To be specific, this chapter expands on three sections. The first section describes the challenges we had to face while the following section describes the implemented native functions. The last section describes in detail how our work can be utilized in a Quality of Service use case and how our tool can provide real-time insights using these functionalities.

Chapter 5 contains a thorough evaluation of our component. In the first evaluation we compare our component over an CPU-based pattern matching implementation. The second evaluation provides details about the achieved throughput along with the accuracy of our model. In the third evaluation we re-evaluate our model's accuracy over a domain specific lexicon.

Chapter 6 outlines the limitations of our work and provide some ideas on how this work can help various machine learning models with their offline training, Chapter 7 surveys prior work, Chapter 8 summarizes this dissertation and points out future research directions.

Lastly, Chapter 9 contains integration details with third-party tools and demonstration snapshots of our work which was presented publicly.

Chapter 2

Background

In this section we provide the necessary knowledge about the sentiment analysis technique we are performing via pattern matching operation and the fundamental native function framework we used to implement our work.

2.1 Java Native Interface (JNI)

The Java Native Interface (JNI)[45] is a native function programming framework in Java Development Kit (JDK) that enables Java code running in a Java virtual machine (JVM) to call and be called by native applications (programs specific to a hardware and operating system platform) and libraries written in other languages such as C, C++ and assembly.

JNI is used when there is already a library written in C/C++ language and we need to use that library in our Java application. Since our pattern matching operation is already implemented in C side, we used this interface in order to board the GPU-accelerated code into JVM.

2.2 Sentiment Analysis

Sentiment Analysis examines the problem of studying texts, like posts and reviews, uploaded by users on forums and electronic businesses, regarding the opinions they have about a product, service, event or idea. The most common use of this technique is the classifying a text to a class. Depending on the dataset and the reason, Sentiment Classification can be binary (positive or negative) or multi-class (3 or more classes) problem. Also, it is often performed on textual data to help businesses monitor brand and product sentiment in customer feedback and understand customer needs.

We are performing this technique via pattern matching operation using as input a sentiment based lexicon in order to analyze customer feedback and learn what makes customers happy or frustrated. By delivering the analysis results to the

business company, they can tailor products and services to meet their customers' needs.

2.3 Pattern Matching

String pattern matching, also called string searching, is the act of trying to locate the occurrence of one or more strings (also called patterns or signatures) within a larger string (e.g. plain-text). String pattern matching, using finite alphabets is a very common technique in order to locate any occurrence of a string pattern into a text. For example, when searching for a string pattern $P = p_1p_2...p_n$ inside a text $T = t_1t_2...t_m$(with lengths n and m accordingly), both characters sequences form a finite alphabet set A.

One of the most popular algorithms in this field is the Aho-Corasick one which was invented by Alfred V. Aho and Margaret J. Corasick in 1975 [9]. The Aho-Corasick algorithm constructs a finite-state machine that resembles a trie with additional links between the various internal nodes. These extra internal links allow fast transitions between failed string matches to other branches of the trie that share a common prefix. This allows the automaton to transition between string matches without the need for backtracking and it's computational complexity is linear.

For our work, we are using an accelerated version of this algorithm created by Dimitris Deyiannis [27] and Giorgos Vasiliadis [61, 63, 60]. More information is provided in the following section.

2.3.1 GPU-acceleration of pattern matching operation

The GPU-accelerated pattern matching operations are offered through an OpenCL library that supports both string searching and regular expression matching operations. The library provides a C/C++ API for processing incoming records, and returning any matches found back to the application. The library can be used transparently by a broad range of applications to offload their costly pattern matching operations to the GPU, and thus increase their overall performance.

Initially, all patterns are compiled to "deterministic finite automaton" (DFA) state machines and state transition tables (see Figure 2.1). The user is able to compile each pattern to a separate DFA, or combine many different patterns to a single one. The compilation process is performed offline by the CPU, usually during the initialization phase of the user application. The state table is then copied and mapped to the memory space of the GPU. At the searching phase, each thread searches a different portion (i.e., a separate message) of the input data stream. In order to fully utilize the data-parallel capabilities of the GPU, the library creates a large number of threads that run simultaneously. The core processing loop splits the input messages, and distributes them for processing to different threads.

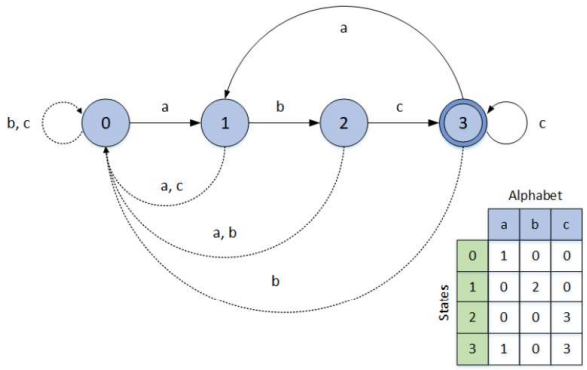

Figure 2.1: The DFA state machine and the state transition table for the regular expression (abc+)+.

During scanning, the algorithm moves over the input data stream one byte at a time. For each consumed byte, the matching algorithm switches the current state according to the state transition table. The pattern matching is performed byte-wise, meaning that we have an input width of eight bits and an alphabet size of $2^8 = 256$. Thus, each state will contain 256 pointers to other states. The size of the DFA state transition table is $|\#\, States| \times 1024$ bytes, where every pointer occupies four bytes of storage. When a final-state is reached, a match has been found, and the corresponding offset is marked. The format of the state table allows its easy mapping to the different memory types that modern GPUs offer. Mapping the state table to each memory yields different performance improvements.

Table 2.1: Sustained PCIe v2.0 Throughput (Gbit/s) for transferring data to a single GPU, whenincreasing the size of data that are transferred at once

Buffer	1KB	4KB	64KB	256KB	1MB	16MB
Host to GPU	2.04	7.12	34.4	42.1	45.7	47.8
GPU to Host	2.03	6.70	21.1	23.8	24.6	24.9

Another thing to consider is the transfer of the incoming data to the memory space of the GPU. A major bottleneck for this operation is the extra overhead, caused by the PCIe bus that interconnects the graphics card with the base system. Unfortunately, the PCIe bus suffers many overheads, especially for small data transfers. To further improve performance, they use a large buffer to store the contents of multiple tuples, which is then transferred to the GPU in a single transaction, every time it gets full. This results in a reduction of I/O transactions over the PCI Express bus, which further results to increase throughput,as can been shown in Table 2.1.

Chapter 3

System Architecture

GPU-accelereted tool consists on native functions specially designed in Java Native Interface in order to access any high computational device such as GPUs or GPGPUs and offload the workload to them. In following sections we will illustrate the design architecture of the GPU-accelerated solution in JVM and describe the additional functionalities we needed to implement in order to process the incoming data and provide real time insights.

3.1 Boarding to JVM

The component is designed to run in a JVM but since the accelerated pattern matching operation is implemented in C side, we need to create a "bridge" which allows the Java code that runs within the JVM to operate with applications and libraries written in other languages. Figure 3.1 illustrates the connection example between the two programming languages using the above interface.

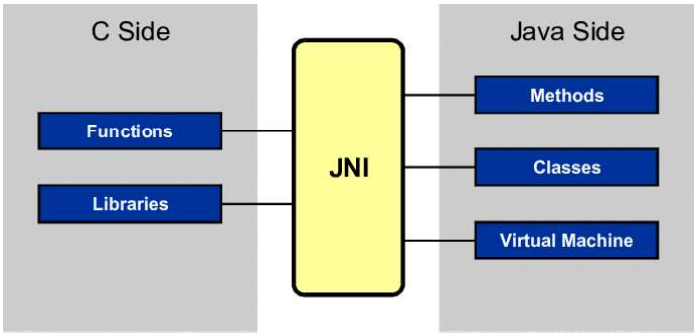

Figure 3.1: JNI allows Java code, that runs within the JVM, to operate with applications and libraries

Ideally our system needs to have three native functions. The first function has to instantiate the target device via parameters. Since we allocated memory for

the device initialization we also need to create a second function to deallocate the above memory and free the device. The main payload will be triggered by a third function which will transfer the available data to the device buffer, executes the pattern matching operation and return the results back to the main Java program.

3.2 Providing insights

Regarding the I-BiDaaS project we are going to use a NoSQL database which has build-in fault tolerance mechanisms known as TerracottaDB [7]. The main concept is to utilize the TerracottaDB as a buffer to temporary store the incoming streaming data.

Once the incoming data have been written to the TerracottaDB, the "GPU-broker" process reads them using a separate Consumer instance. The Consumer instance is responsible to read the entries from the TerracottaDB and store them back-to-back in an array. The array is passed as an argument in a Java Native Interface function and copies the entries to the GPU's buffer. When the buffer is filled or there no other data present in the database, the pattern matching phase can start.

In order to utilize our tool as streaming analytics component we need to implement some additional functionalites. These functionalites will give us the opportunity to process the incoming data in a "rolling window" manner using time-decayed counters, using this method we will be able to aggregate the incoming data in various time window intervals. In each time window we perform **Word Searching** over the data and compute the **Sentiment Score** and also report the **Word Frequencies** along with **Top-Frequent Words**.

Figure 3.2 illustrates how the above functionalites are used to provide real-time insights with a rolled-over window along with time-decayed counters in a "Quality of Service in Call Centers" use case. More details can be found in implementation section (section 4.3).

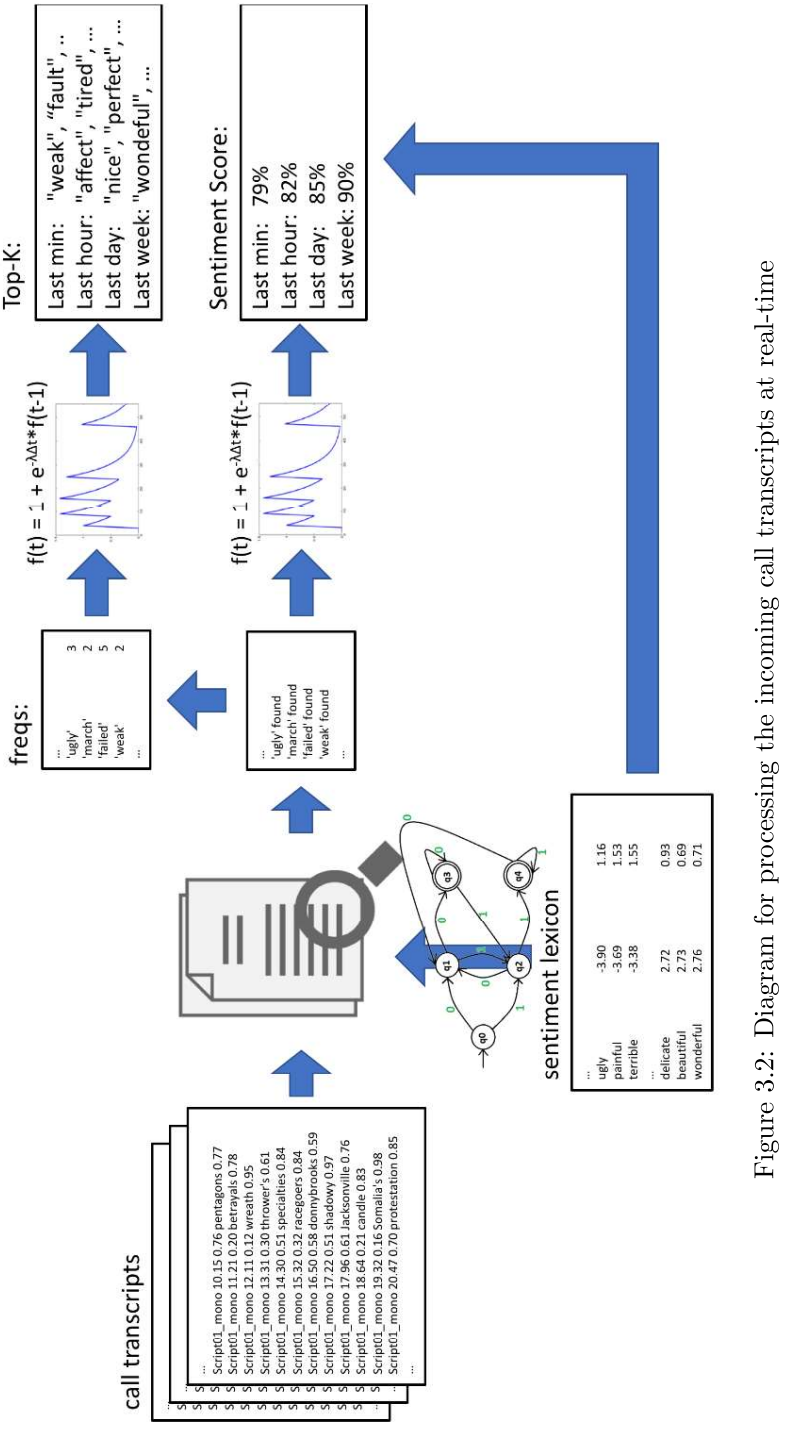

Figure 3.2: Diagram for processing the incoming call transcripts at real-time

Chapter 4

System Implementation

In this section we are going to present and demonstrate the full implementation details of GPU-accelerated module as well as the challenges we encountered and the optimizations we performed.

4.1 Challenges & Optimizations

While implementing our JNI-based work we phased additional challenges in order to gain a better performance from our previous versions. Below we will explain in detail what challenges we had to phase and how we managed to solve them.

1. Memory alignment in Java vs Memory alignment in C, while dealing with characters we found that Java characters use a 16-bit representation while C characters need only 8-bit representation. Our optimization here was to cast the initial steamed data into Java bytes in order to have a more compatible form in our Host side (C side).

2. Initialize the GPU once and run it multiple times. As our module starts, it will generate an Object on Java side which contains the GPU worker instance (on the C side). While the object is created, a JNI function is also triggered and allocates the necessary memory needed for the GPU worker struct. When the worker is ready, we store it's memory address in a type Long field in Java's object. By doing the above method our worker instance lives as long as the Java object is alive and it's fully functionable.

3. Multiple memory copies on JNI side. As we have already mentioned above, a byte array is passed as an argument in the JNI function (C side). The problem here that arrays in Java is treated as Objects. In order to read each entry in the origin array we have to first find the length of the array and then find the corresponding length of each index cell in order to store it into the host buffer. Our main goal here is to minimize these memory copies in order to gain better performance.

4.2 Native Functions

The GPU-based module consist on three main native functions, these functions are `createCobject`, `deleteCobject`, `workerPayload`.

4.2.1 createCobject

This function is called inside the class constructor on our main Java program. More specifically this function will create a C worker instance and initializes the GPU parameters which are needed to start the processing. To keep the C instance alive during Java's execution time we declared a private long variable with the name `Keep_Instance_Alive`, this variable is used to save the instance of our device instantiation. To access the variable from JNI side we created a function with the name `getPtrField`, this function will let us store the C instance (address) into Java's stack frame. This mechanism will help us to make multiple GPU executions without reconfiguring the device each time.

4.2.2 deleteCobject

This function is needed to delete the C instance once the Java program reaches its end, since we do not have any garbage collector in C we need to manually free the memory we allocated from GPU and from C side and eventually free the device.

4.2.3 workerPayload

This is the most important one, it takes the input data from Java side and copies them into the `Host Buffer`, the host buffer is a simple buffer before we move to the final copy to GPU's buffer. Once all the copies are done we need to create the necessary indices in the buffer so each thread can do its work in the according offset. Once the indices have been created, the `gpu_worker` function is triggered.

The above function copies the data from `Host Buffer` to Device's Buffer, triggers the matching process and extract the results from the threads. Firstly, we save the results in an array with the name `ReportMatch`. This is a C array of integers which contains all the patterns which were found within the process stage. This array will return back to Java main program.

4.3 Predicting the Performance of Call Centers

In this section, we present our solution implementation for computing streaming analytics in call centers. This solution will be utilized by Telefonica Investigation y Desarrollo (TID), as part of its Quality of Service (QoS) in Call Centers use case, in order to improve the customer experience. As shown in Figure 3.2, our solution is able to process the recorded text transcripts, which can originate from different call centers at real-time. Given the vast amount of data that needs to be processed, we

are able to demonstrate a faster end-to-end text analytics with GPU acceleration, and provide different types of analytics (such as sentiment score, word frequencies, and most frequent words) for different time intervals (e.g., last minute, last hour, and last day). These analytics can be used for real-time predictions, such as QoS, and help a company to take appropriate actions in order to better understand or control a given situation.

4.3.1 Word Searching

As shown in Figure 3.2, our analysis component takes as input the transcripts as they have been recorded from the corresponding call centers and a lexicon that contains a list of words and a marking if the word has a positive or a negative sentiment. In some cases, the words within the lexicon have a weighted score [46]. Such lexicons can be typically created using machine learning techniques or manually (e.g., by an expert).

The lexicon is used as an input by our component shown in Figure 3.2, which extracts the corresponding words and constructs a deterministic finite automaton (DFA). This DFA is used to find each and every occurrence of them within the stream of transcripts that is received at real-time from the call centers. This processing is accelerated using GPUs, and will produce as output a constant stream of matches (similar to [62, 46, 64]). These matches are then used for two purposes: (i) to compute a universal sentiment score per call center for each time window specified, and (ii) to compute the frequency of each word for each time window specified.

4.3.2 Rolling Analytics

In order to cope with the big amounts of data, many stream processing systems use the notion of rolling data analysis, in which the results are incrementally updated (rolled-over) as new data arrives [16, 18]. These systems, typically, maintain only the data that correspond to specific time-windows (e.g., last minute or last hour) and produce summary statistics only for these time intervals. The key idea is to collect a bunch of data for a specific time range and compute the requested statistics for that data.

It should be noted that the input events can be either synchronous or asynchronous: in the latter case, the events can arrive or stop arriving at any time, while in the former, the events are arriving periodically. When processing synchronous events, the time window is typically programmed as a special mechanism that periodically collects a number of tuples for the requested time window interval, which then is further processed in order to compute the desired statistics. In our solution, we follow a different approach. Each item in the stream processing pipeline is always coupled with a timestamp that is derived either directly from the raw data (e.g., if the data files contain a valid timestamp) or at the time it is received by our system.

Obviously, computing the desired statistics over a time-window is simple, but it can lead to huge memory requirements when the incoming data increases or when it is highly dispersed. As such, they can operate only on small time intervals (e.g.,up to a couple of minutes), due to the excessive memory requirements. This is a major constraint for many applications that need to detect long-term behaviors (e.g., events that occur periodically in an hour-or even in a day-granularity). To overcome the high memory requirements, there have been many different methods and data-structures proposed in [44] that can be used to compute various types of statistics, which do also provide very efficient memory requirements guarantees, at a cost of precision. For instance, as we explain in more detail below, the majority of these approaches are probabilistic, hence the provided statistics are approximate. Obviously, this results in approximation errors in the statistics themselves, although they do not significantly affect the accuracy of the query results. The reason is that exactness is not necessary for the types of queries that we process at real-time, which are related in detecting unusual item trends in the stream or finding the ranking of the top most active items in the stream. Put shortly, the role of the real-time view is to monitor interesting behaviors and provide a more fine-grained dataset for the offline batch processing phase.

To overcome the large memory consumptions of rolling windows, we use the notion of time-decayed counters [24]. The idea behind this is that the value of a counter decays automatically over time using a mathematical formula. Exponential decay is one such popular formula due in part to the relative simplicity with which simple counters can incorporate exponential decay. In particular, it allows to continually reduce, to half, the value of the counter over time. As such, we only keep a simple counter for each feature that is incremented every time we have a new occurrence, and is also decayed using the following formula:

$$cnt = e^{-a.dt} \, . \, cnt \, , \text{ where } e = 2.718, \; a = \frac{log(2)}{halflife}, \text{ and } dt = t_{current} - t_{previous}$$

By using this formula, there is no need for manually sliding time windows or explicitly maintaining rolling counts, as observations naturally decay away over time. The halflife parameter defines the time required for the counter to reduce to half of its initial value. In addition, it is designed for heavy writes and sparse reads, as it implements decay only when necessary or at read time.

Obviously, the counters are not exactly the same as if we have used fixed time windows, even though are sufficient to exhibit the feature trends and reveal momentums over time, without requiring to keep large amounts of data (for each specific time frame), but only a few numbers per feature. In addition, the time-decayed formula provides lifetime flexibility by adjusting the lifetime accordingly, in an ad-hoc manner: we can track trends over just a few minutes or hours, as opposed to more long-term events, in which we are interested in trends over weeks and months.

4.3.3 Sentiment Score Computation

In our scenario, we are not interested to compute the sentiment of a certain entity within a transcript, but rather predict the sentiment out of a whole transcript, which will then be further aggregated to predict the sentiment of the whole call center. As such, the sentiment values of the words that have been found in the text stream of each call center are accumulated into a single score using the mathematical formula described in Section 4.3.2. This score is actually the sentiment score of a specific call center for a given time window, and is actually an indicator of whether the overall customer sentiment is positive or negative.

To keep the score, we use a separate exponential-decayed counter for each of the specified time windows. This score is increased by one when a positive word appears or decreases by one if a negative word appears. In case we use a lexicon with different weights per word, we add the corresponding weight each time. We wanted to keep it as simple as it could be, but on the other hand, if a user has his own dataset with a sentiment score for each word, we can easily use them to increase/decrease the score accordingly. We would like to add that the score can be also negative, and this is also realistic since a specific call center might face a huge amount of complaints for many reasons like long time waiting in line to be served. Separate scores might take place since more than one call center will be available in the database along with a percentage score to ideally visualize how many positive words occurred in this time window. Once the aggregation finishes, we move to the final step, which is to extract the Top-K words occurring from the Hashmap aggregation (described in the next section).

4.3.4 Most Frequent Words

Besides the total sentiment score for each call center, we also compute the Top-K emotion words for each of the selected time windows. These words can be very helpful, as they give extra context and more details regarding the sentiment score.

To aggregate the occurrences of each of the words found, we use a Hashmap. The size of the Hashmap in the worst-case scenario will be as large as the whole lexicon that was fed inside the pattern file. Each entry in the Hashmap is the form of < `word_id`, `counter`>, where `word_id` is the unique identification of the word and `counter` is the corresponding count for this word for a specific time-window, as described in Section 4.3.1. The use of exponential-decayed counters allows the user to configure appropriate time-windows as desired (e.g., for the last minute,hour, day or even week). By doing so, our approach is capable to monitor the frequencies of each word occurring while processing the incoming transcripts and increase the frequencies (counters) of each word along with a decay function to decrease these counters on different time windows.

Every time a specific word appears and it is not present in the Hashmap, a new entry is created, with the current epoch timestamp (in milliseconds) and acount value equal to one. If the word exists in the Hashmap, then we only increase

its counter by one or as many times as the word has been found and update its corresponding timestamp. This counter helps us to keep track of each words frequency occurrence.

The aggregated results are published periodically, every few seconds (configurable). In order to do that, firstly, we need to decay the counters based on the current epoch. This step is necessary because word occurrences change asynchronously within different time-windows. In case a specific word has stopped appearing, its corresponding counter will eventually approach to zero and will be removed from the Hashmap when exceeding a certain threshold (e.g., 0.00001). After the counters have been decayed, we sort them by their corresponding frequencies to find the Top-K entries.


```
1-Minute Window
Data : {"CallCenterID":"CallCenter1","Top10Words":[{"vibración":"6.01"},{"actualizado":"5.93"},
{"heredar":"5.12"},{"enredado":"5.03"},{"terco":"4.76"},{"sostenible":"4.67"},
{"solemne":"4.66"},{"buey":"4.53"},{"desilusión":"4.51"},{"zumbido":"4.48"}],
"SentimentScore":76.04,"TimeWindow":60,"Epoc":1612546433225}

1-Hour Window
Data : {"CallCenterID":"CallCenter1","Top10Words":[{"robada":"97.91"},{"hostil":"97.57"},
{"ebrio":"97.40"},{"despejar":"96.42"},{"Renacimiento":"96.27"},{"sujeción":"96.22"},
{"caos":"95.85"},{"liberación":"95.43"},{"guiar":"95.40"},{"autónomo":"95.21"}],
"SentimentScore":71.55,"TimeWindow":3600,"Epoc":1612546433225}

1-Day Window
Data : {"CallCenterID":"CallCenter1","TopKWords":[{"redundante":"105.68"},{"inspirar":"105.56"},
{"usurpador":"105.32"},{"sarcasmo":"104.22"},{"rampante":"103.2535878994489"},
{"carnicero":"103.12522736666799"},{"aspiraciones":"103.09"},{"obsoleto":"102.99"},
{"pesimismo":"102.86"},{"patético":"102.58"}],
"SentimentScore":71.43,"TimeWindow":86400,"Epoc":1612546433225}
```


Figure 4.1: The aggregated results for different time windows.

4.3.5 Data Model

The aggregated analytics are produced periodically, every 5-10 seconds(configurable), using JSON format. The structure is depicted in Figure 4.1 and, as we can see, we use basic types to represent a single value (e.g., integers, floating point, epoch times, strings), and arrays for bundled values (e.g., sets and tables). In particular, each tuple is of the form:

$$\{\texttt{ccid}, \texttt{Epoch}, \texttt{Time Window Id}, \texttt{Sentiment Score}, \texttt{top K words}\}$$

where `ccid` is the call center id where the entries came from, `Epoch` the timestamp of our processing stage, `Time Window Id` is which window was processed, `Sentiment Score` is the total sentiment score of the specific time window along with the top-k words occurred.

The use of JSON allows post-processing modules to easily interpret the produced results using arbitrary schemes for different purposes. Furthermore, each JSON message can be easily enriched with contextual data. For instance, it can be extended with other data, or tagged by certain modules in the processing pipeline.

Chapter 5

System Evaluation

In this section we present the performance results of our GPU-based streaming analytics component. On the first section, we will evaluate our work over a CPU-based pattern matching algorithm implemented in Java. Moving on, we will present our preliminary evaluation on an illustrated streaming scenario.

5.1 GPU-based vs CPU-based pattern matching

Once we have implemented our work over the JNI framework we need to test it's performance over a CPU-based pattern matching implemented by Robert Bor et al [20]. This work also uses the Aho-Corasick algorithm and it is fully implemented in Java.

Both implementations we will have to perform the pattern matching operation on 100k entries (pairs) and measure the performance with different matching percentage scenarios. The goal here is to determine which device is more suitable to use on a large scale amount of data and if the matching percentage affects the performance of the current device.

Experimental set up: For the evaluation, we acquired a dataset with unique name/surname pairs, from this dataset we took a small sample of 500 pairs and we uploaded each pair in TerracottaDB via the Producer instance. Each pair was uploaded 200 times in order to reach the desired amount of data for this benchmark. Both programs are used by a Consumer instance in order to fetch the uploaded pairs and trigger their payload on the specific device implementation. For this experiment we used a NVIDIA GPU GTX980 and a Intel Xeon E5-2697 CPU. In this setup both implementations are called to perform the pattern matching operation using 60K patterns in 13 different matching scenarios. In each scenario we will replace N patterns within the 60K input pattern file in order to match the desired percentage. Each of the scenarios will be executed 100 times for both implementations, then we will sort the execution time values and report the median of each work on the current benchmark.

Results: Figure 5.1 shows the median values achieved for each implementation in

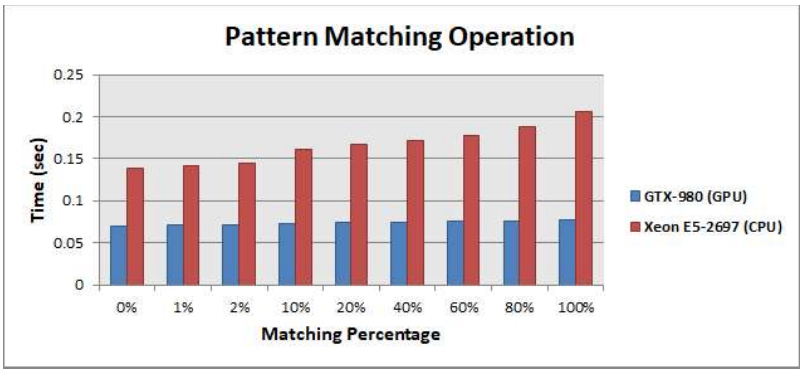

Figure 5.1: Computation time performance over the scenarios

each scenario. As we can see our GPU-based implementation is out performing the CPU-based implementation with a gaining performance starting from 66.28% up to 90.84% depending on the matching percentage. Our implementation has almost a constant time of execution in each scenario, on the other hand the CPU-based performed better on the first scenario where no matching patterns were present while having all matching patterns present the performance decreased.
Concluding remarks: This evaluation clearly states that GPU-based pattern matching is performing better than the CPU-based one, because our GPU-accelerated pattern matching splits the workload among the threads while the CPU implementation performs the workload in a serial manner. Performing our experiments in small set of data the CPU implementation will outperform our implementation and this is due the memories copies that are needed to be performed in order to transfer the data to the device. Thus, our implementation works well on large amount of data.

5.2 Performance of Call Center Evaluation

In this section, we show the performance of our approach for providing real-time sights regarding the performance of Call Centers. This evaluation will be done in two parts using a NVIDIA GPU GTX980. In the first part we will evaluate the time and throughput performance along with the cost of our added functionalities of our GPU-streaming analytics solution, for this part we will generate synthetic data which will represent real transcripts streamed by Call Centers. Moving on, the second part will measure the accuracy of our model over real and labeled data.

Measuring the accuracy of our model will not ended up as expected. Further examination revealed this is a consequence of the specific lexicon we were provided with, which is a generic sentiment lexicon. Thus, in the last section we re-evaluate our model's accuracy using a domain specific lexicon on a different dataset.

Token	Polarity	STD
acertado	0.708	0.149
admirable	0.906	0.125
improperio	-0.542	0.072
inanimado	-0.687	0.063
payaso	0.5	0.0

Table 5.1: Lexicon format (Token = Sentiment Word, Polarity = Sentiment Value, STD = Standard Deviation related with ambiguity of the polarity estimation).

5.2.1 Time & Throughput Performance

Experimental set up: To measure the performance of our work, we created 6 scenarios. Each of the scenario will contain different amount of transcripts which will needed to be processed and provide the real-time insights. For this evaluation we were provided with a Spanish lexicon, this lexicon can be found in [1]. Also, Table 5.1 shows the format of the used Lexicon. More specifically the lexicon contains more than 8K words along with their sentiment value and will be used as an input pattern file in our work. To make this scenario more realistic we will generate synthetic transcripts from a set of words within the lexicon above and each transcript will represent one the three Call Center. These synthetic transcripts were generated as follows. For each transcript we randomly selected 500 words from the lexicon provided from above and upload it in TerracottaDB via the Producer instance. With this set up we will be able to report the amount of transcripts or words processed by the GPU each second, also by filling each transcript with words from the input file of patterns we will make this benchmark more exciting, since the reporting array from the GPU operation will need to report all of the matching patterns for each transcript which has processed. As we said from above, the goal here is to measure the performance which can be achieved from our implementation and the cost of our added functionalities. Each experiment was conducted 100 times, we sort the values and reported the median.

Results: Figure 5.2 shows the total time of processing the input transcripts and aggregate the results. The x-axis represents the amount of transcripts along with the total words in each scenario while y-axis indicates the total time of each of the operation which takes place. Our implementation consists on 3 main operations, the first one is the native call which will trigger the pattern matching operation on the GPU device and return the reporting results, the second operation has to parse the results and create the corresponding Call Center entity which the data are coming from. Once the entity is created it then creates three Hashmap instances in order to store the matched words (patterns) appeared within the transcripts per minute, hour and day. This operation increases the counter of each matched word reported from the GPU, updates the words timestamp (epoch) and also calculates the sentiment score per Call Center entity based on the counters of the

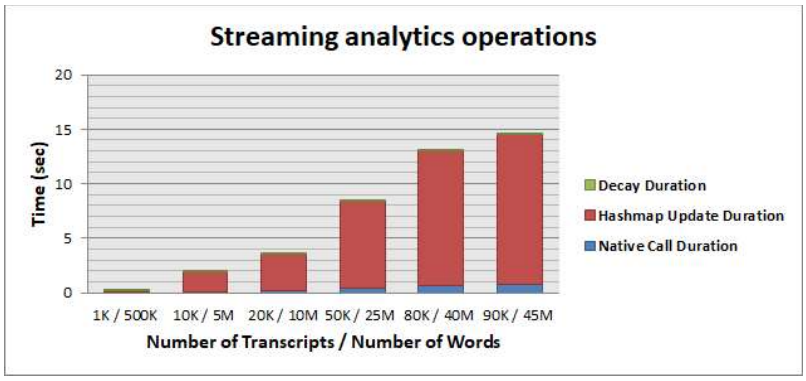

Figure 5.2: Breakdown of each of the operations performed during processing call transcripts

words. Once all the results are accumulated we perform a sorting value operation in order to announce the Top-K words appearing with in the transcripts. The last operation will parse the Hashmap above and perform a mathematical decay function on the counters of each word in each window and also check if the decayed counter reached the desired threshold so it can be removed.

All the operation timers increase as the input transcripts increase, the most time consuming operation is updating the Hashmap values along with calculating the sentiment score for each window timer while on the other hand applying the decay function in each time window.

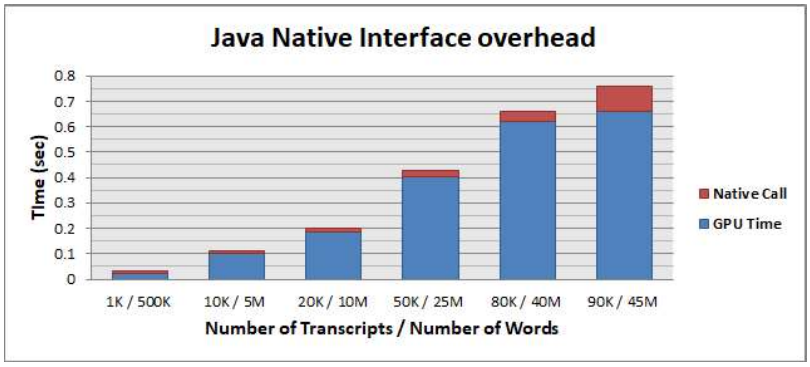

Figure 5.3: Additional overhead added when multiple native calls occur

When performing our measurements we noticed an overhead was added when we used the native function call. This native function triggers the GPU instance from C side, in some cases the data will not fit inside the GPU's buffer at once, so multiple executions are needed to take place. Figure 5.3 shows the actual GPU execution time in which was measured in C part, this execution contains all the memory copies which are performed along with pattern execution and returned

results. Having multiple native calls adds an overhead to the execution depending on the number of calls, for example in the last bar-plot where 90k are present the native call was triggered 7 times while in 80k transcripts only 5 calls are needed to be made. This overhead is minimal when the native call is needed to be triggered once, this can be clearly be noticed in the first two bar-plots with 1k and 10k transcripts.

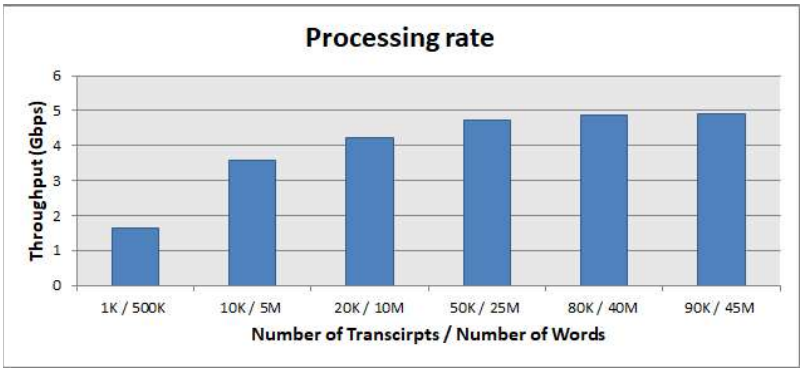

Figure 5.4: Throughput achieved over Streaming Analytics scenarios.

Lastly, Figure 5.4 shows the throughput which achieved in each scenario. For the scenarios which needed more than one execution we sorted the values and reported the median. As you can see our implementation can achieve almost 5 Gbit/s or in terms of words, the GPU execution can process 45M words per second. Also having multiple native calls will cause a performance decrease depending on the number of calls performed.

Concluding remarks: Our evaluation results indicate that our implementation can provide real-time insights yielding high throughput. The most time consuming operation is the Hashmap Update operation which consumes a significant amount of the total time, we plan to optimize as part of our future work.

5.2.2 Accuracy Performance

Experimental set up: In this section we evaluate our solution for computing sentiment score (described in section 4.4.3) in a specific scenario, using 700 call transcripts and we used the same sentiment lexicon both provided by TID. The transcripts are in Spanish language, each of which is of different size. Moreover, they have been properly anonymized and do not contain any sensitive data, such as names, addresses, phone numbers, etc. These transcripts came along with their CSI (Customer Satisfaction Index) value that is based on self-reported feedback. The CSI values range between 1 and 5; transcripts with CSI=[1, 2] are classified as negative while transcripts with CSI=[4, 5] are classified as positive. In the given dataset we did not have any transcripts that were classified as neutral. From our end we will use the sentiment score of each of the words in the transcripts, the

lexicon that we use can be found at [1]. In the end we will normalize the sentiment score to a scale of [1-5], so it can be compared directly with the given CSI. We conducted three different approaches for this evaluation.

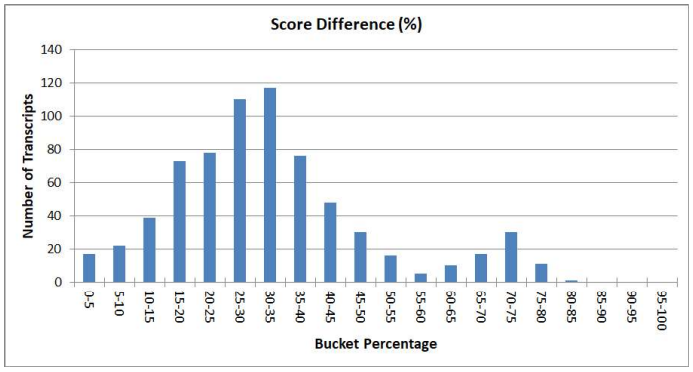

Figure 5.5: The accumulated number of transcripts for a given percentage difference.

Results: In our first experiment we accumulate and normalize the sentiment scores from each transcript and create 20 buckets, each one indicating the percentage difference of the sentiment score with the ground truth CSI. In Figure 5.5 x-axis indicates the buckets we described above and y-axis indicates accumulation of transcripts in each bucket. Most of the transcripts (42%) are accumulated between buckets 5 (20-25%) and 8 (35-40%) while the highest peak is around 30-35% difference with 118 transcripts. Only 5% of the transcripts are accumulated between buckets 1 (0-5%) and 2 (5-10%).
In order to have a better understating we took each transcript CSI and subtracted our GPU CSI that we have acquired. We plotted the above results to a scatter plot (Figure 5.6), x-axis indicated the transcripts id and y-axis indicates the value of the above subtraction. We can clearly state that our approach gave a higher CSI value than the ground truth CSI value in almost 83% of the total transcripts. For this experiment we have also checked if all of CSI values move in the same direction as the ground truth CSI by using Pearson's correlation. Using this method the value which was reported was 0.10, this clearly indicates that our CSI values do not correlate with the ground truth CSI values.

Since the first experiment didn't yield good results we contacted a second experiment but this time instead of trying to compare the GPU sentiment score to the ground truth CSI we will reduce the factors of the problem to two (binary). We took each transcript given CSI and classified the transcripts with CSI value >=4 as positive and the rest as negative. Since our approach has a tendency to give higher values we have to manually find the best threshold to classify each transcript. The optimal threshold we found was to classify transcripts with >=3.5 as positive and the rest as negative.

For this scenario we will regard a "positive" case as a call transcript that

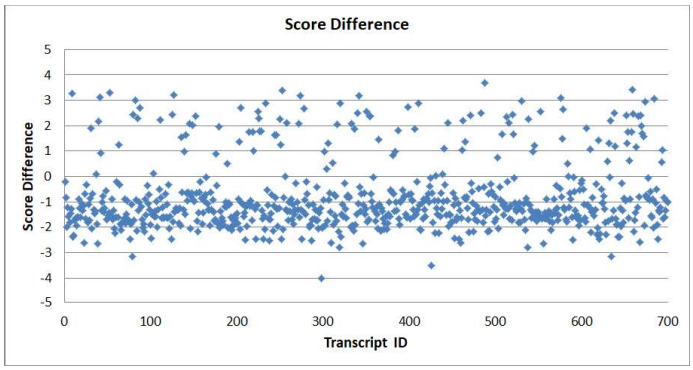

Figure 5.6: The actual difference of Sentiment and CSI scores for each of the transcripts in our dataset.

received a low satisfaction score by the customer, since negative transcripts are the most critical ones to be successfully discovered and reported. Figure 5.7 is a confusion matrix showing the overall performance of our model. In this case our model reported around 75% accuracy.

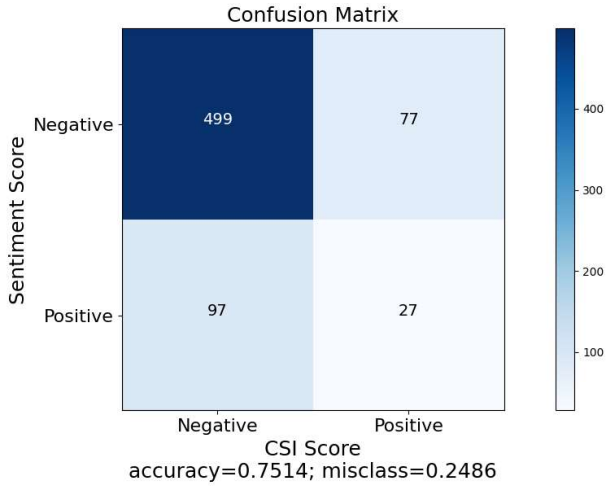

Figure 5.7: Confusion matrix of the overall performance.

Another metric we used for this experiment was the adjusted F-scoring method (see Equation 5.1). The F-score method is a way of combining the precision and recall of the predictive model and evaluates the binary classification model on a dataset. The adjusted F-scoring method allows us to weigh precision or recall more highly. Since negative transcripts are the most critical ones to be successfully discovered and reported, we use the adjusted F-scoring with parameter b=2 since recall is more important in this case. The overall F2-score was 25% which is not

desired.

The third experiment we contacted was a ranking method known as TF-IDF which is a numerical statistic that is intended to reflect how important a word is to a document/transcript in a collection or corpus. For this method we had to offline calculate the IDF (Inverse document frequency) for all the available transcripts in order to measure how much information a certain matched word/pattern provides.

Once the IDF was calculated we had to calculate the TF (Term Frequency),which is a raw count of a term in a document. This can be calculated easily from our approach since our solution reports which and how many matches were found within a transcript. As a next step, we have to multiply each term's frequency found within a corpus with it's according weight(IDF). Having said that our first approach was to correlate again the sentiment score with the ground truth CSI value since more important words, in order to do that we have to multiply the TF-IDF value with each term's sentiment score. This experiment didn't come as expected, since positive words/patterns were ranked higher than the negative ones.

$$F_b = (1 + b^2) \times \frac{preccision \times recall}{(b^2 \times preccision) + recall} = \frac{(1 + b^2) \times TP}{(1 + b^2) \times TP + (b^2 \times FN) + FP} \tag{5.1}$$

Where:

- *precision*: The number of true positives divided by the number of false positives plus true positives.
- *recall*: The number of true positives divided by the number of true positives plus false negatives.
- TP: The number of true positives classified by the model.
- FN: The number of false negatives classified by the model.
- FP: The number of false positives classified by the model.

Concluding remarks: Our preliminary results indicate that the Sentiment scores computed with the specific lexicon do not correlate with the corresponding CSI scores. By further manual examination it was revealed that this is a consequence of the specific lexicon used, which is actually a generic sentiment lexicon for the Spanish language and is biased towards domain-general contexts. Such a lexicon cannot provide a useful indication regarding the resulting customer satisfaction levels; a domain-specific lexicon with appropriately weighted terms tailored for revealing customer satisfaction in the domain of call-centers should be used instead as input.

5.3 Evaluation using a domain specific lexicon

In this section we evaluate our proposed idea solution following the guidelines provided by Andrew L. et al [43] and William L. et al [34]. As their prior works suggest, having a more tailored lexicon suitable for the specific purpose, will yield better results.

Experimental set up : We will evaluate our work over a movie review testing dataset containing near 25k reviews (50% negative and 50% positive) and use as input pattern file a domain specific lexicon, used by Andrew L [2]. Each of the reviews has been labeled with a positive or negative indication along with a variety of features for training a machine learning model. In this experiment we will regard a "positive" case as a negative movie review, since our goal is to correctly classify the negative reviews with the help of a more accurate lexicon.

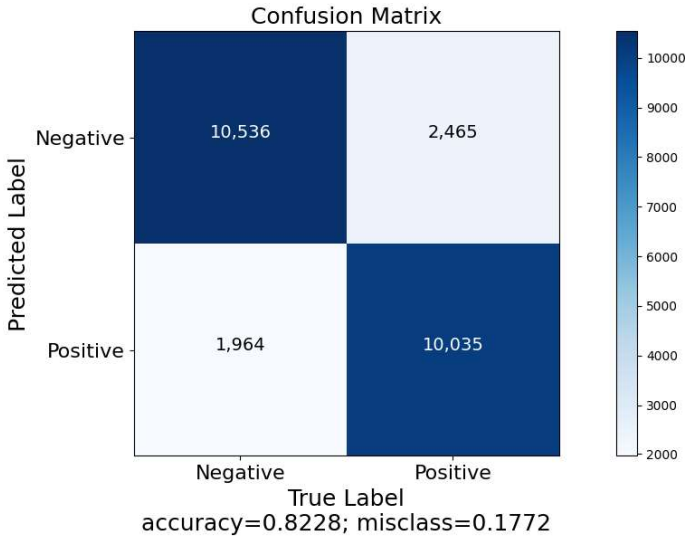

Figure 5.8: Confusion matrix of the overall performance using domain specific lexicon.

Results: Figure 5.8 presents the confusion matrix with the overall performance for binary classification over the dataset. Our model correctly classified 80.28% "positve" reviews. The overall accuracy reported from our model is 82% while the original authors accuracy using only the bag of words lexicon was 88%, this is obviously the main trade-off of our work. Our solution is implemented for aggregating large amounts of data almost real time using only a pattern matching algorithm and try to classify each review based on the accumulated score according to the input sentiment lexicon rather than using any more sophisticated methods used in machine learning models or any other feature vectors. The corresponding F1-score for this experiment is 81.92% and the F2-score is 80.92%.

Concluding remarks: Our preliminary results indicate that our proposed idea

performance is affected by the quality of the lexicon provided. We did not report better results than Andrew L. et al work due to the simplicity of our model but having a tailored lexicon can yield better results rather than using a general purpose lexicon. The above experiment shows that our work can be used in both streaming and offline analysis and provide a good classification of the input data.

Chapter 6

Discussion and Limitations

In this section we will discuss about the limitations of our work along with some directions where our work can be also useful.

6.1 Hashmap operations

As mentioned in section 5.3, operations performed by the Hashmap structures are consuming a significant time of execution. This is due to the size of the lexicon and the frequencies of popular words. One solution to this issue is to allocate the instances of all Call Centers and create the number of desired time windows filled with each candidate pattern word and initialize the frequency counter of the word with zero as the module begins to instantiate the GPU instance. Another optimization which can be performed is the following, instead of parsing the results array and increase the words frequency counter by one we can use an array in order to accumulate the word occurrences and then perform the update operation in Hashmap, for example if a word is reported N times we will accumulate the counters within the array and then perform a single update operation within the Hashmap and reset the corresponding counter with in the array.

Performing the above solutions, will cause a slow start up of our implementation and a memory overhead but in the end no further allocation will be needed and the update of the frequency counters will be as many as the unique words reported.

6.2 Framework Overhead

As mentioned in Concluding Remarks section 5.3 regarding the performance evaluation, using Java Native Interface and performing multiple native function calls add an overhead to the execution depending on the number of calls. A solution to this problem is explore other frameworks or even use a CPU-to-GPU memory mapping technique such as [12], in order to gain performance and extinguish the JNI overhead.

6.3 Model Accuracy

As mentioned in section 5.4, our model's performance accuracy is strongly affected by the quality of the input sentiment lexicon since is data agnostic. Such tailored lexicons are hard to find since special Natural Language Processing (NLP) machine learning models are needed to create such a lexicon. In many works such as [67, 25, 49, 42] are using the TF-IDF statistical method to generate these lexicons. The usual pipeline of such models is as follows, 1) Load the input data and remove stopwords, 2) Perform the TF-IDF method on the clean data and generate the embedding weight matrix, 3) Perform Principal Component Analysis (PCA) on the matrix (Optional), 4) Feed the processed matrix into a classifier model.

From our perspective our implementation can help these models to generate such lexicons by offloading steps 1 & 2 to the GPU in order to speed up the offline training of their model using only a general purpose lexicon (without stopwords).

Chapter 7

Related Work

Pattern matching algorithms are highly used in various applications, not only in computer science (e.g intrusion detection [59, 21, 57], spam filtering [52, 65, 26], digital forensics [4, 3]) but also in other relevant fields, such as computational biology, chemistry and nanotechnology. In advance some of the works such as [22, 23] leveraged pattern matching to sentiment analysis. These works are using domain specific sentiment lexicons in order to extract the sentiment score as we do. The main difference of our work is the variety of devices which the pattern matching operation can be used while their work is only limited to the CPU device. Exploiting high computational will significantly decrease the total time execution and increase their processing rate as we shown in our evaluation section.

Using big data analytics to extract the Costumer Satisfaction Index (CSI) is very important. The Costumer satisfaction index is an attempt to measure how satisfied customers are with a product or a service. The assumption being that the more satisfied a customer is the more likely to stay as a customer. This measurement is applied in many domains such as Manufacturing, Hospitality, but process each of the costumer reviews will help the provider to identify mainly their weak spots. Many works such as [71, 41, 56, 69] achieve the above goal by using Deep learning models boarded on different GPU architectures. Although, all of the above works are limited to offline analysis while our work can be used for both streaming and offline analysis.

In the real-time sentiment analysis area, many works are using machine learning approaches. To begin with, [51] proposed real-time sentiment analysis approach which uses Multinomial Naive Bayes Algorithm with unigram model and follow up by SentiWordNet [15] algorithm using POS tags to classify the tweets. SentiWordNet is similar to the SocialSent tool to offline generate the sentiment lexicon. Moving on, [38] is also performing sentiment analysis on tweets. This work extracts semantic and morphological features from various tweets and then uses a supervised model to classify them. Prakruthi V [48] implemented a scalable system implementation for real-time sentiment analysis in Storm. This work uses a supervised approach to perform analysis on tweets. The primary issue with

these approaches is that it requires manually labeled training data for achieving relatively good accuracy.

In [50] they presented a distributed system to perform real-time sentiment analysis on various social streams. Their work runs atop Apache Storm [31] and uses a sliding window to process the incoming traffic and for the classification they are using a fusion of Multinomial Naive Bayes and Hoeffding Tree classifiers to achieve better results.

G.Amati [19] proposed a scalable Information Retrieval system for near real-time analytics on social networks. This approach uses a supervised learning technique that consists in the use of a linear regression to predict and smooth a sentiment category size on the basis of a cumulative score of documents. A potential bottleneck in this works is decoration component. This component decomposes the incoming traffic extract various information such as timestamps and metadata. The decorator adds significant overhead in the processing phase and needs to be adjusted to the current distributed version of the system.

Imane El Alaou [29] developed a novel adaptable approach that aims to extract people opinion about a specific subject by relying on social media contents. The proposed technique consists to first building a dictionary of word's polarity based on a very small set of positive and negative hashtags related to a given subject, then, classifying posts into several classes and balancing the sentiment weight by using new metrics such as uppercase words and the repetition of more than two consecutive letter in a word. However, the proposed approach still suffers from some shortcomings. First, it does not distinguish the impact degree of the different metrics in order to accentuate a feeling. Second, the system is a prototype designed to assess the ability of automatically constructing dynamic dictionary using small samples.

In [10] proposed a Sentiment Analysis as a Service (SAaaS) framework that abstracts sentiments and opinions from multiple social information services, analyses and transforms into meaningful information. The experiments were conducted with limited data. To be specific, only 404 reviews were used for the evaluation rather than testing the perfomance over real-time large data.

Similar to our our work, HappyMeter [47] a real-time data processing infrastructure to evaluate public sentiment changes in the context of Twitter using a Bag of Words. This work performance, including ours, are highly dependable to the quality of the input sentiment lexicon.

Lastly, taking into consideration the growth trend of big data, hardware vendors proposed CPU-GPU integrated architectures that integrate CPU and GPU on the same chip. This integration provides new opportunities for fine-grained cooperation between CPU and GPU for optimizing streaming processing. With this upcoming trend some works are already being developed such as[70, 40]. In these works they are trying to take advantage of both architectures, and also provide efficient mechanism for handling dynamic stream queries. We also believe using integrated GPUs may reduce the data transfer overhead but a combination of CPUs, integrated GPUs and discrete GPUs working together in order to achieve

high streaming processing rate. Thus, we will explore this idea in the near future.

Chapter 8

Conclusions and Future Work

In this section we present a summary of the contributions of this work (section 8.1) and some thoughts on future work (section 8.2).

8.1 Summary of Contributions

In this work, we develop a GPU-accelerated streaming analytics component which can provide real-time insights from streaming sources such as Call Centers in order to improve their Quality of Service. More specifically, this component is implemented in Java Native Interface and uses the GPU in order perform the simultaneous pattern matching operation over the input data and reports back the matching results. Using this report we are able to aggregate the data in various time windows periods and report the sentiment score along with Top-K words which occurred. In addition, this work can be executed on the vast majority of dataparallel devices, such as AMD and NVIDIA integrated and discrete GPUs. Furthermore, we demonstrated our solution to the public, having three different Call Centers across Spain streaming their call transcripts and induced the insights in real-time.

8.2 Future Work

Our aim for the future developments is to explore new techniques in order to avoid the overhead added from the current framework and find more efficient data storing techniques which will benefit both space and performance. By this way, we will eventually increase the performance for real-time analytics operations.

Latency is also an additional factor that is extreamly important to take into conciderations while we are furthuer developing our tool, since in a noumerous of cases the incoming traffic will cause multiple executions on the GPU device. This latency is created when only a portion of the incoming traffic can be processed by the device while the rest are in queue to be processed. This gap must be

measured and taken into consideration and try to scale our performance using multiple GPUs.

Another important mechanism we should implement is a scheduling program. This scheduler will monitor the incoming traffic and decided where the process should be executed. For example, at a certain point within the day the incoming traffic will cause underutilized GPU executions since the input data won't fill the entire buffer, in this case performing the analysis process on the CPU device will be preferred.

On a later stage where all the necessary implementations take place we then need to compare our tool among different state of the art streaming analytics platforms. This step will help us identify the trade-offs when using our tool compared to existing software.

In addition, as we discussed in Chapter 6, we will try to evaluate the performance gained to generate the emended weighted matrix and also use this matrix as input to a classifier in order to generate domain specific lexicons efficiently.

Finally, we are planning to implement special mechanisms in case of a device crash to leverage the workload to an available GPU device. By scaling our component this way we would gain more perfomance and its tolerance towrdays any false will be increased.

8.3 Conclusion

The GPU-accelerated component uses a sentiment-based lexicon and performs pattern matching operation over the data to extract valuable information such as sentiment score and frequent used words. Also, tool can also can track trends for both short and long term events by manually adjusting the desired time window intervals. In addition, our work can be used in many other different domains that require real-time processing of big volume of data, as well. Moreover, our work can even be used as an offline analysis tool since some companies prefer to store their data.

The evaluation shows the performance of our work in a streaming scenario and shows the performance capabilities using only one device. The performance of our implementation can achieve almost 5 Gbit/s or in terms of words, the GPU execution can process 45M words per second. The accuracy of our model is highly dependable by the quality of the input lexicon.

Chapter 9

Appendix

This work has been funded by the Industrial-Driven Big Data as a Self-Service Solution (I-BiDaaS)project and was demonstrated live to the public providing real time sights from three different Telefonica Call Centers across Spain, for this demo we integrated our component with third party applications. Figure 9.1 shows the integration with Apama streaming platform and our results have been visualized by another component. Apama provides a MemoryStore driver for TerracottaDB.

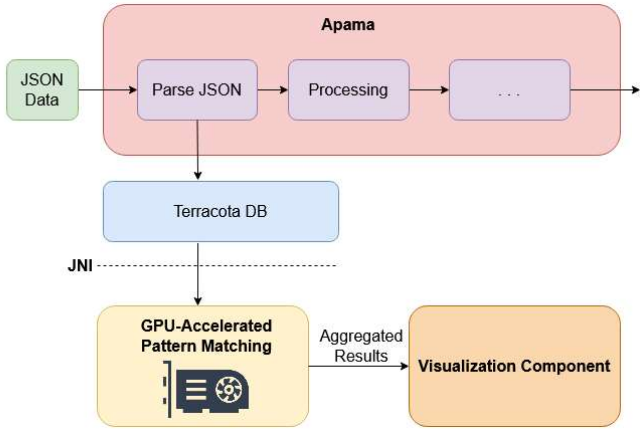

Figure 9.1: Integrating GPU-based component with third party applications

This driver uses the TCStore API to allow Apama to read and write records in TCStore datasets, which may also be read and written by other Apama correlators and non-Apama components such as other Software AG products or custom clients written against the TCStore API. The main concept is to utilize the TerracottaDB as a buffer between the Apama Stream Processing engine and the GPU.

Once the incoming data have been written to the TerracottaDB, the "GPU-broker" process reads them using a separate Consumer instance, performs the pattern matching operation, aggregates the results and then delivers them to the

visualization component via a universal message passing application.

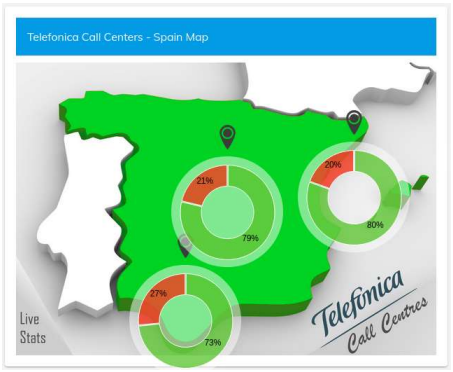

Figure 9.2: Sentiment Score at t_0

Figure 9.3: Sentiment Score at t_1

Figure 9.4: Reporting the performance of the three Call Centers over 5-10 seconds

The following Figures are actual snapshots we that we took when we presented our work to the public. In this demo real transcripts were streamed over three different Call Centers and the following results which were reported from our implementation. Figure 9.4 shows how the sentiment is changing according to the streamed transcripts for each Call Center.

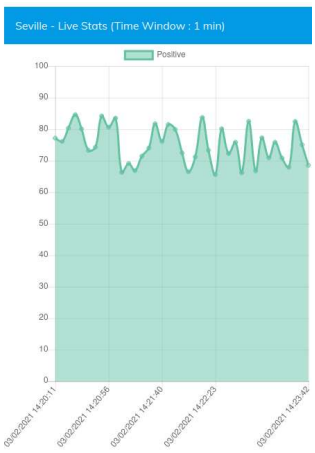

Figure 9.5: Sentiment Score changing in one minute time window

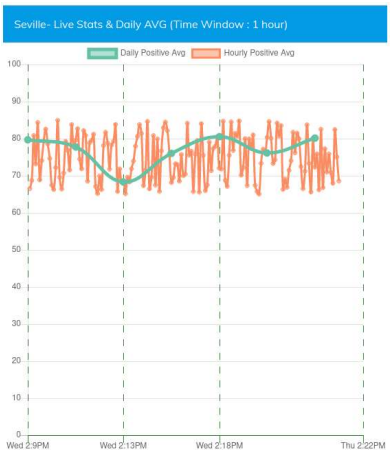

Figure 9.6: Average Sentiment Score over hour time window

Using the Rolling Window method we are able to provide insights for each time period per Call Center, Figure 9.5 shows the difference of the sentiment over

1 minute while Figure 9.6 provides the average sentiment over hour. These results are referred to a specific Call Center located in Seville. Lastly, Figure 9.7 provides the Top-10 words which are more frequently used over the three time window periods.

Seville - Top 10 Words (Sorted by Frequency)

#	Last Minute	Last Hour	Last 24h
1	paz	inexorablemente	inexorablemente
2	puÃ±al	enÃ©rgico	enÃ©rgico
3	lean	derrotado	derrotado
4	indiferencia	insensible	insensible
5	hard	confundido	atestiguar
6	divergente	dark	matador
7	aspirar	peligrar	rupturas
8	discriminar	matador	perjudicial
9	boquete	latente	horda
10	gravemente	perjudicial	peligrar

Figure 9.7: Top 10 words over the three time windows.

Bibliography

[1] Ml-senticon: A layered, multilingual sentiment lexicon (english, spanish, catalan, galician, basque). `http://www.lsi.us.es/~fermin/index.php/Datasets`.

[2] Movie Review dataset: https://ai.stanford.edu/ amaas/data/sentiment/.

[3] Sdhash on GitHub. `https://github.com/sdhash/sdhash`.

[4] Sdhash tool. `http://roussev.net/sdhash/sdhash.html`.

[5] I-BiDaaS Deliverable 4.2. Distributed event-processing engine. `https://www.ibidaas.eu/sites/default/files/docs/ibidaas-d4.2.pdf`.

[6] I-BiDaaS Deliverable 4.3. Streaming analytics and predictions. `https://www.ibidaas.eu/sites/default/files/docs/ibidaas-d4.3.pdf`.

[7] Software AG. Terracotta documentation. `https://www.terracotta.org/documentation/`.

[8] Rodrigo Agerri, Xabier Artola, Zuhaitz Beloki, German Rigau, and Aitor Soroa. Big data for natural language processing: A streaming approach. *Knowledge-Based Systems*, 79, 11 2014.

[9] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, June 1975.

[10] Kashif Ali, Hai Dong, Athman Bouguettaya, Abdelkarim Erradi, and Rachid Hadjidj. Sentiment analysis as a service: A social media based sentiment analysis framework. 06 2017.

[11] Yigal Arens, Chin Chee, Chun-Nan Hsu, and Craig Knoblock. Retrieving and integrating data from multiple information sources. *Int. J. Cooperative Inf. Syst.*, 2:127–158, 06 1993.

[12] Abu Asaduzzaman, Deepthi Gummadi, and Chok Yip. A talented cpu-to-gpu memory mapping technique. pages 1–6, 03 2014.

[13] Big Data Value Association. Technologies and applications for big data value. `https://www.bdva.eu/node/1616`.

[14] S. Athmaja, M. Hanumanthappa, and V. Kavitha. A survey of machine learning algorithms for big data analytics. In *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pages 1–4, 2017.

[15] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. volume 10, 01 2010.

[16] A. Bar, A. Finamore, P. Casas, L. Golab, and M. Mellia. Large-scale network traffic monitoring with dbstream, a system for rolling big data analysis. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 165–170, 2014.

[17] Sonia Bergamaschi, Domenico Beneventano, Federica Mandreoli, Riccardo Martoglia, Francesco Guerra, Mirko Orsini, Laura Po, Maurizio Vincini, Giovanni Simonini, Song Zhu, Luca Gagliardelli, and Luca Magnotta. *From Data Integration to Big Data Integration*, pages 43–59. 05 2018.

[18] Pramod Bhatotia, Umut A. Acar, Flavio P. Junqueira, and Rodrigo Rodrigues. Slider: Incremental sliding window analytics. In *Proceedings of the 15th International Middleware Conference*, Middleware '14, pages 61–72, New York, NY, USA, 2014. Association for Computing Machinery.

[19] Marco Bianchi, Giambattista Amati, Simone Angelini, Luca Costantini, and G. Marcone. A scalable approach to near real-time sentiment analysis on social networks. volume 1314, 12 2014.

[20] Robert Bor. Java implementation of the aho-corasick algorithm for efficient string matching. `https://github.com/robert-bor/aho-corasick`, 2020.

[21] Byungkwon Choi, Jongwook Chae, Muhammad Jamshed, KyoungSoo Park, and Dongsu Han. DFC: Accelerating string pattern matching for network applications. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 551–565, Santa Clara, CA, 2016. USENIX Association.

[22] Jennifer Contreras, Melvin Ballera, Ace Lagman, and Jennalyn Raviz. Lexicon-based sentiment analysis with pattern matching application using regular expression in automata. 03 2019.

[23] Jennifer O. Contreras, Melvin A. Ballera, Ace C. Lagman, and Jennalyn G. Raviz. Lexicon-based sentiment analysis with pattern matching application using regular expression in automata. In *Proceedings of the 6th International Conference on Information Technology: IoT and Smart City*, ICIT 2018, pages 31–36, New York, NY, USA, 2018. Association for Computing Machinery.

[24] G. Cormode, F. Korn, and S. Tirthapura. Exponentially decayed aggregates on data streams. In *2008 IEEE 24th International Conference on Data Engineering*, pages 1379–1381, 2008.

[25] Bijoyan Das and Sarit Chakraborty. An improved text sentiment classification model using tf-idf and next word negation. 06 2018.

[26] Sarah Delany, Mark Buckley, and Derek Greene. Sms spam filtering: Methods and data. *Expert Systems with Applications*, pages 9899–9908, 02 2013.

[27] Dimitris Deyannis. A massively parallel regular expression and string matching engine for commodity hardware. Master's thesis, University of Crete, University Campus, Voutes, Heraklion, GR-70013, Greece, 2017.

[28] Lian Duan and Ye Xiong. Big data analytics and business analytics. *Journal of Management Analytics*, 2(1):1–21, 2015.

[29] Imane El Alaoui, Gahi Youssef, Rochdi Messoussi, Youness Chaabi, Alexis Todoskoff, and Abdessamad Kobi. A novel adaptable approach for sentiment analysis on big social data. *Journal of Big Data*, 5, 03 2018.

[30] Apache Software Foundation. Apache spark - unified analytics engine for big data. `https://spark.apache.org/`.

[31] Apache Software Foundation. Apache storm. `https://storm.apache.org/`.

[32] Venkat Gudivada, Dhana Rao, and Vijay Raghavan. *Big Data Driven Natural Language Processing Research and Applications*, volume 33, pages 203 – 238. 07 2015.

[33] William Hamilton, Kevin Clark, Jure Leskovec, and Dan Jurafsky. Inducing domain-specific sentiment lexicons from unlabeled corpora. 06 2016.

[34] William L. Hamilton, Kevin Clark, Jure Leskovec, and Dan Jurafsky. Inducing domain-specific sentiment lexicons from unlabeled corpora. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 595–605, Austin, Texas, November 2016. Association for Computational Linguistics.

[35] Adeel Hashmi and Tanvir Ahmad. Big data mining: Tools and algorithms. *International Journal of Recent Contributions from Engineering, Science & IT (iJES)*, 4:36, 03 2016.

[36] Dorota Jelonek. Big data analytics in the management of business. *MATEC Web of Conferences*, 125:04021, 01 2017.

[37] J. Jiang and L. Zeng. Research on individualized teaching based on big data mining. In *2019 14th International Conference on Computer Science Education (ICCSE)*, pages 56–59, 2019.

[38] Maria Karanasou, Anneta Ampla, Christos Doulkeridis, and Maria Halkidi. Scalable and real-time sentiment analysis of twitter data. 12 2016.

[39] M. Khader, A. Awajan, and G. Al-Naymat. The effects of natural language processing on big data analysis: Sentiment analysis case study. In *2018 International Arab Conference on Information Technology (ACIT)*, pages 1–7, 2018.

[40] Alexandros Koliousis, Matthias Weidlich, Raul Castro Fernandez, Alexander L. Wolf, Paolo Costa, and Peter Pietzuch. Saber: Window-based hybrid stream processing for heterogeneous architectures. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 555–569, New York, NY, USA, 2016. Association for Computing Machinery.

[41] Cheng Li, Xiaoxiao Guo, and Qiaozhu Mei. Deep memory networks for attitude identification. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, pages 671–680, New York, NY, USA, 2017. Association for Computing Machinery.

[42] Jordi Luque, Carlos Segura, Ariadna Sanchez, Marti Umbert, and Luis Galindo. The role of linguistic and prosodic cues on the prediction of self-reported satisfaction in contact centre phone calls. pages 2346–2350, 08 2017.

[43] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[44] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 346–357. VLDB Endowment, 2002.

[45] Oracle. Java native interface. `https://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/intro.html`.

[46] E. Papadogiannaki, L. Koromilas, G. Vasiliadis, and S. Ioannidis. Efficient software packet processing on heterogeneous and asymmetric hardware architectures. *IEEE/ACM Transactions on Networking*, 25(3):1593–1606, 2017.

[47] Joaquim Perotti and Tina Tian. Happymeter: An automated system for real-time twitter sentiment analysis. *International Journal of Advanced Computer Science and Applications*, 8, 01 2017.

[48] V. Prakruthi, D. Sindhu, and D. S. Anupama Kumar. Real time sentiment analysis of twitter posts. In *2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS)*, pages 29–34, 2018.

[49] W. Pu, N. Liu, S. Yan, J. Yan, K. Xie, and Z. Chen. Local word bag model for text categorization. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 625–630, 2007.

[50] A. H. A. Rahnama. Distributed real-time sentiment analysis for big data social streams. In *2014 International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 789–794, 2014.

[51] R Rajput and Arun Solanki. Real time sentiment analysis of tweets using machine learning and semantic analysis. pages 687–692, 11 2016.

[52] David Sculley, Gabriel Wachman, and Carla Brodley. Spam filtering using inexact string matching in explicit feature space with on-line linear classifiers. 01 2006.

[53] Dipti Sharma, Munish Sabharwal, Vinay Goyal, and Mohit Vij. Sentiment analysis techniques for social media data: A review. In Ashish Kumar Luhach, Janos Arpad Kosa, Ramesh Chandra Poonia, Xiao-Zhi Gao, and Dharm Singh, editors, *First International Conference on Sustainable Technologies for Computational Intelligence*, pages 75–90, Singapore, 2020. Springer Singapore.

[54] Software AG. Apama Streaming Analytics platform. `http://www.apamacommunity.com/`.

[55] Zhaohao Sun, Huasheng Zou, and Kenneth David Strang. Big data analytics as a service for business intelligence. 10 2015.

[56] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. Dyadic memory networks for aspect-based sentiment analysis. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pages 107–116, New York, NY, USA, 2017. Association for Computing Machinery.

[57] A. Tumeo, O. Villa, and D. G. Chavarria-Miranda. Aho-corasick string matching on shared and distributed-memory parallel architectures. *IEEE Transactions on Parallel and Distributed Systems*, 23(3):436–443, 2012.

[58] Ahmed Unnisabegum, Mohammed Hussain, and Mubeena Shaik. Data mining techniques for big data, vol. 6, special issue ,, 08 2019.

[59] S. Vakili, J. M. P. Langlois, B. Boughzala, and Y. Savaria. Memory-efficient string matching for intrusion detection systems using a high-precision pattern grouping algorithm. In *2016 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 37–42, 2016.

[60] Giorgos Vasiliadis, Spiros Antonatos, Michalis Polychronakis, Evangelos Markatos, and Sotiris Ioannidis. Gnort: High performance network intrusion detection using graphics processors. volume 5230, pages 116–134, 09 2008.

[61] Giorgos Vasiliadis and Sotiris Ioannidis. Parallelization and characterization of pattern matching using gpus. 11 2011.

[62] Giorgos Vasiliadis and Sotiris Ioannidis. Parallelization and characterization of pattern matching using gpus. 11 2011.

[63] Giorgos Vasiliadis, Michalis Polychronakis, Spiros Antonatos, Evangelos Markatos, and Sotiris Ioannidis. Regular expression matching on graphics hardware for intrusion detection. volume 5758, pages 265–283, 09 2009.

[64] Giorgos Vasiliadis, Michalis Polychronakis, Spiros Antonatos, Evangelos Markatos, and Sotiris Ioannidis. Regular expression matching on graphics hardware for intrusion detection. volume 5758, pages 265–283, 09 2009.

[65] Rahul Verma and Joydip Dhar. Online spam filter for duplicate or near duplicate message content detection scheme. *Journal of Convergence Information Technology*, 9:23–30, 07 2014.

[66] H. Xu, G. Fan, and K. Li. Improved statistical analysis method based on big data technology. In *2017 International Conference on Computer Network, Electronic and Automation (ICCNEA)*, pages 175–179, 2017.

[67] Zhixiang Eddie Xu, Minmin Chen, Kilian Q. Weinberger, and Fei Sha. An alternative text representation to TF-IDF and bag-of-words. *CoRR*, abs/1301.6770, 2013.

[68] Changwon Yoo, Luis Ramirez, and Juan Liuzzi. Big data analysis using modern statistical and machine learning methods in medicine. *International neurourology journal*, 18:50–7, 06 2014.

[69] Zhigang Yuan, Sixing Wu, Fangzhao Wu, Junxin Liu, and Yongfeng Huang. Domain attention model for multi-domain sentiment classification. *Knowledge-Based Systems*, 155, 05 2018.

[70] Feng Zhang, Lin Yang, Shuhao Zhang, Bingsheng He, Wei Lu, and Xiaoyong Du. Finestream: Fine-grained window-based stream processing on cpu-gpu integrated architectures. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 633–647. USENIX Association, July 2020.

[71] Wei Zhao, Ziyu Guan, Long Chen, Xiaofei He, Deng Cai, Beidou Wang, and Quan Wang. Weakly-supervised deep embedding for product review sentiment analysis. *IEEE Transactions on Knowledge and Data Engineering*, PP:1–1, 09 2017.