

A CONTEXT AWARE NOTIFICATION FRAMEWORK BASED ON DISTRIBUTED
FOCUSED CRAWLING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS OF
THE MIDDLE EAST TECHNICAL UNIVERSITY
BY

MEHMET ALİ AKYOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

AUGUST 2017

Approval of the thesis:

**A CONTEXT AWARE NOTIFICATION FRAMEWORK BASED ON DISTRIBUTED
FOCUSED CRAWLING**

Submitted by **MEHMET ALİ AKYOL** in partial fulfillment of the requirements for the degree of
Master of Science in Information Systems Department, Middle East Technical University by,

Prof. Dr. Deniz Zeyrek Bozşahin
Director, Graduate School of **Informatics**

Prof. Dr. Yasemin Yardımcı Çetin
Head of Department, **Information Systems**

Asst. Prof. Dr. P. Erhan Eren
Supervisor, **Information Systems Dept., METU**

Examining Committee Members:

Assoc. Prof. Dr. Altan KOÇYİĞİT
Information Systems Dept., METU

Asst. Prof. Dr. P. Erhan EREN
Information Systems Dept., METU

Asst. Prof. Dr. Murat Perit ÇAKIR
Cognitive Science Dept., METU

Asst. Prof. Dr. Rahime BELEN SAĞLAM
Computer Engineering Dept., Yıldırım Beyazıt
University

Assoc. Prof. Dr. Sevgi ÖZKAN YILDIRIM
Information Systems Dept., METU

Date:

21.08.2017

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Mehmet Ali Akyol

Signature : _____

ABSTRACT

A CONTEXT AWARE NOTIFICATION FRAMEWORK BASED ON DISTRIBUTED FOCUSED CRAWLING

Akyol, Mehmet Ali

MSc., Department of Information Systems

Supervisor: Asst. Prof. Dr. P. Erhan Eren

August 2017, 54 pages

The amount of data generated from sources on the Web has been increasing on a daily basis. Hence, it is time-consuming to follow all these sources to reach the latest information. The way people access information on the Web is usually pull-based, meaning that they query the Web over time to find the most recent blog posts, websites, news, and even weather reports. Accessing the right information at the right time is crucial for both people and businesses to be more productive and efficient. Moreover, the information that cannot be accessed at the right time may lose its value over time. Traditional pull-based methods for obtaining information may cause important knowledge to be overlooked or too late to be noticed. Technologies like RSS enable people to access information from websites through push-based notifications, but they cannot provide a proper solution for people who are being exposed to too much information at inappropriate times. Accordingly, a push-based context aware solution is needed for people who are in need of accessing the right information at the right time. Although some promising studies in the literature have tried to solve this problem, they appear to be insufficient to meet today's big data requirements. In this study, we propose a context aware notification framework based on distributed focused crawling, in order for people to get notifications on relevant information at the right time and location by leveraging the latest advancements in distributed computing and big data analytics technologies.

Keywords: Big Data, Stream Processing, Distributed Focused Crawling, Context Aware Notifications, Distributed Complex Event Processing

ÖZ

DAĞITIK ODAKLANMIŞ TARAMA TEMELLİ BAĞLAM BİLİNÇLİ BİR BİLDİRİM UYGULAMA ÇATISI

Akyol, Mehmet Ali

Yüksek Lisans, Bilişim Sistemleri Bölümü

Tez Yöneticisi: Yrd. Doç. Dr. P. Erhan Eren

Ağustos 2017, 54 sayfa

Web üzerindeki kaynaklardan üretilen veri miktarı her geçen gün artmaktadır. Bu kaynaklardan üretilen bilgileri takip etmek ve en son bilgilere ulaşmaya çalışmak zaman alan işlerdir. İnsanlar için Web üzerindeki bilgilere yaygın olarak ulaşma yöntemi genellikle sorgulama esaslıdır. İnsanlar en son blog yazılarını, web sitelerini, haberleri ve hatta hava durumu raporlarını bulmak için Web'i belli zaman aralıklarında sorgulamaktadırlar. İlgili bilgiye doğru zamanda ulaşmak hem insanların hem de işletmelerin daha verimli ve üretken olabilmeleri için çok önemlidir. Ayrıca, doğru zamanda ulaşılamayan bilginin insanlar ve işletmeler için değeri ortadan kaybolabilir. Geleneksel sorgulama bazlı bilgiye ulaşma yöntemi önemli bir bilginin gözden kaçabilmesine ya da geç fark edilebilmesine sebep olabilir. RSS gibi teknolojiler insanların web siteleri üzerinden bildirim bazlı bilgiye ulaşabilmelerine olanak sağlarlar da insanların çok fazla bilgiye, uygun olmayan zamanlarda maruz kalmalarına çözüm olamamaktadırlar. İnsanların doğru bilgiye doğru zamanda ulaşabilmeleri için bağlam bilinçli bildirim bazlı bir çözüme ihtiyaç vardır. Her ne kadar literatürde bu problemi çözmeye yönelik bazı çalışmalar olsa da bunlar günümüz büyük veri gereksinimlerini karşılamak için yeterli değildir. Biz bu çalışmada, dağıtık hesaplama ve büyük veri analiz yöntemlerindeki en son gelişmelerden yararlanarak, insanların doğru bilgiye doğru zamanda ulaşabilmelerini sağlayan, dağıtık çalışan odaklanmış tarama temelli bağlam bilinçli bir bildirim uygulama çatısı öneriyoruz.

Anahtar Sözcükler: Büyük Veri, Akan Veri İşleme, Dağıtık Odaklanmış Tarama, Bağlam Bilinçli Bildirimler, Dağıtık Karmaşık Olay İşleme

To my wife and my family

ACKNOWLEDGMENTS

First of all, I would like to express my sincere appreciations to my supervisor Asst. Prof. P. Erhan Eren for his support, generous guidance, comments, and his friendship.

I would also like to thank Assoc. Prof Altan Koçyiğit for his support, guidance, comments, and for sharing his knowledge throughout this research.

There are a lot of people that were with me during this research. Their love, help, and understanding were the only power I had. They are the true owners of this work.

I would like to thank my colleagues; Mert Onuralp Gökalp, Kerem Kayabay, Serhat Peker, Gökçen Yılmaz, Ali Mert Ertuğrul, Özge Gürbüz, Ebru Gökalp, Abdullah Gümüšođlu, Murat Koçak, Ahmet Faruk Acar and Sibel Gülнар. I am also deeply grateful to Ahmet Ada, Veysel Çakır, Talat Gökçer Canyurt, Alper Özel, İbrahim Uzun, Onur Burç, Emre Elgün, Muhammed Ömer Sayın and Mert Murat Bal for their encouragement and friendship.

Special thanks to my beloved wife Şeyma Akyol, my mother Ummahan Akyol, my father Hüseyin Akyol, and my cousin Sevda Akyol. Words cannot express how grateful I am to them.

TABLE OF CONTENTS

ABSTRACT.....	iv
ÖZ	v
DEDICATION.....	vi
ACKNOWLEDGMENTS	vii
TABLE OF CONTENTS.....	viii
LIST OF TABLES.....	x
LIST OF FIGURES	xi
LIST OF ABBREVIATIONS.....	xii
CHAPTERS	
1 INTRODUCTION	1
2 BACKGROUND AND RELATED WORKS.....	3
2.1. Big Data.....	3
2.1.1. Batch Processing	4
2.1.2. Stream Processing	4
2.1.3. Lambda Architecture.....	4
2.1.4. Kappa Architecture	5
2.2. Distributed Computing.....	6
2.3. Complex Event Processing.....	6
2.4. Focused Crawling.....	6
2.5. Related Works	6
3 SYSTEM ARCHITECTURE	9
3.1. URL Sources	10
3.2. Cloud Application Server.....	10
3.3. Database	11
3.4. Mobile Application	11
3.5. Distributed Message Queue	11
3.6. Distributed Processing Unit	12
3.6.1. Distributed Focused Crawler.....	12
3.6.2. Distributed CEP Engine	13
3.7. Message Delivery Unit.....	13
4 PROTOTYPE IMPLEMENTATION.....	15

4.1. Used Technologies	15
4.1.1. Apache Flink	15
4.1.2. Apache Kafka	19
4.1.3. Apache Zookeeper	20
4.1.4. Docker	21
4.2. Implementation	24
4.2.1. Mobile Application	25
4.2.2. Bookmarklet	29
4.2.3. Web Application	30
4.2.4. Distributed Processing Unit Implementation	35
4.2.5. Distributed Message Queue Implementation	36
4.2.6. Storage	37
4.3. Sample Use Cases	37
4.3.1. Use Case 1	37
4.3.2. Use Case 2	38
4.3.3. Use Case 3	38
5 RESULTS AND DISCUSSION	41
5.1. Test Environment	41
5.2. Network Characteristics	42
5.3. Test Scenarios	42
6 CONCLUSION	49
REFERENCES	51

LIST OF TABLES

Table 1. AWS EC2 Server Features	34
Table 2. Host Machine Features	36
Table 3. Cluster Node Specifications.....	41

LIST OF FIGURES

Figure 1. Lambda Architecture	5
Figure 2. Kappa Architecture.....	5
Figure 3. System Architecture	10
Figure 4. Apache Flink Levels of Abstraction.....	16
Figure 5. Apache Flink Stack.....	17
Figure 6. Apache Flink Program Flow	18
Figure 7. Apache Flink Architecture	18
Figure 8. Anatomy of a Topic.....	20
Figure 9. Apache Zookeeper Internals.....	21
Figure 10. Architecture of Docker.....	22
Figure 11. Prototype Software Stack	25
Figure 12. Mobile Application Login Screen	26
Figure 13. Mobile Application Location Screen.....	27
Figure 14. Mobile Push Notification	28
Figure 15. Mobile Application Digests Screen.....	29
Figure 16. Bookmarklet	30
Figure 17. Web Application Twitter Authentication	30
Figure 18. Web Application Registration	31
Figure 19. Web Application Source Page.....	32
Figure 20. Web Application Context Configuration Page.....	33
Figure 21. Web Application Digest Page	34
Figure 22. Apache Flink Crawler Execution Plan	35
Figure 23. Apache Flink CEP Execution Plan.....	35
Figure 24. Network Bandwidth Measurement Results	42
Figure 25. Throughput of Distributed CEP Engine	43
Figure 26. Throughput of Distributed Focused Crawler.....	44
Figure 27. Throughput Increase by Task Managers for Distributed CEP Engine	45
Figure 28. Throughput Increase by Task Managers for Distributed Focused Crawler	45
Figure 29. Memory Utilization of Task Managers	46
Figure 30. Overall Memory Usage of the Cluster.....	47

LIST OF ABBREVIATIONS

RSS	Rich Site Summary
TLD	Top Level Domain
IoT	Internet of Things
BI	Business Intelligence
ETL	Extract Transform Load
CEP	Complex Event Processing
CAS	Cloud Application Server
DSL	Domain Specific Language
SQL	Structured Query Language
JVM	Java Virtual Machine
YARN	Yet Another Resource Negotiator
EC2	Elastic Compute Cloud
GCE	Google Compute Engine
DAG	Directed Acyclic Graph
IETF	Internet Engineering Task Force
AWS	Amazon Web Services
IPC	Inter Process Communication
NLP	Natural Language Processing
CPU	Central Processing Unit

CHAPTER 1

INTRODUCTION

The rapid growth of data sources such as the Web and social media causes a continuous accumulation of information. Scaling up the performance of crawling engines is becoming more of a challenge in the big data era. Search engines like Google, Bing, and DuckDuckGo focus on crawling the entire Web and provide results to individuals when they request information. In spite of the search engines' significant efforts with highly powerful hardware and software, crawling the entire Web is almost impossible in today's world. Moreover, search engines may still fail to provide the latest information on websites because of their web page ranking mechanisms. Web page rankings on search engines depend on a large number of issues, and freshness is only one of many factors in page ranking algorithms.

Most of the websites that search engines spend time, money and effort to crawl are unlikely to provide valuable information for individuals or businesses. Therefore, focused crawling is an appropriate approach to find the web pages that are likely to be relevant to a predefined topic [1]. The focused crawling method proposes to crawl the specific categories, TLD (top-level-domains) or selected topics. According to the study [2], while generic web crawlers may fail to perceive the differences between relevant and related topics in returned results, focused crawling enables us to search for a particular topic continuously and discover any new information on the pre-defined topic. However, accessing the information is still pull-based in which users need to keep querying the topic to be updated about the new information. Therefore, a push-based notification method is required to access the latest information on web pages without intervention. Moreover, if the information loses its value over time, there are costs associated with missing or late identification of new information for people and businesses. The push-based notification method eliminates the process of continuously querying the same topic so that users get notifications as soon as the new information becomes available.

In addition to reaching information on a push-based manner, it is also important to provide new information to users in the proper context, and this is associated with the situation of a person, place, or object [3]. Contexts are categorized into two main categories, which are external and internal contexts. External contexts are usually measured by sensors such as location, temperature, light, sound and air pressure. On the other hand, internal contexts are specified by the users or inferred from the users'

interaction with the system. Users' tasks, goals, emotional state and business processes can be samples of internal context [4]. The users' preferences and behavior may change dynamically, for instance; users may like to have new information about their hobbies at home or on weekends, but they may also want to receive work related information on weekdays or while they are at work. At this point, the utilization of context is particularly important. Context aware systems can adapt their features according to the context without explicit user action [4]. Therefore, context awareness can increase usability and effectiveness of systems by taking environmental context into consideration [4]. However, analyzing dynamically changing contexts together with focused crawling requires processing fast and voluminous data flowing from a variety of context sources, which are characteristics of big data. Therefore, the architecture for context aware systems and focused crawling engines should be highly scalable to support big data processing.

In this study, a scalable architectural framework is proposed to crawl the web pages on a specified topic, and to send notifications to people according to their context preferences via different notification channels; email, SMS, and mobile push notifications in a distributed manner. This enables individuals and businesses to reach the latest valuable information in predefined topics and in proper contexts such as location and time preferences. The proposed architecture also supports the state-of-the-art big data processing, distributed message queues and cloud computing technologies to handle high volume, velocity, and variety of the data extracted from crawled web pages and user contexts. Moreover, the overall performance of the system can also be scaled up quickly when it is needed.

The rest of the thesis is organized as follows; Chapter 2 reviews the background information related to the underlying concepts, the literature of focused crawlers and context aware systems. Chapter 3 introduces the system architecture of the proposed framework. Chapter 4 explains the details of the prototype implementation. Chapter 5 provides the details of the experimental setup and presents the results of the evaluation of the scalability of the framework. Finally, conclusion and future research directions are given in Chapter 6.

CHAPTER 2

BACKGROUND AND RELATED WORKS

2.1. Big Data

Big Data is the research field that is dedicated to the processing, analysis, and storage of large collections of data that is frequently originating from various sources. Big Data is required when the traditional methods of data processing, analysis, and storage technologies become inadequate. There are different requirements that big data techniques and methodologies need to satisfy such as combining a lot of various unrelated datasets, processing of immense amount of structured or unstructured data and extracting hidden information from the data in a time sensitive manner.

Some of the essential characteristics of Big Data are as follows – Volume, Variety, Velocity, Veracity, and Value [5].

- **Volume:** The Volume attribute refers to the scale of the data that is needed to be gathered, stored, and processed. As the traditional single server data processing and storage technologies can be insufficient for massive amounts of data, the volume of the data requires distributed storage and processing techniques.
- **Variety:** The Variety characteristic refers to the diversity of different types of data to be processed. In traditional data processing, it is usually one type of structured data requiring no more than one type of processing
- **Velocity:** The Velocity characteristic refers to the rate at which the data is produced and must be consumed to gain the most of out of the value that you could get from the data, as the value of the information that could be extracted from data can diminish over time. With recent advancements in the field of Internet of Things (IoT) and popularity of social networks such as Twitter, the data generated from the sensors and social networks are fast data and requires special techniques to deal with.
- **Veracity:** The Veracity characteristic refers to the quality of data. In the process of big data, consistency, accuracy, completeness, and integrity of data are

crucial aspects to consider, as the data coming from independent and open data sources can bring about risks and incomplete results.

- Value: The Value characteristic refers to the usefulness of data. It is important to extract useful information at the end of the Big Data processing to gain benefits for an enterprise or an individual.

Depending on the processing requirements of data, Big Data processing can be categorized into two main processing techniques which are Batch Processing and Stream Processing.

2.1.1. Batch Processing

Batch processing deals with the handling of data at certain time intervals. It is the solution for the volume characteristics of big data and requires distributed storage of data. Because of the processing of the immense amount of data at certain intervals, it also results in high latency responses.

Primary uses of batch processing can be BI (Business Intelligence), predictive analytics, prescriptive analytics and ETL (Extract Transform Load).

2.1.2. Stream Processing

Stream Processing deals with processing of the continuous stream of data usually in real time. It is the solution for velocity characteristic of big data. In stream processing, data can be represented as stream or events. Stream data can be continuous and events can occur at certain time intervals. As the small amounts of data processed, it resulted in low latency responses.

Main uses of stream processing can be processing of sensor readings, web logs, mobile application data, patient monitoring, user transactions, and social media analysis in real time fashion.

Usually, most of the data processing frameworks are designed for only batch or stream processing. But some frameworks support both batch and stream processing at the same time. These frameworks can be based on Lambda or Kappa architectures. Even though these architectures enable to process in batch or stream fashion, there are some differences.

2.1.3. Lambda Architecture

Lambda architecture [6] is a data processing architecture for both batch and stream processing. In Lambda architecture, there are two different layers one for the batch

and one for the stream processing. The data is fed to both layers in parallel and results are collected via a serving layer [7].

The underlying architecture of Lambda is depicted in Figure 1.

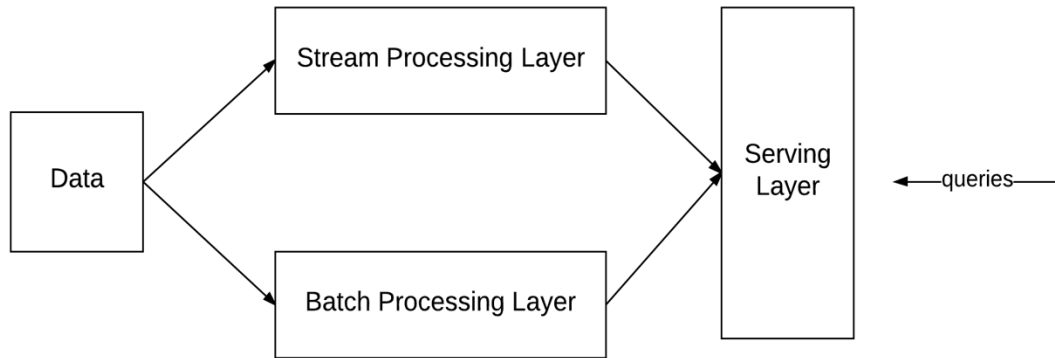


Figure 1. Lambda Architecture

In lambda, the same input data is fed to both a throughput-optimized batch and a latency-optimized real-time layer in parallel, whose results are then made available to the applications via a serving layer. Kappa, in contrast, feeds the input only to a streaming system, followed by a serving layer, which supports both real-time and batch processing by replaying historical data from a logging.

2.1.4. Kappa Architecture

Kappa Architecture [8] can also handle batch and stream processing at the same time, but there is a difference between lambda and kappa architectures. In Kappa architecture, both batch and stream processing are done on the streaming; it does not require any particular layer for batch processing. Batch processing is treated as a special case of stream processing and is handled within the stream processing engine by replaying the historical data from the logging system which can be a database or a messaging queue. Simplified Kappa architecture is depicted in Figure 2.

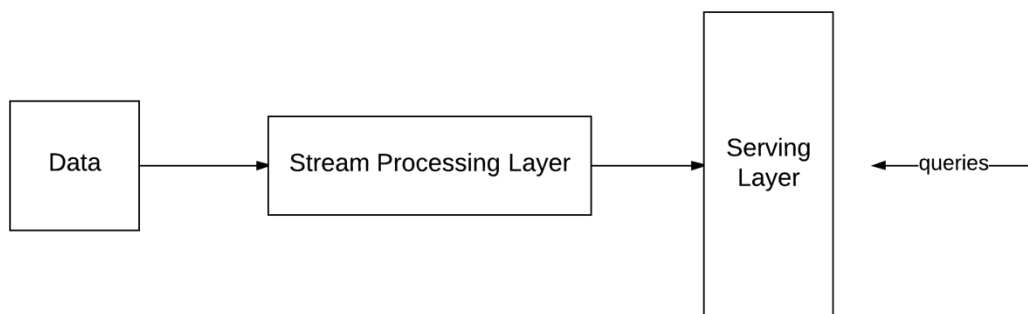


Figure 2. Kappa Architecture

2.2.Distributed Computing

When the processing requirements of a system increase, it becomes not feasible to accomplish the job with a single server implementation. Big Data processing requires an immense amount of computing power along with the storage space. Distributed Computing is one of the enabler technologies of Big Data.

Distributed Computing is the execution of tasks typically in a set of separate but connected servers. Other than Big Data processing in general, some of the uses cases for distributed computing can be telecommunication networks, real time applications, and parallel computing applications.

2.3.Complex Event Processing

Complex Event Processing (CEP) is the handling of multiple events coming from various sources to identify and detect useful events to act as fast as possible. CEP requires real time processing and can be used for various real world applications such as social media posts, stock market data, weather reports, sales leads, customer support tickets and so on [9].

2.4.Focused Crawling

Focused crawling [1] is the technique of crawling the Web pages with a specific focus. The focus could be only crawling the specific list of web pages or web pages in the same category or specific domains depending on the TLD. There are different types of focused crawling approaches to enhance the crawling process such as semantic focused crawlers using an ontology to classify web pages or focused crawlers using machine learning methods to detect whether the page is relevant or not before it is downloaded.

2.5.Related Works

Focused Crawling and Context Aware Notification systems are not new paradigms in the literature, but they need to be transformed to provide a solution for growing data volume and velocity in the big data era. This section reviews the recent focused crawling and context aware notification systems in the literature.

In the literature, there are numerous studies related to focused crawling. In order to evaluate their extensibility and to provide a better solution in the big data era, these studies are analyzed in two main categories, depending on whether their architecture support distributed environment settings or not. Most of the studies [1], [10], [11] related to focused crawling propose a centralized single node solution in which their performance is limited due to processing capabilities of a single server. Thus, these solutions do not provide a viable solution in the big data era.

Several distributed solutions Mercator [12], Ubicrawler [13], BUbiNG [14], [15] and [16] for focused crawling are proposed in the literature. Although they serve different purposes and use cases, they commonly present a scalable focused crawling system architecture to distribute processing components across several nodes. For example, TwitterEcho [17], proposes a distributed focused crawling architecture for Twitter, they aim to collect data from the Twitter and make it available for researchers. Another study [13], proposes a focused crawler for social networks which tracks the specified Twitter topic, using MapReduce programming model to benefit from its inherent support for distributed computations. Mercator is also one of the most notable web crawling architectures in the literature, also used by AltaVista search engine. It is a scalable and extensible web crawler whose components can be distributed across several computing units. However, there is a centralized module in the Mercator's architecture to manage all the crawling process which constitutes a performance issue that affects the overall system.

There are push-based solutions that people can use to get notifications when certain events occur in the web content. Users can use RSS [18] to subscribe to changes in the content of websites that support this technology. Google Alerts [19] can notify users of new articles upon subscription. The main limitation of these techniques is that they require users to subscribe to changes in the websites that support some particular functionalities. Alert Notification System [20] tackles this problem by adding an automated intelligent agent in the communication between the server and the user. This agent detects changes like data about an object becomes part of a web page, a new property is associated with a given object in a web page, and a new value is set to a property of an object in a web page.

Most of the existing solutions do not consider context information when sending push notifications. If users are not given a choice to filter which information to receive in certain contexts, they prefer not to receive any notifications except for the critical ones. This may lead to missing a significant piece of information which is only valuable for a limited amount of time or while the user is in a certain location or situation. In context aware systems, reminders and notifications have a crucial role, as notifying people at the right time in the right channel can increase productivity and efficiency. There are many examples of context aware notification systems in the literature. In the study by Katsiri [21], registered users who are interested in a particular activity are notified. In another study [22], machine learning algorithms are used to manage the incoming notifications. Even though these examples showcase many context aware notification use cases, in our framework, we are dealing with an entirely different use case, delivering the right information in various channels like email, SMS, and mobile push notifications depending on users' preferences like time and location.

In the big data era, we see distributed focused crawling and context awareness as crucial components of information retrieval. Even though there are distributed architectures in literature, they lack the needs of big data era in which there are a lot of social networks, blogs, and websites generating an immense amount of data. Different from the works in the literature that tackle this problem, a new approach can

be utilized, with the help of emerging stream processing engines providing distributed processing and Complex Event Processing capabilities. Context awareness of current solutions cannot meet user expectations. Notification systems should consider more contextual information to be used as part of big data solutions. Users should be able to define complex filters when setting up notification preferences. In the next section, we propose a new contextual framework to introduce a more comprehensive notification system in the big data context to enable people to get relevant information in the proper context.

CHAPTER 3

SYSTEM ARCHITECTURE

In this section, we explain the system architecture of the proposed framework. We go over the roles of each component in the architecture in detail. The architecture is depicted in Figure 3.

The architecture of the proposed framework consists of the following components:

- URL Sources: Twitter, Browser Bookmarklet, and Web application enable users to feed their URLs to the framework.
- Cloud Application Server (CAS): CAS manages User Management, Notification Management, Context Management and Job Scheduling.
- Database: It stores everything related to the URLs, notification preferences, context configurations.
- Mobile Application: It provides context related data to the framework and helps users to get notifications and see the results of the crawled pages as digests.
- Distributed Message Queue: It contributes to creating a data pipeline into the Distributed Processing Unit.
- Distributed Processing Unit: It manages Distributed Focused Crawler and Distributed CEP Engine.
- Message Delivery: It manages different notification channels like Email, SMS, and Mobile Push Notifications.

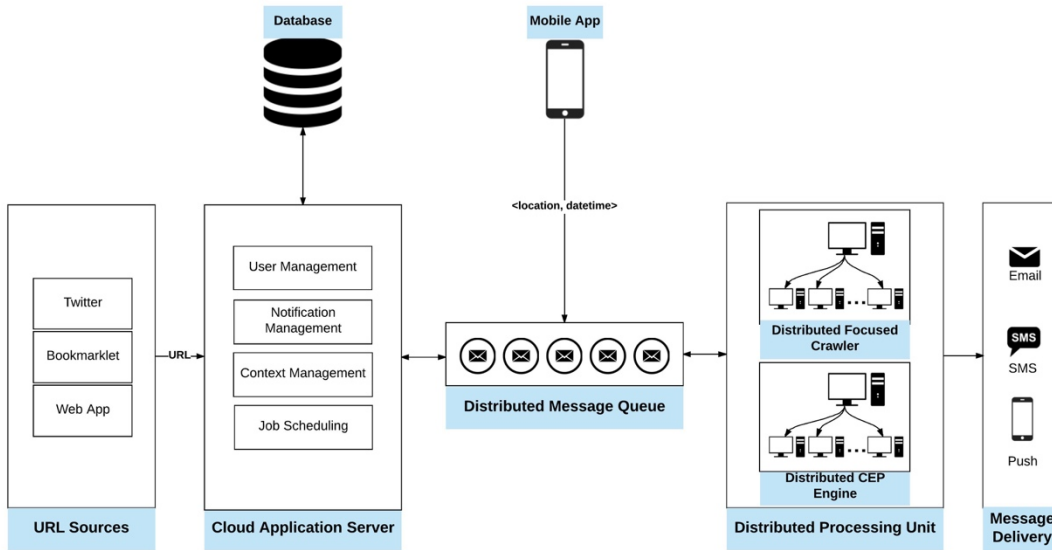


Figure 3. System Architecture

3.1. URL Sources

In order to create a list of seed URLs for each user, the framework needs to be fed with initial URLs to crawl. The seed URLs constitute the starting point for the focused crawling process. There are several ways to feed URLs to the framework. URLs can be added using the Web Application and Browser Bookmarklet by the users. Additionally, URLs can be imported from users' social media accounts such as their Twitter accounts in which they most probably tweet or retweet web pages they read or liked and want to be updated about the changes in them.

3.2. Cloud Application Server

Cloud Application Server is one of the essential components of the framework. It has several roles in the framework including User Management, Notification Management, Context Management and Job Scheduling.

User Management: To use the framework, users need to create an account within the system. Users can create an account via authenticating with their Twitter accounts and must provide an email to complete the account creation. Twitter authentication and email are also used for notification.

Notification Management: There are several ways users can be notified of the new content found on the websites they provide such as Email, SMS, and Mobile Push Notifications. Users can define their notification preferences within the Notification Management.

Context Management: Notifications are sent depending on the particular context that users are currently in. Users can set various context preferences depending on the time and location. For instance, they may choose to get notifications when they are within the specified range of a location and specific time interval. The Context Management administers all these context preferences.

Job Scheduling: There are various jobs which need to be scheduled in the framework. After the URLs are fed into the system, they need to be crawled with different time periods and users can define different time periods for each specific URL. Depending on the preference for the URL, they are pipelined to the Distributed Message Queue for initiating the crawling inside the Distributed Crawler. Additionally, notifications are sent depending on the users' choice, and notification jobs are also scheduled via the Job Scheduling.

3.3. Database

Seed URLs coming from the URL Sources, user preferences related to the context and notification choices of users are stored in the Database. It also helps to preserve the state of some processes within the framework as some of the triggers require the completion of other jobs. For instance, in order for notifications to be initiated, there need to be pages found according to the user queries as a result of the crawling process. Notifications are sent only after those conditions are met. These kinds of crucial states are preserved in the Database.

3.4. Mobile Application

In order to create a context aware system, we need to acquire the context of the users. Mobile Application mainly serves as a source of context data. There are two fundamental types of data we collect from the Mobile Application; the first one is GPS data like latitude and longitude and the second one is the date and time of the mobile phone indicating the date and time of the location where the user resides.

Additionally, Mobile Application also functions as a notification channel in which users can get push notifications on their phones via our Mobile Application. Push notification is a crucial feature along with the Email and SMS as they enable the framework to deliver the content even when they are on the road without requiring them to change their state and comfort. Moreover, users can quickly read the content they get from the Mobile Application as well.

3.5. Distributed Message Queue

Distributed Message Queue creates a pipeline between the data sources and the Distributed Processing Unit. In order to realize the framework, it is important to have a loosely coupled, fault tolerant and scalable architecture. Distributed Message Queue

enables us to build such an architecture. It helps us create a loosely coupled architecture with no direct connections between the sources and Distributed Processing Unit. As a result, this assures the development of individual components such as Mobile Application and Distributed Processing Unit independently, by only following the same interface requirements.

Additionally, in this framework, Distributed Message Queue works as a job queue such that the list of seed URLs and context data is moved to the Distributed Processing Unit to be handled. In this regard, it is crucial not to lose those data before it is processed. Distributed Message Queue provides that fault tolerance by ensuring the multiple replications of the partitioned data over several servers and having failover mechanisms in which when the master server is down, one of the other servers can easily replace the master server without any downtime.

Furthermore, the velocity of data movement into the Distributed Processing Unit is critical regarding the system performance as any delay inside the pipeline will cause the Distributed Processing Unit to underperform and we do not want the data pipeline to be a bottleneck for the data processing. Distributed Message Queue provides a distributed architecture itself to ensure the scalability requirements when the data load is increased.

3.6. Distributed Processing Unit

Distributed Processing Unit has two main sub modules. The first one is Distributed Focused Crawler and the second one is Distributed CEP Engine. As the framework may need to crawl millions of web pages and detect various complex user-defined context patterns for many users and need to deliver notifications in a timely context aware fashion, it is necessary to have a scalable processing unit. If we are to consider the velocity of context data and the requirements to process it, the Distributed Processing Unit needs not only to be distributed but also be able to handle stream processing. Stream processing is a processing technique where the data is processed record by record with very little latency. Therefore, Distributed Processing Unit needs to handle the processing in a distributed fashion with as little as possible low latency.

3.6.1. Distributed Focused Crawler

Distributed Focused Crawler crawls the URLs which are submitted by users or gathered from Twitter. In order to make the crawling process as fast as possible, a distributed infrastructure is needed. The crawler only crawls the URLs pipelined from the Distributed Message Queue, and it goes down to the first depth of the homepage as we are only interested in the freshest content and it also enables us to avoid the dangers that traditional crawlers encounter like spider traps. Spider traps are the infinite loops of links following each other but leading to no useful information [23]. Moreover, in order to create a more efficient crawling process and to be cost effective for the websites we are crawling, we have designed a crawling methodology by

grouping the websites for each different user. Hence, we only download the web pages once for all users submitted it and check each keyword they provided. As a result, we increase the performance of our crawling and be politer when downloading web pages. Additionally, Distributed Focused Crawler also checks for the user queries defined for the URLs. We do not store the content we have crawled, but in memory, we look for the keyword matches. If we can find matches, we send the results to the Distributed Message Queue to be stored in the Database.

3.6.2. Distributed CEP Engine

Other than Distributed Focused Crawler, Distributed CEP Engine resides inside the Distributed Processing Unit, and it is responsible for detecting patterns on streaming user context data which flows to the Distributed Message Queue continuously from the Mobile Application. CEP [41] [42] is the technique of filtering and analysis of the streaming data coming from multiple sources to detect and react to the context promptly. In this framework, it processes the context data to check whether the conditions are met for specific user's notification preferences. If an event is detected for any user, the result is sent to the Distributed Message Queue to be saved into the Database.

3.7. Message Delivery Unit

Depending on the outcomes of the crawling processes, users' context and notification preferences, it is important to deliver the content to the user in a timely manner. Message Delivery component manages the notifications as Email, SMS and Mobile Push notifications. Inside the Message Delivery Unit, some agents periodically look for newly found contents and context matches for particular content sources and users to initiate the notification delivery.

CHAPTER 4

PROTOTYPE IMPLEMENTATION

In this section, first of all, we present the essential technologies we have used, then we describe the implementation details of how we build the prototype, and we present three sample use cases that can be realized with this implementation.

4.1. Used Technologies

4.1.1. Apache Flink

Apache Flink [24] is an open source general purpose processing engine. Its architecture is based on Kappa Architecture. It supports both batch processing and stream processing. Even though its kernel is based on stream processing, Apache Flink represents the batch processing of data as a finite sequence of flows and runs batch applications on a streaming system. Additionally, it also supports Graph Processing and Interactive Processing.

Apache Flink is used by some of the leading companies such as Zalando, King, and Alibaba for different purposes [25]. Zalando uses it for real-time process monitoring [26]. King uses it for real-time analytics [27]. Alibaba uses it for optimizing real time search rankings [28].

Important concepts to consider for Apache Flink are Data Processing Abstraction, DataFlows, Windows, Checkpoints, and Batch on Streaming.

There are different levels of abstraction for developing applications with Apache Flink. As depicted in Figure 4, these are Stateful Stream Processing, DataStream / DataSet API, Table API, and SQL.

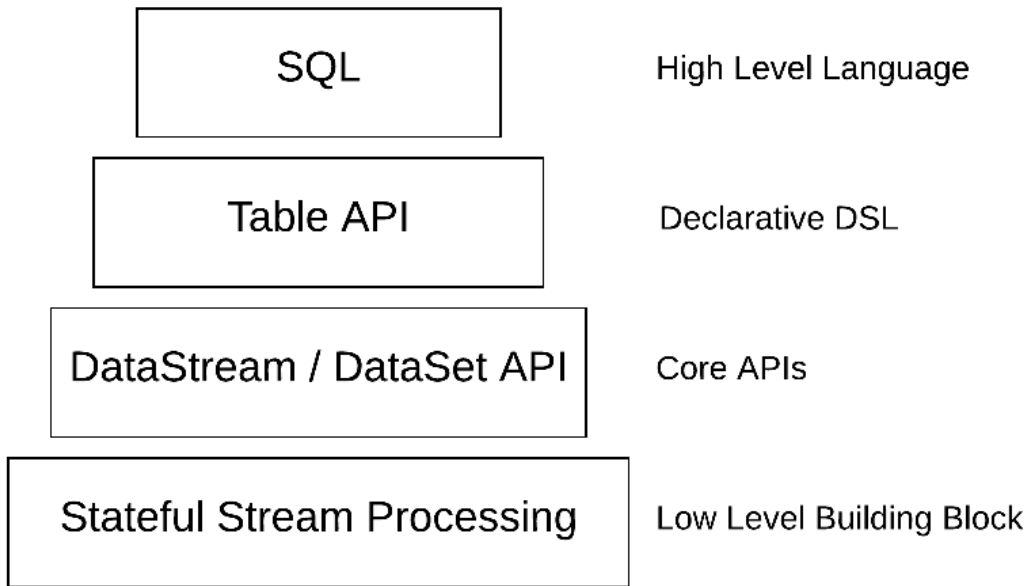


Figure 4. Apache Flink Levels of Abstraction

Stateful Stream Processing is the lowest level of the building block in the abstraction which is also provided by the DataStream API. It enables to process events from a stream in a consistent fault tolerant way. On top of that, DataStream and DataSet APIs, which constitute the Core APIs, are more developer friendly ways to create applications with Apache Flink. DataStream API can be used to process bounded and unbounded streams. DataSet API is used for processing bounded data sets. These APIs provide ways to process data in a diverse set of forms such as joins, transformations, aggregations, state, windows and so on. DataSet API also provides additional functions like iterations and loops. The Table API provides functions like select, project, join, group by and aggregate based on DSL (Domain Specific Language) over dynamically changing tables, which have schemes like in the relational databases. Table API can be used along with the DataStream and DataSet APIs. Lastly, Apache Flink also supports SQL (Structured Query Language) to interact with the data. Very similar to the Table API, SQL queries can also interact with the tables defined in the Table API.

Other than these abstraction levels, Apache Flink also provides additional APIs and libraries for different purposes. In fact, full Apache Flink Stack can be seen in Figure 5 [29].

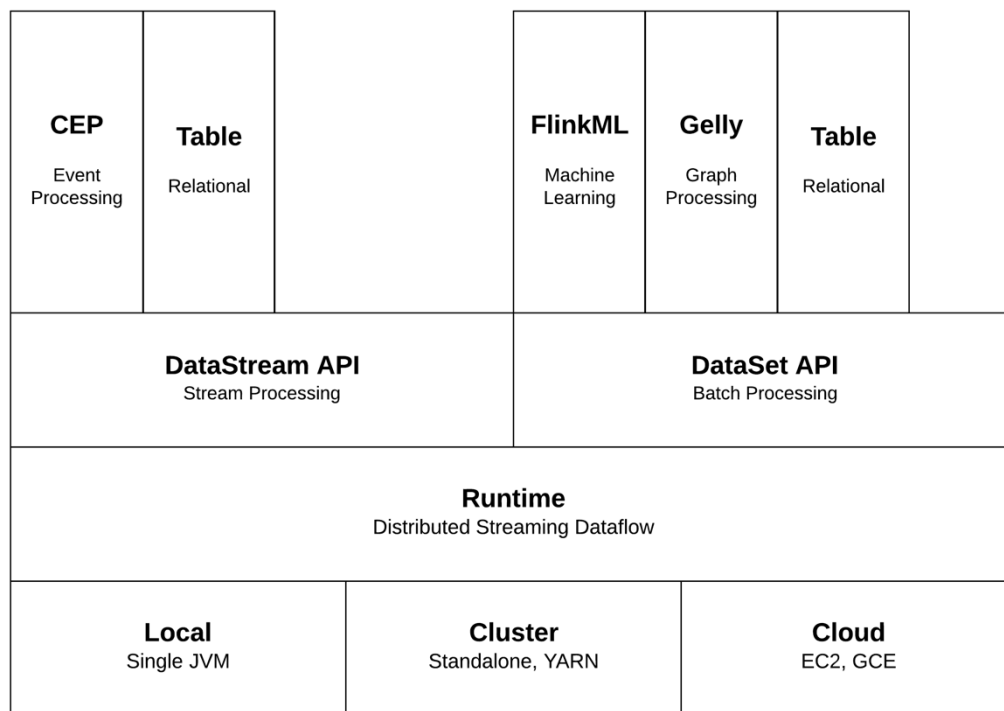


Figure 5. Apache Flink Stack

FlinkML provides implementations of commonly used machine learning algorithms and utilities. CEP library also comes with Apache Flink, allows to detect complex event patterns in a stream of continuous data. Additionally, Gelly library, provides graph processing functionalities to Apache Flink with operations to transform, process, and create graphs.

Moreover, Apache Flink can be deployed in different environments. It runs on the local machine inside the JVM (Java Virtual Machine), it can run on a standalone cluster or cluster managed by YARN (Yet Another Resource Negotiator) and Mesos. Also, it can be deployed on public cloud providers such as AWS (Amazon Web Services) via EC2 (Elastic Compute Cloud) and Google Compute Engine via GCE (Google Compute Engine).

Basic Apache Flink program is depicted in Figure 6 [30]. Data Source represents the incoming data processed by the Apache Flink and Data Sink represents the location where the processed data are sent. In the middle, Transformations represents how the incoming data altered within the Apache Flink. There are different kinds of transformations including but not limited to filtering, grouping, joining, mapping and updating state.

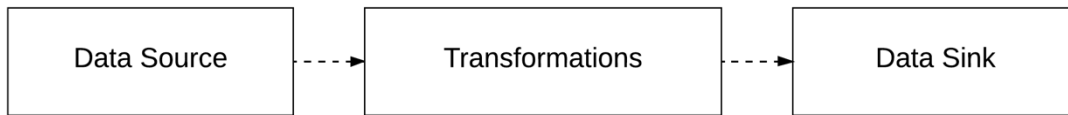


Figure 6. Apache Flink Program Flow

In order to increase the data processing efficiency, Apache Flink takes advantage of technique, that is called Lazy Evaluation. Fundamentally, transformation operations within the Apache Flink program are not executed immediately rather they wait for the actions. Each Apache Flink application is represented by a DAG (Directed Acyclic Graph), which enables to create data flows inside the application.

The architecture of Apache Flink is composed of Client, Job Manager, and Task Manager. The architecture is depicted in Figure 7.

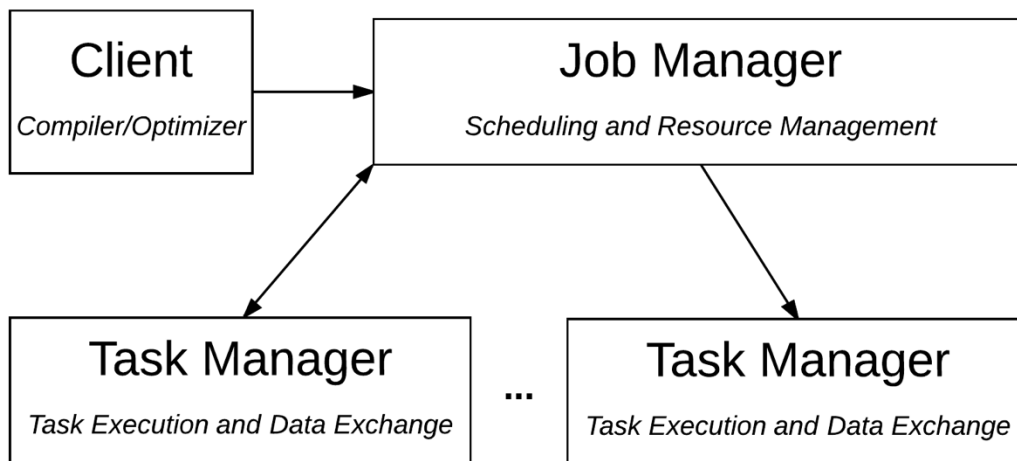


Figure 7. Apache Flink Architecture

The Client is not part of the program execution, but it serves as a way to create and send a data flow to the Job Manager. Job Manager orchestrates the execution, schedule tasks, handles failovers and so on. It can be associated with the Nimbus in Storm Framework. There can be more than one Job Manager to increase the uptime of an Apache Flink application. Task Managers are the ones that process the data and exchange data with each other.

There are also some alternatives to Apache Flink. Apache Spark [31] and Apache Storm [32] are two notable alternatives. However, there exist some deficiencies of them with respect to Apache Flink for our implementation requirements.

First and foremost, Apache Spark is not suitable for native stream processing, but it tries to do it by utilizing micro-batch processing. Additionally, Apache Spark does not have a built-in CEP library which is a crucial requirement for our implementation.

Apache Storm is a stream processing engine, but it also lacks some libraries that already come with Apache Flink. For instance, Apache Storm does not support built-in complex event processing, even though there are some additional libraries like Esper [33] and Drools [34] that can be used with Apache Storm to support the CEP processing, it is still not as easy as using Flink CEP.

4.1.2. Apache Kafka

Apache Kafka [35] is an open source distributed streaming platform based on publish-subscribe data delivery model. It can be used as a distributed message queue. It runs as a cluster of one or more servers and each node called as brokers. Brokers store data published to the topics. Each topic corresponds to a stream of records where each record is a $\langle key, value, timestamp \rangle$ tuple. A topic is a category or feed name to which records are published. To achieve reliability, more than one brokers can store records on each different topic. Additionally, each topic can have zero or more producers/consumers. Consumers subscribe to one or more topics and read records published to those topics at their speed, scale, and convenience. Data is pulled from the broker by the consumers. Each topic in Apache Kafka is partitioned which allows scaling the data across multiple servers and acts as the parallelism unit for Apache Kafka. Each partition has a single leader and zero or more followers. Replication factor represents the total number of replicas including the leader. Typically, there are much more partitions than brokers, and the leaders are evenly distributed among brokers. All reads and writes happen more partitions than brokers and through the leader of the partition. Followers consume messages from the leader like a consumer does and apply them to their log. In the case of leader failure, one of the followers is automatically selected as the new leader.

Additionally, messages within Apache Kafka are written to the file system to persist the data and recover from any failure. The anatomy of a topic can be seen in Figure 8 [36]. The partitions are distributed over the servers and replicated across the configurable number of servers for fault tolerance. The producer chooses which record to assign which partition within the topic. This can be done in a round-robin fashion to balance load or according to some partition function. Each consumer labels itself with a consumer group name and each record published to a topic is delivered to one consumer within each group subscribed to that topic. Kafka handles dividing up the partitions over the consumer instances so that there is a fair share of partitions at any time.

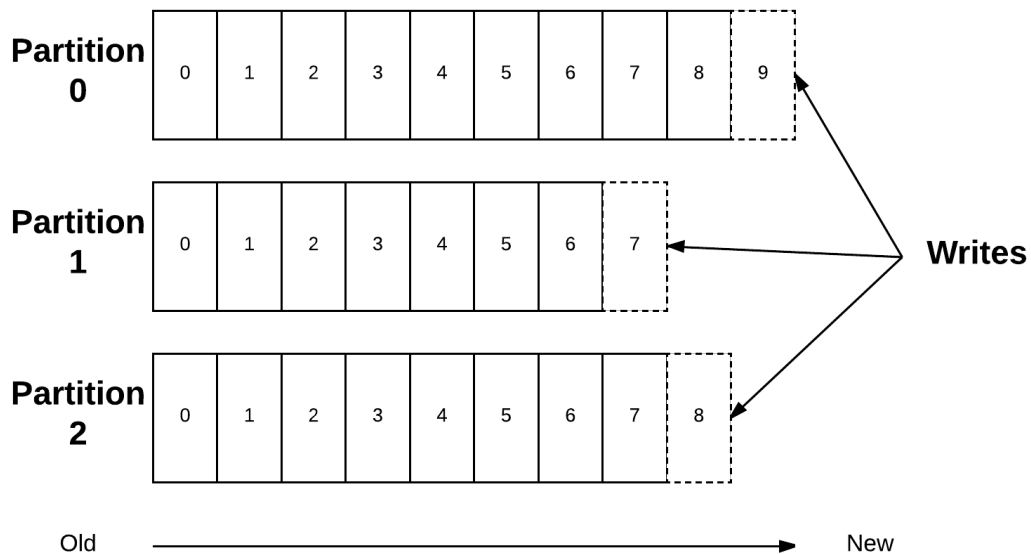


Figure 8. Anatomy of a Topic

Apache Kafka serves as a scalable and fault tolerant point of contact between all producers and consumers of data. It is suitable for real-time streaming applications to reliably get, transform, react to, and store streams of data. Every server node in the Apache Kafka cluster maintains its session with Apache Zookeeper through heartbeat mechanism.

In summary, Apache Kafka provides following guarantees:

- Messages are appended according to the order of arrival to the topic.
- Consumers also process messages according to the order of appearance.
- Fault tolerance by partitioning data across multiple machines.
- Within a cluster of N servers, data can be recovered up to N-1 server failures.

4.1.3. Apache Zookeeper

Apache Zookeeper [37] is a high-performance coordination service for distributed applications. It enables to build applications by providing a set of services such as Name Service, Locking and Synchronization, Configuration Management, and Leader Election. Internals of Apache Zookeeper is depicted in Figure 9.

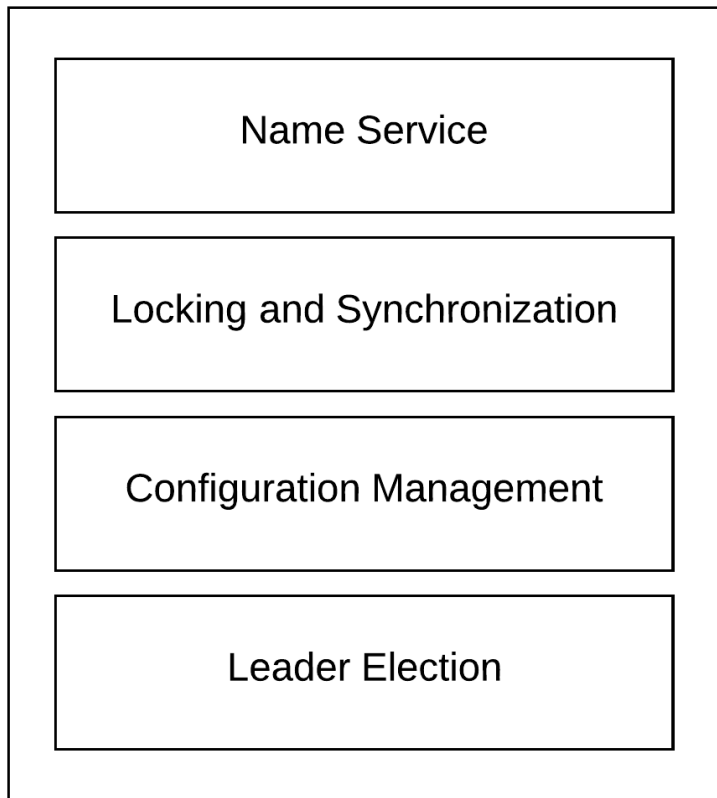


Figure 9. Apache Zookeeper Internals

Name Services provides the name and availability of server nodes and services within the cluster. Locking and Synchronization manage the efficient sharing of resources on the cluster. Configuration Management is the central location for all configuration related information for all server nodes in a cluster. In order to handle failures and recovery, Leader Election manages the leader election.

4.1.4. Docker

Docker [38] is a lightweight virtualization solution that provides container-based virtualization. It is written in Go Language, and it is open source. The architecture of Docker is depicted in Figure 10 [39].

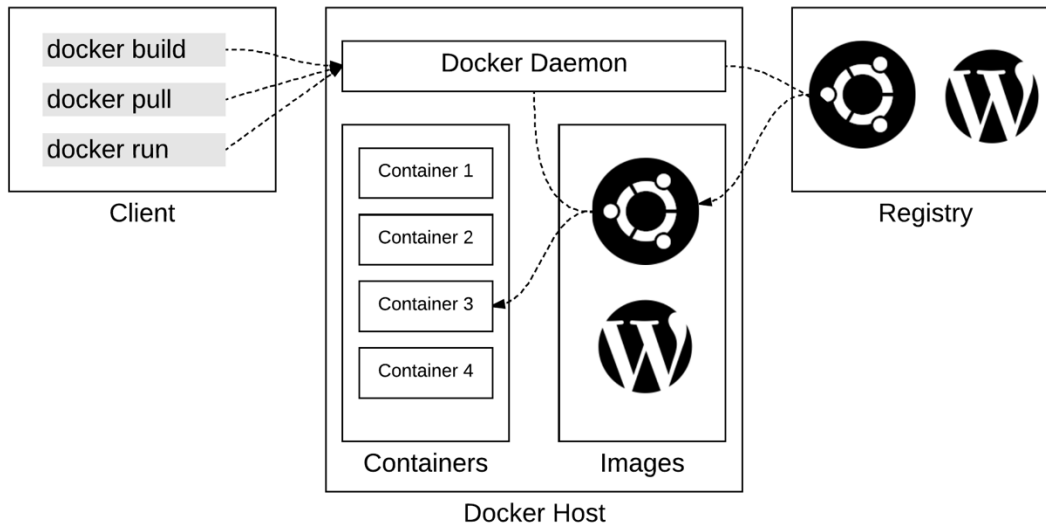


Figure 10. Architecture of Docker

The architecture of Docker is a client-server architecture in which the Docker client (*docker*) talks to the Docker daemon (*dockerd*) to build, run, and distribute the Docker containers. Docker daemon and client can run on the same machine, or it is possible for Docker client to connect a remote Docker daemon. The communication between the Docker daemon and client can be established over different channels such as using the REST API or over the network interface or UNIX sockets.

Docker client is the fundamental entry point for users to interact with Docker functionalities. When the users send commands to the *docker*, the commands are sent to the *dockerd* to be executed. Docker client can work with multiple Docker daemons over the Docker API.

Docker daemon manages Docker objects such as containers, images, volumes, and networks by listening to the Docker API requests. It also operates Docker services by communicating with other Docker daemons.

One of the other important Docker concepts is the Docker Registries. The main functionality of the Docker Registry is to store Docker images. By default, Docker looks for images on Docker Hub [40], but it is also possible to configure a private registry to enable Docker to search for images on that registry. When the *docker pull* command is executed, specified images are pulled from the configured Docker Registry, and when the *docker push* command is executed, the image created or modified on the local machine pushed to the configured Docker Registry.

Using Docker requires creating and using Docker objects such as containers, images, volumes, plugins and other objects.

Docker containers are runnable instances of Docker images. Using Docker API or CLI, containers can be created, started, stopped, moved or deleted. It is also possible to create new images out of containers after adding new layers to it. It is also possible to connect containers to multiple networks or attach storage to it. Containers are isolated from the host server and other containers. However, you can still configure how isolated the network and storage from other containers and the host server. Docker containers are defined by their images and any additional configuration options provided before it is created or started. One important thing to notice is that when containers are stopped, any modifications to their state disappear and if there is any state to preserve, they need to be persisted in the storage.

Docker images are read-only templates providing instructions for creating Docker containers. Images can be built upon other images with additional customizations. For instance, it is possible to create an Apache Flink image after installing Apache Flink on top of an Ubuntu image. It is possible to pull and use images from Docker registries, additionally, by creating a *Dockerfile*, which defines steps needed to build a Docker image, Docker images can be constructed from scratch. Every step in the *Dockerfile* forms a layer in the image and when the *Dockerfile* is updated only the updated layers are rebuilt. Being able to only rebuilt the updated layers is one of the prominent features of Docker that brings about lightweight, small and fast images different than other virtualization alternatives.

Docker Services provides scalability of Docker containers across different Docker daemons. Services enable to define the desired state of the containers. For instance, the number of replications of a service at any given time across multiple daemons determined via Docker Services. Moreover, Docker Services are also load-balanced across all workers, and it abstracts the complexities of distributed nature of the applications to the users.

Docker uses several functionalities of Linux kernel to provide these technologies. Some of the important underlying technologies are Namespaces, Control Groups, and Union File Systems. Namespaces provide isolation to the containers. When containers are created a number of namespaces are constructed for each container. Docker uses following namespaces:

- pid: For process isolation
- net: To manage network interfaces
- ipc: To maintain access to IPC (Inter Process Communication) resources
- mnt: To manage filesystem mount points
- uts: To isolate kernel and version identifies

Control Groups (cgroup) allows Docker to limit the resources that a container can utilize. For instance, CPU or memory usage can be set and limited for containers thanks to cgroup.

Union File Systems (UnionFS) allows Docker to function by creating layers on containers. It is one of the essential building blocks of containers providing fast and lightweight virtualization.

Combining Namespaces, Control Groups, and Union File Systems, Docker creates the containers.

Additionally, there are also useful tools that come with Docker. These tools enrich the features and benefits of Docker. Some of the crucial ones are Docker Compose and Docker Swarm.

- Docker Compose: It enables to define multi-container applications by setting up multiple services on different containers and managing the network configurations.
- Docker Swarm: It is a clustering system for Docker. It enables to deploy containers on multiple physical machines by abstracting multiple host machines into a single host.

4.2. Implementation

The implementation follows the described system architecture in Figure 3 and in order to implement the prototype, we have utilized various open source technologies. We have used Apache Flink, a general-purpose data processing engine, to implement the Distributed Processing Unit. For Distributed Message Queue, we have utilized Apache Kafka. Additionally, for the implementation of the web application, we have used Codeigniter, a web application development framework for PHP. To implement the mobile application, we have utilized Ionic Framework, a framework for developing cross-platform mobile applications. Lastly, for the storage, we have used MySQL database. The software stack we have employed is depicted in Figure 11.

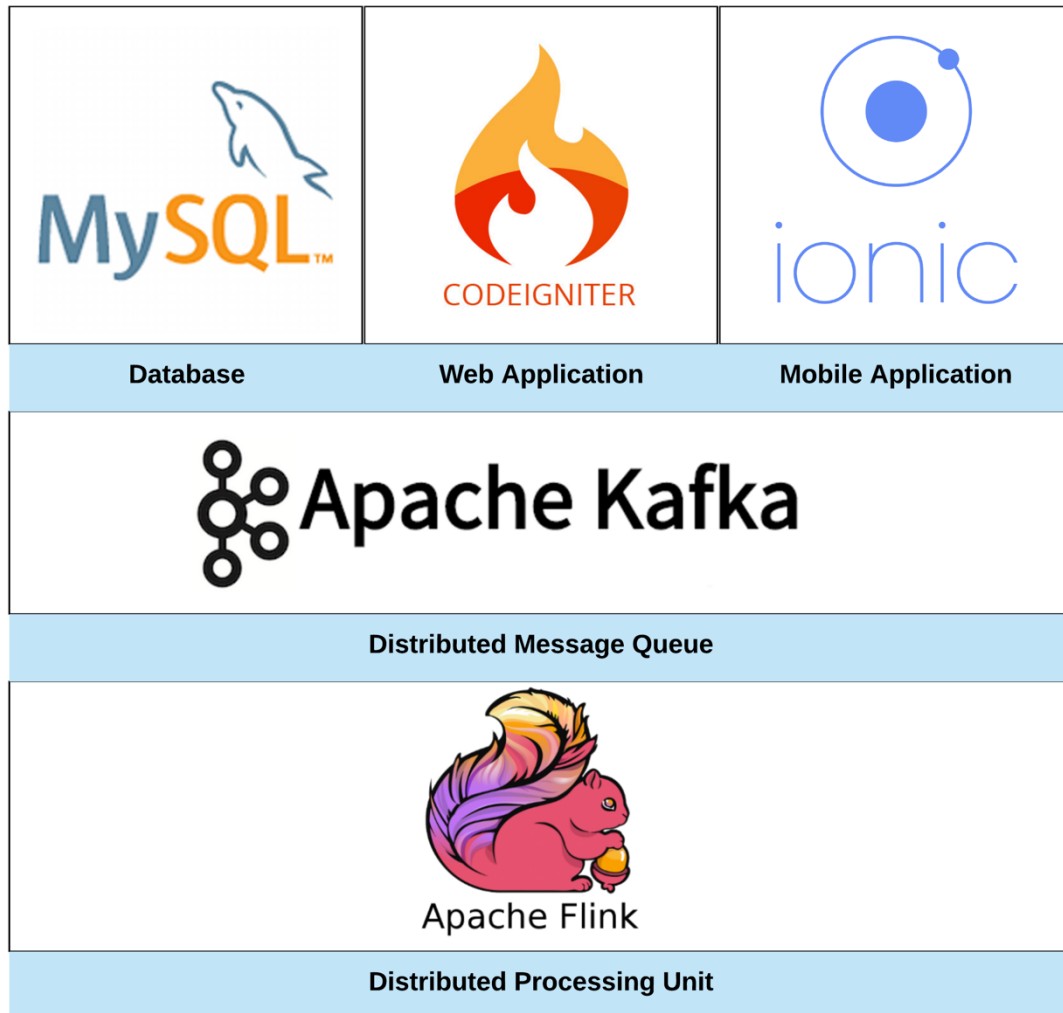


Figure 11. Prototype Software Stack

Next, we explain the details of our implementation and how we have used these open source technologies to construct our prototype.

4.2.1. Mobile Application

The mobile application is the most important part of this framework to gather context data about the users. The application sends time and location of the users and allows them to get context aware digest notifications to read the recently found new pages according to their preferences.

In order to create a secure mobile experience, users need to log in the application through the login screen, which can be seen in Figure 12 using email and password that they created via the web application.

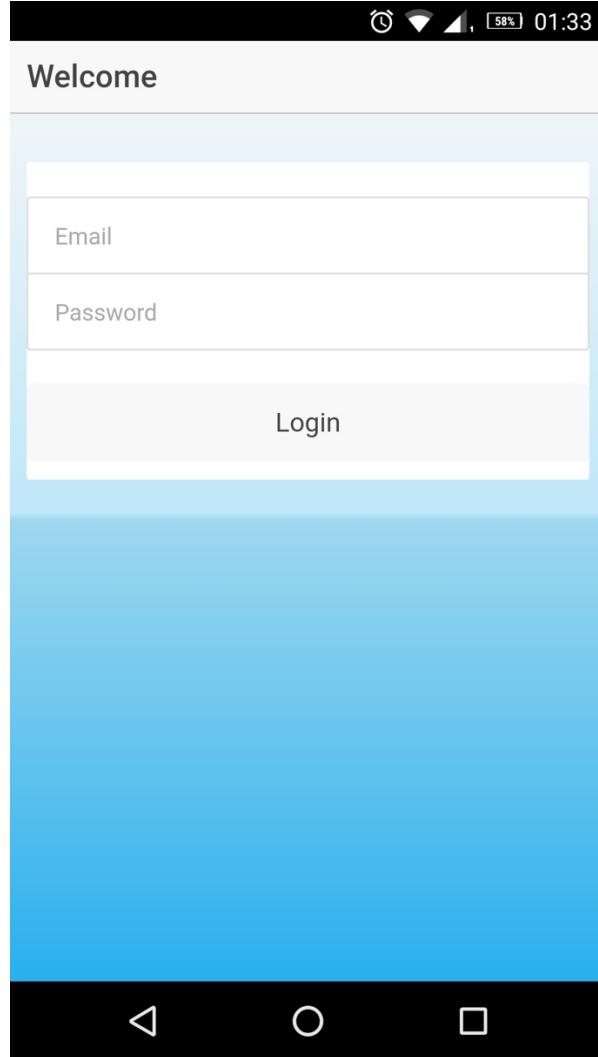


Figure 12. Mobile Application Login Screen

The application also provides time and location in the background to the distributed messaging queue to be piped into the distributed processing unit. In Figure 13 below, the location of the user can be seen in the application. For the test purposes, the current location can also be sent with the user action.

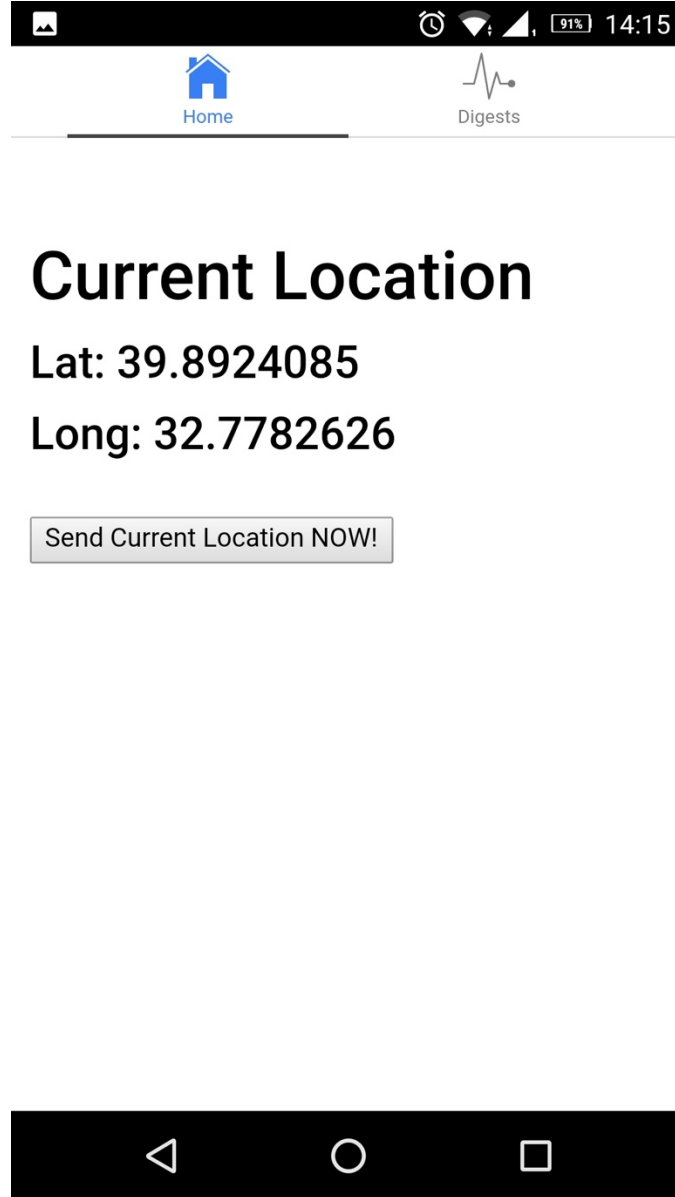


Figure 13. Mobile Application Location Screen

Additionally, when the user queries and context pre-conditions are met, the users get push notifications to their mobile phones over the application. The example of the push notification received can be seen in Figure 14.

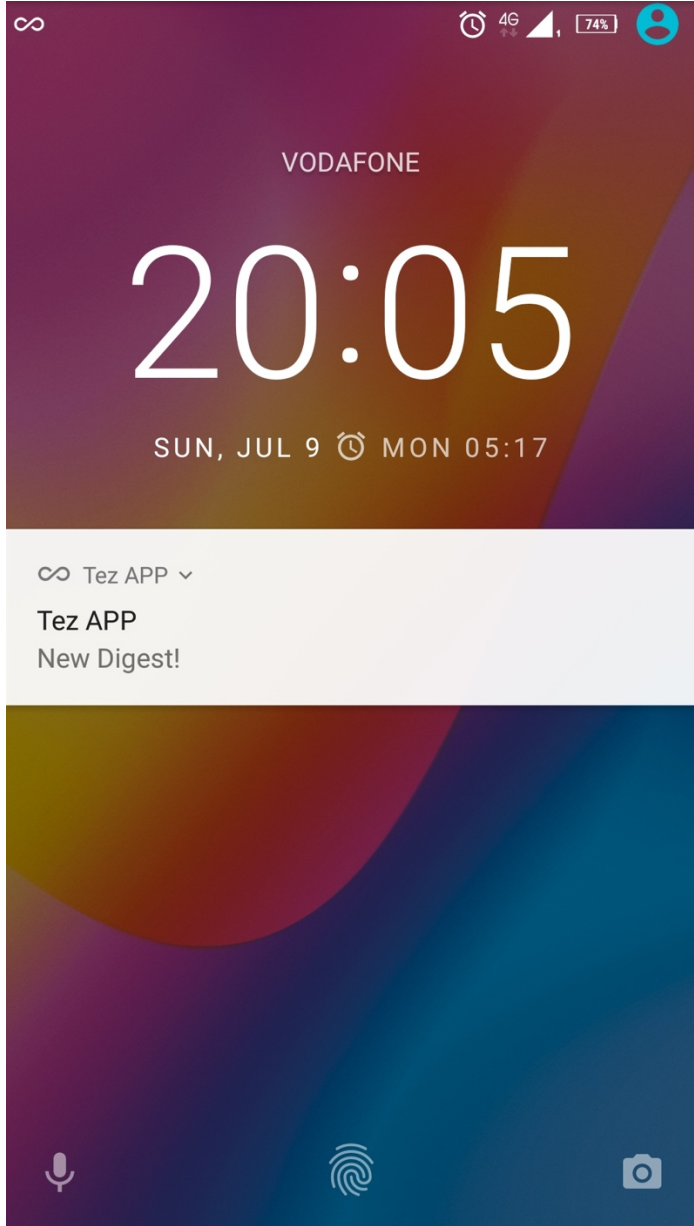


Figure 14. Mobile Push Notification

After the notification is received, users can check digest of new contents and read the pages over the application. The digests are also shown in the application, and the digest page can be seen in Figure 15.

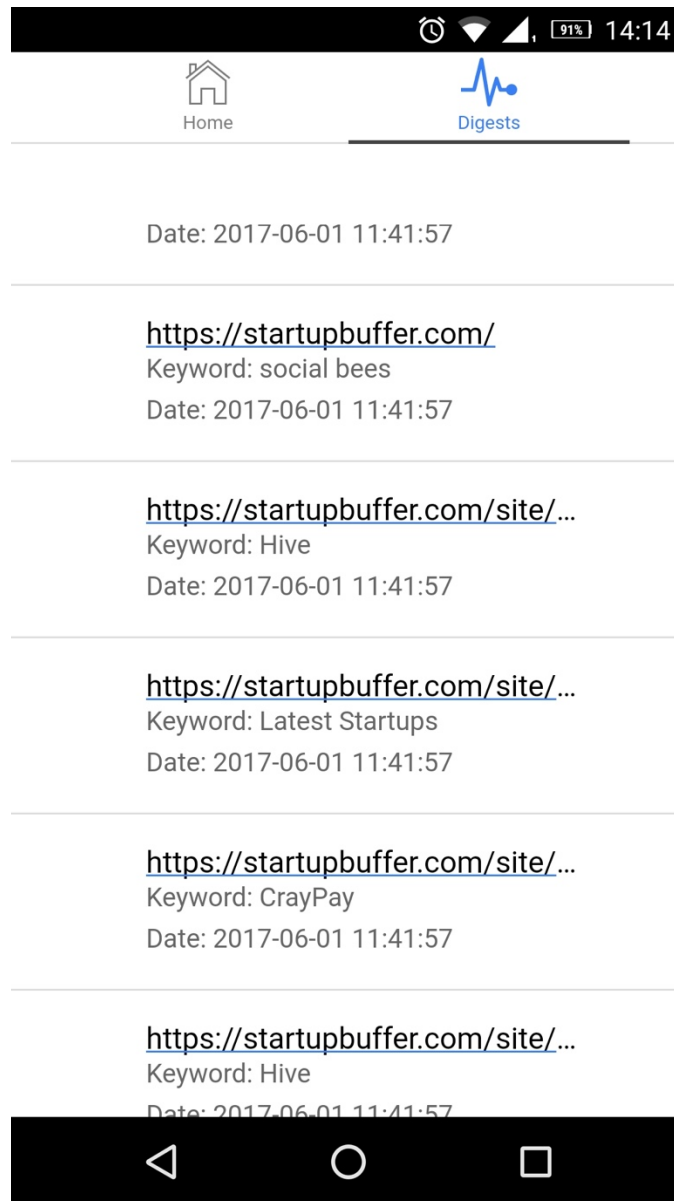


Figure 15. Mobile Application Digests Screen

4.2.2. Bookmarklet

The focused crawling system requires a list of seed URLs to be crawled and in order to make it easier for the users to add new links, we have built a simple browser bookmarklet. Users can drag and drop the bookmarklet from our website into their bookmark lists. Then, when they are browsing the Web and want to keep following the website with the system, they can just click the bookmarklet. We get the current URL and redirect them to our web application to easily save the URL into our system. We have used JavaScript programming language to implement the browser bookmarklet and basically what it does is that along with getting the URL from the

browser, redirects to our web application endpoint with the URL. The bookmarklet can be seen in Figure 16.

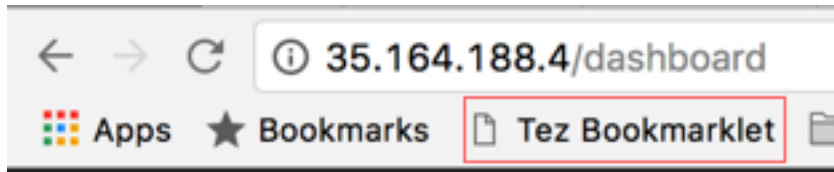


Figure 16. Bookmarklet

4.2.3. Web Application

Within our system, the web application has a significant role regarding users' interaction with the framework. Besides bookmarklet, users can also add URLs inside the web application as well. They can also configure their notification preferences, context aware alert conditions and check the recently found digests.

In order to use the web application, users need to create an account by first authenticating with Twitter and then providing email and password. Twitter authentication also allows us to get URLs from users' accounts and analyze them to create an initial interest list for the users. In Figure 17 and Figure 18 account creation process can be seen.

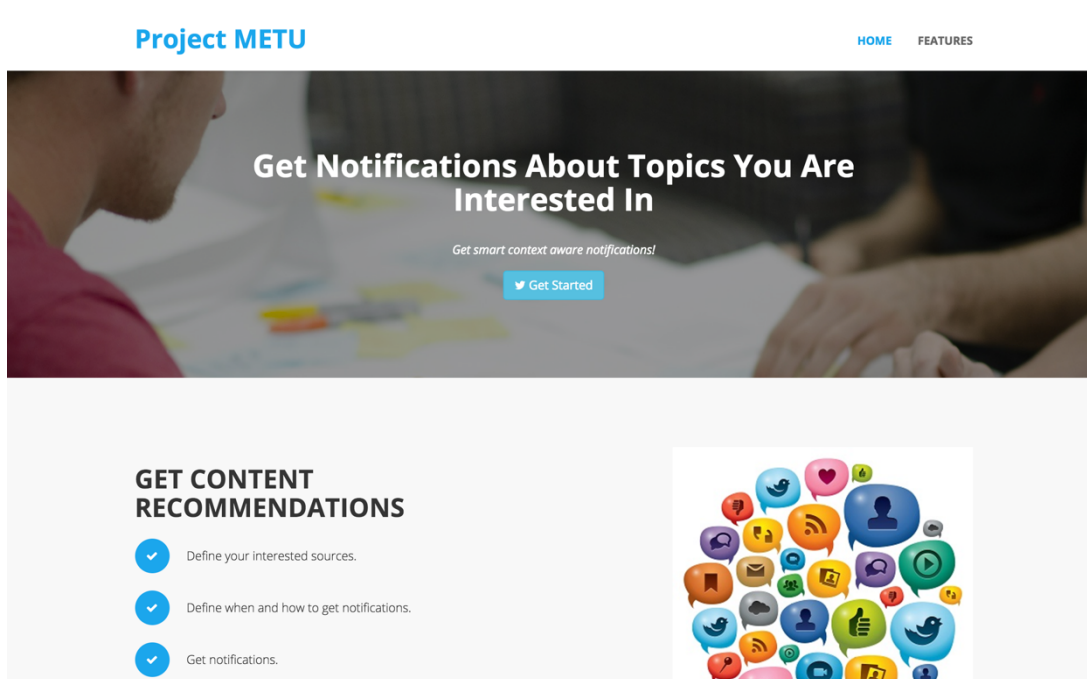
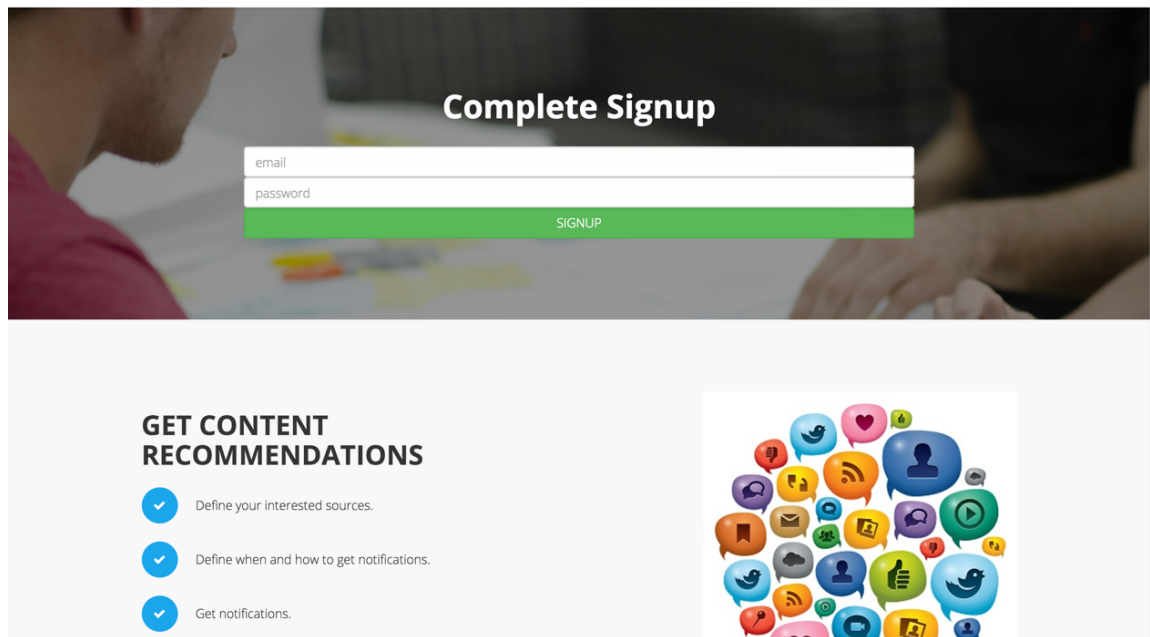


Figure 17. Web Application Twitter Authentication



Complete Signup

email

password

SIGNUP

GET CONTENT RECOMMENDATIONS

- ✓ Define your interested sources.
- ✓ Define when and how to get notifications.
- ✓ Get notifications.

Figure 18. Web Application Registration

After the registration, users can add or update URLs from the dashboard. They also define the keywords they want to make the system look for and they set a color-coded context configuration for each URL. The dashboard screen can be seen in Figure 19.

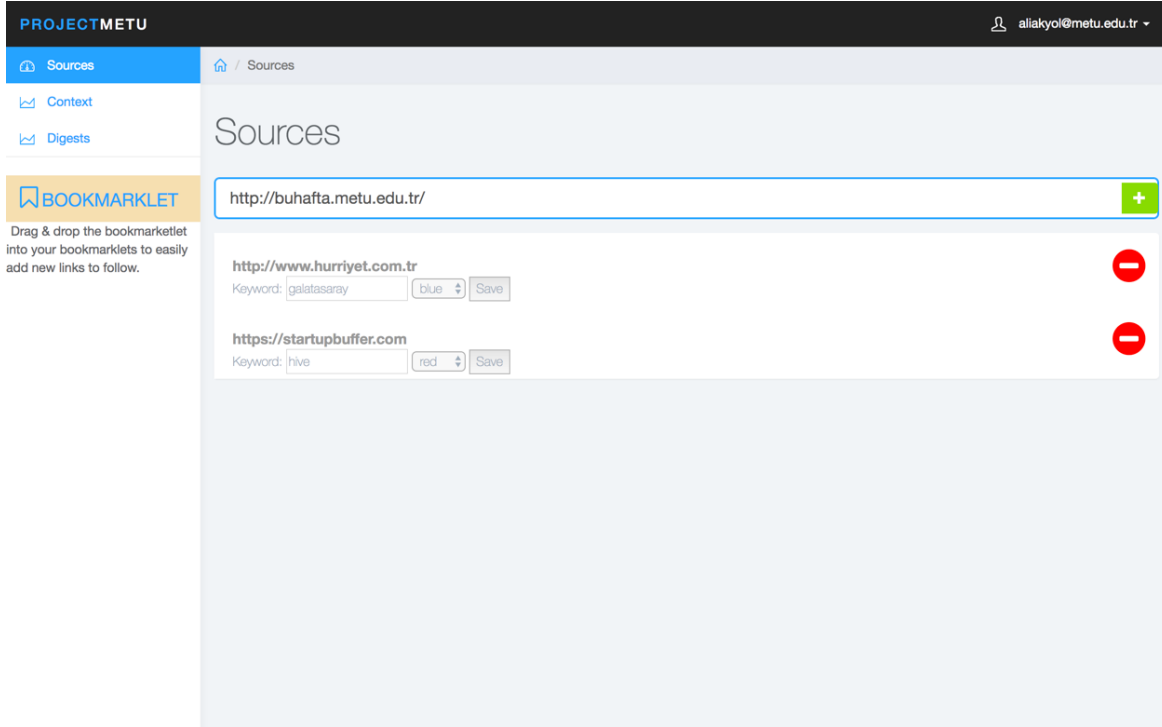


Figure 19. Web Application Source Page

Color-coded context preferences construct an abstraction for all the user preferences related to when and how they want to get notified, that is why it is an important concept for context aware notifications that are proposed by the framework. The context preferences page can be seen in Figure 20. These preferences not only specify the notification time, location and frequency but also crawling frequency for each URL.

For every crawler, it is important to be polite to the websites that are being crawled and respect the *robots.txt*, which is proposed by The Internet Engineering Task Force (IETF) in 1996 [41] to provide regulations for web crawlers to follow. The *robots.txt* is positioned in the root directory of the websites and provides policies regarding the which pages can be crawled or not [42]. We follow the *robots.txt* regulations for each website and allow users to set a crawling frequency of at most once in every 2 hours. Users can also adjust the frequency as *twice a day* and *daily*.

Users can also specify the notification day, time and location. They can specify a time range and set a notification delivery day of weekdays or weekends. While setting the time range, they can adjust the notification delivery whether they want to get notified between or outside of the time range. Also, by providing *latitude*, *longitude*, and *distance* in meters, they can specify the notification location whether they want to get the notification in the specified range from their home or work place.

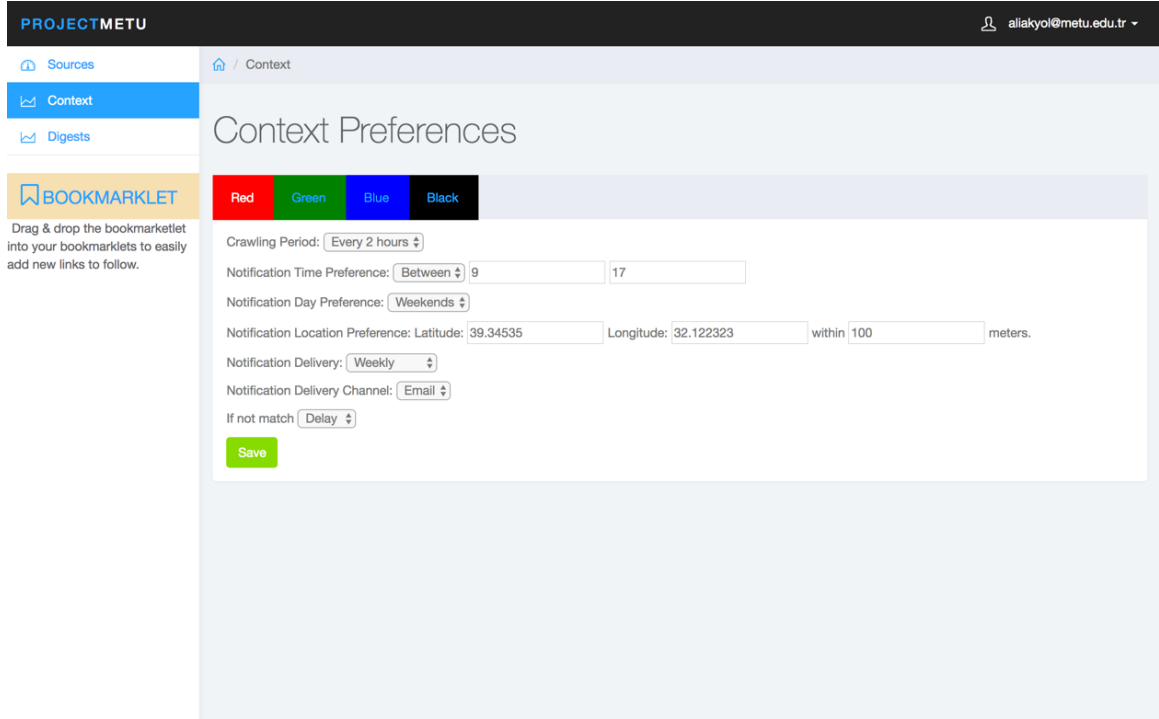


Figure 20. Web Application Context Configuration Page

Additionally, users can set the notification urgency on this page. As for some websites, they may want to get the recently found website immediately when the context conditions met, or they can set to get the notifications as *daily* or *weekly* digests.

Moreover, the color-coded abstraction also allows users to set the notification channels for each URL. The framework supports the following notification channels; email, SMS and mobile push notifications. In order to get the mobile push notifications, users need to install the mobile application.

Lastly, there are some cases where the recently found web pages can be obsolete, or they cannot carry a value for the users. For those cases, users can also specify whether they want to *ignore* the results or *delay* them in the case of context conditions not met for the first time.

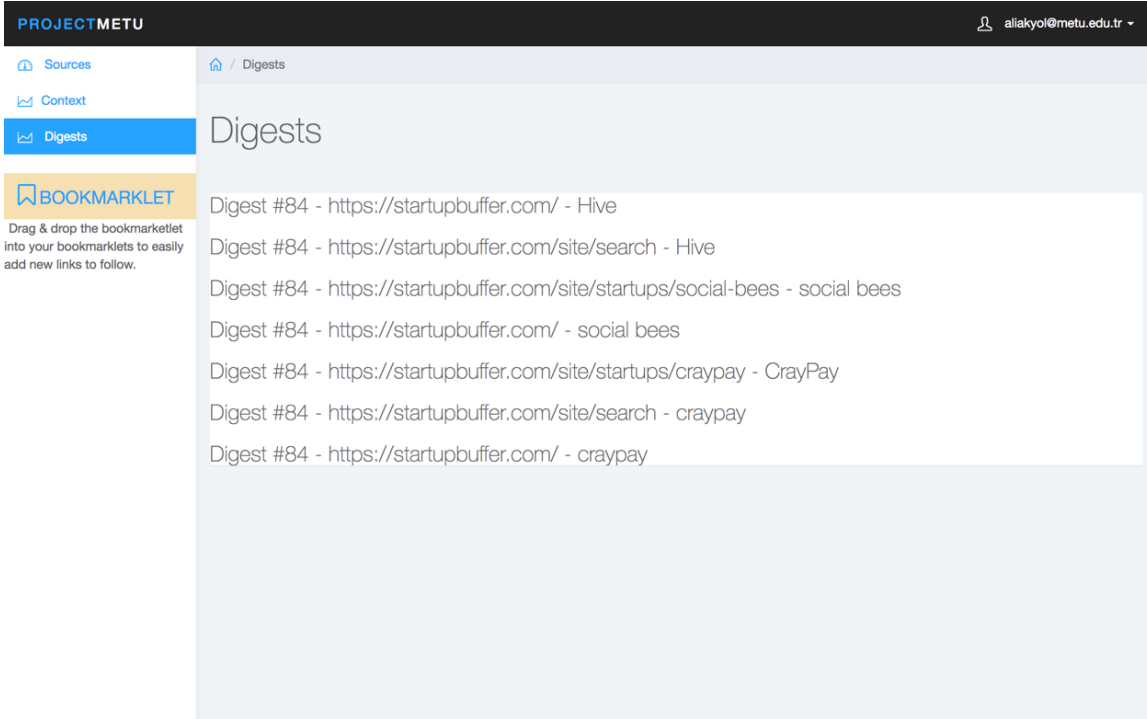


Figure 21. Web Application Digest Page

Lastly, the web application allows users to check the digests of found pages according to their keyword queries. The digests page can be seen in Figure 21.

The web application is deployed on the cloud using AWS EC2 which provides high availability and scalability. For test purposes, the EC2 instance that is used has the following configurations. However, the web application can be easily scaled both vertically and horizontally. AWS offers a range of virtual machines with higher specifications and thanks to tools like AWS Elastic Beanstalk [43], also provided by AWS, can be horizontally distributed.

Table 1. AWS EC2 Server Features

	Model	CPU	Memory	Storage
Features	c4.xlarge	4 Cores	7.5 GB	50 GB

Besides the deployment and scalability of the web application, its backend is implemented using PHP with Codeigniter framework [44]. The Codeigniter is a very lightweight and easy to use framework, following the MVC (Model-View-Controller) design pattern. For the frontend development, HTML, CSS, and JavaScript are used along with the Bootstrap frontend framework [45].

4.2.4. Distributed Processing Unit Implementation

The distributed processing unit of this framework provides two main functionalities. The first one is to make the focused crawling and the second one is to make the context processing.

The focused crawler crawls the URLs provided by the users. The URLs along with the desired keyword queries are fed into the crawler over a distributed messaging queue, in order to prevent any data loss by creating a persistent data pipeline. For the development of distributed focused crawler, Apache Flink is utilized along with the Crawler4J library, which is an open source web crawler library built with Java. We distributed the crawling tasks to task managers inside the Apache Flink cluster. The Apache Flink program flow is depicted in Figure 22.

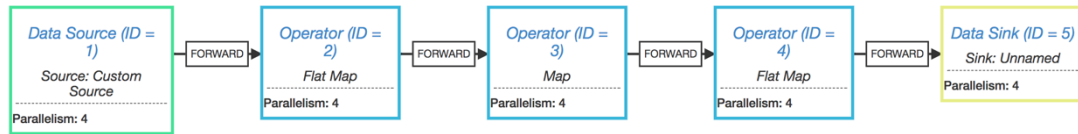


Figure 22. Apache Flink Crawler Execution Plan

Data source block consumes the data from the distributed messaging queue and forwards the URLs and keywords to the other blocks which firstly pre-process the URL and keywords and map them according to the URLs and checks the keyword matches. In the end, the pages matched with the user provided keywords are forwarded to the data sink, which fundamentally produces data for another distributed messaging queue topic.

The context processing unit is also built within the distributed processing unit by utilizing Apache Flink, specifically the Flink CEP library. The Flink CEP program flow is depicted in Figure 23.

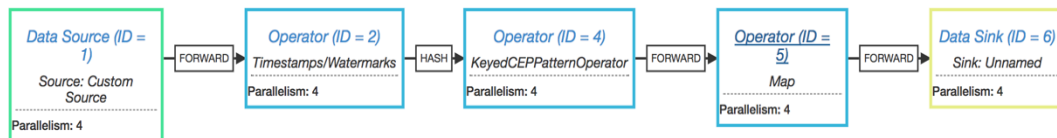


Figure 23. Apache Flink CEP Execution Plan

The data source block consumes the time and location data provided by the users' mobile application over the distributed messaging queue. After that, the data coming from the messaging queue pre-processed and streamed into the CEP operator where the user defined context patterns are checked. If any pattern matches, meaning the

context conditions are met, the result is directed to the data sink, which publishes the data to the distributed messaging queue.

Apache Flink is deployed by using Docker on a host machine with the following specifications.

Table 2. Host Machine Features

	CPU	Memory	Storage
Features	4 Cores	16 GB	30 GB

As discussed before, Docker is a container based lightweight virtualization technology that also makes it very easy to deploy applications on servers. Along with Docker, we have used Docker Compose to quickly create a virtual network between Apache Flink job manager and task managers.

Additionally, Docker deployment has allowed us to scale the number of task managers of Apache Flink quickly. We have tested various numbers of task managers and jobs managers within our virtual network. Docker has helped us to deploy and scale the Apache Flink application.

4.2.5. Distributed Message Queue Implementation

Distributed Message Queue is an integral part of data processing in this framework. Our aim is to build a fault tolerant, reliable and scalable message queue.

Among other alternatives such as Apache Flume [46], RabbitMQ [47], MQTT [48] and so on, we have used Apache Kafka for our implementation of distributed message queue. Among those alternatives to Apache Kafka, some of them are not designed to be used for scalable environments, some of them are not a good fit for our use cases, and some of them are not suitable for other development tools we are using.

Apache Kafka is based on the publish-subscribe data model, and there are topics that each component within our framework are subscribed to. For the distributed crawler, there is one topic for the input of URL and keywords and one output topic for the results of the matches. There are also agents that publish data from the database for the input and agents listening to the output for writing to the database.

Additionally, for the distributed complex event processing part, one topic for the input of location and time data and one output topic for the results of the context matches are used. Mobile application sends the users' location and time in the background to the input topic. There are also agents subscribed to the output topic for the context matches to write them into the database.

The agents are written in Nodejs [49], an event driven Javascript framework for the backend development and they are scheduled to always run in the background.

4.2.6. Storage

In order to store the seed URLs, user preferences, results of the processes and notification digests, we needed a database solution. We have used a relational database solution, MySQL for data storage as it is an open source database solution with a mature community support from numerous contributors.

4.3. Sample Use Cases

Here we present some of the use cases that can be realized with the described implementation above.

In general, the user needs to register via our web application and configure his/her preferences such as seed URLs, filter keywords, notification preferences and channels. Then the seed URLs along with the filter keywords are fed into the Distributed Processing Unit over Distributed Message Queue. Additionally, user location and time collected via the mobile application are sent to the Distributed Processing Unit over Distributed Message Queue. Distributed Processing Unit processes the data, and in the case of any content and context match, users can get notifications like email, SMS, and mobile push notifications.

4.3.1. Use Case 1

Mehmet is an entrepreneur from Ankara, Turkey. He has a company, its office located in Bilkent CyberPark. He wants to know when his company is mentioned in Startup Buffer website, which is a platform for startups, and read the information written about his business. He registers and logs in to the system. He adds <https://startupbuffer.com> URL and adds his company name, “ABC” as the keyword. Then he configures crawling and notification preferences. He sets crawling period as “Daily” as he wants to make the system crawl the Startup Buffer website and query his company “ABC” daily.

He also wants to get the notifications when he is in the office, the day is a weekday and the time is between the working hours. So, he enters the time range “between 9 o’clock and 17 o’clock”, and he selects “Weekdays” as the day preference. He also wants to get notified when he is about to arrive at his office and enters his office location with latitude and longitude along with a range parameter as “latitude: 39.8698192, longitude: 32.7432758 and 100 meters”.

Finally, he selects the notification channel as “email” as he wants to read the news about his company on his work computer when he arrives at his office. He configures

all the settings accordingly. Then, the system starts to crawl the <https://startupbuffer.com> daily and looks for the company name “ABC”, when there is any match, an email is sent to Mehmet when he arrives at his office.

4.3.2. Use Case 2

Seyma is interested in health food ideas. She always looks for new healthy recipes to try. Recently, she tries to find the recipes using “chia seeds” as the “chia seeds” are told wholesome. She always checks <http://www.nefisyemektarifleri.com/>, which is a recipe website, for new healthy recipes. So, in order not to check the website manually to find the recipes, she registers, and logs in to the system. She adds <http://www.nefisyemektarifleri.com/> URL and adds the ingredient name “chia” as she tries to find new recipes with “chia” as the keyword. Then she configures crawling and notification preferences. She sets crawling period as “Weekly” as she wants to make the system crawl the recipe website and query the ingredient “chia” weekly.

She also wants to get the notifications when she at home, the day is a weekend as she works on weekdays and the time is around the afternoon. So, she enters the time range “between 14 o’clock and 16 o’clock”, and she selects “Weekends” as the day preference. She also wants to get notified when she is at home. Thus, she enters his home location with latitude and longitude along with a range parameter as “latitude: 39.9686726, longitude: 32.6531147 and 50 meters”.

Finally, she selects the notification channel as “Push Notification” as she wants to get the notification on her mobile phone. She configures all the settings accordingly. Then, the system starts to crawl the <http://www.nefisyemektarifleri.com/> weekly and looks for the recipes including “chia” as the ingredient, when there is any match, a push notification is sent to her mobile and she checks the new recipes from the mobile application.

4.3.3. Use Case 3

Serhat is a research assistant at Middle East Technical University, and he always checks the cafeteria menu for his favorite meal for his lunch. His favorite meal is “Lamp Tandoori”, and he goes to the cafeteria on the days that his favorite meal is available; otherwise he goes to another place for his lunch. Each day, to decide where to go for lunch, he checks the cafeteria website, <http://kafeterya.metu.edu.tr/> whether the “Lamp Tandoori” is available or not. To be more efficient and not to check the website every day, he registers and logs in to the system.

He adds <http://kafeterya.metu.edu.tr/> URL and adds his favorite meal name, “Lamp Tandoori” as the keyword. Then he configures crawling and notification preferences. He sets crawling period as “Daily” as he wants to make the system crawl the cafeteria website and query his favorite meal “Lamp Tandoori” daily. He also wants to get the notifications when he is in the office, the day is a weekday and the time is just before

the lunch time. So, he enters the time range “between 11 o’clock and 12 o’clock”, and he selects “Weekdays” as the day preference. He also wants to get notified when he is at the laboratory and enters his laboratory location with latitude and longitude along with a range parameter as “latitude: 39.8922581, longitude: 32.7758673 and 50 meters”.

Finally, he selects the notification channel as “SMS” as he wants to get the notification whether he is online or offline. He configures all the settings accordingly. Then, the system starts to crawl the <http://kafeterya.metu.edu.tr/> daily and looks for the favorite meal name “Lamp Tandoori”, when there is any match, an SMS is sent to Serhat just before he goes out for lunch.

Next, we present the load evaluation of the framework carried out within the context of these sample use cases.

CHAPTER 5

RESULTS AND DISCUSSION

In this chapter, we share our findings on the evaluation of the framework. We have tested the framework according to different loads to observe the framework's scalability. In our tests, we mainly monitor the throughput and memory utilization. The results can be seen in the graphs below along with additional discussions.

5.1. Test Environment

In our tests, the cluster is composed of up to 6 slaves and 1 master node. In Apache Flink, the slaves are named as Task Managers and the master node is named as Job Manager. Each Task Manager has 4 cores at 2.5GHz clock speed and 512 MB RAM. The Job Manager also has 4 cores at 2.5GHz clock speed and 2 GB RAM. All the nodes are running Linux 4.4.0-31-generic (x86_64), in particular, Ubuntu 14.04 LTS as the operating system.

Table 3. Cluster Node Specifications

	CPU	RAM
Job Manager	4 x 2.5GHz	2 GB
Task Manager 1	4 x 2.5GHz	512 MB
Task Manager 2	4 x 2.5GHz	512 MB
Task Manager 3	4 x 2.5GHz	512 MB
Task Manager 4	4 x 2.5GHz	512 MB
Task Manager 5	4 x 2.5GHz	512 MB
Task Manager 6	4 x 2.5GHz	512 MB

The cluster is deployed using Docker because of its ease of use, and in our tests, the number of Task Managers have easily increased thanks to Docker. The version of the Apache Flink deployed in our tests is 1.2.1.

5.2. Network Characteristics

The performance of the cluster can depend on many factors of each node’s resources such as the number of CPU cores, clock speed of each CPU, memory and so on. Additionally, network characteristics between the nodes have a crucial effect on the performance. Even though we were deploying each node on the same host machine, we just want to make sure the bandwidth between the Job Manager and Task Managers is not a bottleneck for us. In order to test the network throughput, we have used a bandwidth test software called, iperf [50].

We have installed iperf on the Job Manager node and one Task Manager node. First, we have started iperf on the Job Manager in the server mode, by invoking *iperf -s*. Then, on one of the Task Manager nodes, we have started the iperf in the client mode, by invoking *iperf -c <ip_address>*. The results can be seen in Figure 24.

```

-----
[ ID] Interval           Transfer     Bandwidth       Retr
[  5]  0.00-10.04  sec  38.9 GBytes  33.3 Gbits/sec    45
[  5]  0.00-10.04  sec  38.9 GBytes  33.3 Gbits/sec
-----
sender
receiver

```

Figure 24. Network Bandwidth Measurement Results

As a result, the bandwidth between the Job Manager and Task Managers is calculated 33.3 Gb/s. As expected, this should not be a bottleneck for our tests.

5.3. Test Scenarios

In our tests, we have evaluated the scalability of the framework by testing the processing throughput depending on the task load, memory utilization of each Task Manager and overall memory usage depending on the increase in the number of Task Managers.

The load test tries to simulate the number of users that may use the framework, and we have generated random data for the simulation. We have created random location and time data to test the Distributed CEP Engine’s scalability at the rates of 1000 records / second, 2000 records / second, 4000 records / second, and 8000 records / second. For the tests of Distributed Focused Crawler, we have generated random URL and keywords data at the rates of 500 records / second, 1000 records / second, 2000 records / second and 4000 records / second.

First of all, we have monitored throughput for the context processing by increasing the load of context data being sent to the Distributed CEP Engine. The results can be seen in Figure 25.

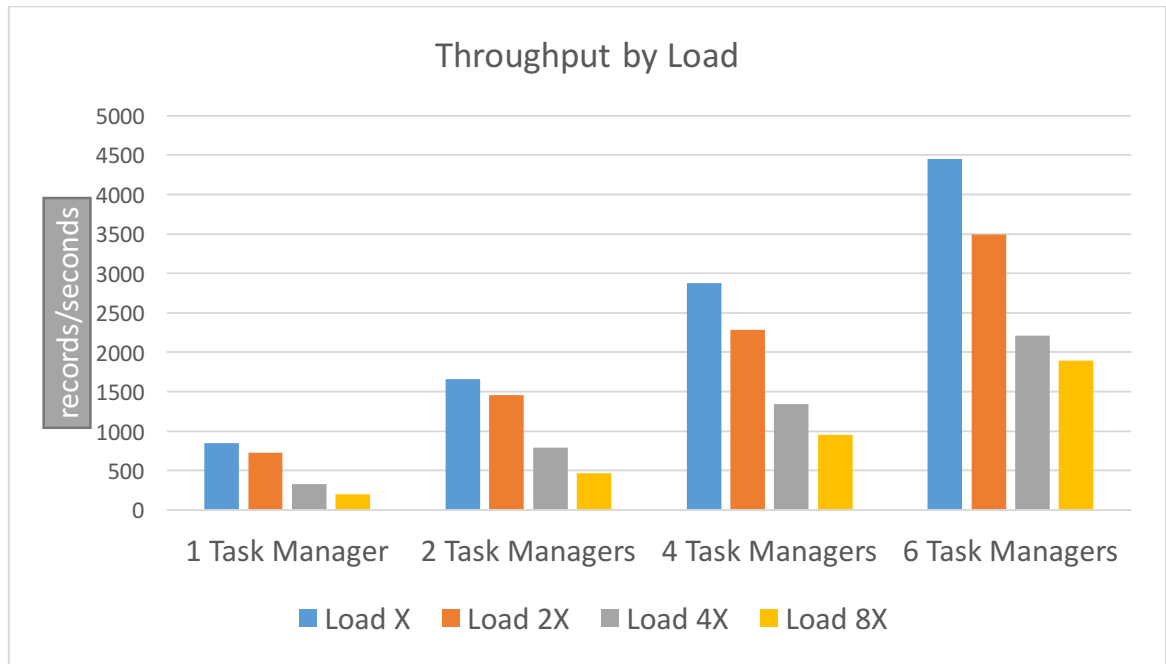


Figure 25. Throughput of Distributed CEP Engine

Then, we have tested the throughput for the Distributed Focused Crawler for different loads. The results can be seen in Figure 26.

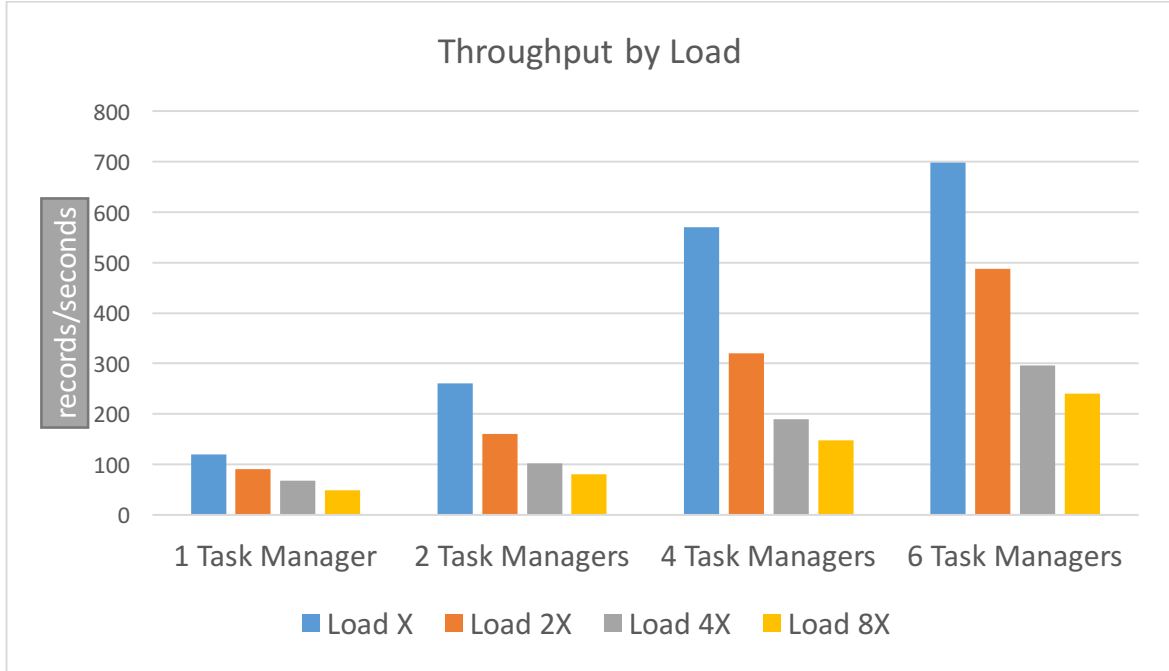


Figure 26. Throughput of Distributed Focused Crawler

In general, both for the Distributed CEP Engine and Distributed Focused Crawler, the throughput decreases depending on the load increase but still stays relatively high enough to produce good results.

Additionally, the performance of the cluster depending on the number of Task Managers with constant load can be seen in Figure 27 and Figure 28 for Distributed CEP Engine and Distributed Focused Crawler respectively.

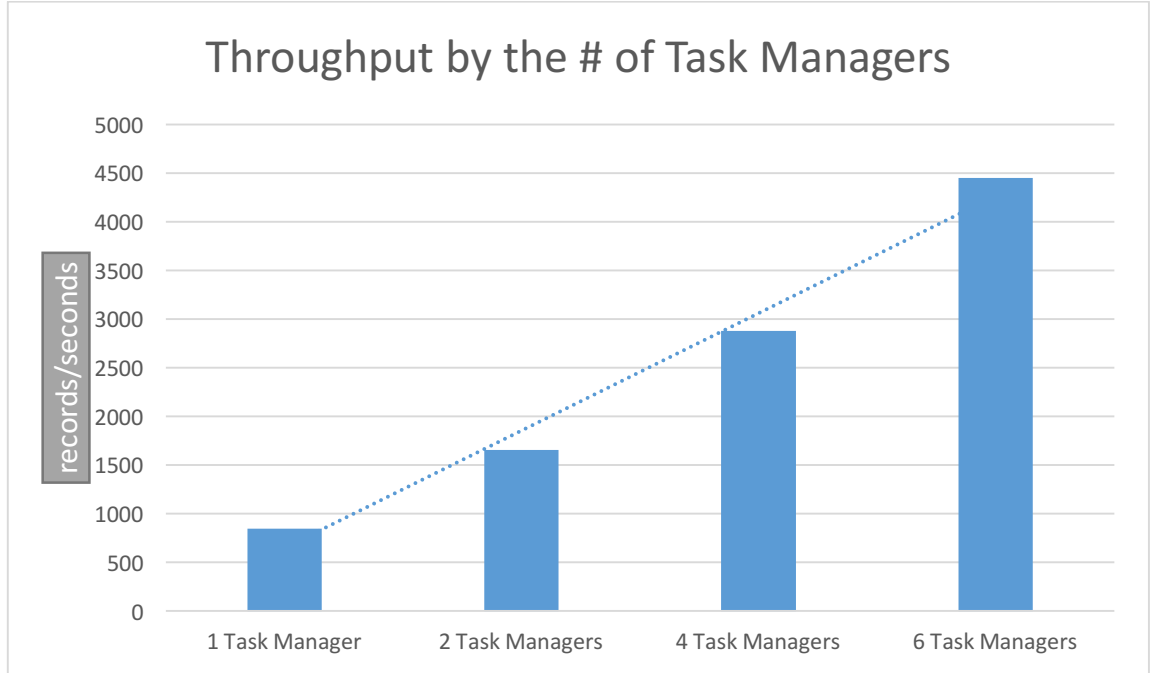


Figure 27. Throughput Increase by Task Managers for Distributed CEP Engine

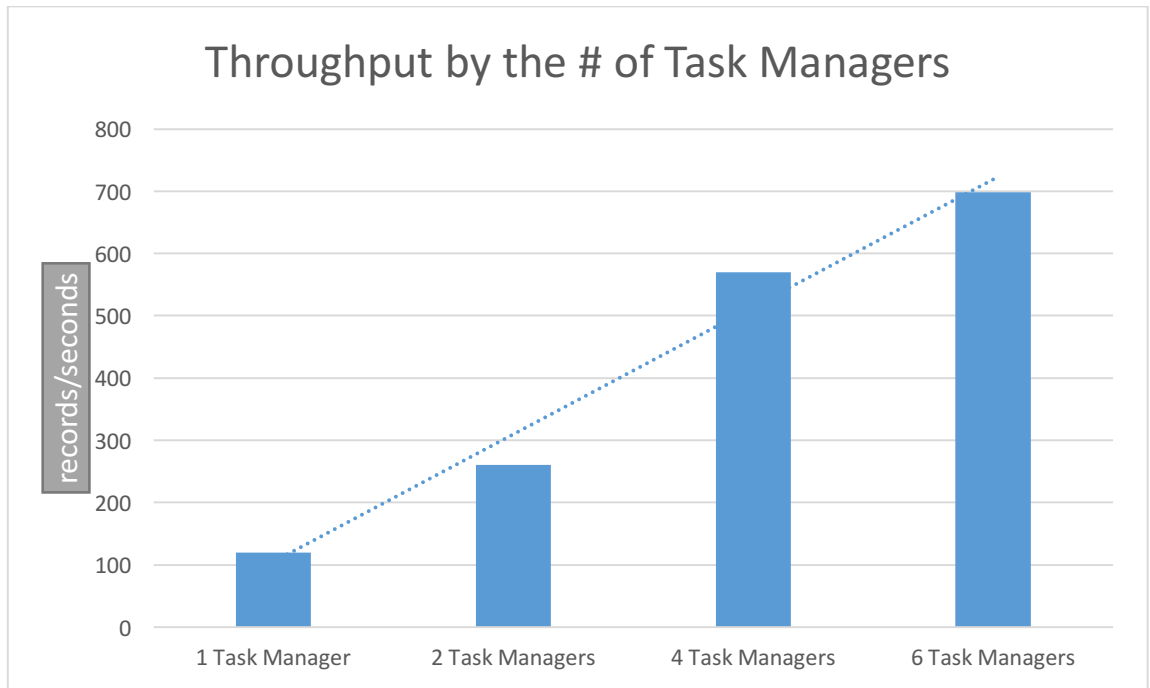


Figure 28. Throughput Increase by Task Managers for Distributed Focused Crawler

Both for the Distributed CEP Engine and Distributed Focused Crawler, the throughput increases when the number of Task Managers are increased. As it can be seen from the baselines, the growth of the performance is almost linear, meaning that we can

keep increasing the throughput if we add more Task Managers to the cluster until the network becomes a bottleneck.

One interesting thing to mention is that, when the load is increased, the throughput decreases. The main reason that causes this result could be all cluster being in the same host machine, including the message queue. Even though in our deployment they behave as different machines they all share the same host machine's resources such as CPU and network.

We have also wanted to examine the memory usage of the cluster. For a constant load, we have measured the average memory usage by the task managers both individually and overall.

The individual memory utilization by each Task Manager can be seen in Figure 29.

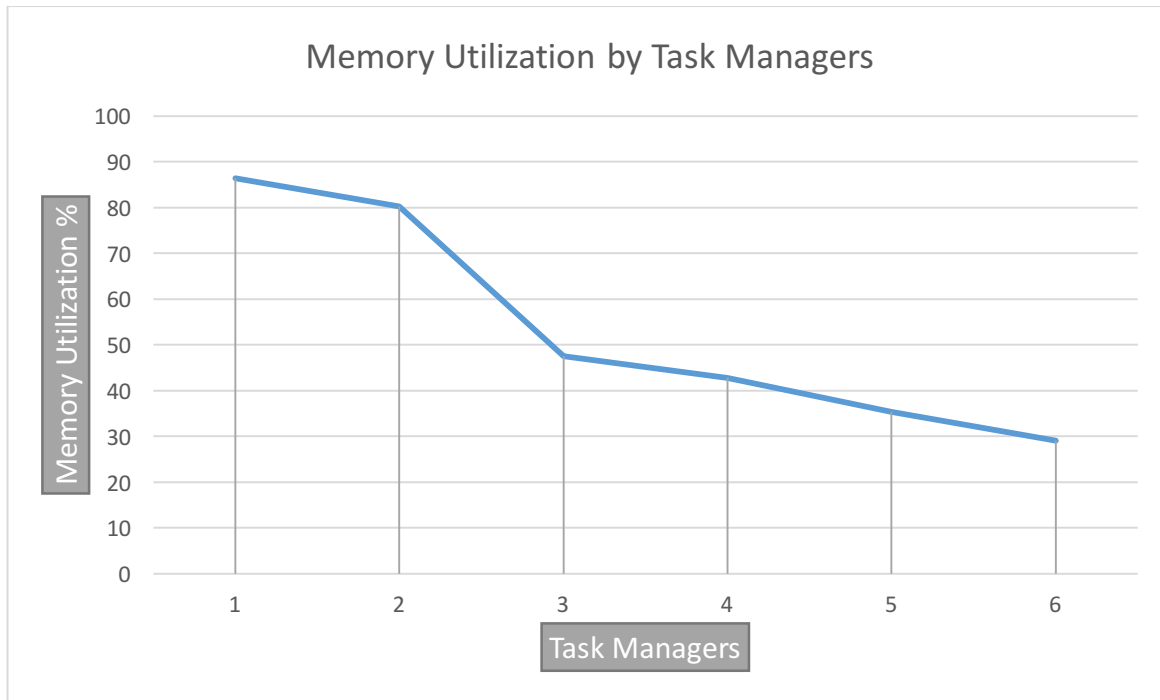


Figure 29. Memory Utilization of Task Managers

The utilization of memory by individual Task Managers decreases when new Task Managers are added to the cluster. This is because Apache Flink serializes data into memory segments rather than allocating spaces in JVM for every object individually [29].

Accordingly, the overall memory usage of the cluster with respect to the number of Task Managers in the cluster is also examined. The results can be seen in Figure 30.

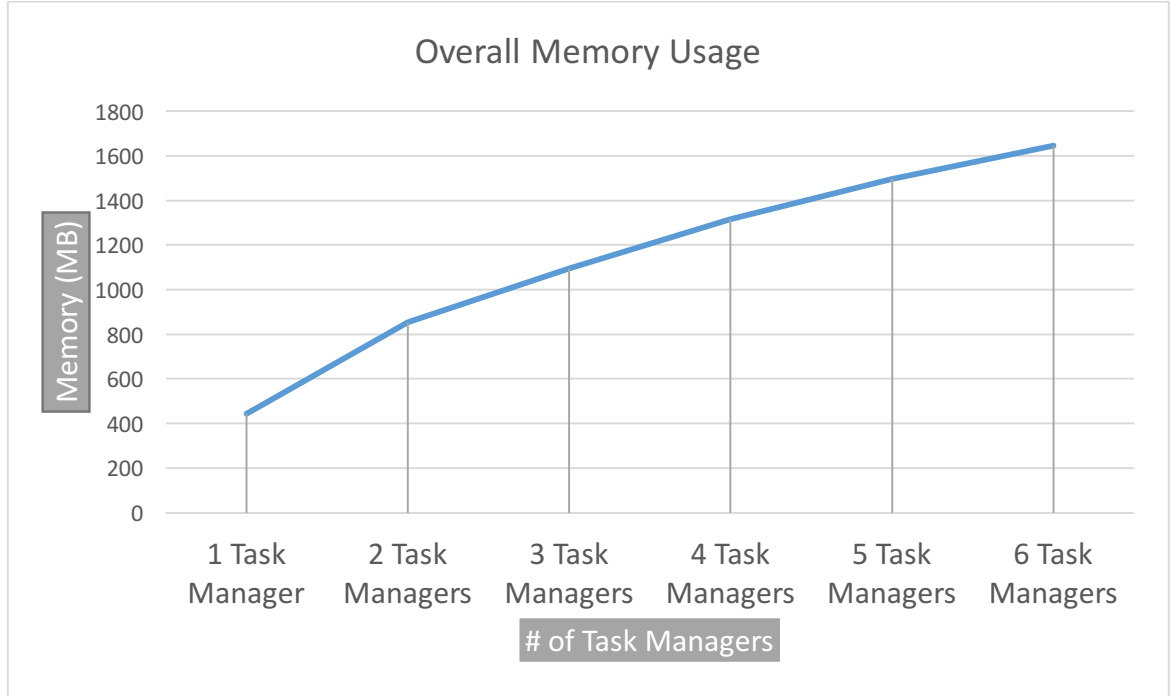


Figure 30. Overall Memory Usage of the Cluster

As a result, we have seen that overall memory usage of the cluster increases by the number of Task Managers, but the ratio of increase does not entirely correlate with the number of Task Managers.

Serving plenty of people with this framework is important for us. The results of both the throughput and the memory utilization tests show that the framework can handle the immense amount of processing hence can scale for a different number of users.

CHAPTER 6

CONCLUSION

Search for information is a fundamental daily activity for people. Obtaining the information as soon as it becomes available is critical. Moreover, some information is only valuable in a certain context. This context can depend on time, location, type of the information, the scope of information, or the combination of these parameters. Furthermore, the end user may wish to receive information via different delivery channels depending on the contextual information.

While traditional pull-based methods for accessing information fail to satisfy the requirements of the current competitive business environment, existing push-based solutions fall short on multiple aspects in this new era. Some of these tools are only usable if the content provider implements the underlying functionality. Focused crawling is used to tackle this problem. Tools that do focused crawling do not have a sufficient distributed architecture to meet the scalability requirements of big data era. Others do not consider context comprehensive enough to allow users to define intended websites, precise situations, and desired delivery channels to retrieve notifications. When users cannot define the exact situation to receive notifications, the notifications become distractions. To avoid unnecessary distractions, users tend to ignore all notifications in which case users have to retrieve information with pull-based methods.

In this study, we propose a context aware notifications framework based on focused crawling to overcome the shortcomings of pull-based information retrieval systems. In this proposed framework, users can define the exact context in which they prefer to receive information. They can choose when, where, and how to receive the information. Notifications become valuable prompts, instead of distractions. Additionally, in order to satisfy the requirements of big data era, our framework takes advantage of latest developments in the big data analytics domain and particularly in stream processing and complex event processing systems. Built on top of an open source stream processing engine, our framework can satisfy real-time processing and delivery requirements. Hence, we have managed to process the context data with high throughput and we have also proposed various crawling strategies to make the crawling more efficient and politer for websites. In order to only find the latest web pages and not to fall into web spider traps, we only crawl up to the first depth of the web pages linked from the homepage and we have grouped the same URLs provided by different users to download them as efficient as possible to be politer for websites.

We are planning to develop the framework further to be more adaptive and to be able to provide more relevant content recommendations by the introduction of Natural Language Processing (NLP) and machine learning algorithms to match web pages not only with user queries but also with content classification and text mining algorithms. In the future, we also intent to monitor not only keyword match but also keyword density inside the crawled web page to better identify the content relevancy.

REFERENCES

- [1] S. Chakrabarti, M. Berg, and B. Dom, “Focused crawling: A New Approach to Topic- Specific Web Resource Discovery,” *Comput. Networks*, vol. 31, pp. 1623–1640, 1999.
- [2] R. Gaur and D. K. Sharma, “Review of ontology based focused crawling approaches,” *ICSCCTET 2014 - Int. Conf. Soft Comput. Tech. Eng. Technol.*, 2016.
- [3] A. (2001). U. and using context. P. and U. C. R. from <http://dl.acm.org/citation.cfm?id=59357>. Dey, AKDey and A.K.~Dey, “Understanding and using context,” *Pers. Ubiquitous Comput.*, vol. 5, no. May, pp. 4–7, 2001.
- [4] M. Baldauf, “A survey on context-aware systems,” *Inf. Syst.*, vol. 2, no. 4, 2007.
- [5] G. Vargas-solar, J. A. Espinosa-oviedo, and L. Zechinelli-martini, “Big Data Technology and Applications,” vol. 590, 2016.
- [6] “Lambda Architecture.” [Online]. Available: <http://lambda-architecture.net/>.
- [7] J. Meehan *et al.*, “Integrating real-time and batch processing in a polystore,” *2016 IEEE High Perform. Extrem. Comput. Conf. HPEC 2016*, 2016.
- [8] “Kappa Architecture.” [Online]. Available: <https://www.oreilly.com/ideas/questioning-the-lambda-architecture>.
- [9] D. C. Luckham, *Event processing for business: organizing the real-time enterprise*. John Wiley & Sons, 2011.
- [10] J. Cho, H. Garcia-Molina, and L. Page, “Reprint of: Efficient crawling through URL ordering,” *Comput. Networks*, vol. 56, no. 18, pp. 3849–3858, 2012.
- [11] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori, “Focused crawling using context graphs,” *Proc. 26th ...*, pp. 527–534, 2000.
- [12] A. Heydon and M. Najork, “Mercator: A scalable, extensible Web crawler,” *World Wide Web*, vol. 2, pp. 219–229, 1999.
- [13] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, “UbiCrawler: A scalable fully distributed Web crawler,” *Softw. - Pract. Exp.*, vol. 34, no. 8, pp. 711–726, 2004.

- [14] P. Boldi, A. Marino, M. Santini, and S. Vigna, “BUbiNG: massive crawling for the masses,” *Int. Conf. World Wide Web - WWW '14 Companion*, no. Ga 288956, pp. 227–228, 2014.
- [15] H. Yan, J. Wang, X. Li, and L. Guo, “Architectural design and evaluation of an efficient web-crawling system,” *Proc. - 15th Int. Parallel Distrib. Process. Symp. IPDPS 2001*, vol. 60, pp. 1824–1831, 2001.
- [16] V. Shkapenyuk, “Design and Implementation of a High-Performance Distributed Web Crawler Vladislav Shkapenyuk Torsten Suel Department of Computer and Information Science Technical Report TR-CIS-2001-03 Design and Implementation of a High-Performance Distributed Web Crawle,” 2001.
- [17] M. Boanjak, E. Oliveira, J. Martins, E. Mendes Rodrigues, and L. Sarmiento, “TwitterEcho,” *Proc. 21st Int. Conf. companion World Wide Web - WWW '12 Companion*, p. 1233, 2012.
- [18] “RSS 2.0 Specification.” [Online]. Available: <http://blogs.law.harvard.edu/tech/rss>.
- [19] K. C. Sia, J. Cho, and H.-K. Cho, “Efficient Monitoring Algorithm for Fast News Alerts,” pp. 1–12.
- [20] M. Gusev, S. Ristov, P. Gushev, and G. Velkoski, “Alert Notification as a new model of internet-based transactions,” in *2014 22nd Telecommunications Forum, TELFOR 2014 - Proceedings of Papers*, 2015.
- [21] E. Katsiri, “A context-aware Notification Service.”
- [22] F. Corno, L. De Russis, and T. Montanaro, “A context and user aware smart notification system,” *IEEE World Forum Internet Things, WF-IoT 2015 - Proc.*, pp. 645–651, 2016.
- [23] A. Paraskevas, I. Katsogridakis, R. Law, and D. Buhalis, “Search engine marketing: Transforming search engines into hotel distribution channels,” *Cornell Hosp. Q.*, vol. 52, no. 2, pp. 200–208, 2011.
- [24] “Apache Flink.” [Online]. Available: <https://flink.apache.org/>.
- [25] “Companies Using Apache Flink.” [Online]. Available: <http://flink.apache.org/poweredby.html>.
- [26] “Zalando.” [Online]. Available: <http://www.zalando.com/>.
- [27] “King.” [Online]. Available: <https://king.com/>.

- [28] “Alibaba.” [Online]. Available: <https://www.alibaba.com/>.
- [29] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, “Apache flink: Stream and batch processing in a single engine,” *Bull. IEEE Comput. Soc. Tech. Comm. Data Eng.*, vol. 36, no. 4, 2015.
- [30] “Introduction to Apache Flink.” [Online]. Available: <https://flink.apache.org/introduction.html>.
- [31] “Apache Spark.” [Online]. Available: <http://spark.apache.org/>.
- [32] “Apache Storm.”
- [33] “Esper.” [Online]. Available: <http://www.espertech.com/esper/>.
- [34] “Drools - Business Rules Management System.” [Online]. Available: <https://www.drools.org/>.
- [35] “Apache Kafka.” [Online]. Available: <https://kafka.apache.org/>.
- [36] “Apache Kafka Introduction.” [Online]. Available: <https://kafka.apache.org/intro>.
- [37] “Apache Zookeeper.” .
- [38] “Docker.” [Online]. Available: <https://www.docker.com/>.
- [39] “Docker Overview.” [Online]. Available: <https://docs.docker.com/engine/docker-overview/>.
- [40] “Docker Hub.” [Online]. Available: <https://hub.docker.com/>.
- [41] M. Koster, “A method for web robots control,” *Netw. Work. Group, Internet Draft*, 1996.
- [42] Y. Sun, Z. Zhuang, and C. L. Giles, “A large-scale study of robots. txt,” in *Proceedings of the 16th international conference on World Wide Web*, 2007, pp. 1123–1124.
- [43] “AWS Elastic Beanstalk.” [Online]. Available: <https://aws.amazon.com/elasticbeanstalk/>.
- [44] “CodeIgniter PHP Framework.” [Online]. Available: <https://codeigniter.com/>.
- [45] “Bootstrap Frontend Framework.” [Online]. Available: <http://getbootstrap.com/>.
- [46] “Apache Flume.” [Online]. Available: <https://flume.apache.org/>.

- [47] “RabbitMQ.” [Online]. Available: <https://www.rabbitmq.com/>.
- [48] “MQTT.” [Online]. Available: <http://mqtt.org/>.
- [49] “Nodejs Framework.” [Online]. Available: <https://nodejs.org/>.
- [50] “iperf - Bandwidth Measurement Software.” [Online]. Available: <https://iperf.fr/>.