NTNU – Trondheim
Norwegian University of
Science and Technology

Large-Scale User Click Analysis in News Recommendation

John Eirik Bjørhovde Nilsen

Master of Science in Computer Science
Submission date: June 2013
Supervisor: Jon Atle Gulla, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Sammendrag

Internett og World Wide Web har tatt over som standard for å lese og finne nyheter. Dette gjør det mulig for nyhetslesere å nøye velge nyhetene som er mest interessante for dem. På grunn av de store mengder av artikler tilgjengelig, kan det være en utfordrende og tidkrevende oppgave å finne den ønskede informasjon. Å forenkle denne prosessen for leserne ville være fordelaktig.

Denne avhandlingen utforsker ideen om å filtrere ut uønskede nyhetsartikler og serverer de nyttige seg til leseren gjennom mobile plattformer. Det er en del av et større prosjekt kalt SmartMedia som fokuserer på bruk av komplekse strategier for å levere nyheter til brukerne. Mens den overordnede strategien er basert på bruk av den totale rammen til brukerne for å gi nyheter, er den spesifikke omfanget av denne oppgaven å opprette brukerprofiler fra brukerens handlinger logget av systemet. Motivasjonen er å utnytte disse profilene i samarbeid med informasjonsfiltreringsteknikker for å bidra til å nå det overordnede målet.

En stor del av denne avhandlingen fokuserer på å implementere Hadoopjobber som oppsummerer brukeren logger inn profiler. I løsningen består hver brukerprofil av to vektorer. En kategorivektor som beskriver brukerens interesser i de ulike nyheter kategorier og en nøkkelordvektor som utnytter enheter definert i nyhetsartikler å analysere på et lavt detaljnivå nivå. Resultatene evalueres og diskuteres til slutt.

Evaluering av effektiviteten og nøyaktigheten av brukerprofilene er vanskelig. Lite reelle data var tilgjengelig i løpet av denne forskningen og faktiske data er nødvendig. Data som agerer reelle brukere er vanskelig å forfalske og er nødvendig for både evaluering og kalibrering av implementasjonen. Dermed er fokus for diskusjonen hvordan man utfører disse to oppgavene når systemet er i produksjon.

Summary

The Internet and the World Wide Web have taken over as the standard for reading and finding news. This makes it possible for news readers to carefully choose the news that is most interesting for them. Due to the large amounts of articles, it can be a challenging and time consuming task to find the wanted information. Simplifying this process for the news readers would be beneficial.

This thesis explores the idea of filtering out unwanted news articles and serving the useful ones to the reader through mobile platforms. It is part of a bigger project named SmartMedia that focuses on using complex strategies for delivering news to the users. While the overall strategy is based on using the total context of users to serve new, the specific scope of this thesis is creating user profiles from user acts logged by the system. The motivation is to utilize these profiles in cooperation with information filtering techniques to help reach the overall goal.

A big part of this thesis focuses on implementing Hadoop jobs that summarizes the user logs into profiles. In the solution, each user profile consists of two vectors. A category vector that describes the user's interests in the different news categories and a keyword vector that exploits entities defined in news articles to analyse at a low granularity level. The results are evaluated and discussed at the end.

How to evaluate the effectiveness and accuracy of the user profiles is difficult. Little real data was available during this research and actual data is needed. Data that replicates real users is hard to forge and is needed for both evaluation and calibration of the implementation. Thus, the focus of the discussion is on how to perform these two tasks when the system is deployed.

Preface

This report presents the work done in my master thesis as a 5th year of student in the Computer Science Master Program at NTNU, Trondheim. The master thesis was carried out in the spring of 2013 at the Department of Computer and Information Science, Faculty of Information Technology, Mathematics and Electrical Engineering at NTNU. Supervision was conducted by two persons; Professor Jon Atle Gulla and Senior Engineer Arne Dag Fidjestøl both from NTNU.

I would like to thank my supervisors for their support and helpful feedback during the semester.

Trondheim, June 18, 2013
John Eirik Bjørhovde Nilsen

Contents

List of Tables

List of Figures

List of Listings

Part I

Introduction

Chapter 1

Introduction

Reading and finding news have changed drastically the last couple of decades. 20 years ago the standard literal news source was paper printed newspapers. Today, the Internet and the World Wide Web have taken over as the standard for this type of media. This has opened the possibility for the news readers to carefully choose the sources that is most interesting for them. Millions of sources spread around the globe has become available to the readers. The vast amount of sources makes it hard to find articles and sources that satisfies a specific users' personal interest. This is an information overload problem.

The websites of CNN[1] and VG[2] are examples of sources that generates news articles. A problem for news readers is that each source delivers different types and quality of news. If a reader wants other types of news, then a different source would be needed. This implies that the reader would have to keep a portfolio of news sources to browse. News aggregation applications solves this problem. Google News and Yahoo! News are examples of news aggregation websites. The research in this thesis focuses mainly on news delivered on mobile platforms, but the domains are very similar. On mobile platforms there exists the same kind of solutions. Zite and Google Currents are examples of mobile news aggregation applications. Still the problem with information overload exists.

Information filtering is a technological approach that in general solves information overload problems. Information filtering methods aims to extract the important information from a large bundle of information with mostly non-important information. In addition to news, email and web search are examples of domains where this approach has been utilized. The most common approach to filter information is by exploiting a profile that represents the preferences of each user. The research in this thesis focuses on exploring how user profiling can help to lighten the burden on the news readers and

[1] CNN website: http://www.cnn.com
[2] VG website: http://www.vg.no.

help them find interesting news stories on a mobile platform.

The work in this paper was done as part of a bigger project names SmartMedia, which will be described below. A news recommender system was the main aim of the project. Development of a module for personalization making user profiles was also key to this thesis. Utilization of a large scale data processing framework named Apache Hadoop were central. Another sub-project of SmartMedia were relying on this module to perform information filtering in the total system.

The remainder of this chapter consist of a short overview of the SmartMedia project. Then the research questions are described. Following this, a section summarises the approach for the module to be created. The results of the research is described after this. Finally, the rest of the report is outlined.

1.1 SmartMedia

The SmartMedia program[1] was established in 2012 by the Department of Computer and Information Science at NTNU. The main objective of the project is to map out new technologies that might give positive effects on the media industry in the current environment. Therefore, close collaboration with the media industry is key for the program.

The SmartMedia program has one ongoing project on news recommender systems that takes advantage of complex strategies to deliver news to the users. This project will be referred to as the SmartMedia project in this report. A lot of subprojects are running concurrently within the overall project, this being one of them. Mobile application development, geospatial filtering, and collaborative filtering are the focus on some of the ongoing subprojects.

Figure 1.1 displays the architecture of the system that was given by SmartMedia. The preprocesssing module takes in article data from RSS feeds. The data is pushed through a pipeline that makes news recommendation possible. An Apache SolR server stores these data. The middleware layer connects all the different subsystems. The iPhone client is where the user interacts with the system. MongoDB is used to store log data. The batch job system is where the development of this research takes place. The arcitechtural view can not be considered complete since a lot of modules like information filtering and collaborative filtering are not present. This is because these modules have not been completely initiated at the time being.

Figure 1.2 shows two screenshots of the news application. The one of the left is a category view of the application and the right one is the front page.

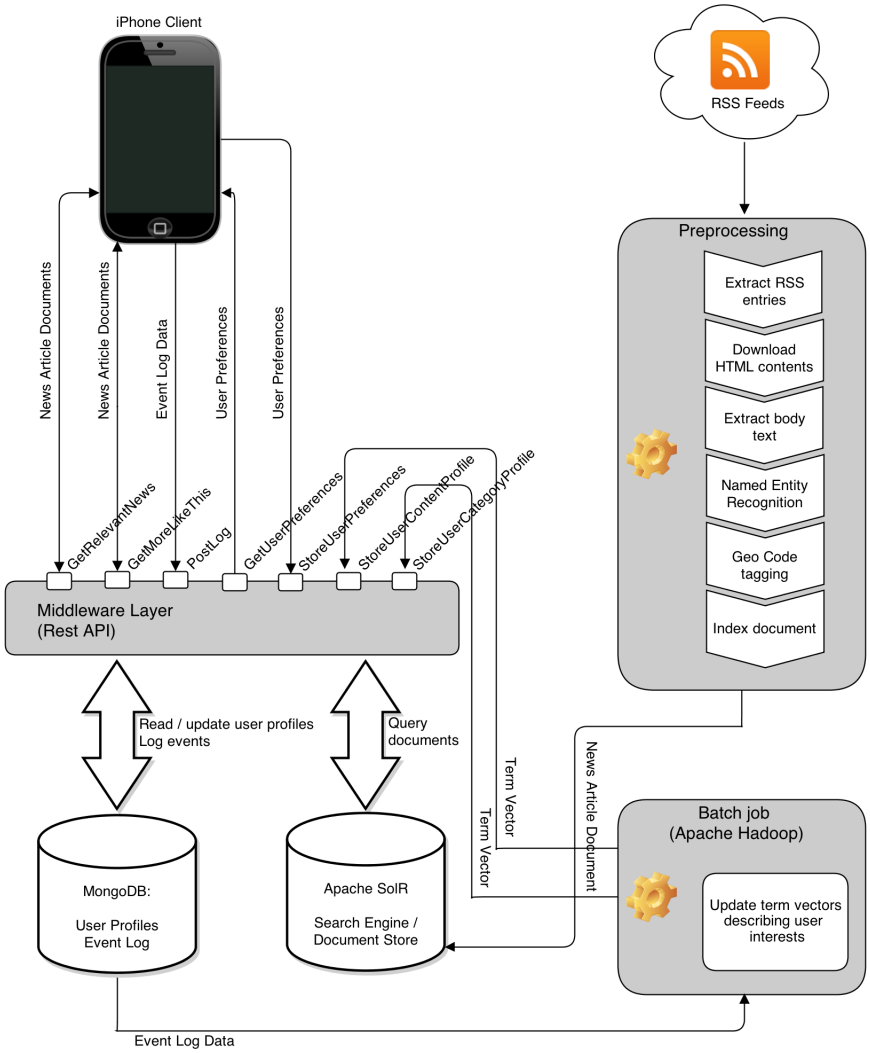

Figure 1.1: SmartMedia project system architecture

1.2 Research Questions

The motivation for this thesis was to explore user click log analysis in the context of a news recommender system and to realize this knowledge in form of a user profiling system. The main target is to study and make user profiles that can be used by recommendation methods in general. There are three research question in this study to aim more specifically what the focus is. They are listed below:

1. What are common approaches to user log analysis for user profile construction?
2. How can user profiles be extracted automatically in news recommender systems without any direct feedback on presented news from users?

Figure 1.2: Mobile application screenshots

3. What architectural and algorithmic approaches should be adopted to deal with the magnitude of user clicks in news recommender systems?

The interpretation of the first question is that a literature review focusing on what prior research have been done within this topic is needed. The second question also states the importance of a literature review, but implementing an approach also is central. The last question is focuses on what technologies that exists to handle large amount of data and also how to use these technologies through implementation. To clarify that this can be considered as a big data problem, an example can be used. Each day 2.000 new articles enters the system. If the user base consists of 10.000 users and each user browses 15 articles a day, then 150.000 views have to be considered. When each display of an article also has a collection of different events with different impact on the result, it is clear that this can be considered as a large-scale operation. If the user base grows beyond this point, it is even further emphasized as traditional programming would be insufficient.

1.3 Approach

A big part of our contribution is a literature review of related work to point out what is the state-of-the-art within this topic. The literature is threefold where one part focuses on concepts generally applicable to the domain, one part that describes what approaches that have been experimented with by others, and one part centralized around a big data and how to process it.

The implementation approach is to use click logs generated by users in the mobile application to make user profiles. When the user browses through the application, the gestures of the users are stored in a log database. These gestures are named user acts in this thesis. Each type of user act has different weight impacting the final output. Vectors were used to store the profiles. This was a design decision made by SmartMedia. To compute the vectors, a large scale computation system called Hadoop is utilized. The implementation is a series of MapReduce jobs that jointly serve as a batch job outputting the vectors. This batch job is intended to run periodically to keep the user profiles up to date.

Weights are constants that need to be calibrated as the system is put into operation and the data can be analysed. Since the system was not put into production, an empirical evaluation of this system was difficult. Therefore, the evaluation strategy focuses on sensitivity analysis of the calibration factors.

1.4 Results

The results of the literature review showed that some work have been done within news recommendation. Little literature have been published in the context of mobile news recommendation, but since it is a subset of news recommendation the domains are very similar. How to value heterogeneous user clicks is absent in all the literature studied, so this is a field that deserves more focus.

It is possible to use Hadoop and MapReduce frameworks to summarize click logs into user profiles. It gets more challenging when introducing several inputs to a job and the time complexity might also suffer from this.

The results of the evaluation implies that the most important user act (open article) can be considered as a generalization of the rest of the events, although it is believed that the other user acts also entails some knowledge to be discovered. The “news” category also seems somewhat dominant in the tests, this could be due to the category distribution or that the module that classifies articles has some weaknesses.

The results also show that one type of user act, named article preview time, is very dominant in the logs. Nearly 80% of the keyword vector entries are, so this might be

viewed upon as redundant data to process when considering cost versus benefit.

1.5 Outline

The rest of this report consists of 8 chapters.
Chapter 2 introduces necessary technological topics.
Chapter 3 describes some of the related work out there.
Chapter 4 focuses on the tools that were central in realizing the module.
Chapter 5 goes deeper into the main strategy of the module development.
Chapter 6 gives a low level insight into the implementation.
Chapter 7 states the evaluation of the user profiling system.
Chapter 8 discusses the finding of this research.
Chapter 9 concludes the report.

Part II

Theoretical Background

Chapter 2

Technological Overview

This chapter gives a technological overview of the relevant background. We start by looking at recommender systems in general. Next we look at the information filtering techniques commonly used. Then user profiles are discussed. The part following describe mobile news recommendation and it's challenges, before looking at the context of our news recommender system.

2.1 Recommender Systems

People often have to rely on the use of recommendations since they lack the needed knowledge to make the decision with highest utility. If, for instance, a person wants to spend the holiday in a new foreign country, then the decision regarding what hotel to stay in might be hard to make. The person could ask friends that have experience with the foreign place to conclude with a better decision than without. This is an analogy describing how a recommender system works. The recommender system serves as the friend that recommends hotels.

The standard recommender system uses peoples recommendation as input which is then transformed and used to make recommendations to other people[2]. The systems vary in how and what they use as input and how they transform this input. Some approaches uses matching between recommenders and users based on similarity measures, while other focuses on the direct aggregation of recommendations.

In [2], a framework for classifying recommender systems is suggested. The framework is built up of two different spaces of dimension. One dimension that focuses on the technical design and one describing the domain. The domain consists of recommended items, the actors in the system, and evaluations. The different spaces and their dimensions are given in Table 2.1 where the domain space have been divided into

two parts.

Table 2.1: Recommender systems framework

Technical design space	- Contents of recommendation - Explicit entry? - Anonymous? - Aggregation - Use of recommendations
Domain space – **Characteristics of items evaluated**	- Type of items - How many - Lifetime - Cost structure
Domain space – **Characteristics of actors and evaluations**	- Recommenders - Density of recommendations - Consumers - Consumer taste variability

2.2 Information Filtering

Information filtering is a "Big Data" problem by nature. The basic idea is to select the information that is important from a large set of mostly unimportant information. The quote below is a definition of information filtering:

> *"Information Filtering is a field of study designed for creating a systematic approach to extracting information that a particular person finds important from a larger stream of information."*[3]

2.2.1 The Need for Information Filtering

Any person with access to the Internet can be an author that publishes material. The consequence[4] of this is that the quantity of information available is gigantic. The quality of the information on the other hand is quite diverse. The main problem with the Internet today is not how available information is, but rather how to find the information of interest. The personal aspect affects this as well; what is important to one person might not be important to another person. The following quote sums this up well:

> "At least 99% of available data is of no interest to at least 99% of the users."
> [5]

2.2.2 Types of Filtering

There are two main types of information filtering. Namely, content-based filtering and collaborative filtering. Content-based filtering[6] is a collection of methods that only uses data related to the specific user to deduce what information is important. This can be done by prompting the user to quantify his/hers desires. It can also be achieved by looking on how the user relates with the items in the system. A combination of the two information sources is also feasible.

Collaborative filtering[7], on the other hand, uses the similarity between different users to perform filtering. A simple example of this is if two users like sports, then recommending information that one user likes to the other would be considered collaborative filtering. A hybrid of the two approaches is also possible and this can improve the total filtering process.

Common for all information filtering methods are that they base the filtering on modelling the user in a way that gains insight into the specific user's preferences[8].

2.3 User Profiles

An user profile is a digital representation of the user. The profile can be used to store the specific user's characteristics. There are many possible ways to represent an user profile. How they are stored depends on what the intention of the profile is. For instance, in an operating system the user profile is used to allow different users safe access to the system. On a social media web site, the user profile is used to represent data about the user to be displayed to other users.

In the context of recommender systems there is also a lot of opportunities. Some important characteristics of a user profile have been defined in [9]. These characteristics are taken from a web usage data mining domain, which can be considered as similar to news recommendation. When converted to a more general domain, these characteristics are:

- Possibility to compare users
- Differentiate the importance of different actions
- Uniform representation of the profiles

Given these distinguishing factors, [9] concludes that representing user profiles as weighted collections are the most pleasing approach. This is basically a vector. The reasoning behind this is that vectors are flexible, easy to understand and easy to use.

2.3.1 User Models

The types of user models are based on two different properties of the model[10]. The type depends on how the users are being modelled and how the models are acquired. The users can be modelled as individuals or in a canonical fashion. In individual models, each user has one interface and individual differences are prioritized. In canonical models the user is considered as part of a group and the priority is to describe what the users in the have in common.

The acquisition model can be explicit or implicit. Explicit means that the user model is based on information provided by the user. Implicit is where the user have no involvement with the process. User feedback and user behaviour are usually the basis for implicit acquisition.

2.4 Mobile News Recommandation

Mobile news recommendation is getting the news you want, at the time you want it and were you want. This with as minimum of work to get it. As an example, if a user is riding a bus and want to check the news. Then instead of finding relevant news manually a mobile news recommendation system could find the news articles wanted for the user.

A challenge of making a mobile news recommendation system is that news are is always new information. This implies that most of the articles will be unseen by the system when it arrives. Users are also unique in their interests, thus making generalization of recommendation difficult and complex methods are needed to serve the individual user the needed information. Quantifying the interests for each individual user is difficult since the granularity of the solution has to be detailed to get the most fitting profile. Factors like time of the day, a user's interests opposed to trending news, and not boxing the recommended articles in to specific categories also plays a role. On mobile platforms, the bandwidth usage also has a part since minimalistic data transfers is in the users interests.

2.5 User Context for Mobile News Recommandation

In our mobile news recommendation system, a user context is made up of the following user features:

1. Time
2. Location

3. User groups
4. News category
5. News items

The time feature entails that the system should be able to serve different news to an user in specific time periods. For instance when a user is at work then one type of news might be more relevant than if the user is at home. Location implies that the location of the user should be a driving factor in selecting news. If a user is in New York, then news from this area is more relevant than news from Tokyo. Then again, if the user has special interest in an area, then this also has to be taken into account. It is also intended to take the underlying social networks of users in the mix. The three first features are outside the scope of this paper.

Each user also has unique interests. Utilizing categories of news to distinguish users interests is a possibility. Entities and items in articles can also be used in this fashion. The categories can be used to give a general view of the users desires. If a user likes technology news better than sports, then the intention is to weed out sports news and recommend more technology news. However, to get a more detailed view of what the user likes, then items in articles can be used. If a user don't like sports in general, but likes a specific football team, then the solution should use these items so that it is taken into account.

Chapter 3

Related Work

This chapter focuses on some of the related work out there. News recommendation projects that have utilized content-based filtering is the aim of the first section. Then collaborative and hybrid solutions are scoped out in the second part. In the third part, user profiling by click logs in other areas are discussed. Finally, how news interests change over time is outlined.

3.1 Content-Based Filtering

Content-based filtering methods have been applied in different areas. Email[11] and web-search[12] are example areas. The approach has also been utilized in the news sector to provide personalized news to users. Common approaches to news recommendation utilizes the user's explicitly stated preferences or the behaviour of the user within the system to filter news. User profiles are generated from this information and used as a basis for the filtering. Not a lot of work has been published regarding the use of these kinds of systems on smartphone platforms, but other areas that are highly related has been explored. Intelligent agents constructed to serve a daily news program for the specific user[13] are one related approach.

PIN[14], a web-based news aggregation system based on adaptive resonance associative maps, which belongs to a family of neural network systems, is another approach. PIN initializes the user profile by letting the user rate a list of keywords, then the profile is altered according to user feedback when the system is used.

The second version of the Daily Learner[15] is a system that uses a client/server based architecture for recommending news. Their idea is to have a common interface for multiple device types. Their research focuses on PDAs, but other types of device could also be utilized. It bases the filtering on both explicitly by allowing users to mark news

stories as interesting which increases the rating and not interesting which decreases the rating. Implicitly they use user behaviour such as entering an article, scrolling through the article, and requesting more information about an article to increase the rating. Their profile consists of marking news articles as interesting and uninteresting on a scale between 0 and 1. Their results states that the effective personalization can be achieved without any user interaction and that the approach is superior to no filtering.

WebClipping2[16] is another system that uses web-based technologies on PDAs to serve users news with collaborative filtering is described. They use user profiles that is initialized by the user providing their interests on different subjects and then generating a keyword database according to this, thus the user profiles consists of a set of keywords. The keywords were gathered from different sources around the web. To keep the profiles up to date, they used the explicit feedback from users to determine whether the profiles should be altered. The feedback would be coupled with specific articles and their keywords. The users would rate the articles on a scale from 1 to 4 (not interesting to very interesting). They also utilized inferring methods by using the total reading time, number of lines read and such characteristics of user behaviour to determine user's interests. Their results states that they infer a user's interest in an article with 90% precision.

What is common for these approaches is that they require the user's input to base their recommendation. Either initially, during the usage of the system, or both.

3.2 Collaborative Filtering and Hybrid Approaches

GroupLens[17] is a news recommendation system that utilizes collaborative filtering. Their approach was to partition the news articles into clusters that are commonly read together. The user models are made of clusters of users that have read and rated the same articles. These two clusters together are used as to perform collaborative filtering.

The first version of Google News recommender[18] utilized collaborative filtering techniques on large-scale data. They use cluster users by applying a MinHash algorithm and PLSI. The focus of this research was scalability, so no evaluation of recommendation accuracy were done.

A hybrid of the two approaches is also possible, where benefits from both methods can be achieved. Claypool et al.[19] utilizes both content-based and collaborative filtering. It is not directly a hybrid approach, since the methods used are separated, but it uses both. Content-based filtering is executed by matching keywords of articles with keywords in the user profiles. Collaborative filtering is achieved by computing similarities

between user profiles. The results are combined in a combination filter that takes each both approaches into account.

Google News tested out a hybrid approach [20]. The collaborative filtering is done by utilizing the work in [18]. Their content-based filtering is done by recommending articles in a specific category that is considered as highly relevant by the user profile. The user profiles are constructed by analysing user clicks on articles. On average they showed that utilizing both methods improved the quality of the recommendations and traffic to the news application.

3.3 User Modelling Based on Click Stream

Most of the research on click streams comes from the web search domain. Lee, Liu, and Cho[21] modelled users based on click stream to enhance personalized web search. Their assumptions was that the specific user's goal could be learned and identified by how other users in the past have used the results of the specific query. Their results states that they can identify the goals of 90% of the queries they studied.

Speretta and Gauch[22] builds user profiles from search history. The user profiles are structured as weighted concept hierarchies. The click logs they used consisted of specific queries and the results clicked on by the users.

Kim and Chan[23] tries to learn a topic hierarchy of interests by utilizing click logs of a user to provide a context for personalization. The hierarchy is a general-to-specific hierarchy and is learned by usage of clustering methods.

Nasraoui et al.[24] proposes a framework for web usage mining. They process user sessions with clustering techniques and summarizes these clusters into user profiles. Then they enrich the profiles with additional log data and domain knowledge. They use category vectors to store the user profiles.

There are little research done regarding how to value and utilize heterogeneous types of user clicks, which is important in our research.

3.4 Changing Interests

User interests changes over time. This makes user modelling difficult. In news reading short-term and long-term interests have been defined[13]. Short-term relates to hot topics like big events and is rapidly changing. Long-term refers the interests of the user without these special events. In [13] they create separate models for long-term and short-term interests. A user profile consists of a collection of articles rated by the user on a scale between 0 and 1 (1 being highly rated). The short-term user profiles

consists of recent events and a nearest neighbour algorithmic approach on the highest rated article is used to recommend new articles. The long-term profile consists of all article ratings and a naïve Bayesian classifier is used to decide what article to recommend.

Liu et al.[20] did some research regarding this subject. Their findings implied that the user interest do change over time. They also emphasized that the general public reflects the current trends. Location of the news and the user's also play a big role when considering such trends. In their research, the long-term user profile consisted of vectors that weighted the categories:

$$\vec{C} = (\frac{N_1}{N_{total}}, \frac{N_2}{N_{total}}, ..., \frac{N_n}{N_{total}})$$

where $\vec{C}$ is the category vector, N_n is the amount of clicks within the specific category, and N_{total} is the total amount of clicks.

To calculate the short-term interest they used the fact that the general population in a area partly represents a specific individual's interest. With short time intervals (e.g. 1 hour) they calculated the general population interest in a specific category to recommend news to individual users. This might imply that content-based methods captures the long-term interests, while collaborative approaches can be used efficiently to calculate short-term interests.

Chapter 4

Big Data Environment

This chapter introduces some of the tools and systems important for this thesis. First, big data in general is described. Second off is a large scale computational platform named Apache Hadoop that uses the MapReduce programming model for parallel processing. The third section describes MongoDB, a database system that is used as a log storage. The two last chapters explains some other related tools that did not get very much attention in the implementation, but should be considered when the system goes into production.

4.1 Big Data

"Big Data" has become a buzzword in most industries. The direct meaning of big data is that the world as whole stores and generates unbelievable amounts of data. According to IBM, every day, $2.5 \cdot 10^{18}$ bytes of data is created[25]. 90% of existing data has been generated during the last two years. These estimates imply an exploding and exponential growth in data stored in the world. Every single search made in search engines is stored; the smallest interaction on social platforms gets stored; videos and digital photos are also in this equation. To phrase the reasoning short: data is generated from everywhere.

Historically, the processing power has limited the possibilities with using large data sets for constructive and revenue increasing ways. As the years has passed, processing power has become cheaper and cheaper. Moore's law[26] states that the number of transistors integrated circuits doubles every second year. The performance of processors has improved even better than this due to faster transistors. This has opened the opportunity to exploit data in new and innovative ways. The general tendency in most businesses is to use big data approaches to get a competitive advantage. This is done by extracting meaning from the stored data.

4.2 Hadoop

Apache Hadoop[27] is an open-source implementation of MapReduce. It is the most popular implementation of the programming model. Hadoop is the definition of a large scale parallel cloud environment for parallel processing of unstructured, semistructured, and structured data types.

4.2.1 MapReduce

MapReduce [28] is a framework for parallel processing. The intended use of MapReduce is on computer clusters. The main intention of the framework is to simplify processing of enormous data sets ("Big Data"). The model has gotten very popular because users find the programming model easy to use. Apache Hadoop is considered as one of the most worthy implementations.

The MapReduce framework consists of two main phases, these are the Map phase and the Reduce phase. The Map phase is when a user defined map function is executed. The function takes in key/value pairs as input and performs some action on the data to produce intermediate results. The results is also on the form key/value. The Reduce phase takes place when a user specified reduce function runs on the intermediate data and reduces the result to a small amount of outputs, typically zero or one output value. The data from the Map phase is shuffled to machines in the reduce phase.

The dataflow through the MapReduce framework can be summarized in six steps:

1. **Input reader:** The input reader partitions the input into splits. Splits are the data unit the map task does the processing on. The typical size ranges from 16 MB to 128 MB. The framework assigns one split to each map task. The input reader reads data from stable storage (e.g. Google File System or Hadoop's HDFS)and generates key/value pairs. The file system is usually distributed.

2. **Map function:** The input to a map function is a collection of key/value pairs which are generated by the input reader. Each pair is then processed by the map function logic and new key/value pairs are generated. Usually a small amout of pairs comes from the execution. The map function outputs spill files in the end are sorted and merged into one structured file. An example of the mapper is given in Figure 4.1. The map function maps the input values to their specific output values.

3. **Combiner function:** The combiner function is optional. The main use of this function is to get better performance. When there are many repetitions in the

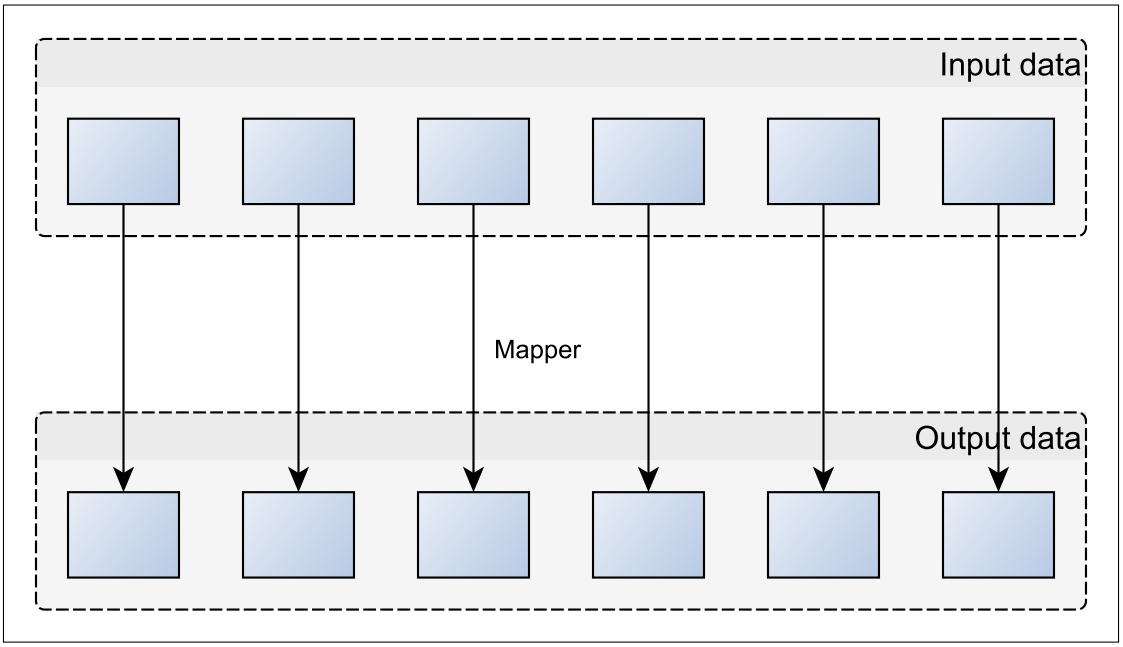

Figure 4.1: Mapping function

halfway results and the implementation of the reduce function is considered commutative and exclusive. When these preconditions are true, the combiner function will partially merge the map results making it so pairs with identical keys will be processed as one collection.

4. **Partition function:** Each Map function output is allocated to a particular reducer by the application's partition function for sharding purposes. The partition function is given the key and the number of reducers and returns the index of the desired reduce. A typical default is to hash the key and use the hash value modulo the number of reducers. It is important to pick a partition function that gives an approximately uniform distribution of data per shard for load-balancing purposes, otherwise the MapReduce operation can be held up waiting for slow reducers (reducers assigned more than their share of data) to finish. Between the map and reduce stages, the data is shuffled (parallel-sorted / exchanged between nodes) in order to move the data from the map node that produced it to the shard in which it will be reduced. The shuffle can sometimes take longer than the computation time depending on network bandwidth, CPU speeds, data produced and time taken by map and reduce computations.

5. **Reduce function:** Once, for each unique key, the reduce function is called. All the associated values for a specific key are processed as a group. The reduce function combines the data according to the implemented logic and outputs the result. Figure 4.2 visualizes how the reduce function works. It aggregates the data coupled with a key into a summary of all the input data.

6. **Output writer:** The output writer simply stores the results of the reduce function in to stable storage as a file. The writer can also be modified to suit storing facilities, an example is to store results in a standard database.

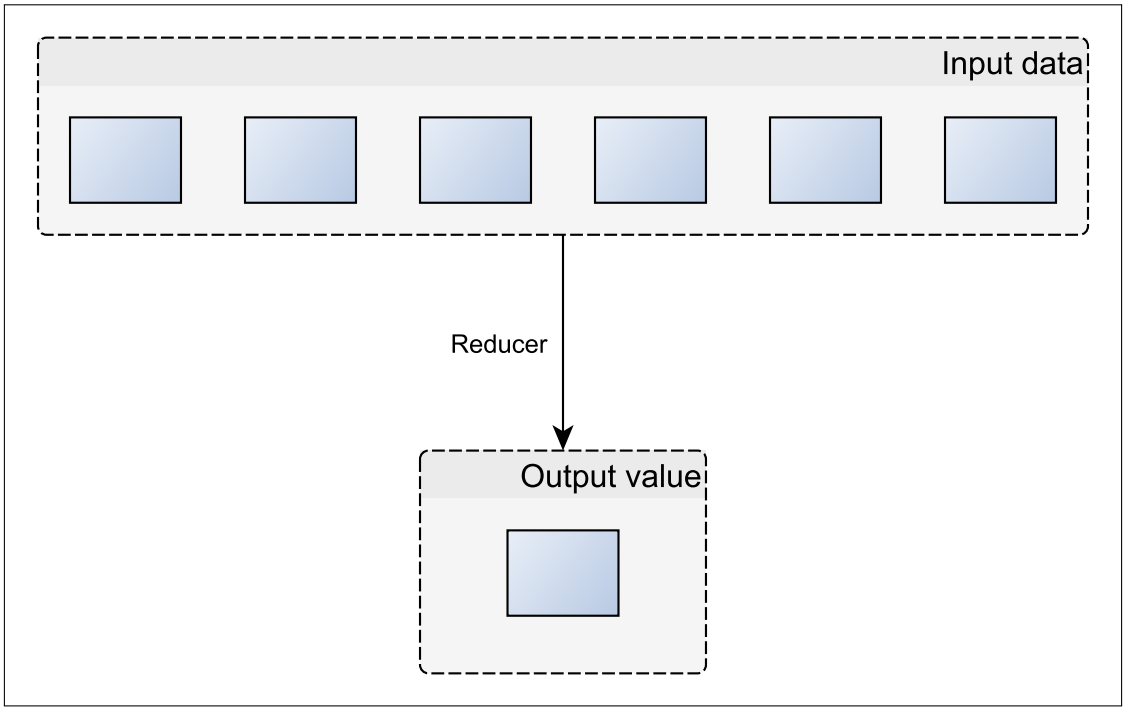

Figure 4.2: Reducer function

4.2.2 Intended Arena

Hadoop is a batch infrastructure with the main intention to run large scale data. It can run on a single machine, but the only activity benefitting from that is the development process of MapReduce jobs. There is a substantial startup overhead in Hadoop. Small problems, which in this context means examples similar to processing some gigabytes of data running on 4-8 nodes, would most likely benefit from another parallel processing environment. This is because the overhead would be substantial in comparison with the processing in a small scaled project. The fraction of influence that the overhead introduces decreases as the problem gets bigger. The real benefit of Hadoop is when the problems get gigantic. The true power of Hadoop shows when processing problems in the tera or petabyte class running on for instance 100 or 1000 nodes.

Hadoop provides a simple programming model that scales in a flat manner. The engineering effort required for regular approaches (e.g. MPI) increases exponentially when the problems scale out of bounds. Hadoop jobs running on 4 nodes would run equally well on 1000 nodes with minimum engineering work needed. One of the reasons for

this is the special purpose distributed file system (HDFS). The filesystem distributes data automatically between nodes and the Hadoop system takes care of the communication between nodes. The only part a developer needs to worry about is writing a map function, a reduce function, and a driver function which is a simple initialization method of a job. Of course the input data needs a definition as well.

4.2.3 HDFS

One of the biggest selling points of the Hadoop system is the special purpose file system, the Hadoop distributed file system (HDFS)[29]. A distributed file system's main purpose is to store big data files. The data should be accessible by multiple clients distributed on a local or wide area network. The regular types of distributed file systems (e.g. NFS) are not well suited for running MapReduce jobs. They usually do not have mechanisms to equally distribute data over nodes and are not very reliable.

HDFS was based on a file system proposed by Google, the Google File System (GFS)[30]. The most important design goal for HDFS is to allow storing very much data. This is done by spreading data over the different nodes in the system. This implies that files larger than a single node's storage will also fit in the system. Another design goal is reliability. More specifically, node malfunction should not influence data availability. Redundancy of data is the solution to this. In a standard configuration each file is stored three times, but it is possible to alter the configuration. The system is also designed to fast access on every scale and more nodes should be the basis of scaling the number of clients able to use the service. And of course it is designed to be complementing the Hadoop MapReduce engine well. HDFS should not be considered as a general purpose file system. Random access is not very efficient as the system is designed for long sequential streams of data.

The basic building piece in HDFS is blocks. A file is split up into fixed size blocks that is spread across the cluster. A single machine that stores data is named a DataNode. Accessing a single file might require cooperation between several machines because of distribution of data. The standard block size in HDFS is 64 MB which is very large when considering standard file systems. The intention is to minimize the needed metadata and faster streaming of data due to less jumping between blocks.

The metadata of the files and blocks is handled by a single machine called the NameNode. The data of the NameNode is also reliable due to redundancy with secondary NameNodes. The reason for centralizing all the metadata is that files follow a write once and read many model. Metadata modification can be done by many machines at in parallel. A single node in charge preserves synchronization.

The basic data flow in HDFS is initiated by retrieval of a file's location by a client. A file can be considered as a record of block locations pointing to the specific DataNodes that

hold the different blocks. This is the only function of the NameNode in the data transfer process; the remaining process parts are directly between the specific client and the specific DataNode. This might also be done in parallel including several DataNodes. An illustration of the HDFS architecture is given in Figure 4.3. The file redundancy in this figure is 2 and there are a secondary NameNode that is a mirror of the main NameNode.

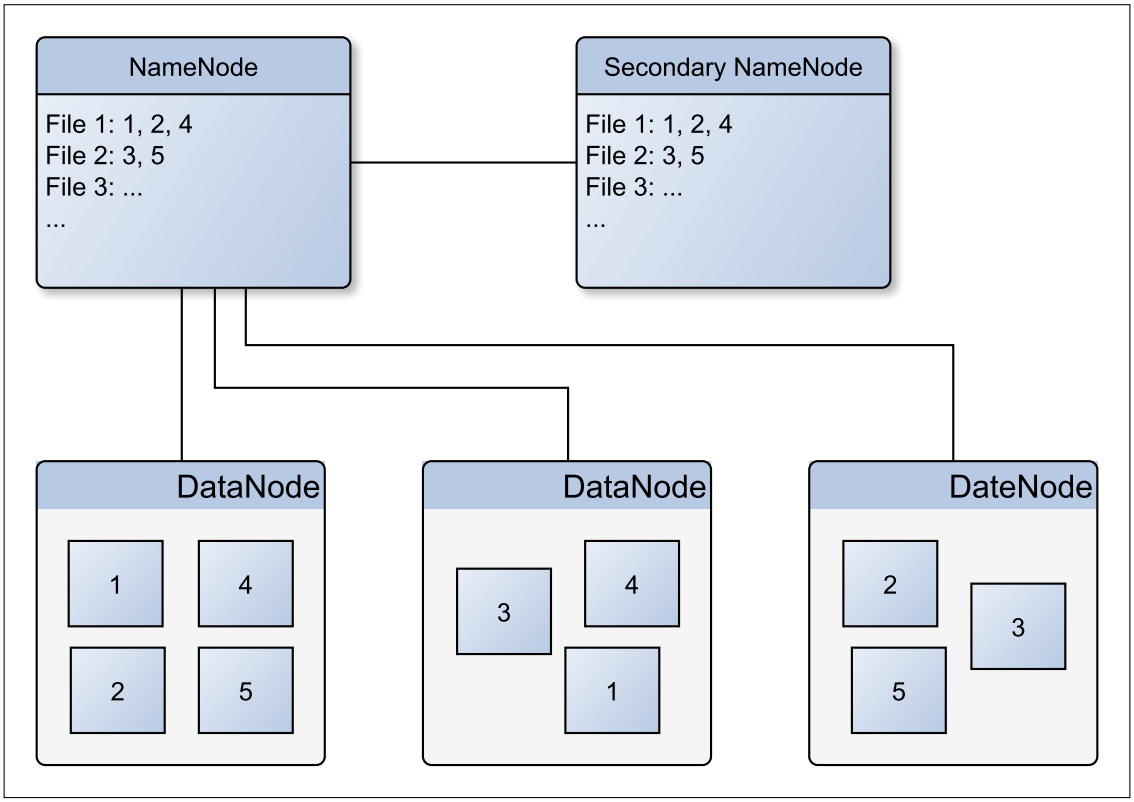

Figure 4.3: HDFS architecture.

4.2.4 Mapreduce Engine

In the most basic sense, one can view Hadoop MapReduce as a black stateless box. The only function of the MapReduce engine in Hadoop is to transform input data lists into output data lists. Data in Hadoop is considered immutable. This implies that Hadoop can only write new output files and not overwrite old files.

A special node named the JobTracker node plays a crucial part in Hadoop. The JobTracker node is where clients add jobs to the system. Clients also use it for getting job information. It can be considered as a single point of failure in the system. If the JobTracker node fails, all ongoing jobs will halt and might have to be restarted.

It distributes the work to TaskTracker nodes in the system. A TaskTracker node can be considered as normal processing node. An architectural goal that minimizes communication cost is location of processing and data. Processing on nodes that have have the needed data minimizes the communication. If this is not a possible option, then nodes on the same rack (nearby nodes) are chosen. The main intention of the JobTracker node is to keep the processing as close to the data as possible while balancing the work evenly between TaskTracker nodes. If a TaskTracker node fails, then the JobTracker relocates the partial job to new available TaskTracker nodes. If the JobTracker fails then a check pointing process keeps track of the system state, thus making it less critical for the overall system.

The MapReduce architecture is illustrated in Figure 4.4. The illustrastion has three TaskTracker nodes and depicts that the entry points for clients is the JobTracker. The HDFS and the MapReduce architecture can together be considered as a layered architecture where HDFS is the lowest level and the MapReduce lies on top. This implies that a DataNode can also be a JobTracker but if the viewpoint is from the HDFS side, then it is a DataNode and vice versa.

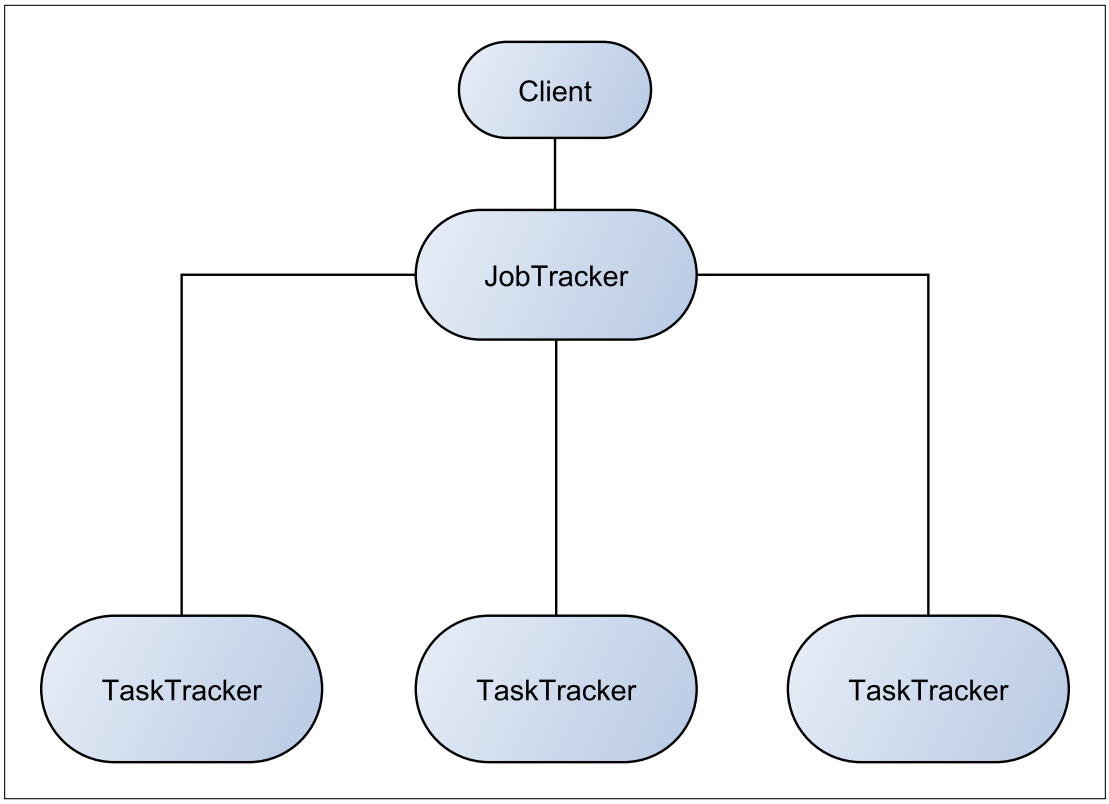

Figure 4.4: MapReduce architecture

4.2.5 The Hadoop Environment

HDFS and MapReduce is considered as the basic components of Hadoop. There also exist other systems and libraries that are widely used. The Hadoop platform has it's base in a big open souce community and thus new systems suited for interconnection with Hadoop and libraries focusing on other apects are added consistently. The most imporant are surveyed in a short fashion below to give an overview of the wide range of possibilites there are.

Avro is a data serialization system. The services provided consists of remote procedure calls, integration with dynamic languages, rich data structures, a compact binary data forma, and a container file to store persistent data.

Cassandra is a distributed database management system. The design goals are to store and handle a large scale of database data in a distributed and reliable fashion.

Chukwa is a data collection system for monitoring large distributed systems. It is integrated with the Hadoop framework and uses HDFS for data storing. It also includes toolkits for working with and analyzing results.

Flume is a sevice designed for delivery of data from applications to Hadoop's HDFS. It is a distributed service for collecting, aggregating, and relocateing big amounts of log data. It is considered robust and fault tolerant with strong reliability and recovery mechanisms.

HBase is an open-source implementation based on Google's Bigtable[31]. It is a extension of the HDFS. Storing in HBase is structured as non-relational colums. MapReduce jobs in Hadoop can use HBase tables as input or output.

Hive[32] extends Hadoop giving it data warehouse infrastructure. It was originally developed by Facebook. HiveQL is the query language, which resembles SQL to some degree. The queries are compiled and executed as MapReducs jobs on Hadoop.

Oozie is a workflow engine for Hadoop. The intention is to allow performingg workflow management and coordination to manage jobs running on Hadoop. Oozie workflow jobs can be considered as Directed Acyclical Graphs (DAGs) of actions. It is integrated with the rest of the Hadoop environment and support most types of jobs Hadoop can perform.

Pig[33] is a high-level framework for creating MapReduce jobs. Pig Latin, the language used, and its execution environment is the contents of Pig. Pig Latin is a procedural scripting language and allows abstraction of regualar Java MapReduce jobs. An important property is UDF (User Defined Functions) which can extend Pig Latin. They can be written in a regular programming language.

Sqoop is a framework designed to transfer raw data between Apache Hadoop and structured data stores such as relational databases in an efficient manner. This includes imports from external data sources, such as data warehouse information, into the Hadoop environment (HDFS or HBase etc.).

Mahout is a project driven to produce scalable implementations of machine learning algorithms. It is a rapidly growing framwork with a strong community behind it.

Zookeeper provides a distributed service for configuration information, naming, distributed synchronization, and providing group services for large distributed systems[34]. ZooKeeper was a sub project of Hadoop but is now an own project since this kind of services are needed for most distributed systems.

4.2.6 Limitations

Some of the main limitation of Hadoop and MapReduce are discussed below.

Security

The main architectural goals for Hadoop are reliability and high fault-tolerance. Security has not had much impact of the development of the framework. This means that relying on the basic Hadoop implementation would not ensure that communication between node-machines in the system is transferred in a safe way. People sniffing on the network would be able to use the communication data.

Costly communication

The output of the Map function can be large, thus the time needed for communication can scale in the same manner. This issue usually comes from badly partitioned data. If different data related to each other could be distributed on topologically close nodes (preferably on the same node), then the problem would be minimized. Unfortunately this is not a service Hadoop provides. The problem is left to the developing engineer, but it is a complex problem to partition the data optimally. This is due to the SIMD type architecture of the Hadoop system. Communication between nodes is supposed to be restricted to make the programming model easier to use. This also implies that embarrassingly parallel problems are what are best suited for this type of application.

Iteration

It is possible to chain MapReduce jobs in a serial fashion. This approach is needed for iterative programming. It is not considered as a straight forward task and it is also costly because of the overhead introduced by many consecutive Map phases.

4.3 MongoDB

MongoDB[35] is a NoSQL[36] type of database. NoSQL databases are designed to be fast when reading and writing data, low cost, scalable, and support massive data amounts. An example of a downside with NoSQL databases is that it does not support SQL, which is considered industry standard. Also, the different NoSQL options is not considered mature since it has gained popularity recently and the implementations are not very old.

MongoDB is a schema-free document type of NoSQL database. A document type database is designed to store big amounts of data and to have efficient queries. Concurrent read and write performance is not the main goals of this class of databases. MongoDB databases reside on a MongoDB server that can host more than one of such databases which are independent and stored separately by the MongoDB server. A database contains one or more collections consisting of documents. Collections inside databases are referred to by the MongoDB manual as named groupings of documents. MongoDB persists documents by a format called BSON which is very similar to JSON but in a binary representation for reasons of efficiency and because of additional data types compared to JSON

4.4 Log Data Transferring Tools

This section focuses on two tools designed for making data transfers to the Hadoop environment better. The first tool discussed is Apache Flume, which is designed for transferring log data from one or several sources to one or several storages. The second one is the MongoDB + Hadoop Connector, which is a MongoDB specific tool.

4.4.1 Apache Flume

Apache Flume[37] is a large scale log aggregation framework. The framework is designed for delivery of data from external applications to an external data store. It is a distributed service for collecting, aggregating, and relocating big amounts of log data.

It is considered robust and fault tolerant with strong reliability and recovery mechanisms. An abstract architectural view of Flume is given in Figure 4.5. A short description of Flume is that it listens to a port for incoming event that it understands. Then it stores the event temporarily. Finally, when the external storage is ready to accept new data, then it forwards the temporal data into the external store. The external source could for instance be a web application and the external storage could be Hadoop's HDFS.

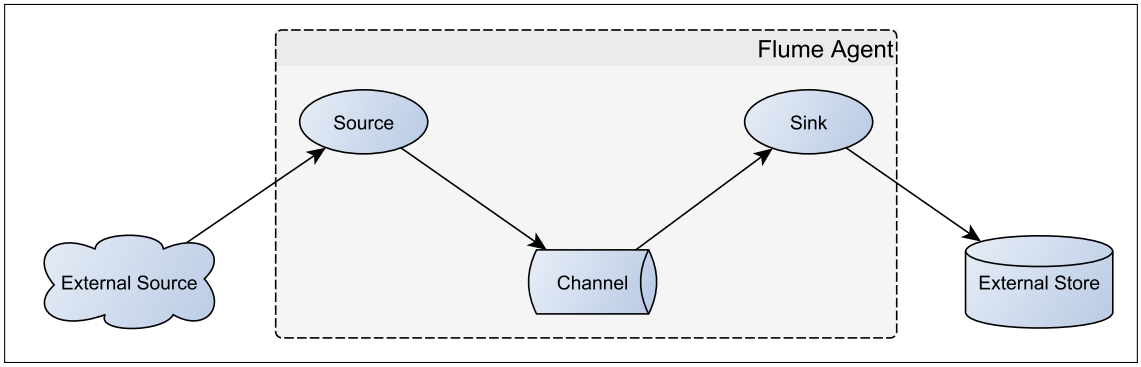

Figure 4.5: Flume Architecture

4.4.2 MongoDB+Hadoop Connector

The Mongo+Hadoop Connector[38] is a collection of plugins designed for Hadoop to improve the connection with MongoDB. A basic Hadoop installation can only read text files from the HDFS without any alteration. The connector makes it possible to feed input to MapReduce jobs directly from MongoDB without going through the HDFS. The connector is written in Java and is basically a translator of MongoDB data. The basic functionality is that the connector forwards a single JSON file from the source collection to the map function. The connector also supports streaming functionality implying that a MapReduce job could start before all data is finished loaded.

4.5 Scheduling Tools

In this section two alternatives for scheduling batch jobs in Hadoop is considered. The first is Apache Oozie which is designed to solve for this type of problem for Hadoop. The second one is a more general Unix-based tool called cron.

4.5.1 Apache Oozie

Apache Oozie is a workflow engine for Hadoop. It acts as a middle-man between the user and Hadoop. The intention is to allow performing workflow management and coordination to manage jobs running on Hadoop. Oozie workflow jobs can be considered as Directed Acyclic Graphs (DAGs) of actions. It is integrated with the rest of the Hadoop environment and support most types of jobs Hadoop can perform.

Oozie is designed to handle two scenarios based on time and on data availability. Workflows that should start periodically (e.g. once a day) and jobs that should start when the needed data is available are these scenarios. The Oozie DAGs are expressed in XML and it supports chaining MapReduce jobs, running jobs concurrently, and also allows to decide between jobs.

When using Oozie, the driver method becomes unnecessary, since the configuration properties should be stated in the XML file. An example of a workflow is given in Figure 4.6. This illustration shows the entry point followed by two MapReduce jobs running in sequence, where the second job is dependent on the results from the first. There is also possible to state what actions to execute when errors happens as shown.

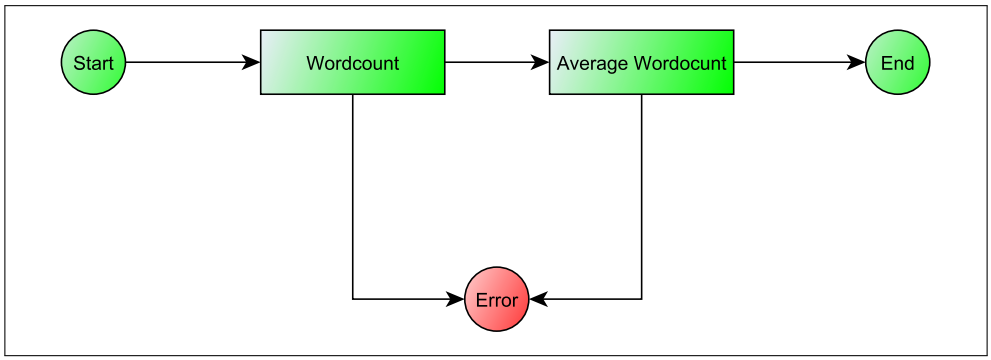

Figure 4.6: Example Oozie workflow

4.5.2 cron

In Unix-based operating systems, which is the usual Hadoop hosting operating systems, there exist a time based job scheduler called cron. The basic functionality is that it allows users to create text files with commands or scripts to be run at given times. This could be utilized for scheduling jobs in Hadoop, but might be best suited for development use and not in production.

Part III

Realization

Chapter 5

Approach

This chapter focuses on the approach for creating the user profiles. Our recommender system is classified in the first sub section. The environment surrounding the scoped module is addressed in the second sub chapter, with the neighbouring interfaces emphasized. The general idea for the solution is explained after this. The fifth sub section explains the creation of logs and what data is available in the logs.

5.1 Recommender System Type

The SmartMedia project is a recommender system. The use case consists of users reading new on the mobile that is recommended to them, with as little explicit user involvement to recommend the new. The system classified in accordance with the framework[2] described in an earlier chapter is given in Tables 5.1-5.3.

Table 5.1: System technical design space

Contents of recommendation	User click logs; possible to override
Explicit entry?	Implicit
Anonymous?	Anonymous
Aggregation	Personalized weighting based on click logs; collaborative approaches
Use of recommendations	Display of news articles

Table 5.1 describes the technical class of the system. The recommendations are based on user click logs that is gathered without prompting the user for the information. This is kind of unusual compared to standard recommender systems where users give infor-

mation. It is possible to override the recommendation process to influence the system, but this is just considered as an alternative.

The aggregation part of the system is where this work is located. The intention is to use a hybrid approach of content-based filtering and collaborative filtering to recommend news. Specifically this work relates to the content-based approach, but it could be used in the collaborative approaches as well. This is due to the fact that both approaches needs good user models. The intention is to make user profiles to be used in content-based filtering.

Table 5.2: Description of items

Type of items	Internet news articles
How many	Thousands per day
Lifetime	1-2 weeks
Cost structure	Bad recommendation unimportant

Table 5.2 describes the items that are to be recommended. The items are news articles published by Norwegian news sites. At the time being, around 2000 articles are generated per day and the lifetime of the articles are considered to be very short. If the recommendation misses, it is not that important, since the user could always just browse ahead. Several bad recommendations in sequence would be unwanted.

Table 5.3: Description of actors

Recommenders	Application users
Density of recommendations	Dense
Consumers	Application users
Consumer taste variability	High

In Table 5.3 the actors are defined. It is basically the users of the system. From a content-based perspective a specific user recommends news articles to himself. From a collaborative perspective the users recommends articles to himself and to others. The density of recommendations is considered to be high the pace of browsing news articles are fast and the variability of the users are also high.

5.2 Environment overview

To give a perspective of where to place the module, a visualized version of the system is given in Figure 5.1. This figure only shows the parts of that system that are crucial for

the module constructed. The arrows represents how the data normally flows within the system. The mobile application forwards user act data, through a mobile connection to the internet, to a centralized server. This server stores the data in a database. The data is then gathered by the Hadoop cluster and processed into profiles. After this, the profiles are sent to the centralized server again for further usage. This implies that the only communication points visual to the module are the log database and the middleware server.

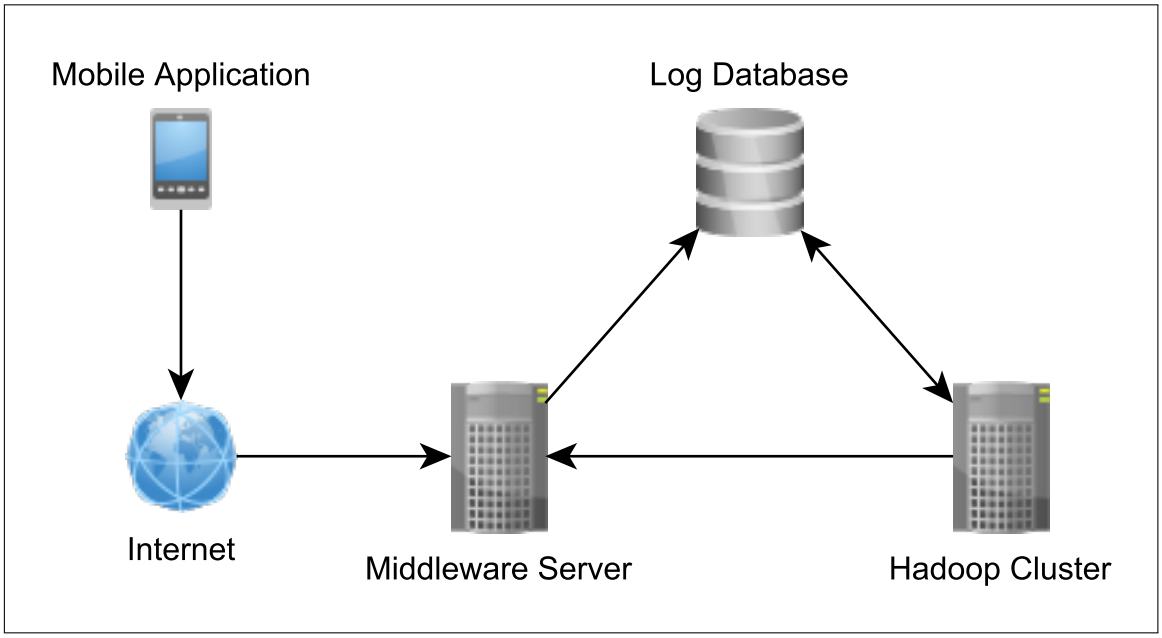

Figure 5.1: Environment overview

5.3 General Idea

From a black-box perspective, we could say that the input to the module is a set of new user acts and the output are user profiles. With a little more detail, the approach taken to create user profiles could be stated in the following steps:

1. Mobile client forwards user acts as they occur to a log.
2. Periodically , a Hadoop installation gathers the newly logged data into the Hadoop environment.
3. The Hadoop installation runs a series of MapReduce jobs.
4. The output of the MapReduce jobs is a set of vectors coupled with the unique user IDs.

The specific system module in scope of this thesis, was a solution that solved the problems stated in step 2 and 3. The reason for introducing MongoDB was an architectural decision made by the SmartMedia project. The implementation of these steps will be explained on a low level in the next chapter.

5.4 Creation of Logs

Whenever a user does an action within the mobile application, the user act is forwarded through a REST API to a MongoDB installation. The data stored per user act is displayed in Table 5.4. The left column consists of the different data fields found in the database and the right column displays the description of the data within the fields.

Table 5.4: MongoDB log data fields

Data field	**Description**
ID	Unique identifier for the user act
User ID	ID of the user performing the act
Article ID	ID of the article coupled with the act
User Act Type	The type of user act
Timestamp	The time the act occurred
Geographical Location	The coordinates of the client at the time of the act
Tags	Keywords related to the article coupled with the act
Categories	The categories classifying the in scope article
Properties	Extra field for additional data

The fields that are important for the module are the user ID, the user act type, the tags, the categories and the properties. The user ID field is key to couple the results with the specific user. User act type is important since knowing what actions that have been performed would allow weighing different acts with regards to what influence they should have on the results. The tags and the categories gives insight into what the profile of the user should since it allows coupling between users and their actions. The timestamps also have a function when gathering and using data in time frames. In the current state of the system, the only use for the properties field is to store duration of timed acts (e.g. time spent viewing an article). The article information (article ID, tags and categories) are not present in acts not involving any article.

The different types of defined user acts are listed in Table 5.5. Each user act follows from a specific action the user perform in the system. Figure 5.2 shows example view from the mobile application. The screenshot on the left is the preview screen for an article and the right one is an instance of the article view. Scrolling over the preview screen will produce an user act that is logs the time the user spends in the view for

that article. The right hand session could produce a series of user acts for the specific article. Preview view time, opened article, and article view time could be the result of opening an article.

Listing 5.1 displays a textual example of an entry in the log database. In this specific case the user shared an article through email. The tags of the article are consists of 6 objects and it is in the news category. Listing 5.2 show a preview view act and Listing 5.3 displays an article view act. These types of acts are generetad when the users are in the views in Figure 5.2. In these acts the properties have a duration that indicates how long the user spent in the specific view.

Table 5.5: User act types

User act type	**Description**
Opened article view	The user opened the full text version of this article
Time spent article view	The time the user spent viewing this article
Time spent preview	The time the user spent viewing the preview of this article
Clicked category	The user clicked on a category
Shared twitter	The user shared the this article via twitter
Shared facebook	The user shared the this article via facebook
Shared mail	The user shared the this article via mail
Starred article	The user added this article to favourites
Viewed map	The user viewed the location this article
Vieved entity view	The user viewed the tags this article
Clicked entity	The user clicked on a tag coupled with this article
Viewed similar article	The user viewed an article similar to this article

Listing 5.1: User act dump

```
{
"_id" : "F4E43303-27D1-4E5B-95A7-2285EC026D9E" ,
"articleId" : "1560609210" ,
"userId" : "bf4d2b7adec01da0ddc8c331788bcdc6" ,
"eventType" : "SHARED_MAIL" ,
"timestamp" : { "\$date" : "1970-01-16T19:10:12.909Z" } ,
"geoLocation" :
        { "name" : "" ,
        "type" : "" ,
        "longitude" : 10.338135 ,
        "latitude" : 63.40136 } ,
"tags" :
        [ "frode granhus" ,
          "henning juul" ,
```

Figure 5.2: Examples of user acts

```
            "flere norske" ,
            "Orkdal" ,
            "Sor-Trondelag" ,
            "Norway"
        ] ,
"categories" : [ "NEWS"]
}
```

Listing 5.2: Time spent preview user act

```
{
"_id" : "7791A4D5-9550-4FEF-9A18-5FDD05CECC3B" ,
"articleId" : "441816664" ,
"userId" : "bf4d2b7adec01da0ddc8c331788bcdc6" ,
"eventType" : "TIME_SPENT_PREVIEW" ,
```

```
"timestamp" : { "$date" : "2013-06-02T16:39:53.928Z"} ,
"geoLocation" :
        { "name" : "" ,
        "type" : "" ,
        "longitude" : 8.00354 ,
        "latitude" : 58.138821 } ,
"properties" : { "duration" : "1.259043"} ,
"tags" :
        [ "politiet" ,
        "Sogne" ,
        "Norway" ,
        "Vest-Agder" ,
        "Kristiansand" ,
        "Norway" ,
        "Vest-Agder"] ,
"categories" : [ "NEWS"]
}
```


Listing 5.3: Time spent article user act


```
{
"_id" : "241B50BE-DFF5-4AAB-A12D-98D4A4606028" ,
"articleId" : "318218311" ,
"userId" : "bf4d2b7adec01da0ddc8c331788bcdc6" ,
"eventType" : "TIME_SPENT_ARTICLE_VIEW" ,
"timestamp" : { "$date" : "2013-06-02T16:41:15.511Z"} ,
"geoLocation" :
        { "name" : "" ,
        "type" : "" ,
        "longitude" : 8.00354 ,
        "latitude" : 58.138821} ,
"properties" : { "duration" : "1.427272"} ,
"tags" :
        [ "agder politidistrikt" ,
        "havnet" ,
        "satt" ,
        "rebelsk mannen" ,
        "operasjonsleder" ,
        "kristiansand slo" ,
        "politiet" ,
        "kristiansand skallet" ,
        "Maharashtra" ,
        "India" ,
```

```
        "Egersund" ,
        "Rogaland" ,
        "Norway" ] ,
"categories" : [ "NEWS"]}
```

5.5 User Profile Vectors

The user model we applied is an individual and implicit type of model. It is individual since it has one unique interface for each single user and the similarity between the users are not considered at all. It is implicit since it only considers user actions to be generated. The user have the possibility to give information regarding how he would like to be modelled, but this is not considered at this level of the system.

The most precise definition of a user profile in our domain is characterizing it as a personalization of all the click log information. The representation of user profiles were chosen to consist of vectors. This is due to that it is a flexible and easy-to-understand way of making a digital representation of a user[9]. It is also computationally efficient and integrates well with recommendation strategies.

Our solution to the problem was using two vectors with paired valued elements. A vector $\vec{C}$ representing categories and a vector $\vec{K}$ representing keywords. A generalized view of these vectors are given below:

$$\vec{C} =< (c_1, v_1), (c_2, v_2), ..., (c_n, v_n) >$$
$$\vec{K} =< (k_1, v_1), (k_2, v_2), ..., (k_n, v_n) >$$

The c and k entries are text elements and the v elements are the numerical weight value of the of the specific category or keyword. The n variable represent the index number. In the category vector this will be a finite static sized number. The keyword vector is dynamic sized and could in theory have infinite entries. New elements in the keyword vector are added to the end. Proposed initialized vectors are given below as examples taken from the final system:

$$\vec{C} =< ("News", 100.0), ("Sports", 13.1), ("Travelling", 7.5),$$
$$("Entertainment", 80.3), ("Domestic", 23.6), ("Technology", 3.14) >$$

$$\vec{K} =< ("46354", 1.866), ("Akatsi", 1.866), ("Forbruker", 1.866),$$
$$("mortenthomassen", 0.933), ("jeg", 0.9444), ("Lodzkie", 19.633),$$

$$("Strømstad", 1.8666, ("PaysdelaLoire", 1.866), ("150", 2.833),$$
$$("RueBenjaminFranklin", 1.866), ("Nordrhein-Westfalen", 1.866),$$
$$("rachelnordtømme", 1.866), ("Nordland", 7.555), (" - ", 0.933), ("3", 2.811),$$
$$("ST13", 19.633), ("2", 81.844), ("BestWesternAnkerHotel", 1.888), ... >$$

The keyword vector is clearly larger than the category vector. The keyword vector also has some odd entries (e.g. "-" or "3"), which implies that there are some weaknesses in the noise handling in the module that tags articles.

The vectors are normalized as a number between 0 and 100. The normalization strategy is to use the largest weighted category or keyword and making all the weights a fraction of this. Then the results are multiplied by 100. This means that the heaviest weighted entities will always have 100 on the scale and all that are less will also be less than 100. A bigger number coupled with the category or keyword means that it is more important. The main reason for choosing this approach was that it suited the content-based filtering module that are to use the vectors.

Another way of normalizing the vector could be to sum all the weights and divide each weight with the sum. This would make all the weights a number between 0 and 1. The reason for choosing the first approach is because it is simple to implement and the weights will not all be very smart fractions. Multiplying by 100 is done out of personal favouring as it is considered more intuitive for human inspection.

Chapter 6

Implementation

This chapter covers the implementation done in the project. Algorithms and code are stated through listings in a java-like pseudocode to explain the logic. The first subchapter explains the intentions and scope of the implementation. Subsection two gives a brief overview. The three following parts focuses the the code that has been written. The calibrating factors are summarized in the last sub chapter.

6.1 Intention and Scope

The main intention for the implementation was to provide a framework for creating user profiles. There are a lot of strategies and algorithms that can be utilized to achieve good models of the users, but evaluation and verification of the approaches are difficult when real data are unavailable. Therefore, instead of constructing a complete final solution, the scope is narrowed down to constructing a framework that requires tuning in coherence with user data before it is put in production.

6.2 Overview

The solution developed during this work can be described as a workflow of processes. The processes ends with the user profiles as the output. A visual version of the workflow is given, as an activity diagram in Figure 6.1.

The black dot is the entry point of the of the compounded job. The black dot with a circle around it means that the activity is finished. The rounded rectangles are processes that needs to be done. When two lines exits an entity and two lines enters an entity, it

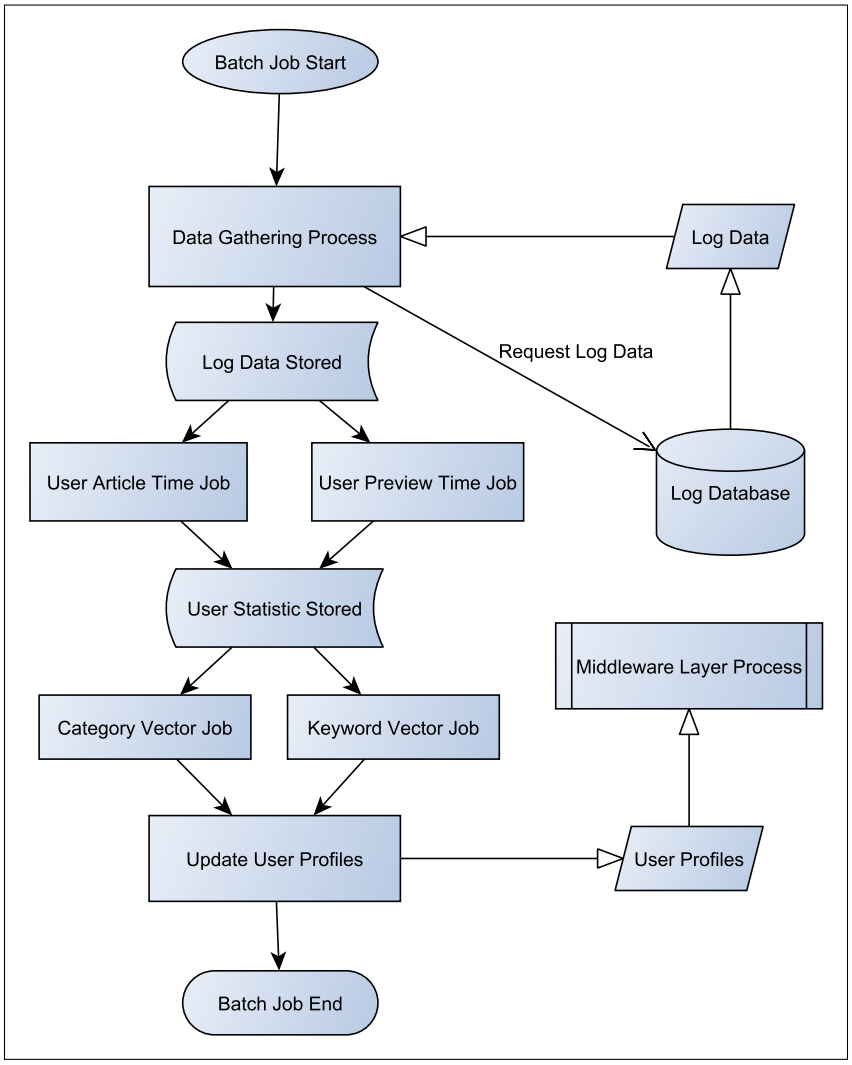

Figure 6.1: Batch job overview

entails, respectively, forks and joins in the activity. Finally, the lines show the order of execution.

In a literal description of the activity, one can say that it starts by gathering new log data. Then statistics per user is computed. This consists of two different jobs that can be executed in parallel. Following this, the user profiles are generated by construction of the two vectors it consists of. The latter two processes can also be run in parallel. The output is forwarded to the centralized system through REST API functions. The batch job is intended to run in a periodic fashion.

Figure 6.2 displays a package diagram of the implementation. I consists of two packages. The designated functionality of the left package is to provide the computational jobs. The right package is implemented as an interface between the log storage and

the Hadoop environment.

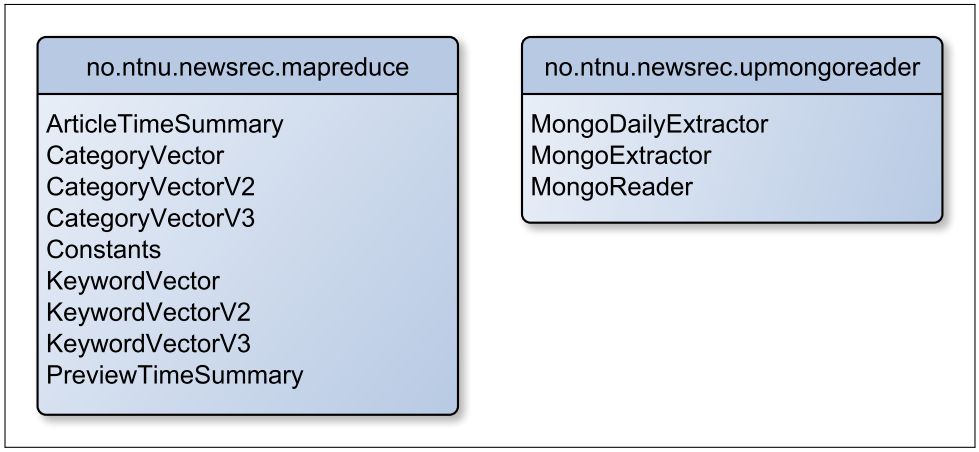

Figure 6.2: Package diagram

The classes of the packages are also visualized in the diagram. The classes in the mapreduce package are mainly the specific MapReduce jobs. It also contains a class with constants common for the jobs. The ArticleTimeSummary and PreviewTimeSummary classes are the jobs that computes statistics related to the users. The CategoryVector and KeywordVector classes contains jobs that calculates the vectors. Each vector has three different versions. These classes are built up by 200-300 lines of code. This is the reason they are described in pseudocode later in this chapter.

The upmongoreader package consists of three classes. The MongoReader class is the interface to the log database. The MongoExtractor class extracts data from the database based on a time interval and the MongoDailyExtractor only gathers data that has been added since the last run. Daily is part of the name because the intention is to run the program on a daily basis. Othe rfrequencies that are more or less rapid should also be possible.

6.3 Importing Data to the Hadoop Environment

Two different programs was created to extract the data from the MongoDB installation to the Hadoop environment. The basic process of this action is to use the MongoDB API to transfer the data over a connecting network. Other approaches for gaining data will is discussed in the discussion chapter.

The first program is taking a simple dump of the collection aimed for (in this case the log collection). A time-range is used to gather the data that has been stored within this range.The data is stored as a text file in the HDFS. This approach will not be optimal when the data amount stored scales, but the program could have its use when first

initializing a new Hadoop installation, when recovering a faulty one, or when focusing on data from specific time intervals.

The second program uses find queries to locate data that has arrived since the last updated extraction was performed. This is the optimal way of gathering the data when using the MongoDB API directly, since the past data is already located on the HDFS and this minimizes the data transfer over the network and time consumption. The last time an update was made is stored on the HDFS and updated when new data has been aquired.

These programs can be executed the Hadoop environment in the same fashion as MapReduce jobs, even though they are not in that category. Two other approaches to filling the HDFS with data will be mapped out in the discussion.

Listing 5.1 shows an example of a single line that the program writes for a specific user act. This is the basic structure the MapReduce jobs work with. It is in JSON format as these types of lines are parsed further down in the activity. The figure has additional spacing to keep it easy to read.

6.4 User Statistics Jobs

Two of the user act types is based on timing: time spent in article view and time spent in preview view. To be able to extract knowledge from these user acts, a strategy is needed. How much time spent in each of the views should be considered as meaningful regarding to the profile?

Six possible strategies were considered:

1. Using global pre set variables as a threshold to decide whether enough time was spent to make it meaningful.
2. Using global variables computed by the data to as a threshold.
3. Using variables computed per user as a threshold.
4. Increasing the impact of the act according to time spent in the views with global weight on the time spent.
5. Increasing the impact of the act according to time spent in the views with globally computed weight.
6. Increasing the impact of the act according to how much time is spent in the views with personalized weight.

The two first strategies were considered as too general. This is due to the fact that every user is unique and will have an unique application usage pattern. The fourth

and fifth strategy could likely perform better that the two first ones, but is somewhat complex. The diversity of users would also have impact on this strategy, since the way of weighting the time spent is global. The sixth approach would likely be the most accurate strategy, but was considered too complex. The decision fell upon strategy three. The rationale was that the trade-off between simplicity and generality was most rewarding.

Two MapReduce jobs was constructed to compute the mean time spent in the views. The jobs were almost equal, only disjoint due to the different event types. Thus, the algorithm described below suits both the jobs.

User statistics map function

The function is described in Listing 6.1. The code simply iterates over the logged user acts and checks whether each separate act can be considered as an timed event. If it is, then it is mapped to the output as a tuple of user-IDs and seconds.

Listing 6.1: User statistic map function

```
void UserStatisticMap(Text userActLogs){

        Iterator lineIterator = userActLogs.lineSplit();

        while(lineIterator.hasMoreLines()){
                if(userActType == timedActType){
                        double seconds = userAct.getSeconds();
                        output(userID, seconds);
                }
        }
}
```

User statistics reduce function

The reduce function is described in Listing 6.2. The reduce function iterates over all the timed user acts related to the specific user and adds it to a sum. It also keeps a count of all the user acts encountered to be used in in the average function which is the simple mathematical function: $A = S/n$. A is average time spent, S is the sum of time spent per user act, and the n is the total of user acts for the specific user.

Listing 6.2: User statistic reduce function

```
void UserStatisticReduce(Text userID,
                         Iterator<Double> secondsIterator){

        double sum = 0:
        double count = 0;

        while(secondsIterator.hasMoreSeconds()){
                sum += seconds;
                count += 1;
        }

        double averageTime = sum / count;
        output(userID, averageTime);
}
```

The output of the reduce function is stored to the HDFS as a text file. It is used in the user vector jobs to determine the impact of the timed events. A more efficient way to do this would be to gather the user times through a database that can look up such information efficiently. A sample output is given in Listing 6.3. The left column in the listing is user-IDs and the right column is the average time computed.

Listing 6.3: User statistics dump

408ae611d7c683fe5d3aecdd33e05d10	5.118645344827586
719adad934ebaad9ce84dd8da293da84	7.923057302837432
a21adabc93ade934ede034ee043e5e43	25.50341930694725
ad28eee9321aee734aa3d732dda9dd21	39.51237823457982
bf798fb8990fbda026423bf7665adee6	22.31612484237698
fa1d3ea243890aecd8cd65ad76cd3231	38.29132703523934

6.5 User Profile Creation Jobs

As the user statistics jobs, the user profile creation consist of two separate jobs. One for creating vectors that weighs the different categories per user and one for weighing the different keywords that is discovered by the user through the articles. The algorithms are very similar, but differs in some ways. The job calculating the category vector outputs a constant sized vector. On the other hand, the output of the keyword vector

job could in theory house infinite entries. The user act input normally contains a lot more keywords than categories per act. This does not force significant alteration to the algorithm, but entails a larger amount of work per job. With this in mind, the two algorithms are described together, with the different properties emphasized.

Three different approaches were implemented. The approaches are separated by the way old data is forgotten. This was done due to the fact that older data becomes less and less relevant as time goes by, as stated in earlier chapters. The three different angles are described in the list below:

1. Only using data within a limited time frame from now.
2. Only using data within a limited time frame from now, where older data are decreased by a constant.
3. Using previously created user vectors degraded by a constant in addition to data newly collected.

6.5.1 User act weighing

All the approaches use the same type of weighing. The possibility of giving the user acts different influence on the profile is considered as an important feature. Sharing an article could say more about the user than if he only viewed the article. What these weights should be is another question that needs an answer.

To support this feature, a constant class was added to the project. The only functionality of this class was to give the jobs access to a hashmap that allowed the algorithm to gather the weighing information with the user act as the key.

6.5.2 The time frame approaches

The two time frame approaches have been compounded due to similarity. The only difference is that the second approach uses a time degrading function.

Time frame map function

The map function is described in Listing 6.4. Due to brevity, keywords and categories are joined together in the listing by the letters kc. The map function takes text files with user act logs as input. It then iterates over the lines in the input files. While iterating it gathers the weight constants for the specific user act. Then it degrades the weight if the the degrading function is active. Finally, it put each of the category tags or keyword tags into a map that is outputted with the user-ID.

Listing 6.4: User profile map function

```
void UserProfileMap(Text userActLogs){

        Map kcMap = new Map();
        Iterator lineIterator = userActLogs.lineSplit();

        while(lineIterator.hasMoreLines()){

                double weight = userAct.getWeight();

                // degrading function
                weight = weight - weight*(days/totalDays);

                for (kc in userAct){
                        kcMap.put(keyword/category, weigth);
                }

                output(userID, kcMap);
                kcMap.clear();
        }
}
```

Mathematically the degrading of the weight is done by the following function:

$$w_u = w_a(1 - \frac{d}{d_t})$$

where the w_u is the weight for the specific user act with time taken in consideration. w_a is the constant value that the type of user act is given. The d is the days that has passed since today. Finally, the d_t is the total time frame (e.g. 90 days if the data in scope is within the last 90 days). This means that data from today is weighted fully since the statement right of the subtraction sign is 0. When going back one day, the weight will be smaller by $1/d_t$ part of w_a.

Time frame reduce function

The reduce function of the time frame approaches are described in Listing 6.5. The input is the a userID and the output from the mapping function. The algorithm iterates over the incoming maps. It then enters another iteration loop, where the elements of the maps are gone through. In these loops it sums all the weights in coherence with the

categories or keywords that the acts is part of. When the iteration is done the output vector, represented by a hashmap, is normalized. The normalization is done by making every entry a fraction of the biggest weighted keyword or category. Finally, the results is forwarded to the the centralized system through the REST API.

Listing 6.5: User profile reduce function

```
void UserProfileReduce(Text userID, Iterator kcMapIterator){

        Map<String, Double> outputMap = new Map<String, Double>();

        while(kcIterator.hasMoreMaps()){
                Iterator category/keyIterator = nextMap.iterator();

                while(kcIterator.hasMore()){
                        outputMap.add(kc, number);
                }
        }

        normalize(outputMap);

        output(userID, outputMap);
        kcMap.clear();
}
```

6.5.3 The previous vector approach

This approach is foundationally different from the two previous approaches. It limits the processing of total user acts, since computation is only performed on newly acquired data. On the other hand, more overhead is encountered in the reduce function due to multiple input sources.

Previous vector map function

The map function algorithm is the same as in the previous approaches (Listing 6.4). The only differences are that the degrading function is not in use and the input would normally be fractionally smaller in this version.

Previous vector reduce function

The reduce function is also similar to the previous approaches. The difference is that the old result from previous calculations of vectors are used to combine the new one. The function is given in Listing 6.6. The weight deducting procedure is just multiplying the old vector by a constant that makes it less important than the new data. Mathematically, the function can be described by vector algebra in the following way:

$$\vec{V_u} = \vec{V_n} + c\vec{V_o}$$

where V_u is the user vector, V_n is the new vector computed by the new user acts, and V_o is the old vector.

Listing 6.6: User profile reduce function

```
void UserProfileReduce(Text userID, Iterator kcMapIterator){

        Map<String, Double> outputMap = new Map<String, Double();

        while(kcIterator.hasMoreMaps()){
                Iterator category/keyIterator = nextMap.iterator();

                while(kcIterator.hasMore()){
                        outputMap.add(kc, number);
                }
        }

        normalize(outputMap);

        Map<String, Double> oldOutputMap = MongoDB.find(userID);

        \\ weight deduction precedure
        outputMap = outputMap + oldOutputMap * c;

        output(userID, outputMap);
        kcMap.clear();
}
```

6.5.4 User Profiling Output

The output of the jobs is user-IDs coupled with the vectors. An example output dump is given in Listing 6.7. This example is taken from a job that has computed the weight on each category that has been encountered by the user. The left column in the listing houses the userIDs. The right column displays the vectors on the form $category = weight$. The keyword vector would have similar structure, but it will usually contain more entries.

Listing 6.7: User profile dump

```
408ae611d7c683fe5d3aecdd33e05d10    {NEWS=100.0,
                                     SPORT=23.0,
                                     TRAVELLING=1.0,
                                     LIFESTYLE=16.0,
                                     ECONOMY=9.0}

719adad934ebaad9ce84dd8da293da84    {TECHNOLOGY=62.0,
                                     NEWS=100.0,
                                     SPORT=23.0,
                                     MOTOR=1.0,
                                     TRAVELLING=6.0,
                                     ECONOMY=9.0}

a21adabc93ade934ede034ee043e5e43    {NEWS=75.0,
                                     SPORT=100.0,
                                     ENTERTAINMENT=13.0,
                                     LIFESTYLE=3.0,
                                     ECONOMY=45.0}
```

6.6 Tuning Factors

To summarize, there are in this solution four main factors that will impact the resulting vectors. The type of forgetting function is important. The solution provides three different algorithms that will all produce different weighted vectors. The specific weighing of the user acts will also make the output vectors react. The time threshold method deciding how many timed user acts also needs to be configured. Finally, the job execution frequency will decide how many new events that are considered per vector.

Part IV

Evaluation and Results

Chapter 7

Evaluation

This chapter describes how we evaluate the system.

7.1 Evaluation Strategy

Since the overall system had not gone live at the moments of this research, the data available was insufficient for evaluating specifically tuned instances of the system. Therefore, to evaluate the the results, a sensitivity analysis of how different configurations would affect a small amount of test user vectors.

7.1.1 One factor at a time (OFAT)

The OFAT method is a simple method used in sensitivity analysis. The basic idea is to change one factor to see how it affects the output. The usual way to do this is by changing one variable and keeping the other variables normal. Then changing the original variable back to its baseline value and repeating the procedure for the other variables. The sensitivity can then be measured by looking at the impact of the changes made. This will produce outputs that is solely dependent on the single variable change that has been made.

A limitation of this approach is that the input space is not fully mapped out. This is because the simultaneous variations of the factors are not considered. This implies that it is impossible to address how the different variables related to each other.

7.2 Weight Sensitivity

In this section we look at how the different weighting constants affect the final result. The experiment was done with data from two test users. Some of the user acts was excluded from this experiment due to lack of data. The OFAT method was utilized in this experiment. It was performed by giving every weighting every event by zero. Following this, an user act type were given a weight of one. After the experiment was done, the weight was switched back to zero again. This was done for all the user act types in the result set.

7.2.1 Hypothesis

The hypothesis for this experiment was that the different user acts should mildly resemble the opened article user act. The reason for this is that most of the acts has to follow this act. The idea is that the opened article act as a basis, while the other types of acts should amplify the weights or decrease the final result.

7.2.2 Results

The results of this experiment is given in Table 7.1 and 7.2. The columns represents the user act type in focus and the rows consists of the categories. The results show that the hypothesis could be true in most cases. This is because the vector values shows, with exception of the preview time type, that the opened article type can be a generalization of the rest of the acts.

The preview time act type adds information about the categories that have not gotten any information from other types of events. This is due to the fact that the user does not seek this information actively, but is forced to give execute these act by the system.

The news category is also always the dominant category. This might imply that the categorization is too general, that a lot of articles in this category are generated, or simply that this is what the users prefer to read.

7.3 Preview Time Vector Size Influence

Because of the results from the last section, it came to presence that the preview time acts affects the size of the vectors a lot. Therefore, an experiment were performed to check if it was. The experiment were performed on the same sets of data as before. The

Table 7.1: User 1 vectors

	Opened article	Article time	Preview time	Similar article	Viewed map
Technology	0.0	0.0	1.348	0.0	0.0
News	100.0	100.0	100.0	100.0	100.0
Sport	0.0	0.0	1.468	0.0	0.0
Entertainment	3.960	19.881	11.611	32.201	16.568
Travelling	0.0	0.0	12.638	0.0	0.0
Economy	0.0	0.0	7.308	0.0	0.0

Table 7.2: User 2 vectors

	Opened article	Article time	Preview time	Similar article	Viewed map
Technology	8.919	3.968	57.145	0.0	7.095
News	100.0	100.0	100.0	100.0	100.0
Sport	9.024	14.052	38.481	29.944	7.178
Entertainment	0.0	0.0	24.443	0.0	0.0
Motor	0.0	0.0	0.524	0.0	0.0
Travelling	8.919	11.904	12.638	29.596	7.033
Lifestyle	0.0	0.0	0.530	0.0	0.0
Economy	0.0	0.0	6.396	0.0	0.0

approach consisted test the vector generation of keywords. One time with the preview time active and one that had it deactivated. Then a comparison of vector lengths were performed.

7.3.1 Result

The result showed that when preview time was active, the total length of the two vectors were 514. When inactive the result were 92. This a very big difference. $92/514 = 0,18$, which implies that almost 80% of the data stored in the keyword vectors is grounded in the preview time user acts.

Chapter 8

Discussion

In this section we discuss the elements in this project.

8.1 Experimental Results

The data utilized in the experiments were very scarce, thus a very critical eye has to gaze upon the results. A large set of data is needed to evaluate the system is needed. In fact, the question regarding how to evaluate the system as a whole is very important. A large amount of data is needed and it can not be auto generated. This is because the data needs to be constructed in a way that resembles users. Creating data from surveys could be a good idea, but this would require a lot of people to attend to get an amount that would suit the system. Generating data from logs of similar systems is another idea that could give a more sturdy evaluation platform.

In the subsections below, we discuss the results of the experiments executed.

8.1.1 Weight Sensitivity

The main finding of this experiment was that preview time acts influence the vectors in a different way than the rest of the user acts. It adds data to categories that no other acts influence. The main questions that needs to be answered is how much influence it should have on the final vector and more importantly if it is needed at all. From a logical perspective, the preview time should have some influence on vectors since it should contain some information of value about the users. This information could however be very inferior to the other value of the other acts. It can be considered as a cost/benefit dilemma. The results show that 80% of the data generated comes from

preview time acts. This is very much and a lot of computation could be avoided by not considering this data.

Getting the weight right is also a problem. Setting them manually would not be a good way of doing it. A better approach would be to use data mining techniques on real data to find the weights. By utilizing machine learning procedures, it should be possible to learn what these factors should be. However, the problem is two-fold. The weighting would change over time, since the recommender system would force the user to look at news that it recommends. If this is not taken into consideration, then the users might get "stuck" at certain types of news, but this a problem that has to be dealed with in the module that serves the user news. A suggestion to avoid this problem could be to present some random news to the user.

8.2 Data Transfer Alternatives

Two alternatives that should work better than the data transferring program described in the implementation part. First off is Apache Flume is discussed as an alternative to this. Following this the Hadoop-Mongo in this environment is described. Finally, we discuss what would be the best option.

8.2.1 Apache Flume

Apache Flume[37] will a more robust and safe way of transferring data to HDFS is used than the approach used in the implementation. Apache Flume guarantees that the data will be transferred since it uses a single-hop guarantee semantics. This provides robust end-to-end reliability. A proposed instance is given as an example in Figure 8.1. In this example, the source is the middleware system and the sink is the HDFS installation.

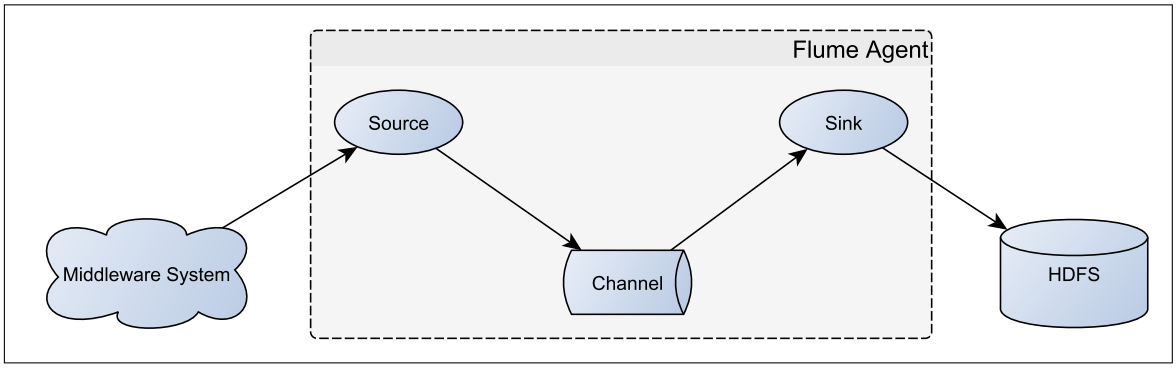

Figure 8.1: Flume instance

Another property that favours Flume is the fact that new events are pushed to HDFS as they arrive. This implies that small messages are sent to the HDFS. In the present state, all data is transferred in a batch, which could be a congestion point when the amount of data scales. Adding new external storage spaces would also be simplifies since it would only require the Flume agent to forward the user acts to this storage as well. As it is now, separate implementation will have to be done for this.

The physical architecture of the system would with the addition of Flume be altered. Figure 8.2 displays a proposed architecture showing how the data flows through the system. The log database has been maintained in the architecture since a separate stable storage for the logs would increase stability due to redundancy.

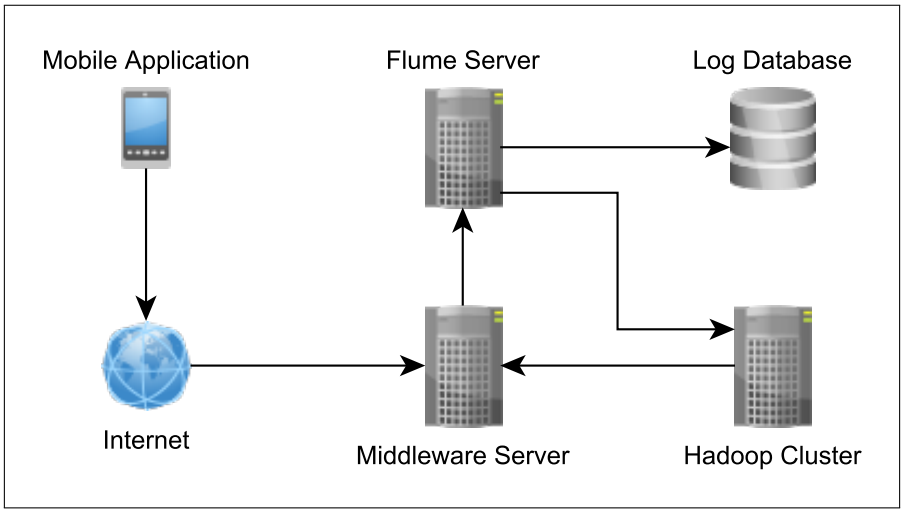

Figure 8.2: Architecture with Flume

8.2.2 MongoDB + Hadoop Connector

Research[39] implies that this connector is very suitable if small amounts of data are to be transferred from MongoDB to the MapReduce job. However, if the solution are to use an analytical approach exploiting large portions of the stored data, then the MongoDB would be inferior to using HDFS directly. The work done in this thesis could be considered as an approach in the first category, due to only utilizing newly acquired data. So in this case the connector should be well suited. However, if the intention is to expand the MapReduce job base to perform more analytical tasks, then using HDFS would be the best choice. Considering both, the last choice would be most beneficial, since the other modules that are to be added could benefit from this. The Flume approach would definitely be best since it removes the need for MongoDB and the integration with Hadoop.

8.3 Scheduling Alternatives

Apache Oozie was not used in the development process, but should be used when the system is deployed. There exists a lot of different tools for scheduling Hadoop jobs, but it is by far the most popular solution which implies that it is well supported. It suits this system well since it is easy to integrate new jobs and introduce new functionality to the system as a whole.

Chapter 9

Conclusion

News recommender systems are designed to serve the news that a specific user want. The Internet and it's vast amounts of articles have made this a possible problem for the news readers. Time-consumption and difficulty of finding the right news would be lowered if it is done correctly.

This thesis considers the idea of using user profiles in a larger process to filter out redundant news articles. The basis for these profiles were the logs that users created while using the recommender application. Usage of Hadoop and it's features when achieving this was also a goal of the research.

The literature review were that some work have been done within news recommendation. Little literature have been published in the context of mobile news recommendation, but since it is a subset of news recommendation the domains are very similar. How to value heterogeneous user clicks is absent in all the literature studied, so this is a field that deserves more focus. Little focus of how the logs were computed into profiles were given, but how they were represented digitally had good descriptions.

How to evaluate the effectiveness and accuracy of the user profiles is difficult. Little real data was available during this research and actual data is needed. Data that replicates real users is hard to forge and is needed for both evaluation and calibration of the implementation. These calibration factors consists of the user act weights, time degrading functions, the time threshold for timed user acts, and the interval between two running jobs.

The evaluation implies that the most important user act (open article) can be considered as a generalization of the rest of the events, although it is believed that the other user acts also entails some knowledge to be discovered. The "news" category also seems somewhat dominant in the tests, this could be due to the category distribution or that the module that classifies articles has some weaknesses. One type of user act, named article preview time, is very dominant in the logs. Nearly 80% of the

keyword vector entries are, so this might be viewed upon as redundant data to process when considering cost versus benefit.

Bibliography

[1] SmartMedia Website. `http://smartmedia.idi.ntnu.no/`. Accessed: 31/05/2013.

[2] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[3] Paul Canavese and Howard Besser. The future of information filtering. *Library and Information Studies A*, 296.

[4] Clifford Lynch. Searching the internet. *Scientific American*, 276(3):52–56, 1997.

[5] C Bowman, Peter B Danzig, Darren R Hardy, Udi Manber, and Michael F Schwartz. The harvest information discovery and access system. *Computer Networks and ISDN Systems*, 28(1):119–125, 1995.

[6] Upendra Shardanand and Pattie Maes. Social information filtering: algorithms for automating "word of mouth". In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217. ACM Press/Addison-Wesley Publishing Co., 1995.

[7] David Goldberg, David Nichols, Brian M Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[8] Uri Hanani, Bracha Shapira, and Peretz Shoval. Information filtering: Overview of issues, research and systems. *User Modeling and User-Adapted Interaction*, 11(3):203–259, 2001.

[9] Bamshad Mobasher, Robert Cooley, and Jaideep Srivastava. Automatic personalization based on web usage mining. *Communications of the ACM*, 43(8):142–151, 2000.

[10] Elaine Rich. User modeling via stereotypes. *Cognitive science*, 3(4):329–354, 1979.

[11] Pattie Maes et al. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):30–40, 1994.

[12] Fang Liu, Clement Yu, and Weiyi Meng. Personalized web search for improving retrieval effectiveness. *Knowledge and Data Engineering, IEEE Transactions on*, 16(1):28–40, 2004.

[13] Daniel Billsus and Michael J Pazzani. A hybrid user model for news story classification. *COURSES AND LECTURES-INTERNATIONAL CENTRE FOR MECHANICAL SCIENCES*, pages 99–108, 1999.

[14] Ah-Hwee Tan and Christine Teo. Learning user profiles for personalized information dissemination. In *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on*, volume 1, pages 183–188. IEEE, 1998.

[15] Daniel Billsus and Michael J Pazzani. User modeling for adaptive news access. *User modeling and user-adapted interaction*, 10(2-3):147–180, 2000.

[16] Ricardo Carreira, Jaime M Crato, Daniel Gonçalves, and Joaquim A Jorge. Evaluating adaptive user profiles for news classification. In *Proceedings of the 9th international conference on Intelligent user interfaces*, pages 206–212. ACM, 2004.

[17] Joseph A Konstan, Bradley N Miller, David Maltz, Jonathan L Herlocker, Lee R Gordon, and John Riedl. Grouplens: applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[18] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280. ACM, 2007.

[19] Mark Claypool, Anuja Gokhale, Tim Miranda, Pavel Murnikov, Dmitry Netes, and Matthew Sartin. Combining content-based and collaborative filters in an online newspaper. In *Proceedings of ACM SIGIR workshop on recommender systems*, volume 60. Citeseer, 1999.

[20] Jiahui Liu, Peter Dolan, and Elin Rønby Pedersen. Personalized news recommendation based on click behavior. In *Proceedings of the 15th international conference on Intelligent user interfaces*, pages 31–40. ACM, 2010.

[21] Uichin Lee, Zhenyu Liu, and Junghoo Cho. Automatic identification of user goals in web search. In *Proceedings of the 14th international conference on World Wide Web*, pages 391–400. ACM, 2005.

[22] Micro Speretta and Susan Gauch. Personalized search based on user search histories. In *Web Intelligence, 2005. Proceedings. The 2005 IEEE/WIC/ACM International Conference on*, pages 622–628. IEEE, 2005.

[23] Hyoung R Kim and Philip K Chan. Learning implicit user interest hierarchy for context in personalization. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 101–108. ACM, 2003.

[24] Olfa Nasraoui, Maha Soliman, Esin Saka, Antonio Badia, and Richard Germain. A web usage mining framework for mining evolving user profiles in dynamic web sites. *Knowledge and Data Engineering, IEEE Transactions on*, 20(2):202–215, 2008.

[25] IBM big data. `http://www-01.ibm.com/software/data/bigdata/âĂŌ`. Accessed: 31/05/2013.

[26] G.E. Moore et al. Lithography and the future of Moore's law. In *Proc. SPIE*, volume 2440, 1995.

[27] T. White. *Hadoop: The definitive guide*. O'Reilly Media, 2012.

[28] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[29] D Borthakur. Hdfs architecture guide, December 2012.

[30] S. Ghemawat, H. Gobioff, and S.T. Leung. The google file system. In *ACM SIGOPS Operating Systems Review*, volume 37, pages 29–43. ACM, 2003.

[31] F. Chang, J. Dean, S. Ghemawat, W.C. Hsieh, D.A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R.E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):4, 2008.

[32] A. Thusoo, J.S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.

[33] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.

[34] P. Hunt, M. Konar, F.P. Junqueira, and B. Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX ATC*, volume 10, 2010.

[35] MongoDB. `http://www.mongodb.org/âĂŌ`. Accessed: 31/05/2013.

[36] Rick Cattell. Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39(4):12–27, 2011.

[37] Apache Flume. `http://flume.apache.org/`. Accessed: 31/05/2013.

[38] MongoDB + Hadoop Connector. `http://api.mongodb.org/hadoop/`. Accessed: 31/05/2013.

[39] Elif Dede, Madhusudhan Govindaraju, Dan Gunter, R Canon, and Lavanya Ramakrishnan. A performance evaluation of a mongodb and hadoop platform for scientific data analysis. 2013.