REAL-TIME C2C MATCHING OF SOCIAL MEDIA MESSAGES

M.R.M. RILFI

148053N

Degree of Master of Science

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

May 2019

REAL-TIME C2C MATCHING OF SOCIAL MEDIA MESSAGES

M.R.M. RILFI

148053N

Dissertation submitted in partial fulfillment of the requirements for the degree Master of Science in Computer Science

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

May 2019

DECLARATION

I declare that this is my own work and this thesis/dissertation does not incorporate any material previously submitted for a Degree or Diploma in any other University of institute of higher learning and to the best of my knowledge and belief, it does not contain any material previously published or written by another person except where the acknowledgment is made in the text.

Also, I hereby grant to the University of Moratuwa the non-exclusive right to reproduce and distribute my thesis/dissertation, in whole or in part in print, electronic or another medium. I retain the right to use this content in whole or part in future works (such as articles or books).

…… ……..

M.R.M. Rilfi Date

The above candidate has carried out research for the Masters' Dissertation under my supervision.

…… ……..

Dr. H.M.N. Dilum Bandara Date

…… ……..

Dr. Surangika Ranathunga Date

Abstract

Social media enables personalization of the Consumer to Consumer (C2C) business model where people could directly do business with each other without an intermediary by sharing their products, services, and consumer requirements. However, messages shared by both the sellers and potential buyers do not reach each other as they are embedded among other social media messages. Moreover, C2C buy/sell interest matching in real time is nontrivial due to the complexities of interpreting social media messages, number of messages, and diversity of products and services. We present a platform for real-time matching of microblogging messages related to product selling or buying in C2C. We adopt a combination of techniques from natural language processing, complex event processing, and distributed systems. First, we extract the semantics of messages such as product attributes and commercial intention of the message either buying or selling using information extraction. Then the extracted buy/sell messages are matched using a complex event processor. Moreover, NoSQL and in-memory computing are used to enhance scalability and performance. The proposed solution shows a high accuracy where commercial intent classification and Conditional random fields based named entity recognition recorded an accuracy of 98.5% and 82.07%, respectively when applied to a real-world dataset. Information extraction, in-memory data manipulation, and complex event processing steps introduced low latency were latencies were 0.5 ms, 5 ms, and 0.2 ms, respectively. For the given setup with modest hardware, we were able to process 3,400 messages per second and overall latency was 0.76 ms.

Keywords: C2C; complex event processing; information extraction; named entity recognition; stream processing;

ACKNOWLEDGEMENTS

First and foremost, I express my sincere gratitude to my advisor Dr. Dilum Bandara. His continuous support, patience, guidance, and advice made the successful completion of this research possible. I am thankful to him for regular meetings and productive discussions despite his busy schedule. I highly appreciate his tolerance during times of my slow progress due to health problems or other issues. It has been a wonderful experience working with him, during which I have acquired and improved new knowledge and skills that will be useful to my career in the future. Also, I like to thank my co-supervisor Dr. Ms. Surangika Ranathunga for her valuable guidance and support.

I would like to thank members of my review committee Dr. Shehan Perera and Dr. Lochandaka Ranathunga for their helpful feedback during my progress reviews.

My sincere thanks go to the Computer Science Department of the University of Moratuwa for facilitating me with the necessary resources throughout the course of my research. I am thankful to Dr. Shehan Perera and Prof. Sanath Jayasena for granted me an Instructor position to me which helped me to carry out my research without any hindrance. Thanks, are to the System Engineers of the Department, especially Mr. Sujith Fernando, for helping me acquire, set up and manage hardware resources. I also thank the department staff in general for their friendly interaction, making my time at the department a fruitful and pleasant one.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ACK	Acknowledgement
B2B	Business to Business
B2C	Business to Consumer
C2C	Consumer to Consumer
CD	Cardinal number
CEP	Complex Event Processor
CRF	Conditional random fields
DAS	Data analytical server
DT	Determiner
FW	Foreign word
GPS	Global Positioning System
HMM	Hidden Markov Model
IE	Information Extraction
IP	Internet Protocol
JJ	Adjective
JSON	JavaScript Object Notation
JVM	JAVA Virtual Machine
MD	Modal
MEMM	Maximum-entropy Markov model
NER	Named Entity Recognition
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NN	Noun, singular or mass

NoSQL	Not Only SQL
OLTP	Online Transaction Processing
POS	Part of Speech
RDBMS	Relational database management system
RDD	Resilient Distributed Dataset
RDF	Resource Description Framework
SQL	Structured Query Language
UI	User Interface
VB	Verb, base form
WARC	Web Archive

CHAPTER 1

INTRODUCTION

This thesis presents a framework to identify potential buyers and sellers in Consumer to Consumer (C2C) e-commerce using real-time matching of social media messages based on product attributes and buy or sell intent. While addressing this research problem, we address several challenges such as information extraction, real-time processing, big data management, and matching of the semantic stream. We used the Conditional Random Fields approach to extract the product attributes, the Logistic Regression to classify the message based on commercial intent, Apache Storm for distributed real-time stream processing, NoSQL database to store the data, and Apache Spark for low latency query processing, and Complex Event Processor to match the buy-sell messages based on semantics of the C2C messages.

1.1 Background

Social media is an interactive computer-mediated technology that facilitates the sharing of ideas, career interests, information, and other forms of expression through building virtual networks and communities. Social media can be considered the most efficient communication medium in the twenty-first century [1]. Today, most people express their opinions freely through social media. In other words, social media has become an influential fact of every aspect of our human life. By design, social media is internet-based and offers users easy and rich digital communication. Users often utilize it for messaging.

Social media initiated as a tool that people used to interact with friends and family but was later adopted by businesses that wanted to take advantage of a popular new communication method to reach out to customers. It not only provides economic benefits to the sellers by increasing the number of buyers, but also opens a new door to the consumers to fulfill their product or service requirements easily and quickly, and cost effectively.

Thus, social media plays a major role in e-commerce. Promotions, recommendations, market-related sentiment analytics, product requirement sharing's, polls, opinion queries, reviews, and trend predictions are some of the ways big business entities interact with social media [1]. Those procedures commonly belong to big business organizations such as Business-to-Business (B2B) and Business-to-Consumer (B2C). In addition to these two business models, the C2C business model is emerging as a consumer-driven alternative [2]. The uniqueness of this model is that it does not have any intermediary between the seller and buyer.

While C2C could vastly benefit from social media as a free channel to communicate between consumer to consumer, conceptually, it can reach anyone in this globe and provide anytime service without any interruption in hidden social media services, handling huge data flow each second. However, the C2C consumers will not apply to advanced social media techniques, rather they just post their product/service requirements as social media messages with a set of simple words or images, it is difficult to interpret what they mean to buy or sell. Also, those messages do not reach a large audience and even the once that reach are buried among so many other messages.

1.2 Motivation

While social media is useful in C2C e-commerce, there is no guarantee that a message will reach potential users who are interested in those messages. Rather the messages will only reach the friends or followers of the message producer sooner or later. If the message can reach a wider audience who may have some association with that message both the buyer and seller could benefit. Moreover, messages do not reach followers in real-time, as followers may access messages based on their convenience and interest. Moreover, due to the asynchronous nature of social media platforms [2] and user behavior, sometimes it can take more than a day. This delay could lead to missed business opportunities to both the buyers and sellers. By nature, social media messages are written in natural language; hence, consists of unstructured text that is difficult to be interpreted by a computer program. Also, these messages arrive at a high velocity making it difficult to apply computationally expensive natural language processing techniques. Finally, the diversity of products and their information make it further

difficult to interpret the content and context of messages.

There are a few existing solutions that partially fulfill such as product recommendation systems, product attribute extraction and product databases. However, these solutions are not geared toward filtering e-commerce related messages social media messages, classifying those filtered messages based on their buy/sell intention, and all other product attributes. Moreover, these solutions do not focus on near real-time processing, big data management and matching sell/buy message semantics based on their product attribute and commercial intent. Consequently, current solutions do not fully address the potential for C2C e-commerce in social media.

1.3 Problem Statement

While C2C consumers can post their product/service related e-commerce messages in social media. The messages will only reach the followers of the consumer with a considerable delay. Most of the time the message will not reach the users who are interested in those messages. Therefore, while social media is a major platform to C2C, there is no effective way to connect potential consumers in real time based on their messages related to products and services. Hence, the problem to be solved by this study can be defined as:

How to develop an architecture/framework for real-time C2C matching, using consumers' text-based social media data?

1.4 Objectives

In this research, our main goal is to implement an architecture/framework that will deliver real-time C2C matching using customers' text-based social media data. Therefore, the above research problem is to be addressed by satisfying the following research objectives:

- To extract the semantics of social media messages and classify messages based on their buy and sell intent
- To develop an indexing structure such that new incoming messages can be matched with previous messages as close to real-time as possible
- To develop a framework to match C2C messages with high throughput and low

latency

- To evaluate the performance of the proposed framework using a real-life dataset.

1.5 Outline

The rest of this task is organized as follows: Chapter 2 explains the context of this research. This study focuses on extracting information from unstructured text, processing message streams, messaging in a distributed environment, and querying and analyzing large amounts of data. Chapters 3 and 4 present proposed approaches to match relevant social media messages in relation to C2C in real time. Chapter 3 explains how to extract product-related attributes from unstructured text, and Chapter 4 explains how to implement a framework for querying and analyzing large amounts of data and managing complex events. Chapter 5 describes the proposed approach using real social media messages, and Chapter 6 summarizes our work and suggestions for future work.

CHAPTER 2

LITERATURE REVIEW

Matching C2C users, i.e., connecting the C2C buyers and sellers that are interested in the same product. Context-based matching using a well-defined structured stream of social media is a complex task that is computationally expensive. Complex event processor is the novel approach to solve this kind of complex computation on structured streams. Another challenge is converting social media messages to be in a well-defined structured form. Such mapping requires information extraction. Moreover, we should consider the large volume of data and the need to process them in real time. This chapter presents related work associated with each of the techniques that we need to solve the research problem. Section 2.1 contains an overview of information extraction though Section 2.2 describes the nature of social media data. In Section 2.3, we present related research about stream processing. Big data persistence and high-frequency data manipulation from the NoSQL database relevant technologies are discussed in Section 2.4.

2.1 Information Extraction

Information extraction (IE) is the task of automatically generating structured information from unstructured and/or semi-structured documents [3]. IE is one of the important applications of Natural Language Processing (NLP). Most IE techniques are text driven, especially when applying on the unstructured/semi-structured text. Unstructured data (or unstructured information) refers to information that either does not have a pre-defined data model or is not organized in a pre-defined manner. Irregularities and ambiguities in the text make it difficult to understand using traditional programs when compared to data stored in fielded form in databases or annotated (semantically tagged) in documents [4]. The unstructured data is mostly generated by systems like social networks, blogs, comments and mobile data content like text messages, e-mail, etc., Therefore, the IE system takes unstructured text and finally produces structured/tabular data. The IE system is a pipeline in the process.

Some of the main important parts of this pipeline are tokenization, Part of Speech (POS) tagging, Named Entity Recognition (NER) and relation detection [3]. Tokenization is the process of taking the text or set of text and splitting them into its individual words at the same time removing certain characters such as punctuations. Then the POS tagging is the expansion of the part of the speech, which assigns parts of speech to each word such as noun, verb, and adjective. There are two main approaches in POS tagging: rule-based POS tagging and stochastic POS tagging. The rule-based method uses contextual information to assign a label to unknown or ambiguous words. Disambiguation is done using the linguistic features of the word, its previous word, its next word, and other aspects [5]. NER classifies named entities from text segments into pre-defined categories such as places, cities, dates, person and organization. The major NER techniques are Dictionary Look-Up, Rule-Based (using lexical, contextual and morphological information), Maximum Entropy Theory-based, Hidden Markov Model, Conditional Random Fields (CRF), and Hybrid methods (Statistical+ Linguistics) [6]. Among those techniques, most of them have solutions with good accuracy. But the NER research field has been growing during recent days and it has got the attention from both academe and business, because of its ability to solve real-world problems.

Structured data is the most preferred form of computational processing. But the social media messages are the most complex form of unstructured data. Sometimes, even humans may face difficulties in reading those social media messages, so it is much more complex for a computer system to understand those messages.

In our research problem, we focused on IE from text messages. There are many methods to convert unstructured text data into a structured form such as text mining approaches, text preprocessing, text representation, vector space model, text classification, text clustering, NER, relation extraction, semantic web, linked data and knowledge base. In our product attribute matching problem, we are focusing on the methods that can apply to the extraction of product attributes. As a requirement, we plan to extract five product attributes such as product name, brand of the product, model of that product, which product group the product belongs to and the selling status of that message. So, we assume the following approaches may help us to fulfill

our requirements such as text preprocessing, text classification, NER, semantic web, and knowledgebase/linked data.

2.1.1 Text Preprocessing

In any text-based processing task, there must be a preprocessing step required. There are a lot of methods that can apply on the text before the actual process such as punctuation removal, numbers, lowercasing, stemming, stop word removal, n-gram inclusion, infrequently used terms, document Indexing and string similarities. Stemming is the process of getting the most basic form of a word.

In any text-based processing task, there must be a preprocessing step required. There are a lot of methods that can apply on the text before the actual process such as punctuation removal, numbers, lowercasing, stemming, stop word removal, n-gram inclusion, infrequently used terms, document indexing, string similarities and removal of hyperlinks [3], [4]. If you look at the important steps, detailed stemming is a frequently used method in text preprocessing. Stemming can be defined as getting the base form of a word. The goal of stemming is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. For example:

am, are, is $\Rightarrow$ *be*

car, cars, car's, cars' $\Rightarrow$ *car*

According to the above transformation, a sentence and result after stemming can be given as:

the boy's cars are different colors $\Rightarrow$

the boy car be differ color

There are a lot of works done on stemming such as Lovins stemmer, Porter's algorithm [7] and Paice/Husk stemmer [8]. The n-gram a has a wide range of applications in text preprocessing, such as improving the interpretation of the multi-word contiguous sequence of tokens of length *n* [9] and improving the interpretability of bag-of-terms [3]. Usually and n-gram is applied with a degree of 1, 2 and 3. However, the problem is the explosion of vocabulary size. String similarities can also be considered as a text preprocessing method. String comparing is the main application of string similarities.

There are many algorithms that can do this by measuring the edit distance between two words usually used in a string comparison, for example, Damerau-Levenstein distance [10]. The string distance is defined as the minimal edits necessary to get one from another [11]. The edit distance can be generalized using a weighted edit distance using a biosequence algorithm named Needleman–Wunsch algorithm [12]. The next common string comparison method in use is Jaccard Distance which works at a token level, checking two strings by first tokenizing them and then dividing the number of tokens shared by the strings by the total number of tokens [13]. The next string comparison algorithm defined by the U.S. Census Bureau for comparing single person names named Jaro-Winkler Distance [14]. The final string similarity algorithm to list is TF/IDF Distance which is based on vector similarity. The basic idea is that two strings are more similar if they contain many of the same tokens with the same relative number of occurrences of each [9].

2.1.2 Named Entity Recognition

NER is a process where an algorithm takes a string of text (sentence or paragraph) as input and identifies relevant nouns (people, places, and organizations) that are mentioned in that string or categorized to a certain topic. NER is useful in various NLP applications such as information retrieval, question answering, and machine translation. When labeling the sequence of tokens two types of labeling methods that are followed widely are BIO and BILOU. Here B, I, and O represent Beginning or Inside or Outside accordingly. Two challenges we need to address in NER are recognition of named entity boundaries and recognition of named entity categories (classes) [15].

There are two main types of approaches followed in the NER process rule-based NER, statistical NER, and NER using deep learning. If we take the rule-based approach it can be either a set of named entity extraction rules, gazetteers for different types of named entity classes, and the extraction engine which applies the rules and the lexicons to the text. Likewise, if we take the statistical approach the following two things are always essential labeled training data and a statistical model. In this case, NER becomes a supervised learning model, and it classifies the sequence of tokens.

The Hidden Markov Model (HMM) is one of the well-known approaches for the sequence labeling problem. Here the states are representing the category or class which can generate the tokens. Also, the classes depend on a few earlier tokens. The trained HMM model is generated by calculating the probability of the above two sets of parameters of different state of classes and generations of tokens in training data. So, the trained model selects the class where it most likely maximizes the product of the above two parameters. State of the art results on the MUC-6 and MUC-7 data using an HMM-based tagger have reported in [16]. Here they used a wide variety of features, which suggests that the relatively poor performance of the taggers used in CoNLL-2002 was largely due to the feature sets used rather than the machine learning method. The limitation in HMM is considering tokens as being independent to each other [17]. A few other richer models exist that overcome the limitations of HMM such as Maximum Entropy [18], the Perceptron [19] and the CRF [20].

2.1.3 Conditional Random Fields

Conditional Random Fields (CRF) is a discriminative model used for predicting sequences. They use contextual information from previous labels, thus increasing the amount of information the model must make for a good prediction. In CRFs the training and test data are sequential, and we must take the previous tag into consideration when labeling a token. CRFs have all the advantages of the Maximum-Entropy Markov Models (MEMMs) but also solves the label bias problem. The main variation between CRFs and MEMMs is that MEMMs use per-state exponential models for the conditional probabilities of the next state given the current state, while a CRF has a single exponential model for the joint probability of the entire sequence of labels given the observation sequence. Therefore, the weights of different features at different states can be traded off against each other [20]. If we consider the difference between the CRF and the Hidden Markov approaches both are applied to label sequential data, even though both differ in many ways. For instance, the Hidden Markov Models are generative and produce the end result by using the joint probability distribution. Instead, the CRF is discriminative and also uses the conditional probability distribution. CRFs do not depend on the independence assumption and ignore label bias.

The CRF model defines dependencies among adjacent tags, based on the assumption that these dependencies are influential [8]. This assumption neglects a few more significant dependence facts, and it confirms that these facts can enhance the efficiency of the CRF approach. Some specialty ontology features are included in CRF, and their result proves that it can increase the performance [7]. In CRF, a result of the transformed Viterbi approach joint with some other rules are used to get the N-Best outcome [6]. Then, some rules are applied to separate the N-Best effects, which might still cause poor results. It proves that it can get a good outcome.

The advantages of the CRF approach is more suitable for PRO NER than other approaches. But, the CRF approach needs some more substantial dependence information in some conditions.

2.2 Social Media Data

Social media messages exhibit the four Vs of big data. First, the *velocity* of a social media messages can be high as thousands of messages are generated by hundreds of millions of users [21]. Hence, the proposed solution should be able to process those messages with the throughput identical to the velocity of the incoming social media messages. In Section 2.3 we discuss how social media messages with high velocity can be handled.

Second, due to the high velocity, social media messages also generate high *volume* of message with time. In a matter of hours to days social media messages could generate hundreds of terabytes of data. Such large volumes not only cause storage issues but also lead to several other challenges such as the need for fast disk writes and low latency searching/querying while maintaining core features of a storage system such as availability, consistency, and partition tolerance.

Another characteristic of the social media message is the *variety* of content. Users of social media generate a variety of content that are mostly unstructured [22], dynamic, and unpredictable. Moreover, messages include different type of content such as text, emoticons, links, videos, and images. Therefore, processing social media messages is not that easy unlike processing a predefined stream with a single type of content.

Veracity is the last challenging nature of social media. The content of social media

apart from unstructured shows some nature of uncertainty. Also, there is a chance that messages are interpreted with different meanings by different people. The content of social media is too small, so it contains a lack of information, which makes it hard to using only social media.

2.3 Stream Processing

From audio stream to Global Positioning System (GPS) Stream and sensor stream to social media stream, all these streams are not only just for consumption, visualization and filling the storage, but we can also utilize those streams to get more benefits in real-time as well as in the future. Stream processing is one of the trending technologies in the era of big data. You can see the usage of stream processing from stock exchange predictions to weather forecasting and disaster management.

Since stream processing is a part of the big data ecosystem, it must be capable of handling the stream data flow in big data dimensions, especially high velocity (in real-time stream processing) and a large volume of data (in batch processing) [4]. There are several other issues like guarantying the data consistency, availability (failover mechanism), and complexity.

There are several use cases with different requirements in stream processing. In some scenarios, the stream should be processed at very high speed and the result should be produced in real-time. In some other cases, a little delay may be acceptable. In the worst-case delay, like one hour or one day, it is suited for some large data processing conditions. According to the time taken for the processing of the stream, we divide the stream processing into two; real-time stream processing and/or near real-time stream processing, and batch processing. Stream processing has several aspects such as real-time processing, distributed stream processing, decentralized stream processing, guaranteed stream processing, publisher-subscriber mechanism, fault-tolerance, and scalable. If you take the real-time processing dimension in stream processing, this real-time term is based on the time, latency and a few other factors such as data motion, and data access. We can divide the big data process into two types; batch processing and real-time processing. Table 2.1 compares the differences between batch processing and real-time processing. Even if we use the term real-time processing, it is near real-time if any system can process a single element of the data in less than 1 millisecond

can consider as near real-time processing. Next, we cannot avoid the third option, which is a hybrid solution of real-time and batch processing. An example of such a processing model is the Lambda Architecture that we discuss in detail in the following section, Section 2.4 big data storage and processing. If we take the real-time processing, there are a few requirements to fulfill; conditions such as keep the data moving, having a stream-based SQL, handle the stream.

Table 2.1 Difference between batch processing and real-time stream processing.

Batch Processing	Stream Processing
Data is at rest	Data is in motion
Batch size is bounded	Data is essentially coming in as a stream and is unbounded
Access to entire data	Access to data in the current transaction/sliding window
Data processed in batches	Processing is done at event, window, or at the most at micro batch level
Efficient, easier administration	Real-time insights, but systems are fragile as compared to batch

Imperfections (Delayed, Missing and Out-of-Order Data), Generate Predictable Outcomes, Integrate Stored and Streaming Data, Guarantee Data Safety and Availability, Partition and Scale Applications Automatically and Process and Respond Instantaneously [23]. If we focus on a few important factors mentioned above. One of the important requirements is to make sure that each message is processed. In this aspect, the process has three types in the real-time process according to the requirements; at most once, at least once and exactly once. Table 2.2 lists the characteristics of each processing type.

Table 2.2 Three types of guaranteed processing in real-time processing.

At Most Once	At Least Once	Exactly Once
Subscribes to data from the start of the next window	Operator brought back to its latest checkpointed state and the upstream buffer server replays all subsequent windows	Operator brought back to its latest checkpointed state and the upstream buffer server replays all subsequent windows
Ignore the lost windows and continues to process incoming data normally	Lost windows are recomputed & application catches up live incoming data	Lost windows are recomputed in a logical way to have the effect as if the computation has been done exactly once
No duplicates & no re-computation	Likely duplicates & re-computation	No duplicates & re-computation
Possible missing data	No lost data	No lost data

Figure 2.1 shows that Spout A emits a tuple T(A), which is processed by bolt B, and bolt C, which emits the tuples T(AB) and T(AC), respectively. So, when all the tuples produced due to tuple T(A) – namely the tuple tree T(A), T(AB), and T(AC) – are processed, we say that the tuple has been processed completely. If the message reaches the destination task, it is considered as a completed message, otherwise, it is a failed message. To implement guaranteed message processing, one of the recognized ways is sending the acknowledgment to the root emitter. Figure 2.2 shows each task of sending an acknowledgment to the spout if the message is successfully processed. If the task is not sent the acknowledgment to the spout that message considered as a failed message. So according to the guaranteed messages processing optimization either at most one, at least one and exactly once, these failed messages may process again until it processed successfully.

The next important feature in a real-time stream processing system is stream grouping. Currently available stream grouping types are Shuffle grouping - which randomly partitions the tuples, Fields grouping - which hashes on a subset of the tuple attributes/fields, all grouping - which replicates the entire stream to all the consumer tasks and Global grouping - which sends the entire stream to a single bolt [24]. Figure 2.3 illustrates the distribution of messages among tasks in each stream grouping type.

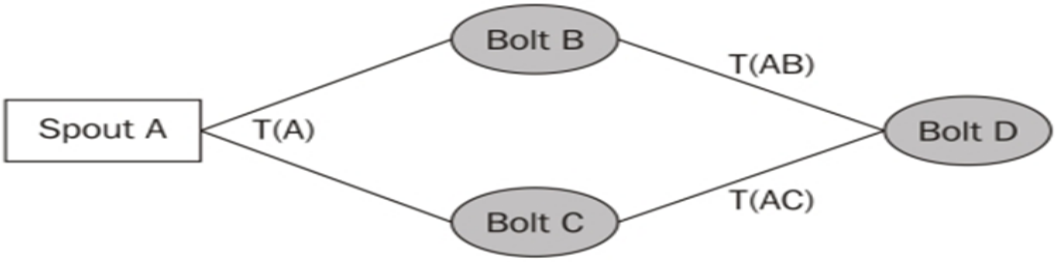


Figure 2.1 Typical stream processing with multiple sequence of task.

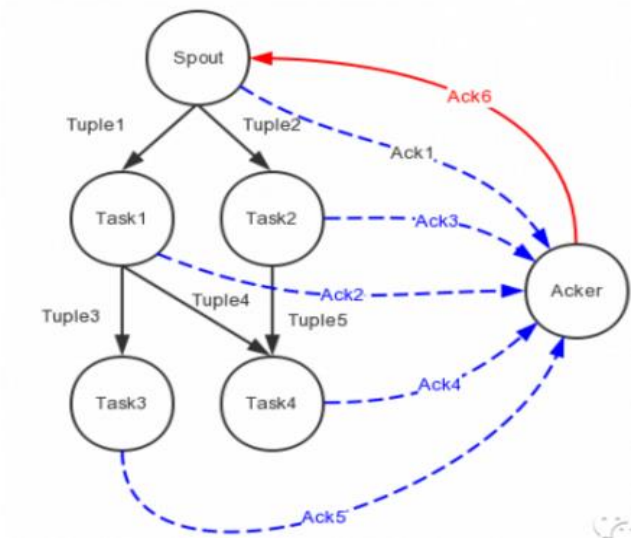


Figure 2.2 Acknowledgement mechanism in Storm.

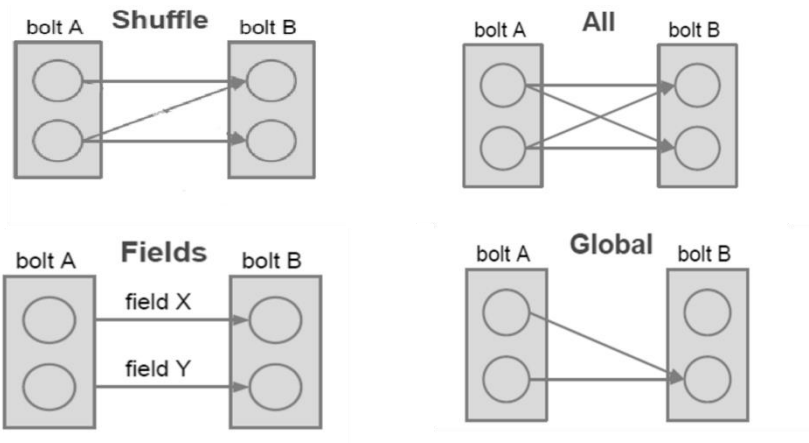


Figure 2.3 Four types of stream grouping.

2.4 Big data persistence and high frequent data manipulation

In this section, we analyze a few main challenges in dealing with big data, specifically big data manipulation. The problem is that we are storing the big data in a NoSQL database, which contains hundreds of millions of data entries distributed among multiple nodes, but the requirement is to be able to query the data from NoSQL with low latency and high frequent queries. Here, we going to discuss four solutions for this problem, namely NoSQL native solution, Lambda architecture, Kappa architecture, and in-memory computing.

Several NoSQL database products currently in use, and among them Cassandra, it performs well compared to other products [25]. If we analyze the arrangements in the Cassandra NoSQL database for big data, when writing, initially the new data goes to the commit log, then the data is stored in the memory for a while. *Memtable* is a memory caching data structure where if the memtable reaches the upper bounty, the data is permanently flushed into the stable (Sorted Strings Table), which is in the disk.

During the read operation or data manipulation, Cassandra consults a bloom filter that checks the probability of a table having the needed data. If the probability is good, Cassandra checks a memory cache that contains row keys and either finds the needed key in the cache and fetches the compressed data on disk or locates the needed key and data on disk and then returns the required result set.

Next, we discuss the lambda architecture [26]. Here, rather than directly reading from the NoSQL table, a pre-computation process takes place and creates batch views. As illustrated in Figure 2.4 the data comes to the system and splits into two layers, the batch layer and the speed layer, for processing. Managing the master immutable dataset and pre-computing the data views are the two main functions of the batch layer. In the middle, the serving layer generates the indexes from the batch view to enable low-latency data manipulation. Any data query can be obtained by combining both results from real-time data and batch data.

As illustrated in Figure 2.4, Kappa Architecture is derived from Lambda Architecture. There is no batch layer in Kappa architecture. Instead of the batch layer, it is cloned into two real-time streams.

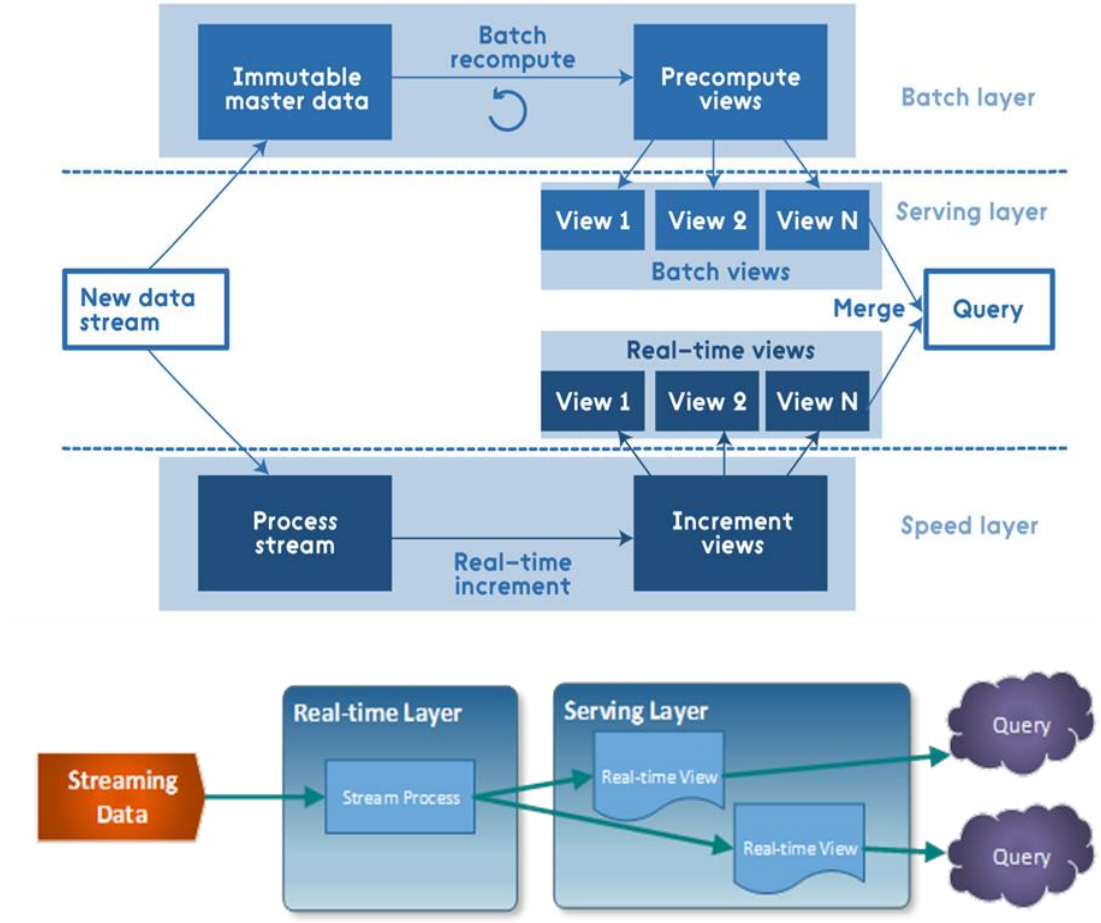


Figure 2.4 Lambda architecture and Kappa architecture

Source: Adapted from [25].

2.5 Existing Solutions for C2C matching

We are unable to find an existing solution that matches real-time C2C matching the research domain, but there is some part of our problem that has been previously tried by a few people. First, if we consider the product attribute extraction section, people have approached this matter in two different ways, Rule-Based methods NER PRO is mainly run using some manually created or automatically generated rules. Pierre [27] has developed an English NER system that allows product names to be found in product reviews. Like string template matching, a simple Boolean product name classifier was used. However, in this way, many new product names are not included in corpus training, and results may be worse in certain situations.

Statistical model-based approaches usually apply statistical models integrated with

some heuristics and external knowledge bases. For example, a bootstrap method using two consecutive learners (syntax-based decision list and hidden Markov model) is presented in English NER [25]. The main advantage of this method is that it avoids manual annotation of the practical training corpus, but it has two problems. The other is that it relies heavily on parser performance. In [26], another very similar approach was used. There, all O & M ontology XML tags have been mapped to the OWL concepts. Although the author describes access to annotated data on SPARQL queries, SPARQL queries generate large amounts of traffic at low sample rates, so for applications that need to access sensor data in real time, it is not effective. Therefore, there remains a need for solutions for real-time semantic annotation as well as for the efficient representation of sensor data knowledge in dynamic environments such as smart cities.

2.6 Summary

In this chapter, we discussed the abstract architecture and implementation, which includes our main components such as IE, Bigdata, and social media, stream processing, bigdata persistence, and high-frequency data manipulation from a NoSQL database. Furthermore, in IE, we discussed the main approaches we used, such as text preprocessing and social media message normalization. Then we moved into NER, here we presented the algorithms we used such as CRF and Logistic Regression. In Section 2.2, we analyze the nature of social media data, particularly the bigdata feature applied to social media messages. In the next section, we moved into the performance problem. First, we drill down the available technologies in stream processing, particularly the types of stream processing are batch processing and real-time processing. Then we discussed guaranteed stream processing and parallelism in distributed real-time stream processing. Finally, we discussed bigdata persistence and high-frequency data manipulation. Here, we discussed available technologies for managing huge data, especially a fully decentralized distributed NoSQL that has high availability and low latency. Then we finished with low latency data manipulation in a very high-frequency stream environment. We found the following technologies such as batch processing, micro-batch processing, material view, and Lambda Architecture.

CHAPTER 3

RESEARCH METHODOLOGY

The objective of the research is to find the matching social media messages related to the C2C business model. Figure 3.1 illustrates the three main aspects involved in this problem, namely Information extraction (IE) from row social media messages, matching the messages based on the semantics, and process above two in real-time.

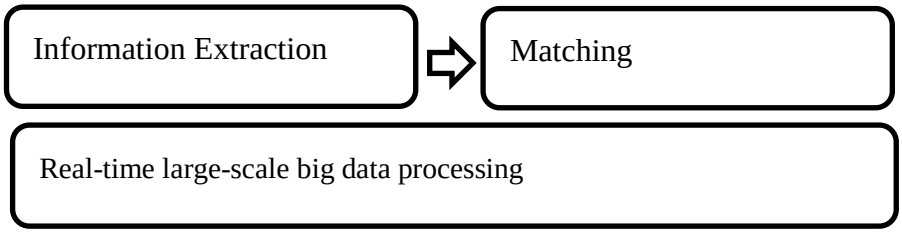


Figure 3.1 Association between three aspects of our research problem.

Table 3.1 lists the three main aspects and technologies. Section 3.1 presents the methodology of IE. The matching methodology is presented in Section 3.2. In the next section, we talked about the approach we applied in the implementation of real-time big data processing. Finally, in Section 3.4 we presented the high-level architecture of our research.

Table 3.1 Three aspects and relevant technologies.

Information Extraction	**Matching**	**Real-time Big Data Processing**
Gazetteers	Complex event processing	Stream processing
Machine learning	Business analytics	Distributed computing
Linked data		NoSQL database
Conditional random fields		In-memory computing
Logistic regression		Search & indexing
Named entity recognition		

3.1 Information Extraction

As the social media messages are in unstructured form, it is difficult to compare messages with each other and find the matching set of messages. There are several reasons for this difficulty. One of the differences between the C2C social media

messages and the product listing on general e-commerce sites like eBay or Ali Express is the organized nature of the content. While most e-commerce product content is in semi-structured form, social media messages are in unstructured plain text. Moreover, C2C-related social media messages not only contain the product details as the user can add irrelevant content to the message, e.g., but the messages may also contain greetings with product attributes such as the example given in

Figure 3.2. Number 1 highlights the product attribute and the rest of the text is irrelevant to the product. At the same time, the message contains non-dictionary words as well (highlighted as number 2). The third reason is the inaccuracy in messages such as typos or confusion. For example, the correct wording of the product should be *"Sony DSC -H400"*. At the same time, the second message mostly contains the product attribute in its text. If we compare both the messages using Levenshtein distance [28], there is a large variation between the two messages. However, producing the semantics or structured form of a message from the plain unstructured message is not straight forward. IE needs to produce structured messages; in our case, as we use statistical IE.

3.2 Matching

As seen in Figure 3.3 to be able to match messages, we need to:

1. Work with a stream-based data flow
2. Detect matches between real-time messages
3. Detect matches between messages in near real-time
4. Match both real-time live messages and offline stored messages
5. Fulfill the availability and horizontal scaling needs
6. Match both complete matching and partial messages

To fulfill all the above requirements, we came up with a combination of multiple emerging technologies. For example, complex event processing is used to process a structured stream input data flow to find matches among real-time data and near real-time data stream using time windows. To perform a high-frequency search with low latency over a large database, we implemented a combination of big data technologies like NoSQL database, data indexing, in-memory computing, and map-reducing. In the upcoming sections, we discussed those methodologies in detail.

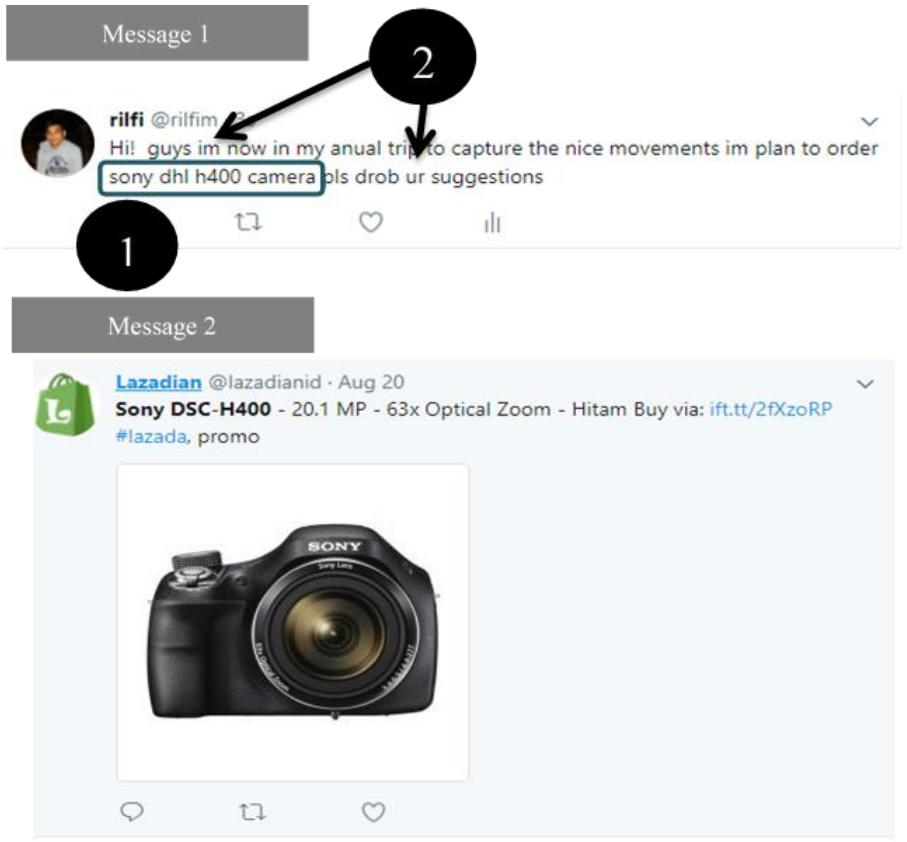


Figure 3.2 Two sample tweets with a mix of product attributes and greeting.

3.3 Real-time big data processing

IE and matching phases can handle a single message at a time and have relatively high latency. Therefore, under the high input rate experienced in social media, messages will get queued increasing the overall response time. The process-level distribution could be used to solve this issue. To make it work in an optimized environment, we split the above two main processing units such as IE and matching into several small building blocks. Then, according to the workload/complexity and the latency of each component will be assigned with a parallelism index in our distributed stream processing environment.

3.4 High-level architecture

Figure 3.3 illustrates the data flow between each building block of our system. As illustrated in Figure 3.4, the IE phase is used to convert the unstructured social media messages into a structured form which is taking place in the Storm cluster. The

matching component is responsible for finding the matching messages among the structured message stream produced by information extraction part. Matching process happening inside the WSO2 DAS. To process this social media messages in real-time, both IE of the messages and matching the messages must work with low latency in any complex use cases. With a standalone application we cannot achieve this.

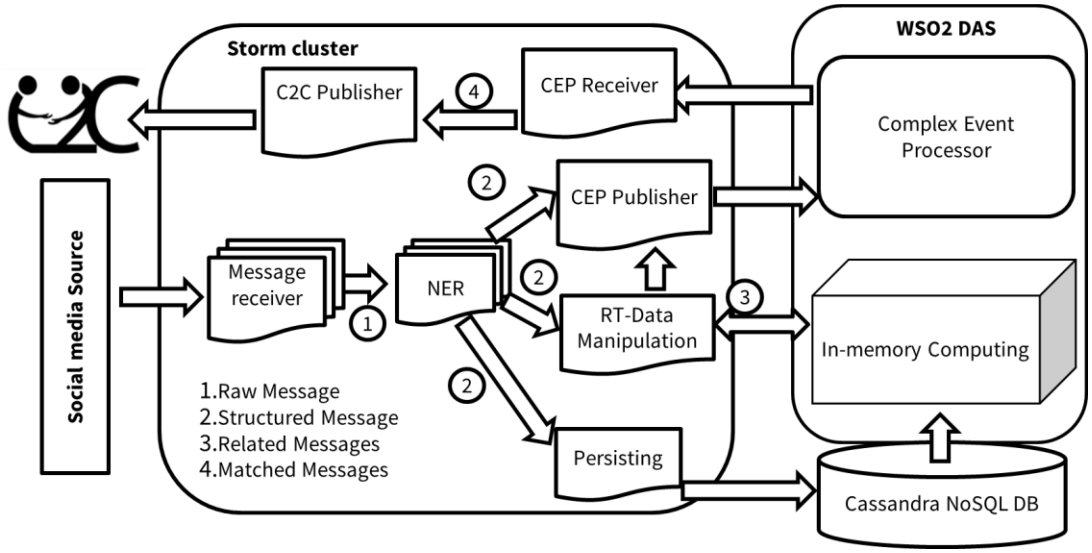


Figure 3.3 High level architecture of our framework.

The real-time implementation facilitates three internal requirements, namely processing social media message stream in near real-time, storing the huge volume of semantics generated by stream processing, and real-time query processing. Mostly the IE and some other supportive process were executing on top of the distributed parallel stream processing.

So, with the help of distributed stream processing, each subtask executes multiple instances parallelly. The list of sub-tasks and the purpose is given in Table 3.2. According to the requirement, the parallelism of a task can be adjustable, e.g., five message receiver instances, ten NER instances, and two persisting instances. Because of this parallel execution feature provided by the distributed stream processing implementation, the incoming social media messages can be processed in near real-time with low latency.

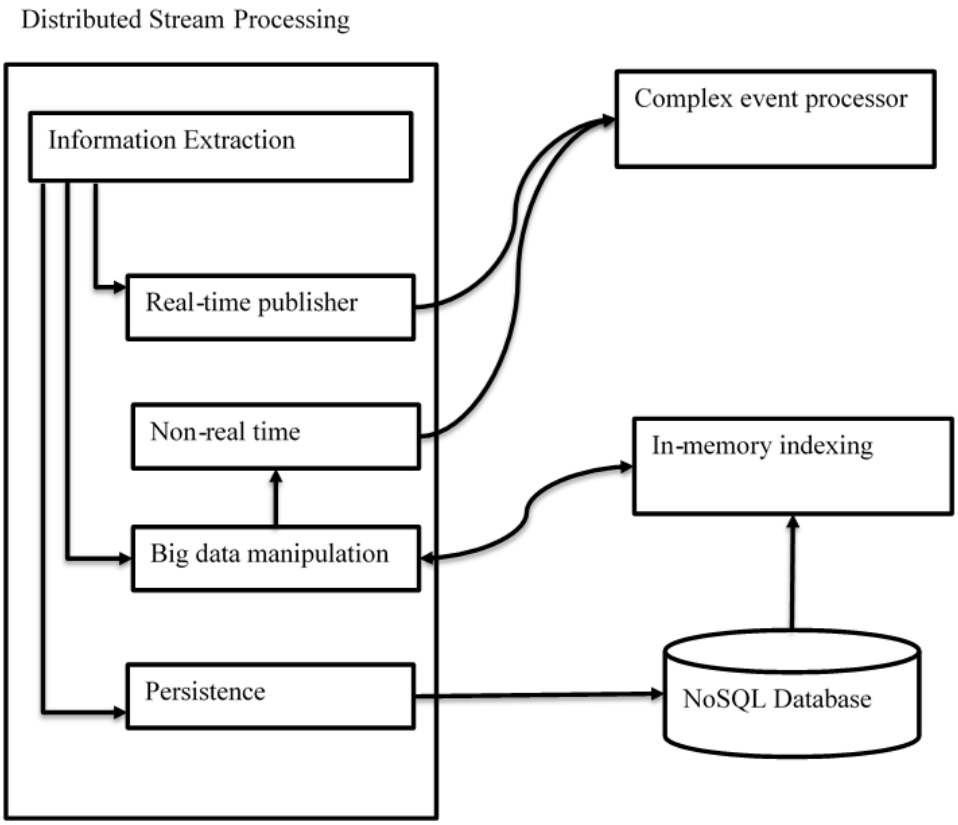


Figure 3.4 The data flow between IE, In-memory, NoSQL and CEP.

The raw/plain text of social media messages is collected by a message receiver. Then, the messages will reach the IE task, named as Named Entity Recognition (NER). The IE part is split into several subtasks to extract various product attributes such as product name, model name, product group, brand name, and commercial intention of that message. Next, the semantics stream of the messages is replicated into three streams and NoSQL database, in-memory computing, and complex event processor. The NoSQL database receive and store messages for future use. In-memory computing system converting the real-time message stream into a query which manipulate the related messages from messages stored in the NoSQL database. The Complex Event Processor (CEP) takes the real-time message stream as one input stream and related stream produced by the in-memory system as the second input and match the messages through based on a predefined set of rules. To connect to each of the above units, we implemented another three-stream data transferring connector, namely persisting bolt, real-time data manipulation bolt and publishing bolt to connect with the CEP.

Table 3.2 Components implemented using distributed stream processing.

Task	Purpose
Message receiver	Retrieving social media messages
NER	Converting the unstructured social media messages into structured form/ semantics through named entity recognition.
Persisting	Persisting the extracted semantics to the NoSQL database
Real-time data manipulation	Search and retrieve the resulting from the NoSQL database according to the query parameters taken from incoming semantics by connecting to the in-memory computing component
CEP publisher	Publishing the semantics stream to the complex event processor
CEP receiver	Receiving the matched messages from complex even processor and sending those matchings to the C2C publisher
C2C publisher	This the final task connecting the consumers through notifying social media users about the messages relevant to the message published.

So, the same data stream is used for different purposes by different units to find out whether the extracted semantics from real-time match the other messages in the real-time stream and offline data. Matching between real-time and offline data is more challenge than matching among real-time data. Such matching is challenging because of high velocity and volume of social media stream. while storing all the messages in NoSQL database, we simultaneity convert each real-time message to act as a query on already stored buy/sell messages on NoSQL database. Therefore, the velocity of the live social media stream determines the frequency of the queries. However, such high frequency queries are difficult to handle with most databases. Moreover, all these queries may need to scan all the data stored in the database linearly increasing the query time with respected to the number of messages already stored in the database.

In our solution, we use a NoSQL database to handle four Vs of big data [29] while distributing both the storage and workload across the multi-node distributed cluster. Through the persisting bolt first replicated stream will reach the NoSQL database. Each message in the real-time stream will make a query on the NoSQL database to find potentially matching more complicated entries. To handle these high-velocity queries, we use in-memory computing, materialized views and indexing. This increases the throughput of query processing while reducing the latency. The real-time data manipulation bolt is responsible for converting the semantics into the CEP publisher bolt.

Finally, the third replication of real-time semantics occurs and the results from the query processing are sent to the CEP through the CEP publisher. The CEP receives both the real-time semantics and the results of the queries collected from the NoSQL database. The CEP has several execution plans to process those messages and will finally produce the matches from both streams. In the end, the matching details will reach the right users through the CEP receiver and the C2C publisher [30].

CHAPTER 4

IMPLEMENTATION

In this chapter, we explain the detailed? implementation of our solution. Our solution contains four main technical aspects IE, real-time stream processing, big data processing and matching the semantics of the social media messages. First, we start with IE on Section 4.1 and later in Section 4.2 we explain the real-time and big data processing mechanisms. Also, in Table 4.1 listing the major aspects applied in our implementation, relevant tools, and their applications.

Table 4.1 Applied tools and their functionality

Tools	Aspect	Application
Nltk	Information Extraction	Python Natural Language Toolkit
LingPipe	Information Extraction	Tool kit for processing text using computational linguistics
Fuseki	Information Extraction	SPARQL server for manipulating Linked data in triple RDF format
Gate	Information Extraction	A general architecture for text engineering
Spark	Information Extraction/real-time big data processing	Fast and general engine for big data processing, with built-in modules for streaming, SQL, machine learning and graph processing
WSO2 Data analytical server	Matching	An analytics platform that analyzes data streams in real time
WSO2 Complex event processor	Matching	Helps identify the most meaningful events and patterns from multiple data sources, analyze their impacts, and act on them in real time
Apache Storm	Real-time big data processing	Distributed real-time computation system
Apache Zookeeper	Real-time big data processing	Distributed, open-source coordination service for distributed applications
Apache Kafka	Real-time big data processing	Real-time data pipelines and streaming apps
Apache Cassandra	Real-time big data processing	Distributed NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure

4.1 Information extraction

In our case, we have two main requirements related to IE the NER of product attributes and classification of the messages. Both requirements needed training dataset. The detailed set of the process is illustrated in Figure 4.1. The accuracy of a supervised machine learning model increases with the size of the training dataset. So, a very large training data set is necessary to get good accuracy.

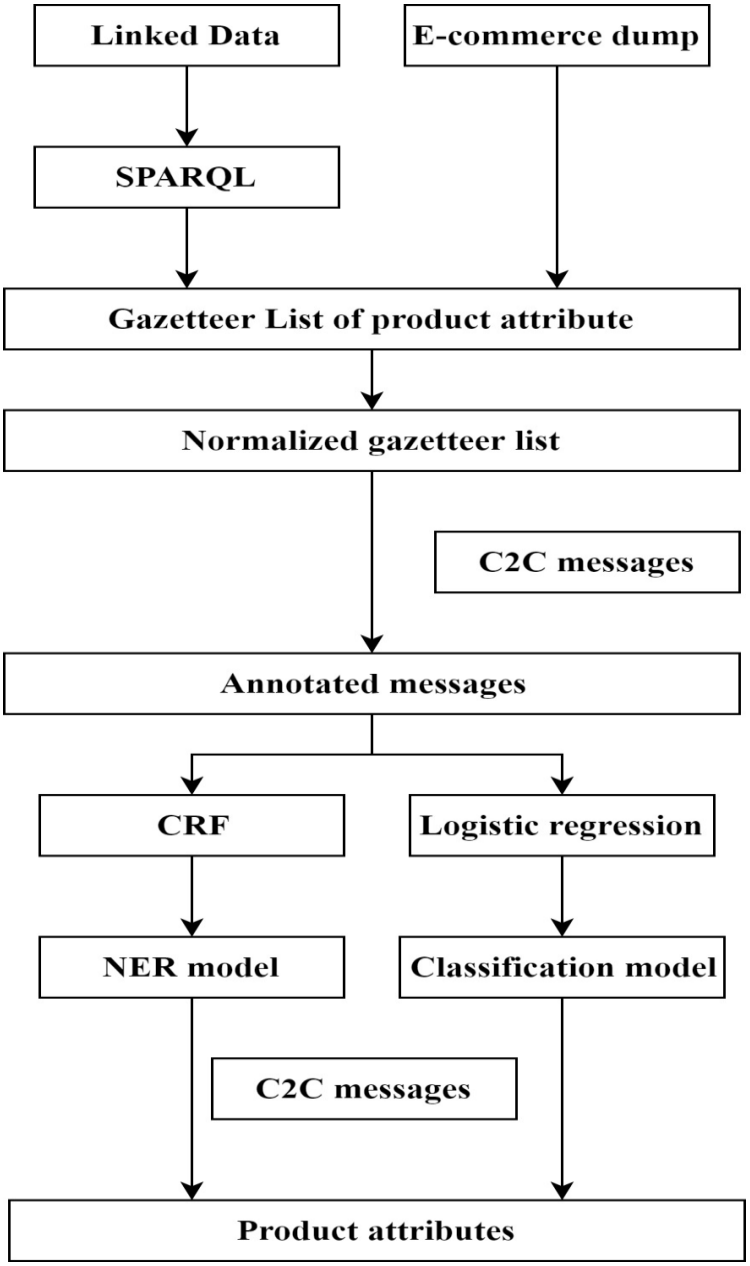


Figure 4.1 Information extraction process

As we were unable to find any proper training dataset with product attribute labeling, we generated a large size of the training dataset. Manually labeling each message will take a long time. Therefore, we came up with an automated training idea. In our case, we needed to recognize product attributes such as product name, brand of the product, model of the product and several other functional/performance attributes. As shown in Figure 4.2, to create a label for each attribute we needed the global list of the whole values of that attribute e.g., only if we have the whole list of brands, are we able to label the brand in our training set. So, we needed to collect a large list of terms for each product attribute.

4.1.1 Gazetteer list Generation from Linked Data

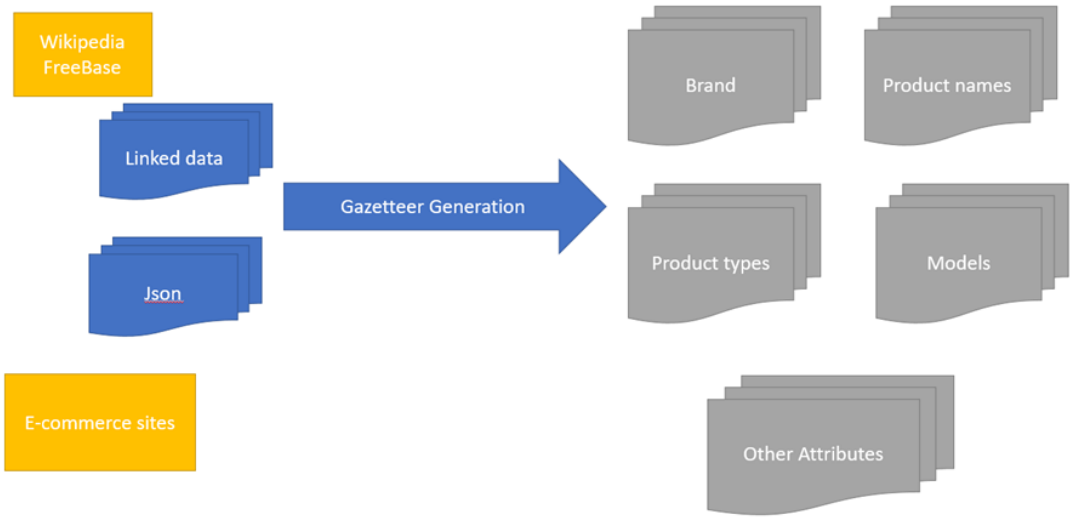


Figure 4.2 Data sources of gazetteer lists belong to product attributes.

Basically, the gazetteer list is a list of text phrases, which is one approach in NER to identify a named entity. But there is no gazetteer list publicly available for identifying product attributes. So, we planned to prepare a set of gazetteer lists for product attributes such as brand, product, and model. So, we identify certain datasets relevant to the product details that are in a semi-structured format such as product details on Wikipedia, Product listings on e-commerce sites, review records of product listings.

Our first data source is Wikipedia data, which is available in different forms such as ontologies, Linked Data, RDF, semi-structured data, XML, n-quads, and n-triples.

Table 4.2 Sources of Product Linked data and statistics.

Domain	num_pages	num_pages_with_triple	num_triples
searsoutlet.com	341924	341779	4953189
overstock.com	347846	307498	6301109
macmall.com	401944	150677	1659587
apple.com	391539	299242	40755099
abt.com	115539	54616	218580
microcenter.com	459921	13364	216767
target.com	2007121	582093	6689999
alibaba.com	7136	3873	46653
shop.lenovo.com	41465	10681	555219
walmart.com	1711517	1683145	15163284
shop.com	1754368	1752746	26619496
sears.com	659	0	0
newegg.com	37393	19904	169466
frontierpc.com	187184	187059	28738447
techspot.com	386273	365260	17152324
pcrush.com	292904	0	0
microsoftstore.com	24163	12239	476781
dhgate.com	50099	22896	461643
tesco.com	222802	22476	636409
selection.alibaba.com	607410	583711	9651092
flipkart.com	63112	7571	57401
boostmobile.com	2487	105	263
bestbuy.com	389146	107763	11140272
techforless.com	361234	280569	1158915
tomtop.com	13306	6797	850590
samsclub.com	22788	8034	146019
membershipwireless.com	457	0	0
costco.com	54274	30837	128469
bjs.com	40930	26445	270573
ebay.com	413144	357165	9037049

Here, we focus on product linked data. This linked data is a graph net between product related entities. These datasets are also called triple data, which are subject, object and predicate. As illustrated in Figure 4.3, each attribute of the product contains three parts such as subject, object, and predicate. Product data is a collection of multiple sources which are listed in Table 4.2.

<http://www.w3.org/1999/02/22-rdf-syntax-ns#type><http://schema.org/Product><http://deals.ebay.com/> .
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type><http://schema.org/Offer><http://deals.ebay.com/> .
<http://schema.org/Offer/priceCurrency>"USD" <http://deals.ebay.com/> .
<http://schema.org/Offer/price>"$59.99" <http://deals.ebay.com/> .
<http://schema.org/Offer/availability>"Null" <http://deals.ebay.com/> .
<http://schema.org/Product/name>"Acer Iconia One 7\" Tablet - 8GB Memory - Android OS" <http://deals.ebay.com/> .
<http://schema.org/Product/url><http://www.ebay.com/itm/Acer-Iconia><http://deals.ebay.com/>

Figure 4.3 Single product entry in triple format.

Here the statistics of crawled data, such as number of pages, number of pages with a triple, and triple were given. This data set contains 6 million pages from the multiple sources mentioned, and in size will be over 350 GB of compressed data in WARC format. Here in this dataset, we identified 250 different product-related semantic attributes. Few semantic entries are listed in Table 4.3. But in our research, we only focused on a few semantic fields of products, such as product name, product group, product brand, product model and some others.

Table 4.3 Few semantic entries from our product linked dataset.

condition	country of origin - assembly	load rating
contact pressure	country of origin - components	magnet
sound pressure level (spl)	country/region of manufacture	manufacturer part number
3.5mm_jack	detachable cable	manufacturer's part number
accessories furnished	driver diameter	max. input power
accessories included	ean	max. nominal continuous input power
adapter lead	ear coupling	max_input_power
assembled product weight	earpiece	maximum frequency
attenuation	earpiece design	maximum input power
audio frequency	features	maximum_input_power
audio interfacetype	fit design	minimum frequency
battery specification	frequency bandwidth	model
bluetooth version	frequency characteristic	model number
brand	frequency range	mpn
bundled items	frequency response	net weight
cable	frequency response (headphones)	noise canceling
cable length	frequency response (microphone)	noisegard noise compensation
cable type	frequency response range	nominal impedance
cables_included	frequency_response	operating time
characteristic sound pressure	function	package
charging time	gender	part number
codecs	headphone type	pick-up pattern

So, to store the data we used an RDF (Resource Description Framework) [31] based server called Apache Jena Fuseki [32]. Apache Jena Fuseki is also a platform to run

the manipulation queries on top of linked data. This query language is commonly known as SPARQL i.e., a semantic query language for databases [33]. In our case also we need to manipulate multiple product attributes separately from this linked data server. Table 4.4 shows the list of semantic URI references belonging to product attributes, which is the result of the following SPARQL query.

So, the following queries were used to generate the pre-normalized version of the gazetteer list.

Product name manipulation query

```
SELECT DISTINCT ?productName
WHERE
 { GRAPH ?g {
    ?s <http://schema.org/Product/name> productName
            }
  }
```

Product model manipulation query

```
SELECT DISTINCT ?productmodel
WHERE
 { GRAPH ?g {
    ?s <http://schema.org/Product/model> ?productmodel
       }
 }
```

Product brand manipulation query

```
SELECT DISTINCT ?productBrand
WHERE
 { GRAPH ?g{
    ?s <http://schema.org/Brand/name> ?productBrand
        }
 }
```

From the RDF data store, we are able to generate a certain large size of the gazetteer list of the various product attribute.

Also, we found semi-structured product data set from some other resources which are in json form. Here mainly we collected from two sources metadata of Amazon product listings [34] and gold standard Product Feature Extraction from webdatacommon [35].

Table 4.4 Semantic references belonging to product attributes.

http://schema.org/Product/offers
http://schema.org/Product/name
http://schema.org/Product/url
http://schema.org/Product/gtin13
http://schema.org/Product/review
http://schema.org/Product/mpn
http://schema.org/Product/aggregateRating
http://schema.org/Product/brand
http://schema.org/Product/color
http://schema.org/Product/model
http://schema.org/Product/description

Table 4.5 Dataset summery for gazetteer generation

Dataset no	**Source**	**Type**	**Obtained Product Attributes**
Dataset1	Linked data	N-Quarts(RDF)	Product_name,model and brand
Dataset2	Amazon	json	Product_name, category, brand
Dataset3	Web data common	json	Product_name, brand, product_type, manufacturer and phone_type

This dataset includes product reviews, rating, and metadata from Amazon, including 142.8 million reviews spanning May 1996 - July 2014 [36]. This dataset includes reviews (ratings, text, helpfulness votes), product metadata (descriptions, category information, price, brand, and image features), and links (also viewed/also bought graphs). In this dataset the brand and product category ware very useful to our experiment.

Here in this dataset, the records ware included from various e-commerce sites. The labeled set contains out of 500 product entities, while the distinct labeled properties are 338 in total. The product entities were labeled as JSON objects [37]. Here the dataset includes many product attributes for each product. It may vary from product to

product. The headphones contain 36 attributes, phones contain 32 attributes and tv contains 76 attributes.

So, from these three datasets, listed in Table 4.5 we were able to collect noisy product attributes of product name, product model and product brand. From the above process, we generated a gazetteer list for the above product attributes, which helped us in labeling the training dataset to produce the machine learning model to NER. At the same time, product category and sale status ware two classification problems. We needed to classify each message to a suitable product group, as well as classify the message to the right sale status among buy, sell, and neither. Our second dataset was used to generate the classification of the product category. But all this data is very noisy, so we implemented certain normalization techniques, which are described in the following section.

4.1.2 Product domain Named Entity recognition

For NER we needed a labeled dataset but, in this case, the domain is very large. So manually labeling a small number of messages will not give an accurate result. Using gazetteers, we managed to label a significant number of messages. In the above step, we directly produced three gazetteer lists belonging to the product name, product model, and product brand. Even though those gazetteer lists were very noisy, we came up with a normalized version of the gazetteer list through several steps, which are mentioned below.

Normalizing the gazetteer list

For each product attribute, we got a very large number of noisy lists. Those lists had a lot of problems, such as duplicated entries, case sensitive, containing prefixes, containing suffixes, containing symbols, containing numbers, very long words and a large number of words [38]. First, we filtered the entries that were less than four words. Then, we removed the entries containing stop words, using the python NLTK library.

Also, if any entry had very long individual words, they were removed from the list. To reduce the noise, we filtered the entries with numbers and symbols. Later, we generated separate files according to the number of words, so we had 4 files for single word entries e.g., maco, mage, malo, maps, tomato, maxx, and meco, two-word

entries, e.g., mosey life, mount pros, music hall, then tri word entries e.g., sp studio lighting, sport supply group, stone case company.

For the single word list of brands names, we checked whether the word is a dictionary word or not. If it is a dictionary word, we remove that word from the list. However, we added popular brands that are even dictionary words, such as apple and orange. When checking the dictionary, we used Wordnet [39] because helps to identify all the forms of dictionary words. Also, we allowed single words that were the concatenation of two dictionary words e.g., microelectronics. For the entries that had more than one-word dictionary words, it was not an issue. Later, we sorted each list according to the length (number of characters). In the end, we made a single list by merging all 3 lists together according to the word size, and secondly length. Finally, we were able to generate a brand gazetteer list with nearly 15000 entries. Most of the entries are relevant to the C2C business model.

4.1.3 Training dataset Generation using the gazetteer

We had 5 datasets for different named entities brand, model, product name, product group, and sale state. But these gazetteers generated above are used only for the brand, model, and product name. We only selected the above five attributes due to the research constraints and we are looking to expand the number of attributes in future research. The datasets are a collection of social media messages from different sources such as Twitter feeds, and product listings from different C2C and B2C websites such as eBay and Amazon. The challenges in this training process are the labeling segment not having a static length, i.e., the number of words in each labeling element is different. Other than that, when thinking about automated training /labeling task we cannot expect that the words will exactly match an element in the gazetteer list, e.g., *GW Security Inc* vs. *gwsecurity* in this case has very few differences such as the space between GW and security, missing of Inc in training set, difference in case. The next challenge was the overlapping between two elements e.g., *smartworks consumer products* and *smartworks*. If both are in our gazetteer list, and if both represent two different brand names, then there is a conflict as one of the brand name is a part of another brand. The next challenge here is the occurrence of multiple entities of the same type, i.e., multiple occurrences of product entity in a single message. So even

after detecting an entity, we cannot stop the process for the rest of the text as we have to check against the gazetteer from the beginning.

4.1.4 String comparison in automated training

In our case, when we automatically trained the dataset it was a complicated task to recognize the multi-token named entities. In our gazetteer list, we have elements up to 3 words after normalizing. As the gazetteer generation list is sorted according to a number of words primarily, and the character length, the lengthy element will be labeled, and the others ignored if that matched multiple elements. This was implemented using n-gram [40]. First, the social media messages will go through an *n*-gram with the size from 3 to 1. Then, there will be a segment with n number of words next, that must compare with the gazetteer list. As we said earlier, a single instance may be located with little difference in both places. So, to overcome this, we implemented a few string similarity techniques. We only applied the string comparison to the entities with a length of more than 5. For smaller words, it may not work well. So we used the Jaccard Distance [13] based character distance edit method of 0.2 edit distance, as well as a token-based Jaccard Distance of 0.2.

4.1.5 NER using Conditional Random Fields

Conditional Random Fields (CRF) is a conditionally trained unidirectional graphical model. It can apply to standard linear-chain structure. So we selected the CRF [20] approach over the maximum entropy classifier, hidden Markov model [20] approach over maximum entropy classifier, hidden Markov model and maximum-entropy Markov model. The above four methods are relevant to sequential labeling. The Maximum-entropy Markov model makes decisions at each element independently However, the hidden Markov model overcomes the drawback of not showing good performance on the unknown label sequences. The Maximum-entropy Markov model overcomes the shortages and combines the advantages of maximum entropy and the hidden Markov model. So, we selected CRF it overcomes the problem with the Maximum-entropy Markov model such as label biased problem and the CRF avoid bias problem. The maximum-entropy Markov model is locally normalized but the CRF is globally normalized. The CRF is now widely used in many domains because of its state-of-the-art results.

Let $X_{1:N}$ denote the subset of a message and $Z_{1:N}$ the label of the named entity. Here, the Z is the normalization factor. Feature Functions are a key component in CRF. In our case, the feature function deals with many features e.g., the feature function will generate binary values. It is 0 if the current word is "sony", and if the current state Z_n is "BRAND". If this feature condition is satisfied,

$$f_1(z_{n-1,}z_n, x_{1:N}, n) = \begin{cases} 1 \; if \; z_n = barnd \; and \; x_n = sony \\ 0 \; otherwise \end{cases} \quad (4\text{-}1)$$

Unlike other algorithms, CRF depends on other positions as well. So, the feature function will calculate the previous and next label as well e.g., if the current label is brand and the next label is a product, the feature function for X_{n+1} will activate. The following Figure 4.4 is a graphical model presenting the CRF. X_I represents the word elements and y_i represents the labels.

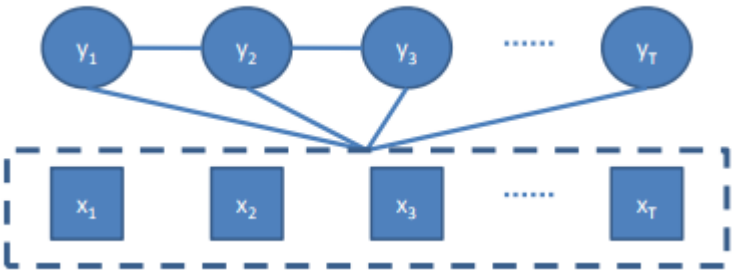


Figure 4.4 Graph view of CRF model.

In NER there are serval machine learning based approaches. The whole NER process was done using a Java NLP framework called LingPipe [41]. First, we implemented a training corpus compatible with Lingpipe.

Also, the ratio between the training dataset and the test dataset was 8:2. After the creation of the corpus, we must tokenize the corpus. In tokenizing the corpus, we implemented the BIO based tagging standard. This means that for each label, it may contain one or more words. So for the labeling, we placed each word in separate lines, then if the word was not in the label, it was labeled as –O, if the word was at the beginning of the label, it was labeled -B, if the word was part of the label but not at the start, then it was labeled as -I. In the model generated, feature extraction is a major

component. We implemented the following feature extraction techniques.

4.1.6 Product group and commercial intention classification

The next information we extracted from the C2C related social media messages were product group and commercial intention. Both of these attributes were not directly extracted from messages. Rather, each message classifies into predefined categories. In product classification, we have 3 categories that are available in our collected dataset, namely, electronics, musical instruments, and cellphones. Likewise,

Table 4.6 List of features used to classify the messages.

Selling Feature	Buying features
containing URL	No URL
many product specifications	No product /fewer product specifications
certain terms such as • I'm selling • buy cheap • cheap • for sale • check out • low price • deal • shop • reasonable • sell • buy • purchase • retail	Usage of certain terms such as • have to buy a • I want a • looking for a • how much the • planning to buy • urgent need • need a • ready to buy • can pay to • wanted • for buy • to buy
Less abbreviations / non-dictionary words	Usage of abbreviations

in commercial intention classification, we have another three buckets, namely, selling, buying, and nether. Here, we used the Logistic Regression Classifier. In our total corpus, nearly 1 million messages were included. The corpus was divided into a training set and test set in an 8:2 ratio.

4.1.7 Feature extraction on the classification

Feature extraction is an important aspect of any supervised machine learning method. In this logistic regression-based classification problem, we included several features

to classify both product group and commercial intention [42]. The features are listed in Table 4.6.

POS sequence features are also very important in commercial intent classification. In Table 4.7, we listed those POS sequences with example messages and the syntax tree for each POS sequence. The list of POS sequences mostly occurred among C2C e-commerce social media messages. Also, the selected sequences are specific to certain commercial intent, either buying or selling.

Table 4.7 POS based features

POS Sequence attributes	Status
VB JJ NN CD	sell
NN CD CD	sell
NN JJ CD	sell
NN CD CD NN	sell
NN IN DT JJS	sell
VB DT JJ NN	buy
VB JJ NN CD NN	sell
NN JJ CD	sell
JJ NN FW	buy
FW MD	buy
VB JJ NN	buy

4.2 Real-time stream processing

At the end of the IE, we will have the semantics of the messages. The next thing is matching relevant C2C messages based on the semantics. But the C2C social media matching use case is a big data problem, so we must solve this from different aspects. Both the IE and C2C message matching have two different big data challenges. To the IE we to make sure that the C2C messages are processed in real-time. So, we implemented a real-time IE solution for a high-velocity C2C social media stream using distributed real-time stream processing. So, the next problem is real-time C2C matching based on C2C. We had two challenges: real-time query processing from the

NoSQL database and real-time and batch-based matching within complex event processing. We explained the detailed implementation in the following sections.

4.2.1 Real-time information extraction

In big data processing, we had mainly two processing types of first real-time processing and batch processing. To implement the real-time IE we used distributed stream processing. We implemented the distributed stream processing using Apache storm. We can divide the implementation into two sections: physically distributed cluster and programmable logical topology. In the following topic, we explained about creating the cluster and the tools used in disturbed stream processing and the supportive services needed to achieve this, such as message passing, data bus, publisher-subscriber agent, distributed coordinator, and master-slave architecture.

Real-time distributed stream processing cluster implementation

As we explained earlier, to achieve real-time IE using distributed stream processing a distributed solution was implemented on top of the multi-node cluster. We used Apache Storm [43]as the software framework. It is a master-slave architecture given in Figure 4.5 to connect all the nodes in the Storm cluster Apache Zookeeper coordination service was used. This distributed infrastructure providing us the parallelism to achieve real-time processing.

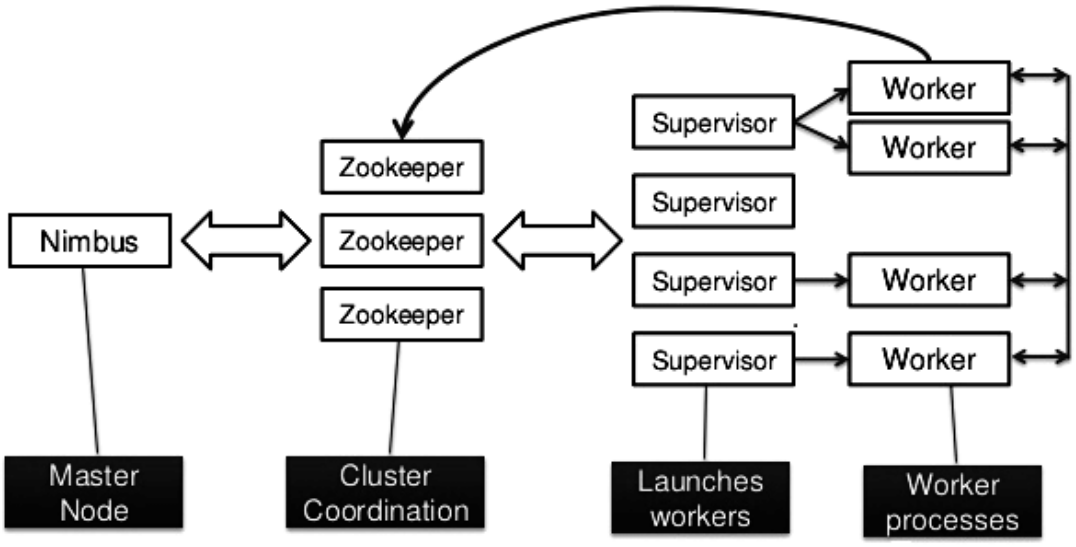


Figure 4.5 Apache Storm physical architecture.

Distributed coordination service

Coordination service is a core of the storm architecture. In Storm, Zookeeper is used as the coordination service. So, we first implemented a single-node Zookeeper cluster. In the Storm-distributed cluster, Zookeeper provided many services such as sharing configuration information among every node in that cluster and coordinating the various processes. Zookeeper acts as the storage space in the Storm cluster to store all the data and task records. Finally, it provided the ability to ensure availability through a multi-node cluster. As discussed earlier, using a distributed election approach, Zookeeper elects a leader and rest of the nodes act as followers.

To keep the cluster working continuously, the live/running nodes should be greater than n > 2. Also, as recommended, we assigned an odd number of nodes for our experiment. If we use 4 nodes, the minimum number of live nodes should 3 (3 > 4 / 2). So here, only one node can die. If we use 5 nodes while keeping the system available, two nodes can die.

The Zookeeper ensemble configuration is given below. Here the tickTime is the basic unit of time in milliseconds used by ZooKeeper. It is used to send heartbeats, and the minimum session timeout is twice the tickTime value. The four nodes of the Zookeeper cluster are mentioned in each ensemble configuration, such as a server.1,

After implementing the coordination cluster using Zookeeper with four server nodes, the second node was elected as the leader and all other three nodes were assigned as followers.

Real-time distributed parallel stream processing cluster

We used Apache Storm as the real-time distributed stream processing engine. As the main requirement for the Storm cluster implementation, the Zookeeper-based coordination service cluster was implemented above. In the next step, we implemented the physical distributed environment/infrastructure with both software and hardware resources to fulfill the real-time IE requirements. So, we implemented a six-node Storm cluster with 2 nimbus nodes and 4 supervisor nodes.

The nimbus nodes were configured by providing the IP address of the Zookeeper

clusters in the first place. In our Storm cluster, we assigned three Zookeeper node clusters, so we mentioned those addresses in the configuration. At the same time, in our Storm cluster, we assigned two nimbus nodes to overcome the failover. At the end, the Storm cluster was implemented and up and running, and we could monitor the total cluster using the Storm UI. Our distributed real-time IE infrastructure was completed at this point. In the next section, we explain the software implementation of our distributed processing of IE on top of this infrastructure.

Table 4.8 NER modules in stream processing topologies

Name of the unit	Type	Functionality
TwitterSpout	Spout	Receiving the tweets
brandNERBolt	Bolt	Brand Named Entity recognition
productNERBolt	Bolt	Brand Named Entity recognition
nerjoiner	Bolt	Merging both NER output streams into a single stream
StateClassificationBolt	Bolt	Classifying the message either product selling or product buying or neither
GroupClassificationBolt	Bolt	Classifying the messages among several product groups
classifierJoiner	Bolt	Merging both classifiers outputs streams into a single stream.
ModelRecognizerBolt	Bolt	Recognizing the model names from messages

Distributed real-time information extraction

We already did the core of the IE for regular application through sequence labeling / NER models and classification models. Also, we implemented the multi-node distributed cluster environment using Apache Storm and Zookeeper. In this section, we are going to explain the implementation of real-time IE for the high-velocity social media stream. Our solution has a clear architecture and implementation to solve this issue. In the storm cluster framework, they use the term Storm topology for this distributed parallel processing model. So, our storm topology contains two types of processing modules, spouts and bolts. The spouts response for receiving the stream and bolts for processing the stream. So, in our core Storm topology, we have a single bolt to receive the Twitter stream. At some time, we have 7 bolts in our core IE topology, as shown in Figure 4.6. In the upcoming sections, we will discuss additional bolts for the additional feature implementations. In Table 4.8, we listed the spouts and

bolts. Rather than working as a pipeline of sequence processing one after the other, we implemented a total parallel solution where each module mentioned above works independently. The graphical structure of our topology is shown in Figure 4.6.

The NER bolts can extract multiple NER entities such as a message containing multiple brands in a single message. So, we are producing the output of each NER bolts as a set, as shown in Figure 4.7.

4.2.2 Parallelism in Information Extraction

In our real-time distributed IE cluster implemented using Apache Storm, we have a multi-node cluster including a nimbus, Zookeeper and supervisor nodes. But the supervisor nodes are the execution part of the cluster. Each supervisor node is capable of running multiple JVMs. Each JVM is called a worker process in Storm terminology. The number of worker processes/JVMs in a single node can be configured with the topology definition. In our cluster, we only use 3 supervisor nodes and, in each node, we configured four workers processes/JVMs.

Now we have 16 JVM/worker processes going to process out IE modules. The next important aspect in parallelism in Apache Storm is the executors in the word threads. We can assign the number of threads/executors for each module in our topology, as listed in Table 4.9.

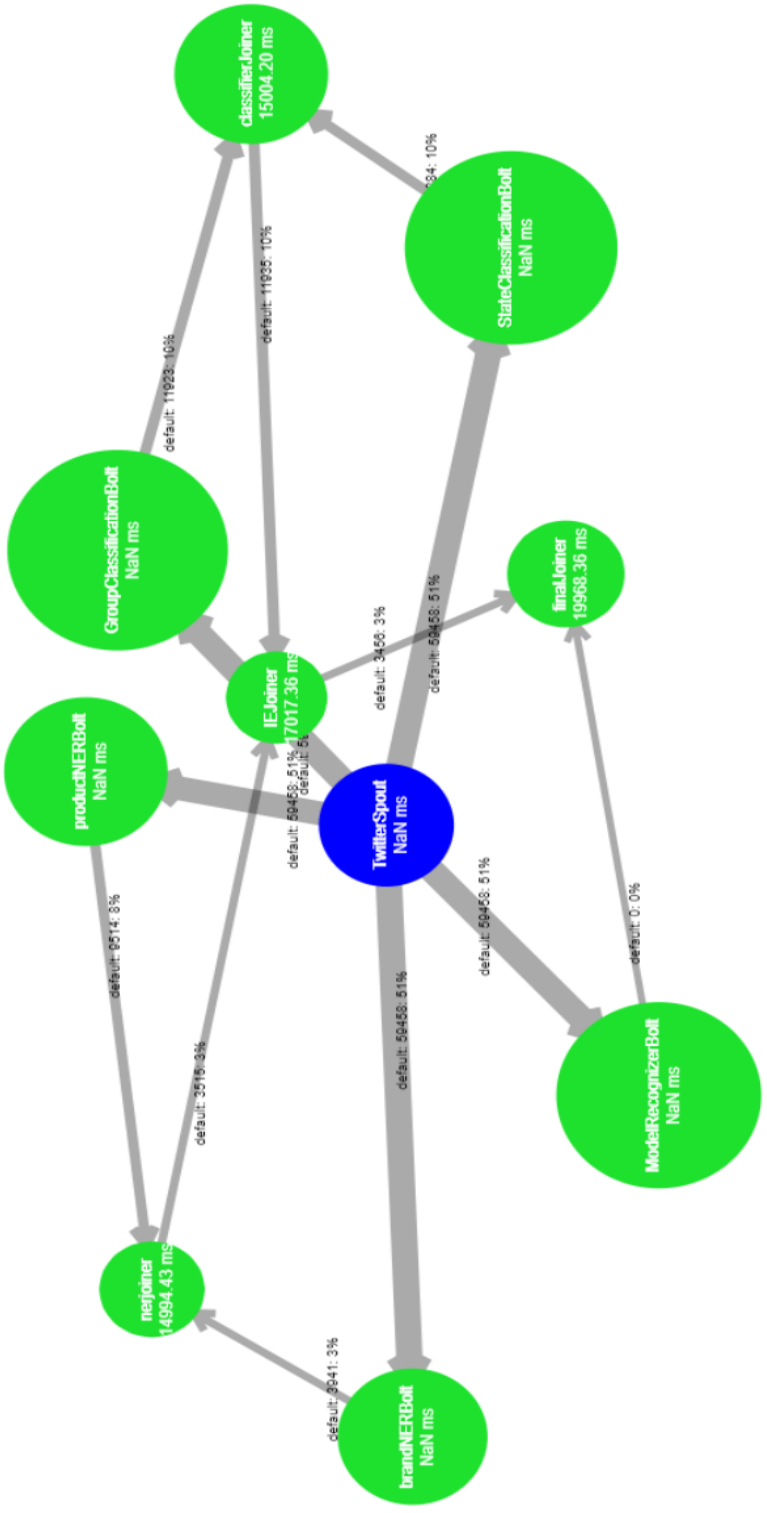


Figure 4.6 Storm topology graph representation.

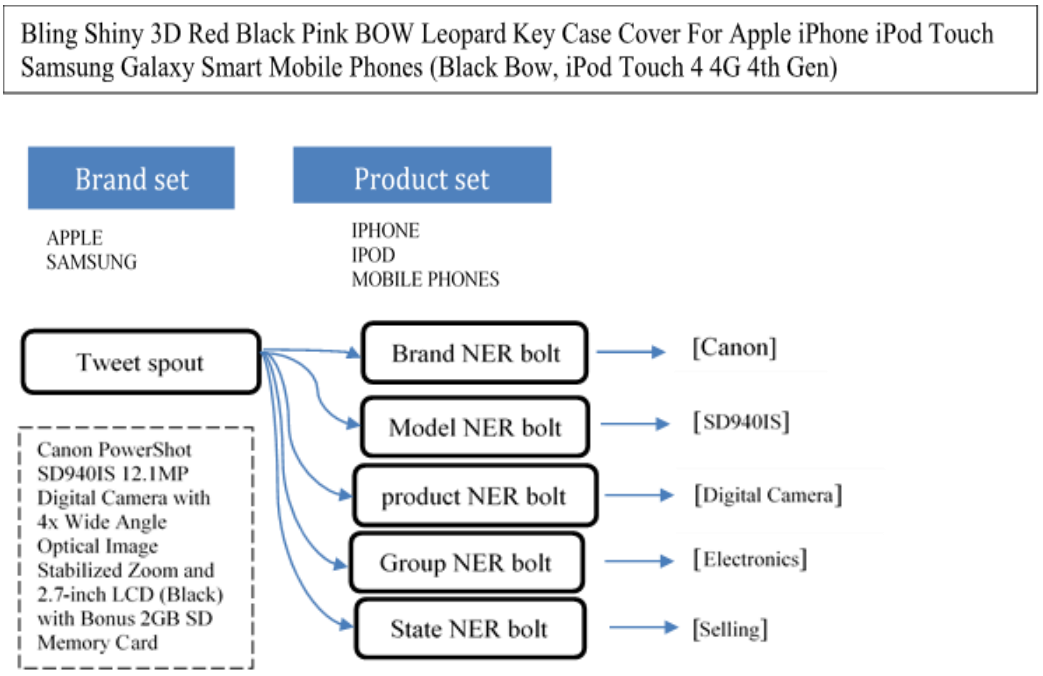


Figure 4.7 Sample input and output from each NER module.

So accordingly, we need 30 executers for each node, and 10 executors will be assigned. In each node, we have four worker processes so for distributed 10 executors among four worker processes like three executers for first two worker process and two executors for rest of the two workers process the clear picture illustrated in

Table 4.9 Each module in the Topology and their parallelism.

No	Module	No of Executers	No of Tasks Per Executer	Parallelism of Module
1	Twitter Spout	1	1	1
2	Brand NER Bolt	6	2	12
3	Product NER Bolt	5	2	10
4	Model NER Bolt	5	2	10
5	NER joiner	1	1	1
6	Group Classification Bolt	5	2	10
7	State Classification Bolt	5	2	10
8	Classifier Joiner	1	1	1
9	Final Joiner	1	1	1
	Total	30	14	56

The next parallelism concept in Apache Storm is Task, which means the instances of each module in each executor. The numbers of instances mentioned in Figure 4.8 illustrates the distribution of the task.

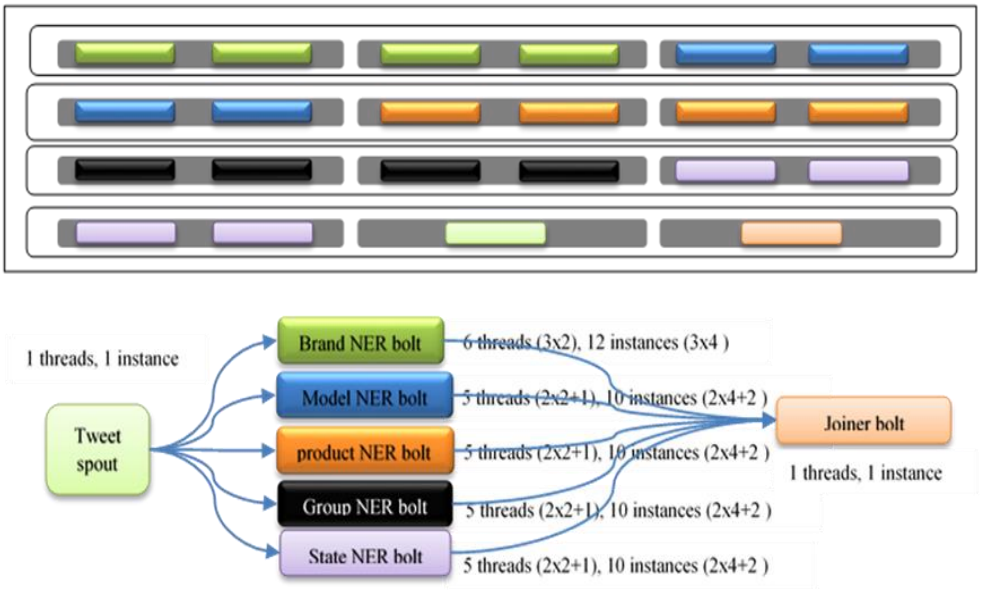


Figure 4.8 Task distribution in a single server node

Total parallelism

As mentioned in Table 4.9 BrandNER, we assigned 6 threads, and in each thread, 2 instances likewise, other 4 modules assigned 5 threads each running 2 instances and rest of 4 modules assigned single threads and each running single instances so finally, in our distributed stream processing cluster, we are running 56 instances parallelly.

Matching the C2C messages

In this matching requirement, we have considered a few things. First, matching between real-time streams, then matching between real-time and recent data/recently persisted data. If we need non-real-time matching, it is necessary to store the recent messages in data storage to retrieve later. In our use case, the data source is social media, and social media is a well-known example of big data. So, we needed big data storage to persist in our recent data. The next requirement is manipulating the relevant recent messages from big data storage according to the real-time messages. This real-time message stream has a very high velocity. Each message from the real-time stream

is considered as a query to manipulate the relevant messages from big data (recent messages persisted in NoSQL database). But the challenge is manipulating the data from the NoSQL database with high-frequency reading speed (High frequent data manipulation with low latency). To overcome this problem, we applied the in-memory indexing approach. Finally, matching stream-based data using conditional stream joining stream routing and real-time matching detection. But implementing these functionalities for a huge data stream with very high velocity is not easy. So, to overcome this issue, we applied a complex even processor with low latency, high throughput, and rich platform with many rich stream-based functionalities.

4.3 Bigdata Storage and in-memory computing

In the abstract, we applied three main technologies to solve the real-time big data matching:

1. NoSQL database
2. In-memory indexing
3. Complex Event processing

We applied in-memory indexing of the NoSQL storage to achieve this. In this section, we will see how to implement the NoSQL database. In our solution, we used Apache Cassandra as our NoSQL database [44]. It can manage a huge volume of data through distributed storage, a horizontally scalable nature, and flexible consistency options.

Our Apache Cassandra database will be receiving hundreds of thousand records in a short time. Our purpose of storing this huge amount of data in the NoSQL database was only because of making the matching between the real-time stream and offline data stored in NoSQL database. So, we should run a query on top of the NoSQL database based on the semantics of the real-time stream. Apache Cassandra shows a better sequence read performance record.

We have many reasons to select the Cassandra NoSQL [45] database. In our use case, we have a huge data storage, high writing rate, zero updates, and very low delete rate. Cassandra was implemented and optimized with the above features. Also, as a requirement, we must query the data in a very high frequency and low latency. But it is not straight forward, and the reading performance of the Cassandra is not that good.

Even though it supports secondary indexing, which is a helpful feature. Due to this reading problem, we did not apply multi-field queries while reading. Rather we only used a single parameter (brand name) queries. But our matching needed comparison between multiple fields. However, Cassandra does not support that kind of read requirements.

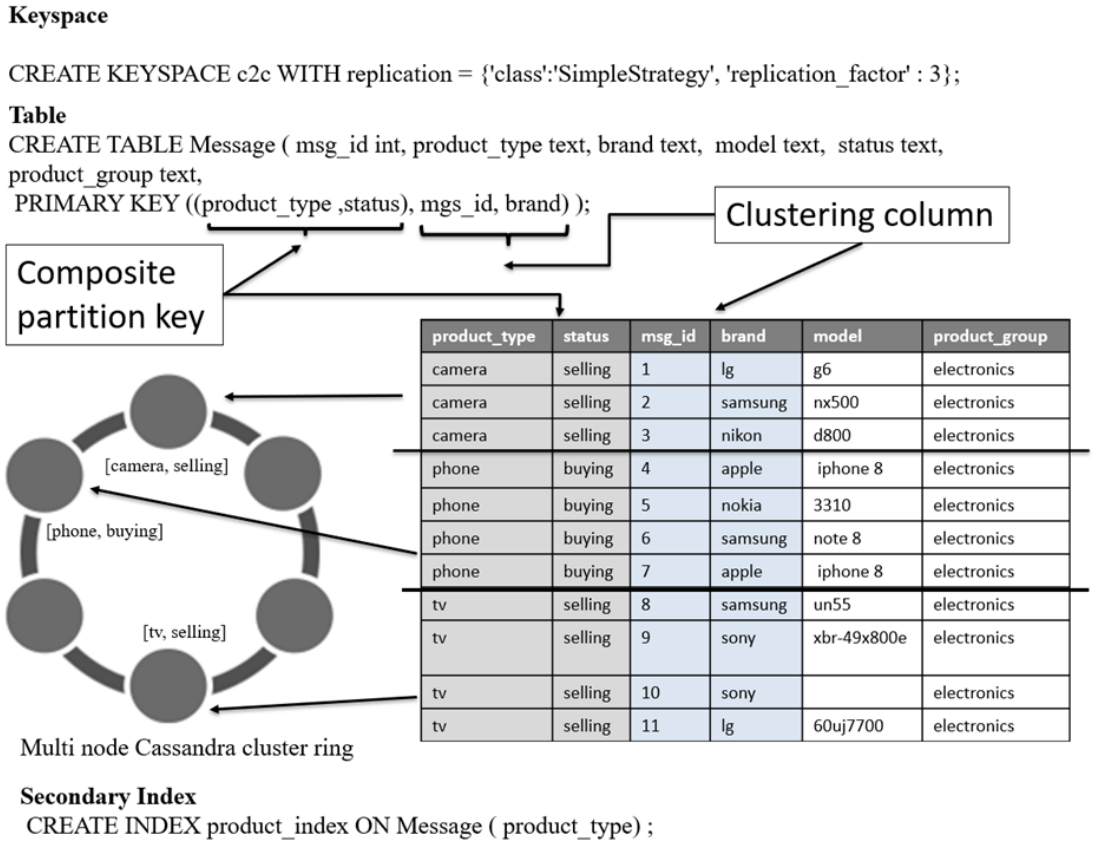
To overcome this reading issue, we used an in-memory data storing technology on top of the Cassandra NoSQL database, as shown in Figure 4.10. The in-memory storage makes 10-100 X faster than read from Cassandra directly. We used Apache Spark to achieve this. It is using an immutable in-memory storage facility called “Resilient Distributed Datasets (RDDs). We can create immutable distributed memory RDDs for our Cassandra tables, which is explained in detail in the following pages. Therefore, the client will access spark RDDs rather than Cassandra directly. We implemented a multi-node cluster with each node containing Apache Cassandra and Apache Spark.

After that, we created the schema according to our persisting stream. Cassandra is different from RDBMS in many ways. Instead of a database, Cassandra uses a key space. So, in Cassandra, we created a data model/schema considering our stream nature, indexing and a lot of other optimizations. Unlike RDBMS, in Cassandra, data is stored in different nodes and it does not support all types of queries that RDBMS support. Moreover, data modeling must be based on the query we need rather than being data-centric. To optimize the reading capability, we considered a few things: partitioning the data and secondary index.

In Cassandra, according to the partition key, the rows persisted to different nodes according to the hash values. So, it is easy to retrieve the data of a partial key from particular nodes. Also, if we want to use a column as a query parameter, it must be a part of the composite partition key and must be a secondary index.

Figure 4.9 illustrates our Cassandra data model and the data partition and row distribution. In our model, product type and selling status were assigned as the composite partition key. So, the row distribution among the nodes takes place according to the partition key. At the same time, the clustering column helps us to maintain the order. As illustrated in the above diagram, all the rows belonging to the

camera and with the status set as selling were stored in the first node and the rest of the rows stored in different nodes according to the partition key. Also, we added a secondary index to our data model. In Cassandra, if we want to create a secondary index, that column must be included in the partition key. In our case, the product type is already included in the partition key. At the same time, it is not good to select high cardinality or low cardinality columns as a secondary index. In our data model, the product type suits the secondary index. Also, it enables as to query the data based on a secondary index.



product_type	status	msg_id	brand	model	product_group
camera	selling	1	lg	g6	electronics
camera	selling	2	samsung	nx500	electronics
camera	selling	3	nikon	d800	electronics
phone	buying	4	apple	iphone 8	electronics
phone	buying	5	nokia	3310	electronics
phone	buying	6	samsung	note 8	electronics
phone	buying	7	apple	iphone 8	electronics
tv	selling	8	samsung	un55	electronics
tv	selling	9	sony	xbr-49x800e	electronics
tv	selling	10	sony		electronics
tv	selling	11	lg	60uj7700	electronics

Figure 4.9 Read optimized Cassandra data model.

Boosting read performance using in-memory computing

The Cassandra-based solution was optimized for our use case. To improve the read performance, we applied the in-memory storage and in-memory computing component on top of the Cassandra NoSQL database. So here, we are using Apache Spark, which

creates immutable storage units called Resilient Distributed Dataset (RDD). RDD is the fundamental data structure of Spark, which makes data reading multiple times faster compared to direct reading from the disk. So here, we are loading our table in Cassandra to spark RDD and then manipulating the queries with very low latency.

4.3.1 Matching between real-time and persisted data

So, from the above Cassandra-Spark combination, we are able to query big data in near real-time. This query results are considered as non-real-time data/ persisted relevant data to the upcoming real-time messages. As illustrated in Figure 4.10 this step we are going to take the real-time stream and the relevant messages persisted in the NoSQL database, into our matching module as two different streams. The matching module was implemented using a complex event processing technology.

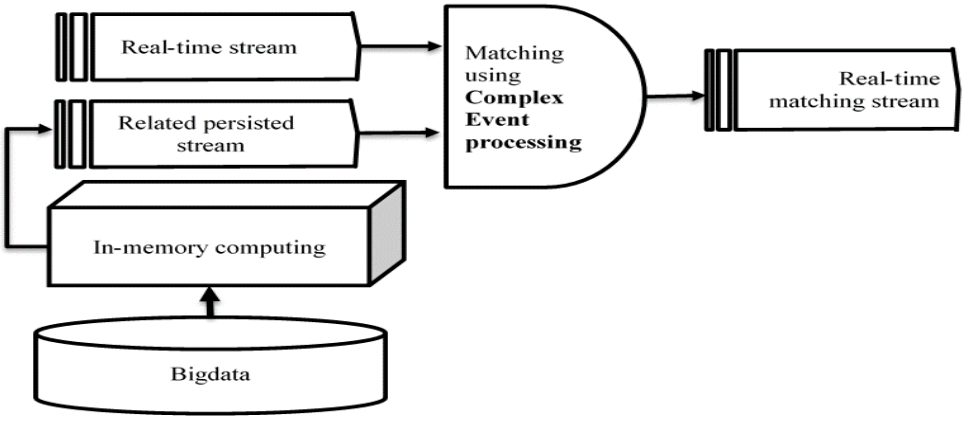


Figure 4.10 Matching component implemented using CEP.

In CEP, we implemented many executions plans. Some of the execution plan visualizations are given in Figure 4.11. Each execution plan does different tasks. First, we collect the real-time stream, then we split the stream into 5 different streams based on the available semantics such as brand name, the product name, model name, product group, and commercial intention. Also, another split stream from real-time stream goes to a persisting execution plan where each message will remain for one second before being stored in the NoSQL database. The other four splitters end with separate execution plans. The execution plans are designed to find the matching semantics among the messages, particularly the complete stream containing all the product semantics, which will be compared with all the partial and complete streams, and all

other partial streams compared with the streams accordingly. When a new message reaches the CEP apart from real-time to real-time matching, the system runs a query on in-memory storage containing the data loaded from the NoSQL database.

To fulfill our matching requirement, we are using WSO2 CEP, an efficient open-source complex event processing engine. Here, we receive two streams into the complex event processing i.e., real-time stream and relevant persisted stream. The following execution plans were part of our matching module.

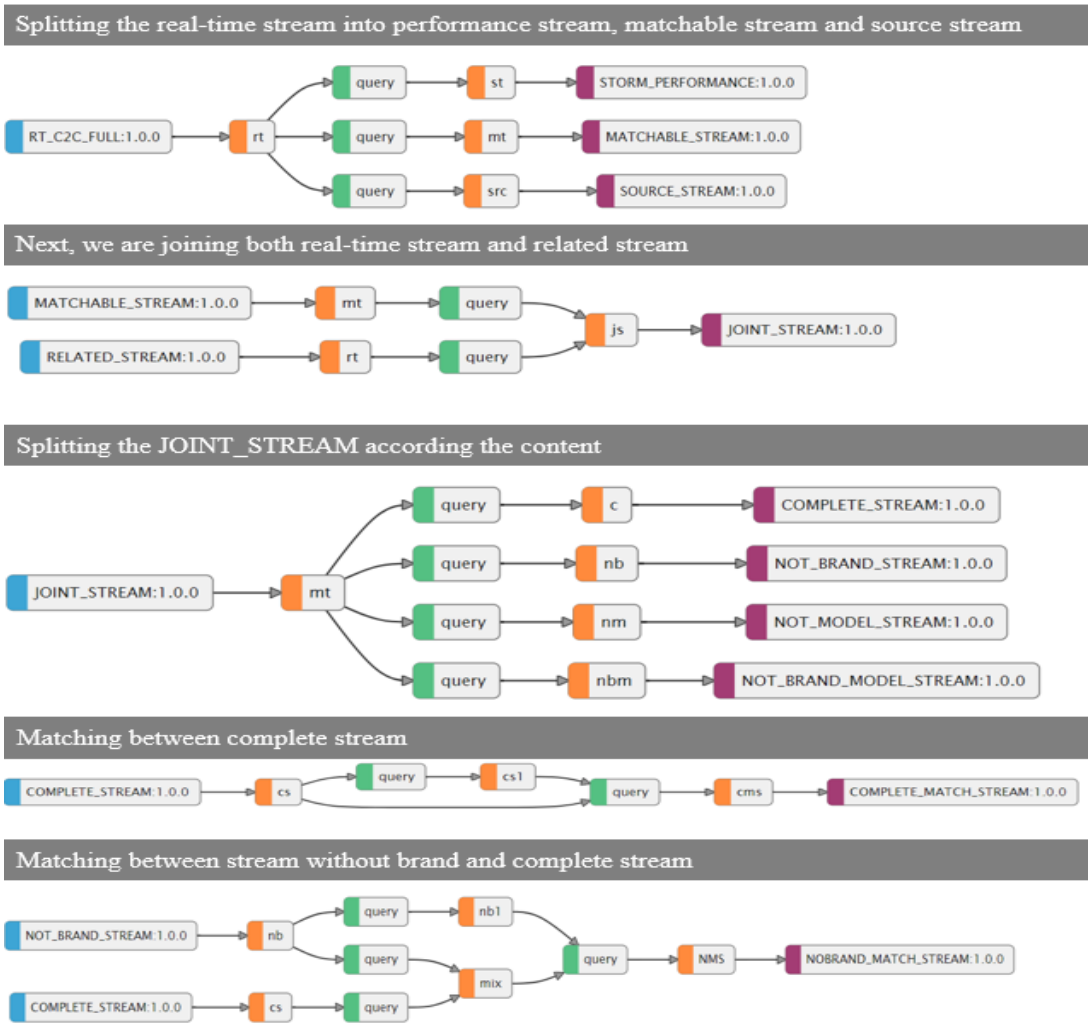


Figure 4.11 Stream and execution plans in CEP.

Window-based operations on matching using complex event processing are illustrated in Figure 4.12 We used two types of windows operations in our stream-based matching process Time based windows and length-based window. Here, we store the incoming messages into a queue i.e., time window. It means we are storing all the messages in the queue for a certain time e.g., 5 minutes. Comparing each real-time message i.e., length window with size 1 with messages stored in the queue. If we found the match between messages stored in those two windows, then we will concatenate both matchable messages into a single stream element and return as a matched message. In Figure 4.12 the basic abstract of our window implementation is shown.

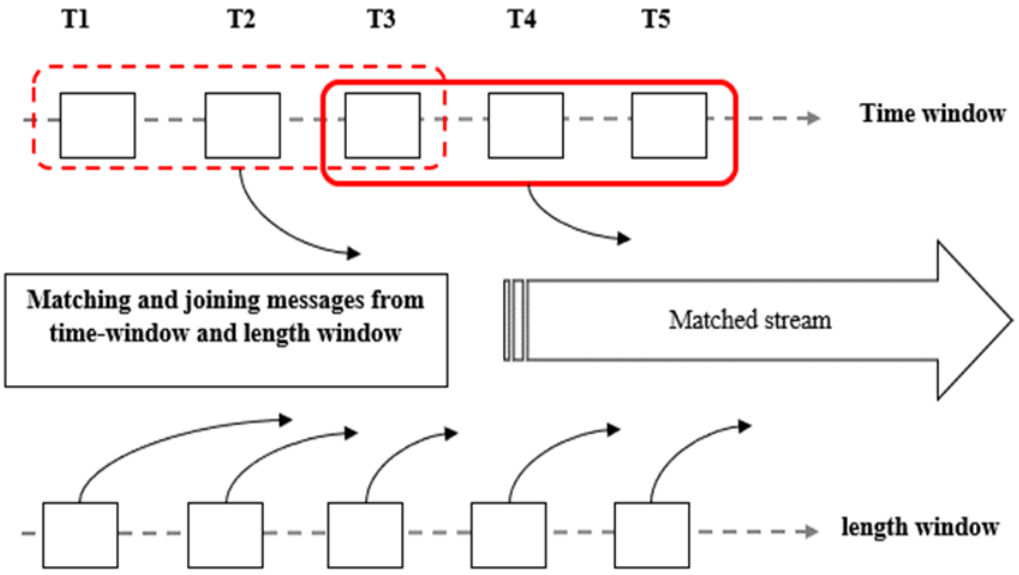


Figure 4.12 Time and size window based matching operation.

4.4 Non-functional aspects

As a research project, it is not an end-to-end solution to use by the public. But I can direct some suggestions to make this research as a real working solution. If we start from the beginning of the chain of process, the very first thing is receiving social media messages. For our experiment using a large number of messages are offline because of testing the performance. In a real-world application, there are many concerns about retrieving social media messages and privacy policy issues. We can introduce a

hashtag through some marketing approach to the people who are interested in our service. Then, if any message includes our hashtag, it is captured by our system.

Next, as a real-world application, prospective performance is a major concern. Now our solution is implemented to work in a distributed environment and gives a good performance result. But when we handle the performance aspect these days few things must be considered, such as load balancing, modular based development, component level scalability and microservice architecture.

The next aspect of wide range applicable nature is that the solution can be used better for some other domains that have a similar requirement and pattern. With the current implementation, it is particularly designed to work with the product domain. If we wanted to adopt this solution to some other domain first, we must make changes to the machine learning model. Our current solution has a product attribute extraction model. So, if we are using for a different domain, we must create a separate domain model and from the knowledge base, we are able to generate the training data set for various domains. We must make some changes in the logic of matching.

4.5 Summary

In this chapter, we described the detailed implementation. Mainly this chapter was divided into three sections such as IE in Section 4.1, real-time stream processing in Section 4.2 and big data storage and in-memory computing in Section 4.3. In IE, we discussed the training dataset preparation, NER using Conditional Random Fields and extracting commercial intend using logistic regression. In the next section, we concentrated on distributed real-time stream processing. Here, we apply the distributed stream processing techniques on IE to achieve a near real-time IE with very low latency for a very high-velocity social media stream.

Finally, the matching part of the C2C social media messages was explained in Section 4.3, which includes persisting data in a NoSQL database, high-frequency data manipulation from a NoSQL database using in-memory computing and matching the messages among real-time stream and non-real-time data using complex event processing.

CHAPTER 5

RESULTS AND ANALYSIS

In results and analysis, we present the full nature of our system. As a multi-component system, we have results for each component in our solution, as well as the results and performance of the total system. Mainly, we present two types of measure results; accuracy measures and performance measures. Some components only have accuracy results, some have both accuracy and performance, and the rest of the components only have performance measures. As a multi-component system, that includes information extraction, big data manipulation, and matching using complex event processing. Section 5.1 presents performance metrics. The experimental setup explained is presented in Section 5.2. Section5.3 presents the results of classification-based information extraction at the same time 5.4 presents CRF-based NER results. The following Sections 5.5 and 5.6 present the performance results of our distributed stream processing implementation, Big Data read-write performance and the Complex Event Processor.

5.1 Experimental setup

Our experimental setup is heterogeneous, which means many different computers with different configurations can be used based on availability. For the IE model creation, we used a computer with the following configuration: i7 core processor with A clock speed of 2.70 GHz, no of cores: 2, no of threads:4 cache sizes: 4 MB and CPU model: Intel® Core™ i7-7500U. The memory of the computer is 16 GB. As a software tool, we used the Lingpipe library with a heap size of 12 GB. As a dataset, we used messages collected from Twitter. The ratio between the training data and test data was 8:2.

5.2 Performance Metrics

Our IE design was to extract five named entities i.e., product type, brand, model, selling status and product group. To extract these five named entities, we applied three different techniques, such as sequence statistical model using CRF, classification using logistic regression, and rule-based IE using regular expressions. Among the five

entities to extract product type and brand, we used a CRF and to extract product group and selling status we used the logistic regression classification approach, and finally, to extract the product model, we used regular expressions.

$$accuracy = \frac{true\ positive + true\ negative}{True\ positive + false\ positive + false\ negative + true\ negative} \quad (5.1)$$

$$precision = \frac{true\ positive}{True\ positive + false\ positive} \quad (5.2$$

$$recall = \frac{true\ positive}{True\ positive + false\ negative} \quad (5.3)$$

$$F1 = 2 * \frac{precision * recall}{precision * recall} \quad (5.4)$$

5.3 Results of product classification using logistic regression

Here, we extract two product attributes by classifying the messages using logistic regression. First, we classify the messages into product groups such as electronics, cellphones, and music. The second classification is classifying the messages between selling, buying, and neither. We followed the k-fold cross validation for the evaluation to ensure the results represent the whole dataset.

Table 5.1 Accuracy of our both classification models

Classifying	Accuracy	Recall	Precision	F-measures
Product group	0.949	0.949	0.941	0.945
Selling status	0.99	0.991	0.991	0.991

The chart given in Figure 5.1 shows the distribution of three different product groups, such as cellphones, electronics, and music related products. In Figure 5.2, you can observe that the gradual improvement in the accuracy depends on the size of the dataset. We got a high accuracy in both classifications. For the product group

classification, our evaluation shows 0.949 accuracies and for the selling status, classification shows even more accuracy, 0.99. Table 5.1 lists the associated accuracy measures, such as accuracy, precision, recall, and F1. We cannot determine a model considering only the accuracy if one category represents most of the data points where accuracy is not a good measure for assessing model performance. While recall expresses the ability to find all relevant instances in a dataset, precision expresses the proportion of the data points in our model says was relevant actually were relevant [46]. We want to maximize either recall or precision at the expense of the other metric. We want to find an optimal blend of precision and recall where we can combine the two metrics using what is called the F1 score. The F1 score is the harmonic mean of precision and recall taking both metrics into account. The following plots show the gradient of the accuracy with the training size.

$$true\ positive\ rate = \frac{true\ positive}{True\ positive + false\ negative} \qquad (5.6)$$

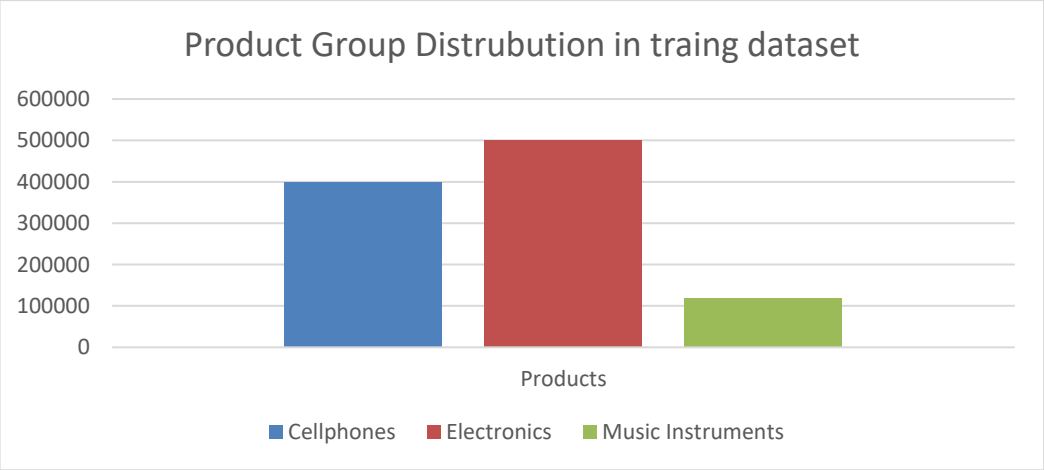


Figure 5.1 Product distribution in our training dataset

Next, an indicator of a perfect model is the Receiver Operating Characteristic (ROC) curve shown in Table 5.2 and Table 5.4. The ROC curve shows how the recall vs precision relationship changes as we vary the threshold for identifying a positive in our model. A ROC curve plots the true positive rate on the y-axis versus the false positive rate on the x-axis. Finally, we can quantify a model's ROC curve by calculating the total Area Under the Curve (AUC), a metric which falls between 0 and 1 with a higher number indicating better classification performance. An alternative to the ROC curve is the Precise Return Curve (PRC). Although used less often as ROC curves, PRC is suitable for unbalanced data sets.

Table 5.2 Area measures of PR and ROC Under curve

Classification Name	**Area Under PR Curve**		**Area Under ROC Curve**	
	Interpolated	**Uninterpolated**	**Interpolated**	**Uninterpolated**
Product group classification	0.9764	0.9747	0.9863	0.9863
Selling status classification	0.9986	0.9981	0.9984	0.9982

Table 5.3 Accuracy measures of CRF models.

Classifying	**Accuracy**	**Recall**	**Precision**	**F-measures**
Product name	0.8397	0.9036	0.9223	0.9128
Brand Name	0.8207	0.8733	0.9316	0.9015

Precision recall curves are often zig-zag curves, which often go up and down. Thus, the precision recall curve intersects much more often than the ROC curve. This makes comparisons between curves difficult. However, for perfect testing, curves near the PRC (described later) are better than curves near the baseline. In other words, curves over other curves have higher power levels. In Table 5.3, the measured values of the area measurement sub-curves for PR and ROC are very close to 1, which is a good indication of the best model.

$$false\ positive\ rate = \frac{false\ positive}{false\ positive + true\ negative} \qquad (5.7)$$

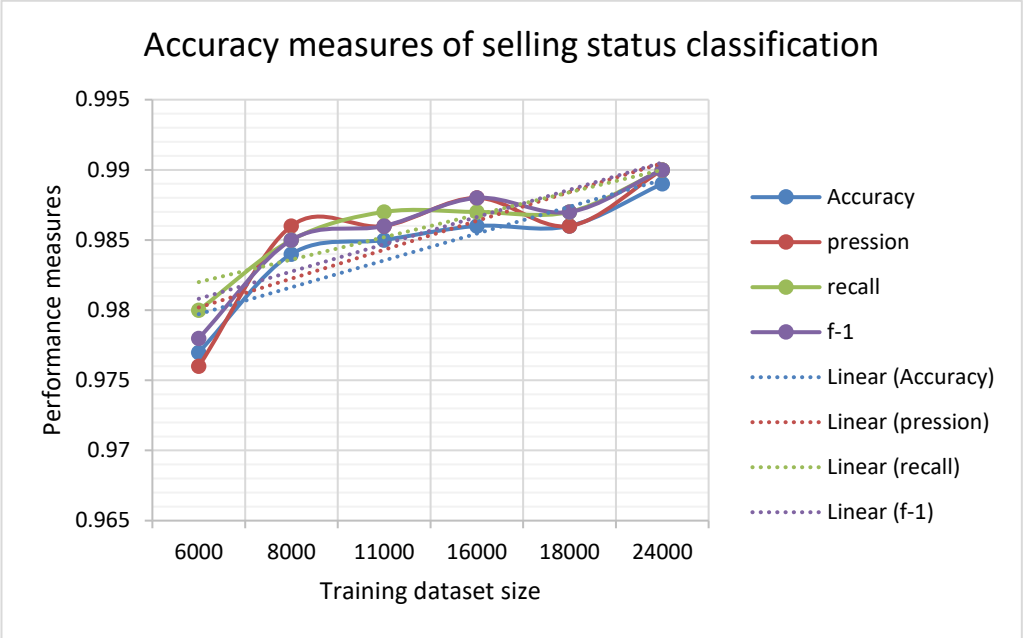


Figure 5.2 Selling status model accuracy Vs training set size.

5.4 Results of CRF-based product attribute extraction

For the rest of the product attributes we used the CRF to recognize the named entities.

If we take the accuracy of the CRF models, which are listed in Table 5.3 it shows that the accuracy of the product name recognition model is 0.84 and the accuracy for the brand name is 0.82. This is a good result and you can observe the improvement in the accuracy according to the training dataset size in Figure 5.3.

Table 5.4 Area measures of PR and ROC Under the curve

NER model name	Area Under PR Curve		Area Under ROC Curve	
	Interpolated	Uninterpolated	Interpolated	Uninterpolated
Product Name	0.9328	0.9219	0.9980	0.9980
Brand Name	0.9517	0.9515	0.9983	0.9983

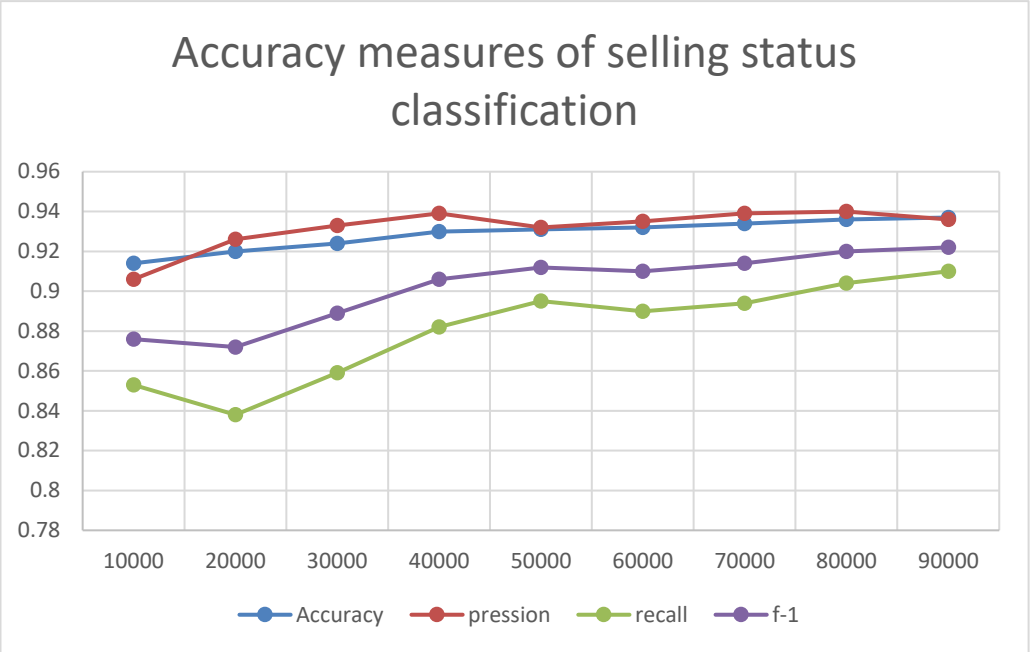


Figure 5.3 Product group model accuracy Vs training set size

5.5 Distributed information extraction performance

If we take the performance of the IE, as mentioned in the implementation details in Section 4.2, this part is totally distributed altogether. We implemented nine components in this distributed system. If we move to the latencies of each component, it varies depending on the workload of the component. You can see the latency details in Figure 5.4. where apart from joining components, others take around 0.5 milliseconds.

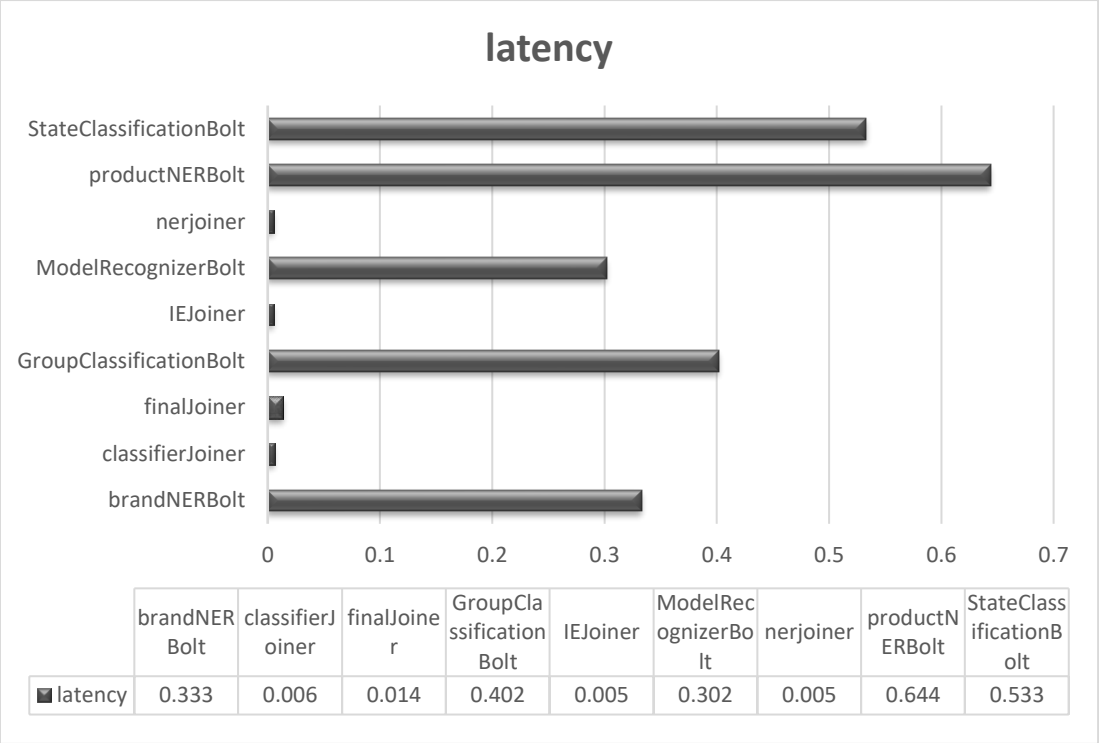


Figure 5.4 Average latency of each information extraction modules

5.6 Results of high-frequency data manipulation from NoSQL

Big data manipulation is a major part of our solution. With the combination of the Apache Cassandra NoSQL database and Apache Spark in-memory computing, enable us a high-speed data manipulation. The following results show the write and read performance. In Figure 5.5 it illustrates the writing operations per second and Figure 5.6 shows the average latency of the writing operation. We achieved around 350 messages written per second as throughput. So, our writing performance shows an average latency of 4 ms. At the same time, the data manipulation performance is shown in Figure 5.7 where the throughput of the data manipulation from the NoSQL database reached around 9500 readings per seconds. In Figure 5.8, it presents the throughput of matching between the real-time stream and manipulated data from the NoSQL database where we achieved more than 82500 matches per second. At the same time, the throughput of the matching among real-time stream is presented in Figure 5.9. This

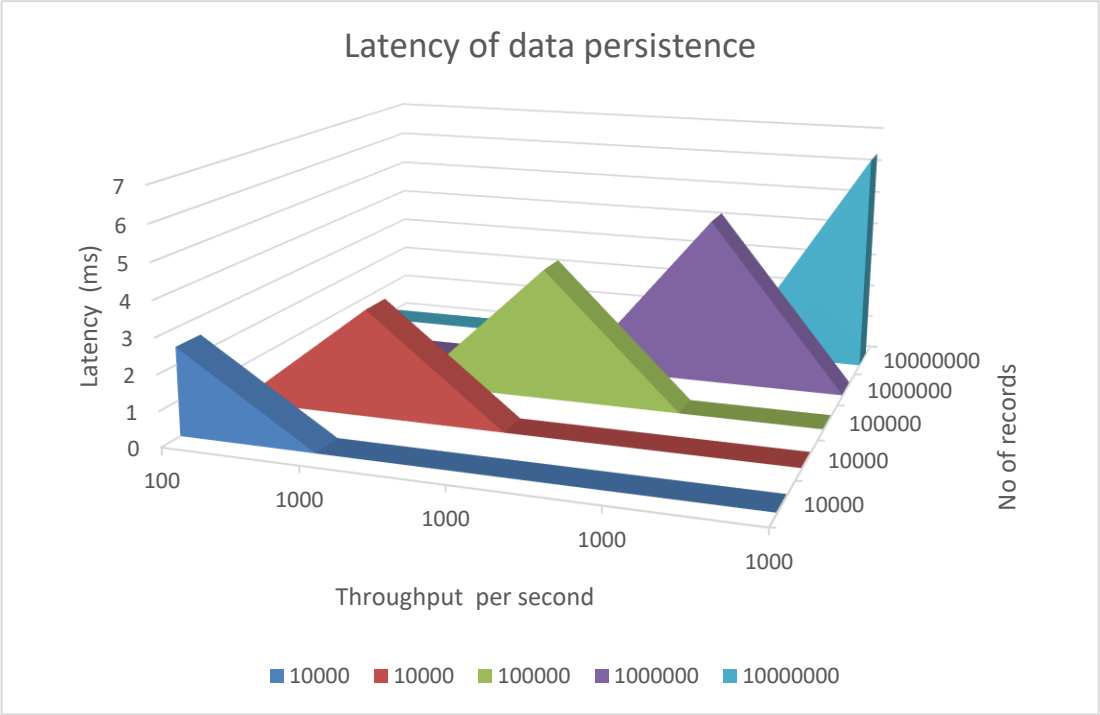


Figure 5.5 Data persistence latency Vs size of the database and throughput.

time we achieved more than 165000 matchings per second. Finally, the overall performance is presented in Figure 5.10 and Figure 5.11. In Figure 5.10 we show the overall throughput and reached around 4700 messages per second and as a latency, we reached 0.76 milliseconds as presented in Figure 5.11.

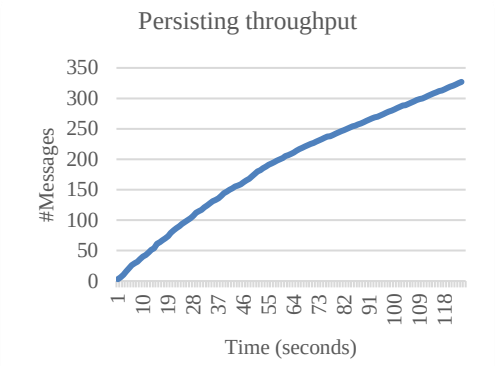


Figure 5.6 Persisting throughput

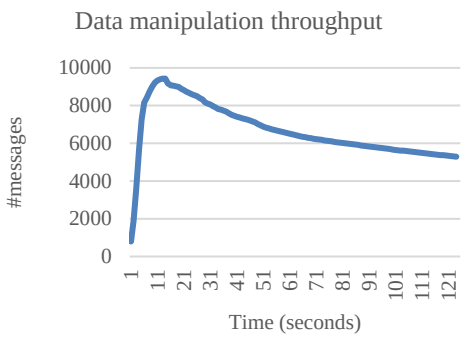


Figure 5.7 Data manipulation throughput

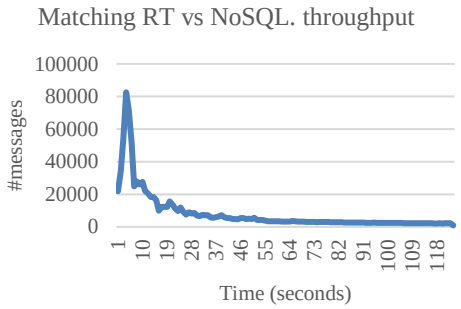


Figure 5.8 Matching RT vs NoSQL. throughput

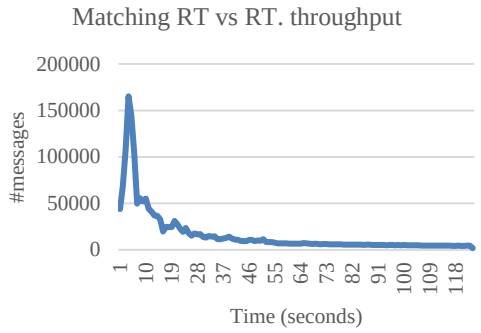


Figure 5.9 Matching RT vs RT. throughput

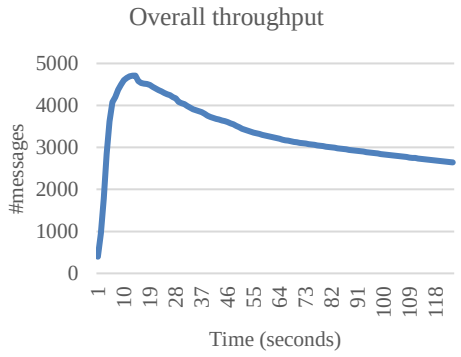


Figure 5.10 Overall throughput

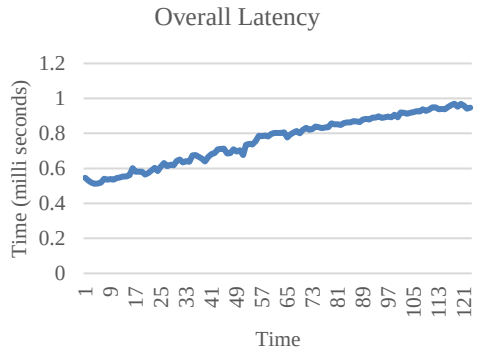


Figure 5.11 Overall latency.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

Many industries started to adapt to big data technologies. Even ordinary people adapted to depend on social media for many of their requirements, particularly in the e-commerce industry where the influence of social media is very high. But there are many challenges and problems in the social media-based e-commerce industry. One of the problems we discovered in social media-based Consumer to Consumer (C2C) is that product offers do not reach the right consumer.

As an outcome, we can pull real-time microblogging messages as social messages and in near real-time, we were able to notify the C2C end users with the matching of their message. Due to our solution, many C2C uses will benefit from real-time. To make that possible most part of our solution is distributed among the multi-node cluster.

Initially, our system extracts product attributes from raw messages using machine learning based IE. It is the part that controls the accuracy of the whole system. As our trained product extraction model, the end results give good accuracy. Next, the output is delivered in real-time or near real-time. In this aspect as well, we were able to achieve good performance with very low latency. We applied three technologies to achieve near real-time, such as distributed stream processing, in-memory computing, and complex event processing. Third, according to the requirement, our system is scalable in runtime. Finally, our system was implemented to overcome failover occasions, in other words, our implementation ensures the availability of the system.

The internal modules and algorithms enable high accuracy and real-time performance. For example, Conditional Random Fields (CRF) in NER in-memory computing in big data manipulation and Complex Event Processing in matching the semantics. Apart from the core components, we used a few tools in our solution, such as Lingpipe, Apache Storm, Apache Spark, WSO2 CEP, and WSO2 DAS.

When we consider the performance of our implementation, we can divide the performance records into each main component in our system. In the abstract for the IE, the average latency is 0.5 ms. The latency of the persisting data on NoSQL is around 6 ms. Even though it takes a little more time, it will not affect our overall performance because we always retrieve the data that persisted 1000ms before at the same time the persistence reached 350 writes per second as throughput. The high-frequency data manipulation shows a high throughput as 9500 events per second, which the matching between the real-time stream and non-real-time showed 82500 events per second as throughput and matching among real-time messages showed 165000 events per second. Overall, it takes 0.7 ms latency on average and shows 4700 events per second as maximum throughput.

6.2 Research Limitations

We can list a few things as our research limitations. First, in creating the NER model we were able to create a single model for all our product attributes classes. Due to the limited resources, rather than a single NER model, we created separate models for each product attribute. Secondly, in the real-time distributed stream processing system for each task, we created separate bolt components with different numbers of multiple instances. For each instance of a task, the Apache Storm facilitates to maintain a state. But as a single task, we do not have a common state among all its instances.

Even more, if you take the variety of products, it is a huge variety of products that are available in the market. But in our research, we only implemented the solution for very few products. Again, if you take the number of attributes of different products, it varies from product to product. But our solution was only designed for very few attributes such as product names, brand names, product group, and some more. The next limitation is that with the new products in the market and a new set of attributes, our system can recognize named entities that are not given in the training dataset. But always in machine learning, if we increase the size of the training dataset, we can produce the result with more accuracy. Now, we are using CRF and logistic regression algorithms in our systems but if we move to the deep learning algorithms such as LSTM and word embedding, we can produce more accurate and good performance results.

6.3 Future Work

For our research, we only considered a few products attributes that we extract from the raw messages, but each product has a different number of attributes. Some product even contains hierarchical attributes. But in our current solution, we have only 5 attributes, such as product name, product brand, product model, product group and product selling status. So, we have more scope to work on this product attribute NER.

Related to this product attribute extraction, we can focus on more things such as feature engineering and the deep learning approach, which will increase the accuracy of the system. As mentioned in the research limitations, if we have enough hardware resources, particularly GPU, we can try for a single NER model for all the product attributes.

Also, as mentioned in the research limitation, we can improve the distributed real-time stream processing in many ways, such as enabling a synchronized approach to share the same state between a different instance of the same task and enabling the runtime auto resource scalability and load balance. Also, as a successful methodology, we can follow deep learning approaches in NER.

In C2C matching we implemented the message matching based on 5 product attributes. But there are a lot of opportunities that are not discovered that can help to match C2C uses such as user behaviors in social media, profile details, friends and their similarities. Using that information, we can predict the user needs and be able to connect the consumers directly

In this era, almost all the products and services somehow reach social media, in different ways. So, through collecting and storing that product information as a global open source product database, it opens a new opportunity for C2C community.

The next opportunity that we can move forward in is that most social media messages not only contain the text details, it also contains an image of the product. In this scenario, we already worked on extracting the product attributes from the text messages. We can convert this structured product attribute and the image of the product will lead us to product recognition from the image and related opportunities, including product matching based on social media product images.

Different types of databases are available in use such as relational database, NoSQL database, graph database, document database, even more, have databases to store specific information such as DNA database and a Drug database. Each database has its pros and cons. Each type of database has been optimized for different use cases. But here, the product information structure shows a huge amount of records and replicating attribute terms. Also, sometimes, a product is a composition of multiple products or may be part of another product. So, finding an optimized way to arrange the product data itself is a challenge.

REFERENCES

[1] Y. C. Rathod, *Methods and systems for brands social networks (bsn) platform*. Google Patents, 2011.

[2] V. N. Gudivada, D. Rao, and V. V. Raghavan, "NoSQL systems for big data management," in *2014 IEEE World congress on services*, 2014, pp. 190–197.

[3] M. J. Denny and A. Spirling, "Text Preprocessing For Unsupervised Learning: Why It Matters, When It Misleads, And What To Do About It," *Political Analysis*, vol. 26, no. 2, pp. 168–189, Apr. 2018.

[4] F. Sun, A. Belatreche, S. Coleman, T. M. McGinnity, and Y. Li, "Pre-processing online financial text for sentiment classification: A natural language processing approach," in *2014 IEEE Conference on Computational Intelligence for Financial Engineering Economics (CIFEr)*, 2014, pp. 122–129.

[5] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60.

[6] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," *Lingvisticae Investigationes*, vol. 30, no. 1, pp. 3–26, 2007.

[7] "The Porter stemming algorithm: then and now | Program | Vol 40, No 3." [Online]. Available: https://www.emeraldinsight.com/doi/abs/10.1108/00330330610681295. [Accessed: 23-Jun-2018].

[8] C. D. Paice, "An Evaluation Method for Stemming Algorithms," in *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, NY, USA, 1994, pp. 42–50.

[9] R. R. Larson, "Introduction to Information Retrieval," *Journal of the American Society for Information Science and Technology*, vol. 61, no. 4, pp. 852–853.

[10] G. V. Bard, "Spelling-error tolerant, order-independent pass-phrases via the Damerau-Levenshtein string-edit distance metric," in *Proceedings of the fifth Australasian symposium on ACSW frontiers-Volume 68*, 2007, pp. 117–124.

[11] E. Brill and R. C. Moore, "An Improved Error Model for Noisy Channel Spelling Correction," in *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, Stroudsburg, PA, USA, 2000, pp. 286–293.

[12] "Comparison of biosequences - ScienceDirect." [Online]. Available: https://www.sciencedirect.com/science/article/pii/0196885881900464. [Accessed: 26-Jun-2018].

[13] M. U. S. Shameem and R. Ferdous, "An efficient k-means algorithm integrated with Jaccard distance measure for document clustering," in *2009 First Asian Himalayas International Conference on Internet*, 2009, pp. 1–6.

[14] W. E. Winkler, "Overview of record linkage and current research directions,"

BUREAU OF THE CENSUS, 2006.

[15] B. Mohit, "Named Entity Recognition," in *Natural Language Processing of Semitic Languages*, Springer, Berlin, Heidelberg, 2014, pp. 221–245.

[16] G. Zhou and J. Su, "Named entity recognition using an HMM-based chunk tagger," in *proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, 2002, pp. 473–480.

[17] D. M. Bikel, S. Miller, R. Schwartz, and R. Weischedel, "Nymble: A High-performance Learning Name-finder," in *Proceedings of the Fifth Conference on Applied Natural Language Processing*, Stroudsburg, PA, USA, 1997, pp. 194–201.

[18] A. L. Berger, V. J. D. Pietra, and S. A. D. Pietra, "A Maximum Entropy Approach to Natural Language Processing," *Comput. Linguist.*, vol. 22, no. 1, pp. 39–71, Mar. 1996.

[19] M. Collins, "Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms," in *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, Stroudsburg, PA, USA, 2002, pp. 1–8.

[20] J. Lafferty, A. McCallum, and F. Pereira, "Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data," *Departmental Papers (CIS)*, Jun. 2001.

[21] G. Bello-Orgaz, J. J. Jung, and D. Camacho, "Social big data: Recent achievements and new challenges," *Information Fusion*, vol. 28, pp. 45–59, 2016.

[22] A. Gandomi and M. Haider, "Beyond the hype: Big data concepts, methods, and analytics," *International Journal of Information Management*, vol. 35, no. 2, pp. 137–144, 2015.

[23] M. Stonebraker, U. Çetintemel, and S. Zdonik, "The 8 Requirements of Real-time Stream Processing," *SIGMOD Rec.*, vol. 34, no. 4, pp. 42–47, Dec. 2005.

[24] A. Toshniwal *et al.*, "Storm@Twitter," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2014, pp. 147–156.

[25] H. Gokavarapu, "Exploring Cassandra and HBase with BigTable Model," *Indiana University Bloomington*.

[26] Z. Hasani, M. Kon-Popovska, and G. Velinov, "Lambda architecture for real time big data analytic," *ICT Innovations*, pp. 133–143, 2014.

[27] J. M. Pierre, "Mining knowledge from text collections using automatically generated metadata," in *International Conference on Practical Aspects of Knowledge Management*, 2002, pp. 537–548.

[28] E. S. Ristad and P. N. Yianilos, "Learning string-edit distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 5, pp. 522–532, May 1998.

[29] A. B. M. Moniruzzaman and S. A. Hossain, “NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison,” *arXiv:1307.0191 [cs]*, Jun. 2013.

[30] S. Suhothayan, K. Gajasinghe, I. Loku Narangoda, S. Chaturanga, S. Perera, and V. Nanayakkara, “Siddhi: A Second Look at Complex Event Processing Architectures,” in *Proceedings of the 2011 ACM Workshop on Gateway Computing Environments*, New York, NY, USA, 2011, pp. 43–50.

[31] J. Z. Pan, “Resource Description Framework,” in *Handbook on Ontologies*, Springer, Berlin, Heidelberg, 2009, pp. 71–90.

[32] A. Jena, “Apache Jena Fuseki,” *The Apache Software Foundation*, 2014.

[33] O. Hartig, C. Bizer, and J.-C. Freytag, “Executing SPARQL Queries over the Web of Linked Data,” in *The Semantic Web - ISWC 2009*, 2009, pp. 293–309.

[34] R. He and J. McAuley, “Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering,” in *proceedings of the 25th international conference on world wide web*, 2016, pp. 507–517.

[35] P. Petrovski, A. Primpeli, R. Meusel, and C. Bizer, “The WDC gold standards for product feature extraction and product matching,” in *International Conference on Electronic Commerce and Web Technologies*, 2016, pp. 73–86.

[36] J. McAuley and A. Yang, “Addressing Complex and Subjective Product-Related Queries with Customer Reviews,” in *Proceedings of the 25th International Conference on World Wide Web*, Republic and Canton of Geneva, Switzerland, 2016, pp. 625–635.

[37] P. Petrovski, A. Primpeli, R. Meusel, and C. Bizer, “The WDC Gold Standards for Product Feature Extraction and Product Matching,” in *E-Commerce and Web Technologies*, 2016, pp. 73–86.

[38] L. Derczynski, A. Ritter, and S. Clark, *Twitter Part-of-Speech Tagging for All: Overcoming Sparse and Noisy Data*. .

[39] G. A. Miller, “WordNet: a lexical database for English,” *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[40] W. B. Cavnar and J. M. Trenkle, “Ngram-based text categorization,” in *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, 1994, pp. 161–175.

[41] B. Carpenter, “LingPipe for 99.99% recall of gene mentions,” in *Proceedings of the Second BioCreative Challenge Evaluation Workshop*, 2007, vol. 23, pp. 307–309.

[42] B. Hollerit, M. Kröll, and M. Strohmaier, “Towards Linking Buyers and Sellers: Detecting Commercial Intent on Twitter,” in *Proceedings of the 22Nd International Conference on World Wide Web*, New York, NY, USA, 2013, pp. 629–632.

[43] A. Jain, *Mastering Apache Storm: Real-time Big Data Streaming Using Kafka, Hbase and Redis*. Packt Publishing, 2017.

[44] A. Chebotko, A. Kashlev, and S. Lu, “A Big Data Modeling Methodology for Apache Cassandra,” in *2015 IEEE International Congress on Big Data*, 2015, pp. 238–245.

[45] N. Neeraj, *Mastering Apache Cassandra*. Packt Publishing Ltd, 2013.

[46] W. Koehrsen, “Beyond Accuracy: Precision and Recall,” *Towards Data Science*, 03-Mar-2018. [Online]. Available: https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c. [Accessed: 10-Jun-2019].