

A dissertation submitted to the **University of Greenwich** in partial fulfilment of the requirements for the Degree of

Master of Science in
Enterprise Systems and Database Administration

Relevant Tickets Retrieval Solution

Name: Riad Darawish
Student ID: 000820865

Supervisor: Dr Mohammed Hassouna
Submission Date: 31st January 2015

Word count: 18,123

Relevant Tickets Retrieval Solution
Riad Darawish
Computing & Mathematical Sciences, University of Greenwich, 30 Park Row, Greenwich, UK

Abstract

The central purpose of this project is to develop a relevant ticket retrieval solution for ProgressSoft's helpdesk system. This will facilitate obtaining the relevant tickets for a new ticket and thus improve customer service performance. In response to the challenges of helpdesk's system data growth and scalability, the solution uses a distributable algorithm to analyse and index the system tickets. The solution adopts vector space scoring to rank the relevant tickets for a newly arrived ticket. A data set from the system was used to test the implementation.

A literature review was conducted to explore some of the approaches used in retrieving relevant documents. It also sheds light on the process of building an information retrieval system for unstructured data, the distributed inverted index and the MapReduce programming model. It also discusses the search mechanism that ProgressSoft's helpdesk system employs to find the related tickets for a query.

Scrum development methodology was used to develop the solution. A project plan has been created which consists of four sprints, with each ending with a working component. The sprints are: fetch the text from helpdesk tickets, index and analyse text, build a TF-IDF matrix, and build the search API. Each of these sprints includes phases for gathering requirements, analysis and design and implementation.

Keywords: information retrieval; inverted index; Hadoop; MapReduce; Lucene API; Sqoop; helpdesk system; tickets

Acknowledgements

I would especially like to thank Dr Mohammed Hassouna for agreeing to supervise me. His guidance, support, advice and comprehensive feedback are greatly appreciated, as they have enabled me to complete my project to a high standard. In addition, I would like to thank all of my lecturers and tutors who have taught me during my degree.

I want to give special thanks to ProgressSoft which has contributed to my project by providing a sample of helpdesk data.

Finally, I would like to thank both Dr Mohammed Hassouna and Dr Simon Scola for agreeing to attend a demonstration of my project.

Contents

ABSTRACT	2
ACKNOWLEDGEMENTS	3
1. INTRODUCTION	7
1.1 RESEARCH QUESTION	7
1.2 ORGANISATION OF THESIS	8
2. LITERATURE REVIEW	9
2.1 REVIEW OF RELEVANT-DOCUMENT RETRIEVAL APPROACHES	9
2.2 SEARCH MECHANISM FOR PROGRESSOFT'S HELPDESK.....	10
2.3 OVERVIEW OF THE PROPOSED APPROACH	10
2.4 INFORMATION RETRIEVAL FOR UNSTRUCTURED DATA	11
2.5 INVERTED INDEX	12
2.6 MAPREDUCE PROGRAMMING MODEL	13
2.7 RUN-TIME SYSTEM.....	14
2.8 DEVELOPMENT APPROACH AND ASSUMPTIONS	15
3. APPLICATION DEVELOPMENT	18
3.1 PRODUCT BACKLOG.....	18
3.2 SPRINT 1: FETCHING THE TICKET TEXT	18
3.3 SPRINT 2: TEXT ANALYSIS AND INDEXING	27
3.4 SPRINT 3: BUILDING THE TF-IDF MATRIX	36
3.5 SPRINT 4: BUILDING THE RETRIEVAL API	41
4. RESULTS AND EVALUATION	48
4.1 EVALUATION OF RETRIEVAL RESULTS	48
4.2 EVALUATION OF RESEARCHING TIME.....	64
4.3 EVALUATION OF INDEX PERFORMANCE.....	65
5. CONCLUSION	69
5.1 CHALLENGES AND FUTURE RECOMMENDATIONS	69
6. REFERENCES	71
7. APPENDIX	74

List of Figures

Figure 1. The Two Parts Of An Inverted Index.....	12
Figure 2. The Execution Of The Mapreduce Program.....	14
Figure 3. Scrum Alliance (Softwaretestinghelp.Com, 2015).....	16
Figure 4. Ps Helpdesk Ticket Structure	19
Figure 5. The Average Length Of A Ticket's Description	20
Figure 6. The Average Length Of A Ticket's Text.....	20
Figure 7. Jiraissue Table.....	22
Figure 8. Jiraaction Table	22
Figure 9. Use-Case Diagram For Fetching The Ticket Text.....	24
Figure 10. Code Of Jiaraaction_Concat Plsql Procedure.....	25
Figure 11. Sqoop Import Command	25
Figure 12. Extracted Records In An Hdfs File	26
Figure 13. The Structure Of The Term's Postings List In Json Format	28
Figure 14. Use-Case Diagram Of Text Analysis And Indexing	30
Figure 15. Text Analysis And Inverted Index Mapreduce Job Execution	31
Figure 16. Inverted Index Stored In Hadoop Sequencefile.....	34
Figure 17. Index File Contains A Dictionary Of All Terms In Inverted Index.....	35
Figure 18. The Code Of Main Method Of Invertedindexdriver.....	35
Figure 19. Use-Case Diagram Including "Building Tf-Idf Matrix"	38
Figure: 20. Generating Tf-Idf Matrix	39
Figure 21. The Algorithm For Computing Vector Space Scores.....	422
Figure 22. Use-Case Diagram For Retrieving The Relevant Tickets	433
Figure 23. The Retrieval & Scoring Process & The Euclidean Lengths Mapreduce.....	45
Figure 24. Precision Formula (Manning Et Al, 2008)	49
Figure 25. Recall Formula (Manning Et Al, 2008).....	49
Figure 26. Precision And Recall (Manning Et Al, 2008).....	49
Figure 27. Ps-Helpdesk Tickets' Hierarchy.....	500
Figure 28. The Results Of Query "Duplicate Micr Error"	51
Figure 29. Precision-Recall Curve Query 1	52
Figure 30. The Results Of Query "Timeout Cheques"	53
Figure 31. Precision-Recall curve query 2.....	53
Figure 32. The Results Of Query "File Not Found In Edms"	55
Figure 33. Precision-Recall Curve Query 3.....	56
Figure 34. The Results Of Query "Unreachable Destination Cheques"	57
Figure 35. Precision-Recall Curve Query 4.....	58
Figure 36. The Results Of Query "Ldap Authentication Issue"	59
Figure 37. Precision-Recall curve query 5.....	62
Figure 38. Averaged 11-Point Precision-Recall Graph Across 5 Queries	62
Figure 39. Researching Time Vs Number Of Tickets	64
Figure 40. Evaluation Of Indexing Time Depending On The Number Of Tickets.....	67
Figure 41. Ticket Indexing Time For Each Round	67

List of Tables

Table 1. Product Backlog	18
Table 2. Product Backlog After Sprint 1	26
Table 3. Product Backlog After Sprint 2	36
Table 4. Product Backlog After Sprint 3	41
Table 5. Description Of Retrieval Phase Use-Cases.....	44
Table 6. Product Backlog After Sprint 4	47
Table 7. Test Queries And Their Information Needs.....	51
Table 8. Precision-Recall Values Of Query 1	52
Table 9. Precision-Recall Values Query 2.....	54
Table 10. Interpolated Precision – 11-Recall Levels Query 2	55
Table 11. Precision-Recall Values Query 3	5656

Table 12. Interpolated Precision – 11 Recall Levels Query 3.....	57
Table 13. Precision-Recall Values Query 4.....	58
Table 14. Interpolated Precision – 11 Recall Levels Query 4.....	59
Table 15. Precision-Recall Values Query 5.....	60
Table 16. Interpolated Precision – 11 Recall Levels Query 5.....	61
Table 17. Computing the average of 11-point interpolated precision for five queries	622
Table 18. Calculating The Mean Average Precision For Five Queries	63
Table 19. The Execution Time For Each Job	66
Table 20. Ticket Indexing Time Per Round.....	67

1. Introduction

Many companies tend to operate intelligent helpdesk systems in response to the strong need to provide high-quality customer service. It is reported that 70% of customers leave a company, not because of the price or product quality issues, but because they do not like the customer service (Customerservicetrainingcenter.com, 2015). Helpdesk officers often provide support for a variety of products, therefore they have to be familiar with many manual operations. Furthermore, sharing experience and knowledge between staff is often difficult. For these reasons, it is important to obtain an intelligent helpdesk system to overcome these challenges (Wang et al, 2011).

For decades, there has been much interest in applying case-based recommendation functionality in several disciplines, including product suggestions in e-commerce, holiday suggestions and jobs suggestions in employment (Smyth, 2007). The emergent helpdesk systems also tend to offer the recommendation functionality to provide the potential set of tickets (also referred to herein as documents) that are similar to a new helpdesk ticket (also referred to herein as a query). Here the system ranks the potential tickets by computing a score for each matching ticket.

Vector space scoring is a simple algorithm to measure the similarity of the documents to a query by assigning a score to each (query, document) pair. It is rigidly based on viewing each relevant document as a vector of a given weight, and then having it compute a score between each query and document.

Existing systems employing relevant document retrieval features suffer from one limitation: the computing and storage cost for indexing the documents and calculating the weights of the terms can be very high for large document collections. To optimise the computation, a few techniques have been proposed. However, the performance of these methods is still limited by the processing ability of the single computer. Ideally, the proposed work presents a distributed solution that is developed to offer improved processing power to analyse and index the document text and calculate the weights in a large data set.

In this paper, a scalable relevant document retrieval solution is proposed which is based on distributed algorithms for indexing the documents and computing the terms' weights on a MapReduce framework, and more specifically its open source implementation, Hadoop. The performance of the parallel method and retrieval adequacy are evaluated and compared. The proposed solution is developed to serve ProgressSoft's (PS) helpdesk system and the results are displayed using a web interface that is borrowed from PS's helpdesk system.

1.1 Research Question

A primary aim of this project is to develop a relevant ticket retrieval solution to serve PS's helpdesk system to facilitate obtaining the relevant tickets for a new ticket and thus improving the customer service. In response to the challenges of helpdesk's system data growth and scalability, the solution employs a distributable algorithm to analyse and index

the system's tickets. It uses a vector space model to rank-order the relevant tickets for a new ticket. A data set from PS's helpdesk system was used to test the implementation.

The key questions that will be investigated are: How efficient is using a distributable algorithm for indexing the documents and computing the terms' weights on a MapReduce framework to scale up with the data growth of the helpdesk system data?; and how accurate is using vector space scoring as a similarity measurement to obtain the relevant tickets?

1.2 Organisation of Thesis

The chapters in this project are:

- Literature review – this chapter covers the main areas of research for this project. It starts by discussing the approaches of retrieving relevant documents. It then presents an overview about the search mechanism used in ProgressSoft's helpdesk system to find a list of relevant tickets for a query. The literature review briefly introduces the framework of the proposed solution. The definition of information retrieval for unstructured data is then presented. It introduces the concept of inverted index and its programming model. A development approach is defined with the appropriate assumptions for this project.
- Development – this chapter reflects the development process of the application which is divided into sprints.
- Results – this chapter reveals the reliability and performance of the proposed distributed indexing approach and assesses the relevancy of the adopted similarity measurement.
- Evaluation – this final chapter will evaluate the project and describe the main challenges; the project concludes with future perspectives of enhancing the retrieved related tickets.

2. Literature Review

2.1 Review of relevant-document retrieval approaches

Typically there are a number of approaches that have been used to retrieve the relevant documents for a query. Among them is case-based reasoning (CBR), in which each document is described manually for decision rules and scoring criteria (each document just needs a small set of key characteristics to describe it). The system then processes a new document to produce a special format that it can match to the stored representations (Chang et al, 1996). The disadvantage of this approach is the need for human intervention to process the stored documents, thus for a large collection of documents this approach will be difficult to implement.

Chen et al (2002) presented CBR architecture focused on enhancing processing time for retrieving similar cases on a large-scale. They proposed a different CBR architecture from the general one by using a parallel indexing method, called bitwise indexing, and constructing a Similarity Mapping list which contains a list of pre-computed possible similarities to speed-up the calculation of similarities. The solution was tested efficiently to scale up vertically.

Weiss et al (2000) proposed a lightweight document-matching system to match new documents to old and to rank the retrieved documents by assigning a score or relevance. The prototype was developed to reduce the amount of text data to be indexed by only processing the text from document titles and other keywords extracted from other sources, including those manually assigned, and the most frequent words in the document body. Although the results show that the implementation provides good program performance measures, the implementation faces the challenge of providing a good recall fraction due to the system using no word-stems (except that plurals are mapped to singulars), thus the query's words must exactly match the terms stored in the data structure. Another challenge highlighted by Wang et al (2011) is that these systems have difficulty capturing the semantic meanings of the new and past requests due to the similarity measurements used which are based on keyword matching.

Wang et al (2011) presented a helpdesk system based on summaries of clusters formed by semantically related solutions. The system ranks the past cases based on their semantic relevance to the request, groups the relevant cases into different clusters using a mixture language model and symmetric matrix factorisation, and summarises each case cluster to generate recommended solutions. The survey they conducted shows that their approach has better user satisfaction than the traditional keyword-based ranking.

Work proposed by Ali et al (2011) for a document retrieval system used the proximity of terms in the document to give a higher score to documents which contain more nearby terms as those of the query. Although the proposed indexing approach obtained higher precision and recall compared to other similarity measurements (cosine scores, dice and sum of weights), the sample for the experiment included only 340 documents.

In this paper, a relevant document retrieval solution is proposed to perform the text analysis and inverted index process in a distributed manner. At the retrieving phase, vector space scoring is used as a similarity measurement that takes its input from a pre-computed (to

speed-up the calculations of similarity) term frequency-inverse document frequency (TF-IDF) matrix generated from a distributed inverted file.

2.2 Search Mechanism for ProgressSoft's Helpdesk

ProgressSoft uses a JIRA helpdesk system, which is a proprietary issue tracking product, developed by Atlassian (Atlassian.com, 2015). The system provides a text search functionality which is built entirely under the Apache Lucene API umbrella. The search function supports a number of queries, the most important of which are:

- Modifying query terms such as wildcard searches, fuzzy searches and proximity searches
- Boosting a term which allows the user to control the relevance of a document by boosting its term
- Boolean operators which allow terms to be combined through logic operators

These queries result in a ticket that either matches or does not match a query (Manning et al, 2008). As Raghavan and Nayak (2014) state, these queries are good for expert users with precise understanding of their needs and the collection, but are not good for the majority of users as most users are incapable of writing Boolean queries (or can but think it's too much work). Furthermore, most users do not want to screen through thousands of results. This is in contrast with ranked retrieval models, such as the vector space model, in which users largely use free text queries rather than a precise language with operators for building up query expressions, and the system decides which documents best satisfy the query (Manning et al, 2008). In accordance with that, it is required for the helpdesk system to rank the tickets matching a query or the text content of a new ticket and then order them based on their score from highest to lowest.

Another point that has to be considered is the rapid data growth of the PS helpdesk system. Lucene's inverted index is used for the aforementioned search functionalities and is able to handle the current number of the tickets. However, according to the helpdesk manager, it is expected that the current number of stored tickets will at least triple within a year due to the dramatic increase in the number of customers. Therefore, any proposed solution has to consider this growth. Unfortunately, Lucene provides no facilities for scaling. Lucene's indexing and searching throughput allows for a sizable amount of content on a single modern computer (McCandless et al, 2010).

2.3 Overview of the Proposed Approach

The proposed relevant ticket retrieval solution is centred on two themes: the first is to use a distributable approach for indexing and text analysing to overcome the storage and computation challenges when dealing with explosive data; the second is to adopt a vector space model to obtain the most relevant tickets ranked according to their similarity to the text content of a new ticket.

The main characteristics of the system's process flow are:

1. Fetching the text contents of the documents and storing them in a distributed file system.
2. Full indexing of the documents using a distributed index algorithm to obtain a distributed inverted index partitioned across several machines.
3. Pre-calculating weights of all terms in the inverted index dictionary and storing them in a distributed TF-IDF matrix that is partitioned across several machines.
4. Generating a Map-File for the distributed index file which contains all the terms mapped to byte offset of their postings lists in the distributed index file.
5. Generating a Map-File for the distributed TF-IDF file which contains all the ticket IDs mapped to byte offset of their vectors in the distributed TF-IDF file.
6. At the retrieval phase, vector space scoring is used to calculate a score for each relevant ticket by using information obtained from the distributed inverted index and TF-IDF matrix through their Map-Files.

Implementing this approach provides the scalability for processing a large helpdesk data set when it is difficult to perform indexing on a single machine. This approach is inherited from the technique used by web search engines when there is a large number of web pages that need to be indexed. To reduce the hardware costs of the distributed computation and storage, the MapReduce programming model is adopted, which is a general architecture for distributed computing. MapReduce is designed for large computer clusters. The point of a cluster is to solve large computing problems on inexpensive service machines that are built from standard parts (processor, memory and disk) as opposed to on a “supercomputer” with specialised hardware (Manning et al, 2008).

2.4 Information Retrieval for Unstructured Data

Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers) (Manning et al, 2008). In IR parlance, objects to be retrieved are generically called “documents” which in our case are the tickets of the PS helpdesk system. Our problem is in line with the field of IR, in which one must organise the unstructured text (free-text) of a system's tickets in a particular way to be able to retrieve the most useful tickets for the summary or the subject of a new ticket. Each ticket records the interactions between the customers and the helpdesk's technical staff in free-text that is stored in a relational database. The definition of IR covers all data and information problems, including our problem, with unstructured data which do not have an overt structure that makes it easy for the computer to construct. On the other hand, Manning et al 8) see that in reality, almost no data are truly “unstructured”. This is true of all text data if you count the latent linguistic structure of human languages.

2.5 Inverted Index

Suppose that you must determine the tickets that contain the statement “The application runs out of memory” from the helpdesk data set. One way that most people think about it would be to go through all the ticket records, noting which contain any of the words in that statement and excluding those that do not contain any of those words. This process is referred to as grepping through the text and it might be useful in some cases but for processing a large number of records it is inefficient (Manning et al, 2008). Grepping through text can be effective, especially with the speed of modern computers; it often allows for wildcard pattern matching through the use of regular expressions (Manning et al, 2008). The solution is less than optimal when users have to process large document collections quickly, allow for more flexible matching operations, and find the best answer among many tickets that contain given words (Manning et al, 2008).

Another way to find the tickets that satisfies a query is to index the documents in advance. The indexing process can be defined as a process that transfers the information contained in the text to a different representation space that can be treatable by a computer (Ali, Abdelkrim and Mebarek, 2011). Inverted index or a postings file (Knuth, 1968) were popular in the document retrieval landscape. Today, however, most retrieval engines for full-text search rely on an inverted index, which uses a term to provide access to the list of documents that contain the term (Dean and Ghemawat, 2008). The basic concept of an inverted index is illustrated in Figure 1.

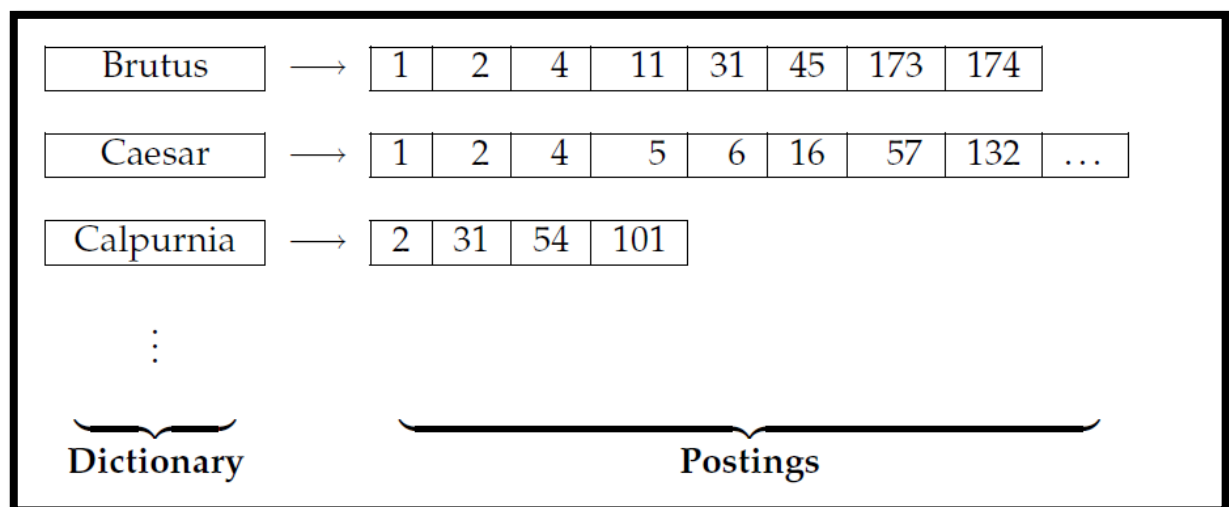


Figure 1. The two parts of an inverted index. The dictionary is commonly kept in memory, with pointers to each postings list, which is stored on disk (Manning et al, 2008).

Inverted indexes are efficient because their search strategy is based on vocabulary (the set of distinct words in the text), which is usually much smaller than the text, and thus fits in the main memory (Arroyuelo et al, 2014). For each word, the list of all its occurrences (positions) in the text is stored. Those large lists may be stored in secondary storage, or in the main memory of the cluster nodes. Inverted indexes are suitable only for searching

natural-language texts (which have clearly separated words that follow some convenient statistical rules) (Baeza-Yates and Ribeiro-Neto, 2011).

The growing amount of text data demands efficient parallel and distributed indexing mechanisms to be adopted to manage large resource requirements and unpredictable system failures. Parallel and distributed indices built using commodity hardware like personal computers (PCs) can substantially save costs because PCs are produced in bulk, achieving economy of scale (Luk and Lam, 2007). For this reason, it was decided to build the inverted index in a distributable manner using the MapReduce programming model.

2.6 MapReduce Programming Model

The distributed concept of the inverted index can be constructed using MapReduce. Dean and Ghemawat (2008) define MapReduce as a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real-world tasks can be expressed in this style. Google was one of the first organisations to adopt this style to solve the problem of wanting to download the whole of the Internet and index it to support search queries (Perera and Gunarathne, 2013).

The primary motivation to adapt this programming paradigm for coding our inverted index is for the purpose of scalability as the program is executed in parallel on a cluster environment that consists of a large number of commodity machines. The run-time platform executes MapReduce's code on the cluster's nodes and it also manages the fault tolerance. Using this programming model facilitates building the inverted index without being concerned how to parallel its computation. As Dean and Ghemawat (2008) state, this programming approach allows programmers without any experience of parallel and distributed systems to use the resources of a large distributed system.

The secondary motivation is the simplicity of the paradigm. MapReduce provides the developer with a clean and easy to use library; hence, he or she has to specify the computation in terms of a map and a reduce function, and underneath, the system automatically splits the computation across multiple machines, schedules machine usage, and handles network communication and machine failures (Gog et al, 2011).

The basic MapReduce programming model is described as follows (Dean and Ghemawat, 2008):

1. The MapReduce computation takes a set of input key/value pairs, and produces a set of output key/value pairs.
2. The user of the MapReduce library expresses the computation as two functions: map and reduce.
3. The map, written by the user, takes an input pair and produces a set of intermediate key/value pairs.

4. The MapReduce library groups together all intermediate values associated with the same intermediate key and passes them to the reduce function.
5. The reduce function, also written by the user, accepts an intermediate key and a set of values for that key. It merges these values to form a (possibly) smaller set of values; typically just zero or one output value is produced per reduce request. The intermediate values are supplied to the user's reduce function via an iterator. This allows users to handle lists of values that are too large to fit in memory.

2.7 Run-Time System

Hadoop is the run-time system that will be used to execute the processing logic of the inverted index which is coded using MapReduce. Hadoop is a Java-based open source project. Users have to only write the processing logic, and Hadoop executes the logic while handling distributed aspects such as job scheduling, data movements and failures (Perera, 2013). Hadoop is the most widely known and widely used implementation of the MapReduce paradigm (Perera and Gunarathne, 2013).

When the processing logic is executed, Hadoop carries out the following steps (Perera and Gunarathne, 2013) (Figure 2):

1. Hadoop breaks the input data into multiple data items using new lines and runs the map function once for each data item, giving the item as the input for the function. When executed, the map function outputs one or more key/value pairs.
2. Hadoop collects all the key/value pairs generated from the map function, sorts them by the key, and groups together the values with the same key.
3. For each distinct key, Hadoop runs the reduce function once while passing the key and list of values for that key as input.
4. The reduce function may output one or more key/value pairs, and Hadoop writes them to a file as the final result.

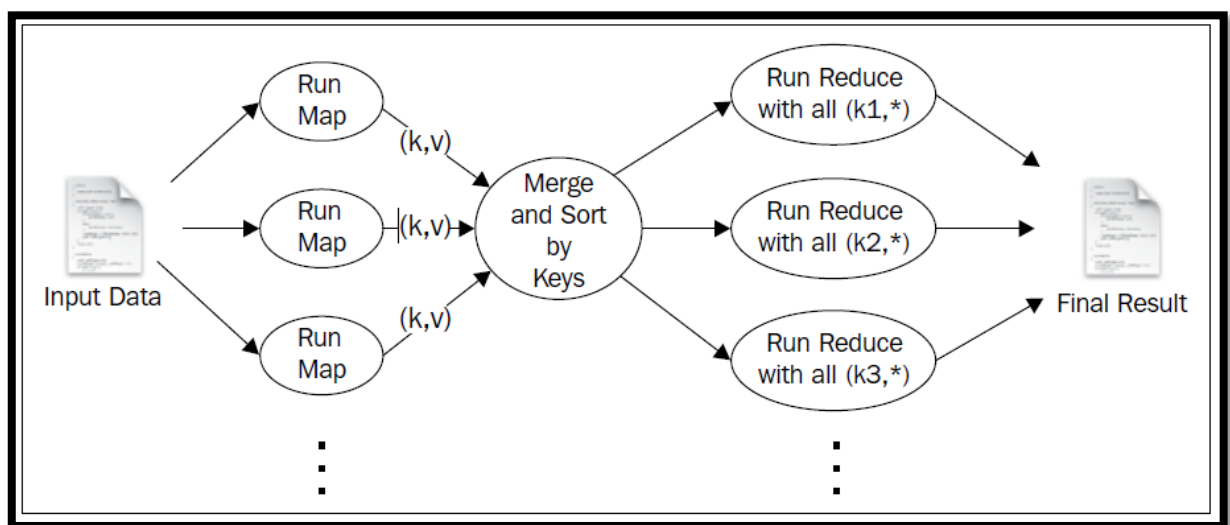


Figure 2. The execution of the MapReduce program (Perera and Gunarathne, 2013)

2.8 Development Approach and Assumptions

Due to the development complexity of the proposed system and to insure delivering it on time, we must follow a development methodology to provide guidance on how the system should be developed. There is single development methodology suitable for use by all projects (CMS, 2007), so we need to examine the available methodologies to choose the most suitable development approach.

Software development life cycle (SDLC) is the traditional methodology for software development and its most common model is the waterfall process model. (Istqbexamcertification.com, 2015). Different models of SDLC share the phases of gathering and analysis, design, implementation or coding, testing, deployment and maintenance. Each phase produces deliverables required by the next phase in the life cycle, therefore they have to be executed in order (Istqbexamcertification.com, 2015).

Traditional life cycle development is based on the principle that methodologies should focus on defect prevention, therefore it is required to get it right the first time. Thus, effort has to be put into planning and determining the specification in the beginning stages of the project (Kyriakidou, 2014). For the proposed software, it is difficult to prevent what it is unknown and at the earlier stages it is not easy to predict or speculate on all the specifications of the final product. This is because the knowledge and the skills needed for this area of study will be gained gradually through the progress. For these reasons, it is difficult to follow any traditional life cycle methodologies to develop the proposed solution.

Agile methodologies was created in the mid-1990s to maintain simplicity and focus on developing systems incrementally (Kyriakidou, 2014). It offers a way to develop software that is less extensive and with less detail, which brings quick and active processes (Picek, 2009). These principles and others are stated in the Agile Manifesto that was published in 2001 by Agile Alliance:

“Individuals and interactions over processes and tools; Working software over comprehensive documentation; Customer collaboration over contract negotiation; Responding to change over following a plan.”

2.8.1 Scrum

A number of development methodologies are based on the agile approach, among them are Scrum, Extreme Programming (XP) and Lean Software Development (Picek, 2009). Scrum is used to develop MapReduce applications and the developers of the project (Thorn, a replacement for Java - Implementing MapReduce for the Thorn language) justify their choice as Scrum has the following advantages:

- Scrum is known to produce good outputs for projects which need to be achieved in a relatively short period of time.
- The Scrum method enforces frequent updates on the state of the project and this enables users to know where the project is heading and what stage it is currently in.
- Scrum allows supervisors as well as team members to follow the progress of the project and assess the amount of work that has been done and that is left to do.

- It is good for an open-ended project where it is difficult to assess precisely the complexity of the tasks that need to be completed. This allows users to work towards stable components, thus allowing more control over the progress of the project.

Scrum is a framework for dealing with complex new product development, and is an alternative for traditional approaches with predictable requirements that are typically adapted by systems development. However, Scrum is learning about work and the process as it attempts “to put the chaos in a box, making the most of uncertainty” when the process is too complicated (Scrumtrainingseries.com, 2015).

The main processes of Scrum are addressed by Scrum Alliance (2015) which has three primary roles, listed below and in Figure 3:

1. A ranked wish list is constructed by the product owner and it includes future system features called a product backlog.
2. The development team take a number of the highest priority functions or features from the wish list during the sprint planning to form the sprint backlog. The team then discuss how to implement the chosen pieces.
3. The sprint usually lasts between two to four weeks when the team completes its work. However, they meet each day to evaluate its progress (daily Scrum).
4. The role of the Scrum Master is to ensure the team is focused on the sprint’s goals.
5. At the end of the sprint, the work should be delivered to a stakeholder (product owner).
6. After the sprint, the team reviews its work.
7. The next sprint starts when the team takes a number of other features from the product backlog and starts working again.

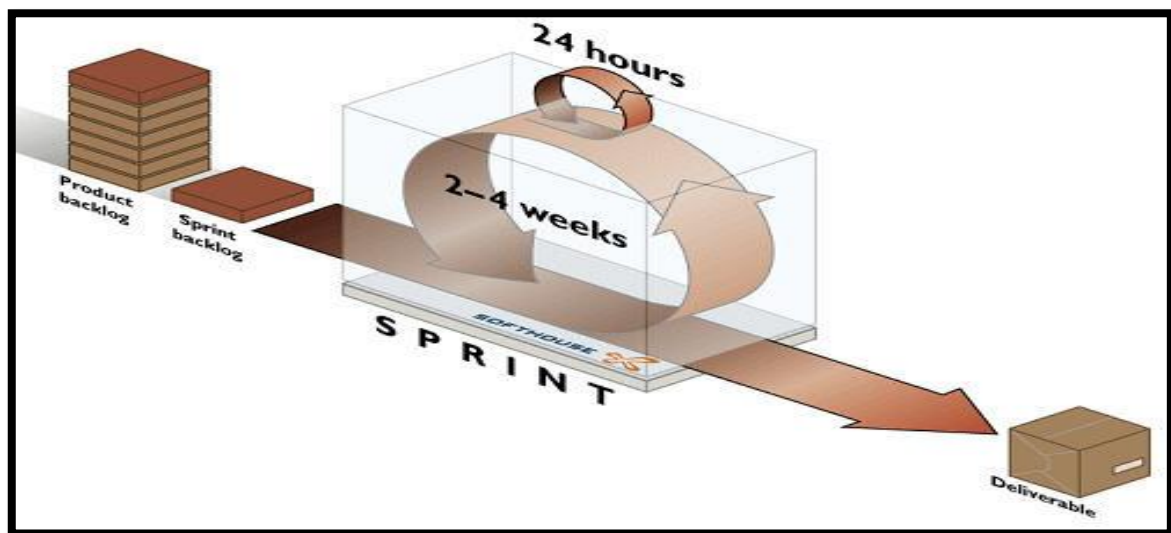


Figure 3. Scrum Alliance (Softwaretestinghelp.com, 2015)

2.8.2 Assumptions

As mentioned above, the Scrum approach requires teamwork; however, the project is an individual piece of work. Therefore, the Scrum approach will be initiated by one person

and the main principles of constructing a backlog, planning a sprint, and developing a piece of software during the sprint will be followed.

The search API returns the relevant tickets to a new ticket regardless of the product that the new ticket is opened for. It is assumed that it is the decision of the caller to filter the retrieved results according to specific criteria such as the product name, country and customer. These attributes can be captured from the ticket ID, for example: ticket ID = QA00QCB-48, country code = QA, product ID = 00 (PS-ECC), customer ID = QCB.

3. Application Development

3.1 Product Backlog

According to Scrum's process framework, creating a product backlog is the initial step of product development. The product backlog typically includes a brief description of the main features of the desired software product ordered based on their priorities. The first change lists the known and best-understood requirements (Scrumguides.org, 2014). The product backlog of the future system is presented in Table 1.

Feature No.	Feature Name	Priority	Sprint	Status	Note
1	Fetching ticket text	1	1	In-progress	Extract the text from all HD tickets and store it on HDFS.
2	Text analysing and indexing	2	2	Planned	Perform text analysis on the text contents of the tickets and construct the inverted index.
3	Building TF-IDF matrix	3	3	Planned	Generate TF-IDF matrix by using the inverted index data.
4	Developing the search API	4	4	Planned	Develop a Java API to retrieve and score the relevant tickets.

Table 1. Product backlog

Each feature is developed within a sprint which can have one of the following statuses: planned, in-progress and completed. The Gantt chart for the development steps is provided in Appendix A.

3.2 Sprint 1: Fetching the Ticket Text

3.2.1 Gathering Requirements and Analysis

To understand exactly what has to be done in this phase to proceed to the following phase, it is useful to identify what input the next phase (Text pre-processing and indexing) requires from this phase.

The second phase involves building a distributed inverted index which will be constructed using MapReduce which demands the data be represented in the form of key/value pairs. Therefore the extracting phase has to output the text contents of helpdesk tickets structured in key/value pairs. In this case the key is the ticket ID and the value is the text of the ticket.

Choosing the document unit

The first step for building any information retrieval is to identify the document unit for indexing, i.e. determine indexing granularity level (Manning et al, 2008). For the PS helpdesk system, the focus is to find the tickets that are related to a particular query, therefore it is suggested to consider the entire ticket as a document unit. However, more analysis is needed to decide whether the length of ticket is suitable to consider it as a document unit or whether we might need to treat the ticket sections as document units. Manning et al (2008) state that if the units get too small, important passages could be missed as terms are distributed over several smaller documents; if the units are too large false matches could occur and relevant information could be difficult to find.

To answer the question related to indexing granularity, we need to study the ticket structure and the length of the ticket sections (Figure 4):

1. Ticket structure

The ticket's contents are presented within a web document that consists of three major text fields:

- A. Ticket summary, which contains a brief description of the issue's subject
- B. Ticket description, which is filled in using a detailed description of the issue
- C. Ticket action body, which is a sequence of comments about the issue or the interactions between a customer and the service team

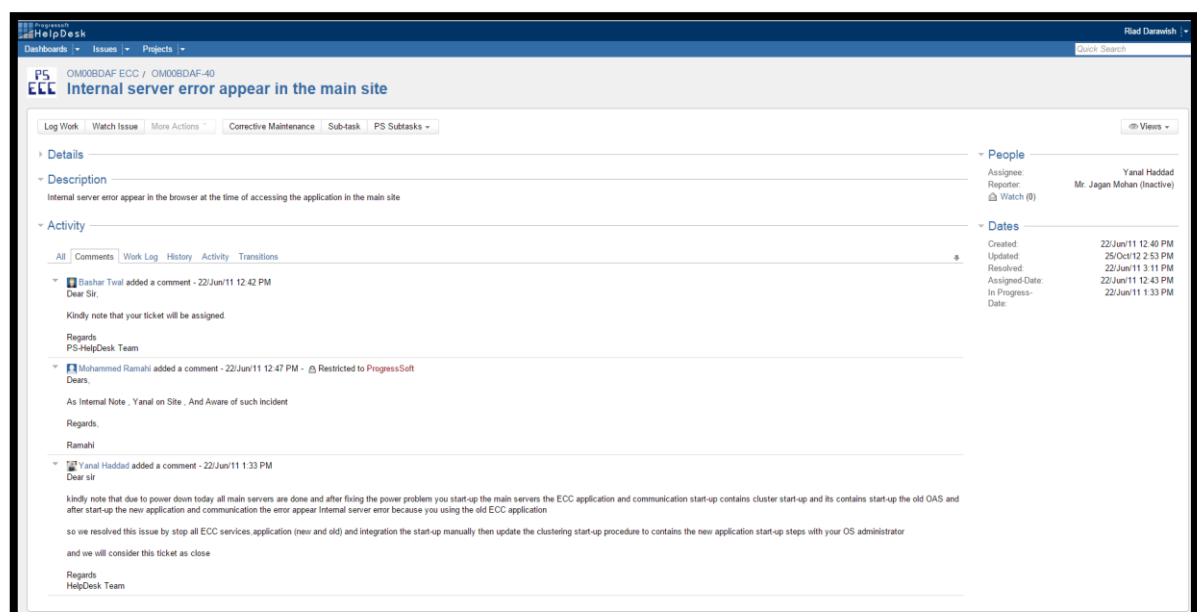


Figure 4. PS helpdesk ticket structure

2. Statistics about the ticket length and section lengths

Obviously, the summary field is small enough to be considered as a unit because its maximum length is 255 characters. However, the maximum capacity of a ticket's description is very large as its value is stored in a database column of CLOB data type which has a limit up to 4GB (Docs.oracle.com, 2015). The question here is what is the average length of a ticket's description in the collection? This average can be calculated as shown in Figure 5.

```
SQL> SELECT SUM(LENGTH(DESCRIPTION)) FROM JIRAISSUE WHERE DESCRIPTION IS NOT NULL;

SUM(LENGTH(DESCRIPTION))
-----
4944072

SQL> SELECT count(*) FROM JIRAISSUE WHERE DESCRIPTION IS NOT NULL ;

COUNT(*)
-----
17708

SQL> SELECT 4944072/17708 FROM DUAL;

4944072/17708
-----
279.19991
```

Figure 5. The average length of a ticket's description

As shown in Figure 6 the length of 279 bytes is still too small to consider the ticket's description as a document unit compared to 1986 bytes, which is the average length of a ticket's text in the collection.

```
SQL> SELECT SUM(LENGTH(JE.DESRIPTION)) + SUM(LENGTH(TJ.ACTIONBODY)) FROM JIRAISSUE JE , temp_jiraaction TJ WHERE tj.issueid = J
E.ID;

SUM(LENGTH(JE.DESRIPTION))+SUM(LENGTH(TJ.ACTIONBODY))
-----
38817792

SQL> SELECT count(*) FROM temp_jiraaction ;

COUNT(*)
-----
19542

SQL> SELECT 38817792/19542 FROM DUAL;

38817792/19542
-----
1986.37765
```

Figure 6. The average length of a ticket's text.

It becomes clear from the figures of the ticket summary and description that those values reflect the length of one or two sentences. According to Manning et al (2008), treating individual sentences as mini-documents will have a precision/recall trade-off. As a result of this analysis, it is clear that the average size of a ticket is suitable to use it as a document unit.

Obtaining the text of tickets

After identifying the document unit, we are required to find the best approach to acquire the raw content (textual part) from these units to be indexed. There are two main ways to obtain the text content from the ticket sections (summary, description and comments).

Since the tickets are web documents, web crawling can be one of the approaches. The web crawler fetches a set of pages in the fetch queue, extracts links from the fetched pages, adds the extracted links back to the fetch queue, and repeats this process many times (Perera and Gunarathne, 2013). McCandless et al (2010) recommend a number of web crawlers that are used by giant search engines like Google and Yahoo, among them are the following:

1. Solr (<http://lucene.apache.org/solr>), a sister project under the Apache Lucene umbrella, has support for natively ingesting relational databases and XML feeds, as well as handling rich documents through Tika integration.
2. Nutch (<http://lucene.apache.org/nutch>), another sister project under the Apache Lucene umbrella, has a high-scale crawler suitable for discovering content by crawling websites.
3. Grub (<http://www.grub.org>) is a popular open source web crawler.
4. Heritrix is Internet Archive's open source crawler (<http://crawler.archive.org>).
5. Droids, another subproject under the Apache Lucene umbrella, is currently under Apache incubation (<http://incubator.apache.org/droids>).
6. Aperture (<http://aperture.sourceforge.net>) has support for crawling websites, file systems and mail boxes and for extracting and indexing text.
7. The Google Enterprise Connector Manager project (<http://code.google.com/p/google-enterprise-connector-manager>) provides connectors for a number of non-web repositories.

The second approach is to export the database records that contain the text of the tickets. This could be easier than the crawling approach because there is no need to parse HTML documents and the all text can be obtained by a simple select statement rather than adding another layer of software to do this task.

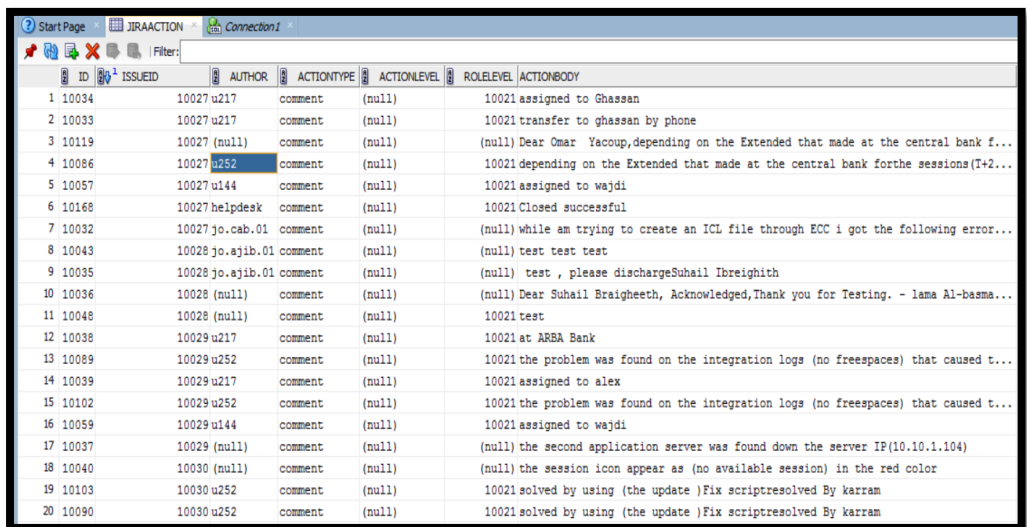
All the textual parts of tickets are stored in two tables:

1. JIRAISSUE: This table contains the ticket summaries and descriptions (Figure 7).
2. JIRAACTION: This table contains the ticket comments (Figure 8).

The two tables can be combined based on a common field which is named in JIRAISSUE as ID and in JIRAACTION as ISSUEID. The corresponding records of every ticket in the JIRAACTION table hold the same value of ISSUEID, therefore we must concatenate them in a single record using a PLSQL procedure JIRAACTION_CONCAT to store the result in a staging table called TEMP_JIRAACTION.

Start Page		JIRA ISSUE	Connection1					Actions...
Filter:								
ID	PKEY	PROJECT	REPORTER	ASSIGNEE	ISSUETYPE	SUMMARY	DESCRIPTION	
11782	AE00NSAU-2005	10024 ae.mbs.02	u193	5	CS	bank services had a failure today	CS bank services had a failure today which	
11794	QA00SCB-2004	10076 qa.scb.01	u319	5	List	of Brnches for all the Banks	Please provide the script to extract the list of Br	
11825	QA01CFA-2001	10078 qa.cfa.12	u323	6	-GFI	Fax Maker	Dearwe need the product key for our GFI fax maker	
11838	FTRECC-2	10096 jo.eccu.01	u324	7	development	the Log Files	To contain detailed information about the operation	
11840	SY01HMSO-2001	10091 sy.hmsc.12	u003	5	question		How are you?and wish you happy new year.Pls. find th	
12063	SY01HMSO-2004	10091 sy.hmsc.11	u332	5	Problem	in folder permissions	when we want to give permissions to group on old fo	
12068	JO00ARAB-2036	10027 jo.arab.02	u193	5	disconnection	between AB, IIAB and CBJ com...	disconnection between AB, IIAB and CBJ communication	
12072	QM00PS-18	10155 u243	u243	5	Failed	to connect to the server	the batch was not uploaded from scan page to next q	
12083	AE00NRAD-2026	10024 ae.nbad.03	u231	5	Stuck	in reverse posting	The attached cheque was stuck in Reverse posting fo	
12085	JO00ABC-2004	10025 jo.abc.01	u342	6	LogoutUserInterval	activation	Dears,,You Are Kindly Requested To Activate The Log	
12090	JO00ABC-2005	10025 jo.abc.01	u193	5	New User	Creation	Dears,,Kindly, We Have The Following Case: 1- We	
12095	JO00BOJ-2003	10030 jo.boj.01	u193	5	database server	and history exchange query	full system load and cpu usage is is very lowdw ser	
12112	QA00DB-2016	10067 qa.db.01	u100	6	Removing	unprocessed cheques	Dear Helpdesk, kindly remove the cheques which coul	
12115	JO00JAB-2006	10040 u231	u259	8	Investigating	Account request message	Al-Tayeb to Investigate Account request message.The	
12116	QA00SCB-2029	10076 qa.scb.01	u355	5	31 cheque	under Comm. Pending Checks	pls check and do the needful.	
12169	QA00IBQ-2002	10069 qa.ibq.01	u319	5	Installing	New Printer	We are facing a problem with installing T3230 printe	
12172	QA00ABQ-2003	10063 qa.abq.02	u319	6	Banks	name with branch codes	Kindly provide us by banks name and branch code	
12178	QA00CBQ-2002	10066 qa.cbq.01	u319	5	TIME OUT	CHEQUES FROM VARIOUS BANKS.	We have today also 149 cheques still waiting for reg	
12181	QM00PS-14	10155 u060	u013	5	Security	Auditing for user management	When a user which have user management privilege make	
12192	QA00QNB-2005	10075 qa.qnb.01	u319	6	Move	cheques to from Master History database	I have new business requirements to move all replie	

Figure 7. JIRAISSUE table



ID	ISSUEID	AUTHOR	ACTIONTYPE	ACTIONLEVEL	ROLELEVEL	ACTIONBODY
1	10034	10027 u217	comment	(null)		10021 assigned to Ghassan
2	10033	10027 u217	comment	(null)		10021 transfer to ghassan by phone
3	10119	10027 (null)	comment	(null)		(null) Dear Omar Yacoup, depending on the Extended that made at the central bank f...
4	10086	10027 u252	comment	(null)		10021 depending on the Extended that made at the central bank forthe sessions(T+2...
5	10057	10027 u144	comment	(null)		10021 assigned to wajdi
6	10168	10027 helpdesk	comment	(null)		10021 Closed successful
7	10032	10027 jo.cab.01	comment	(null)		(null) while am trying to create an ICL file through ECC i got the following error...
8	10043	10028 jo.ajib.01	comment	(null)		(null) test test test
9	10035	10028 jo.ajib.01	comment	(null)		(null) test , please discharge Suhail Ibreighith
10	10036	10028 (null)	comment	(null)		(null) Dear Suhail Braigheeth, Acknowledged, Thank you for Testing. - lama Al-basma...
11	10048	10028 (null)	comment	(null)		10021 test
12	10038	10029 u217	comment	(null)		10021 at ARBA Bank
13	10089	10029 u252	comment	(null)		10021 the problem was found on the integration logs (no freespaces) that caused t...
14	10039	10029 u217	comment	(null)		10021 assigned to alex
15	10102	10029 u252	comment	(null)		10021 the problem was found on the integration logs (no freespaces) that caused t...
16	10059	10029 u144	comment	(null)		10021 assigned to wajdi
17	10037	10029 (null)	comment	(null)		(null) the second application server was found down the server IP(10.10.1.104)
18	10040	10030 (null)	comment	(null)		(null) the session icon appear as (no available session) in the red color
19	10103	10030 u252	comment	(null)		10021 solved by using (the update)Fix scriptresolved By karram
20	10090	10030 u252	comment	(null)		10021 solved by using (the update)Fix scriptresolved By karram

Figure 8. JIRA ACTION table

Moving Text Data to HDFS

The next phase involves building a distributed index that processes the input data (text content of tickets) using Hadoop. Hadoop comes with a distributed file system called HDFS, which stands for Hadoop Distributed File System. HDFS is designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware (White, 2012). Although distributed jobs in Hadoop can be run against data on local file systems, HDFS was chosen to store the input data because:

- Data scalability and availability
- Hadoop performs better with processing data stored in HDFS rather than local file system because it tries to collate the data with the compute node, so data access is fast since it is local (White, 2012). This is the core of MapReduce which is known as a data locality.

- Benefiting from partitioning the data across a number of separate machines using cheap commodity hardware.

Apache Sqoop, an open source project, can be used to move the ticket records from the source (Oracle database) to the destination (HDFS). Apache Sqoop is designed for efficiently transferring bulk data between Apache Hadoop and structured data stores such as relational databases (Sqoop.apache.org, 2015). It provides an easy to use, fast and efficient way to extract and load data into the HDFS (Perez-Goytia, 2015). It also supports incremental imports from RDBMS to HDFS and it is easily extensible as it works with the most popular RDBMS such as Oracle, MySql, SqlServer and DB2. Sqoop can also give five times faster performance if it is plugged in with Quest Sqoop (Worms, 2013).

3.2.2 Design Sprint 1's Process

The ordered steps needed to complete this phase can be described as follows:

1. Build a staging table TEMP_JIRAACTION in helpdesk database.
2. Execute PLSQL procedure to concatenate the corresponding comment records for each ticket in table JIRAACTION to one record in TEMP_JIRAACTION table.
3. Execute Sqoop command line to fetch the data from the database tables and write the results to HDFS.

The Unified Modelling Language (UML) use-case diagram can be used to visualise every step of development and can be applied to the analysis and design of information systems using the Agile approach (Iyaniwura, 2012). The use-case diagram for sprint 1 is shown in Figure 11. The primary actor at this phase is the developer who will execute the use-case “Writing data to HDFS” (step 3) which involves “Concatenating records” (steps 1 & 2) at least once. The “include” relationship between use-cases leads to the execution of the “Concatenating records” in order to execute the “Writing data to HDFS” use-case. PS helpdesk is an external system and is therefore presented as a secondary actor. The system boundary shows what is included in the system.

Since this is the first sprint, the UML use-case diagram is simplistic, however it will become more complex during the next sprints.

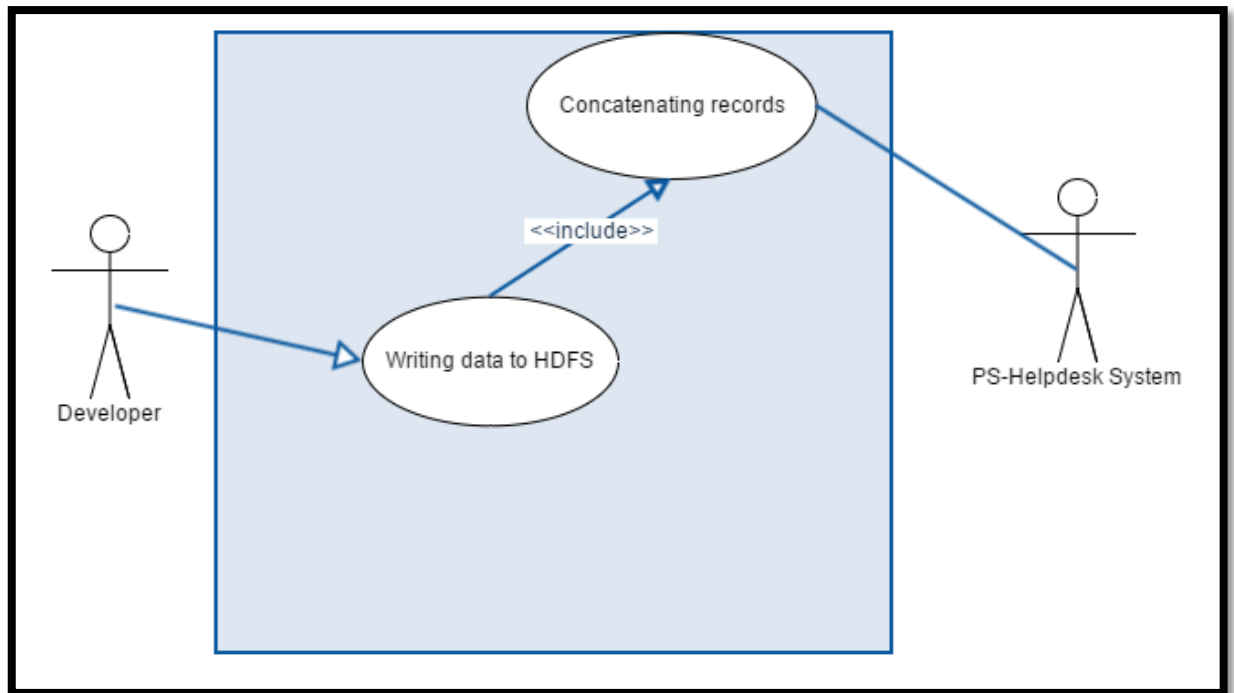


Figure 9. Use-case diagram for fetching the ticket text.

3.2.3 Implementation of Sprint1's Process

As discussed in the Requirements section, the appropriate way to obtain the textual part of tickets is to retrieve their corresponding records from the database instead of crawling their web documents. Firstly, staging table TEMP_JIRA ACTION has to be created with two columns, "ISSUEID" and "ACTIONBODY". Secondly, a PLSQL procedure is created and then executed to populate the staging table from table JIRA ACTION. Finally, a Sqoop command is constructed with SQL select statement to extract the records from the tables JIRA ISSUE and JIRA ACTION and write them into an HDFS file.

1. Create TEMP_JIRA ACTION table

SQL Command:

```
CREATE TABLE TEMP_JIRA ACTION (ISSUEID NUMBER (18,0),
ACTIONBODY CLOB);
```

2. Create JIARA ACTION_CONCAT PLSQL procedure


```

1 CREATE OR REPLACE PROCEDURE JIARACTION_CONCAT
2 IS
3 CURSOR C_JIRACTION IS SELECT DISTINCT ISSUEID FROM JIRACTION;
4 CURSOR C_ACTIONBODY (V_ISSUEID JIRACTION.ISSUEID%TYPE) IS SELECT actionbody
5 FROM JIRACTION WHERE ISSUEID=V_ISSUEID ORDER BY ID;
6 REC_ACTIONBODY C_ACTIONBODY%ROWTYPE;
7 V_ACTIONBODY jiraaction.actionbody%TYPE;
8 BEGIN
9     FOR REC_JIRACTION IN C_JIRACTION
10     LOOP
11         DBMS_LOB.CREATETEMPORARY(V_ACTIONBODY, TRUE);
12         OPEN C_ACTIONBODY(REC_JIRACTION.ISSUEID);
13         LOOP
14             FETCH C_ACTIONBODY INTO REC_ACTIONBODY;
15             DBMS_LOB.APPEND(V_ACTIONBODY, REPLACE(REC_ACTIONBODY.actionbody, chr(10), ' '));
16             EXIT WHEN C_ACTIONBODY%NOTFOUND;
17         END LOOP;
18         INSERT INTO temp_jiraaction VALUES(REC_JIRACTION.ISSUEID,V_ACTIONBODY);
19         COMMIT;
20         CLOSE C_ACTIONBODY;
21     END LOOP;
22 END;
23 /

```

Figure 10. Code of JIARACTION_CONCAT PLSQL procedure

3. Creating Sqoop command

Sqoop import command is used to transfer the data from the Oracle database to Hadoop. Before Sqoop is used, a release of Hadoop must be installed and configured. The Sqoop import command is shown in figure 11.

```

45 sqoop import --connect jdbc:oracle:thin:@//192.168.0.10:1521/helpdesk --username supp_hd -P --query "SELECT JIS.ID, '##ENDOFID' || JIS.PKEY || '##ENDOFFKEY',
46 REPLACE(JIS.SUMMARY, CHR(10), ' ') || '##ENDOFSUMMARY', replace(replace(JIS.DESRIPTION,CHR(10), ' '),CHR(13), ' ') || '##ENDOFDESCRIPTION',
replace(replace(JAC.ACTIONBODY,CHR(10), ' '),CHR(13), ' ') || '##ENDOFACTIONBODY' FROM SUPP_HD.JIRAISSUE JIS JOIN SUPP_HD.temp_jiraaction JAC ON
(JIS.ID=JAC.ISSUEID) WHERE \${CONDITIONS} " --fields-terminated-by '\t' --num-mappers 1 --target-dir /data/helpdeskData --split-by JIS.ID

```

Figure 11. Sqoop import command

The SQL statement used in the Sqoop import command selects the following columns:

1. JIRAISSUE.ID: Unique ID that identifies every issue provided by helpdesk system.
2. JIRAISSUE.PKEY: Unique key that identifies every issue provided by helpdesk.
3. JIRAISSUE.SUMMARY: Brief description of issue subject.
4. JIRAISSUE.DESRIPTION: Detailed description of issue.
5. TEMP_JIRACTION.ACTIONBODY: Concatenated values of the column JIRACTION.ACTIONBODY.

The retrieved columns are segregated with specific notations to help identify the ticket section in every line during the streaming of the file for the indexing process.

The retrieved results are stored in files under the HDFS directory “/data/helpdeskData”. The imported data were enough to fit in a single file because the size of the exported data is less than the default block size of HDFS (64 MB). The following figure shows what the imported data look like.

```

10083 ##ENDOFID 300071B-139##ENDOFKEY postdated##ENDOFSUMMARY 200704261000029##ENDOFDESCRIPTION when making qa to postdated chege batches from the page which contains the batch # message appear
(post dated checks cant be urgent) from the batch details its accepted.Dear Saleem Bader, Ayesah Amawi had been dispatched at 10:30am to resolve this issue Yanal Haddad ProgressSoft HelpDesk Team Email
: Helpdesk@Progresssoft.com Web:http://194.165.134.179/otrs/customer.pl assigned to Ayesah Amawi Dear Saleem Bader, We believe that the above issue is now resolved. The solution to this issue is cpoing new f
iles to APP serve('BatchInfoTag.tag', 'ChequeInfoTag.tag') Please Confirm Resolution. Ayesah Amawi ECC Support Team Email: Helpdesk@Progresssoft.com Web:http://194.165.134.179/otrs/customer.pl thank you fr
om Eng.Mohammad AL-Masa' deh thank you from Eng.Mohammad AL-Masa' deh ##ENDOFACTIONBODY
10086 ##ENDOFID 3000803-142##ENDOFKEY error message##ENDOFSUMMARY 200704261000056##ENDOFDESCRIPTION During the period of the parallel there has been an error which occurred more than one tim
e "failed to retrieve error description " when entering any module(Inward, Outward, Reports) Yanal Haddad ProgressSoft HelpDesk Team Email: Helpdesk@Progresssoft.com Web:http://194.165.134.179/otrs/cus
tomer.pl Dear Fayyad Mujahed, wajdi Sahaweh & fadi Hamleh had been dispatched at 2:00 to resolve this issue Yanal Haddad ProgressSoft HelpDesk Team Email: Helpdesk@Progresssoft.com Web:http://194.165
.134.179/otrs/customer.pl Dear Fayyad Mujahed, We believe that the above issue is now resolved. The solution to this issue is by restarting the OC4J and CATALINA Please Confirm Resolution. Wajdi Sah
aweh ECC Support Team Email: Helpdesk@Progresssoft.com Web:http://194.165.134.179/otrs/customer.pl Closed. Closed. ##ENDOFACTIONBODY
10081 ##ENDOFID 30005CB-136##ENDOFKEY Problem in scanning outward clearing##ENDOFSUMMARY 200704251000085##ENDOFDESCRIPTION When users are trying to scan the outward clearing, and press scan
bottom he got the error message scan error " the operation has been completed successfully" and he will not be able to scan any cheque until he restart the PC. I attached the below attachment. Dear Mohammed
Milkawi, Jehad Ghareeb had been dispatched at 9:15am to resolve this issue Yanal Haddad ECC Support Team Email: Helpdesk@Progresssoft.com Web:http://194.165.134.179/otrs/customer.pl Dear Mohammed Milkawi, R
egarding to the scanner problem, the message didn't appear since the operating system for the workstation request had been restarted. However, we couldn't take any technical action to solve this situation excep
t check the "ECCJordanScanner.ocx" version and it was installed successfully. For further Assistance, please contact Islam Matal'a from your IT Department site. Please Confirm Resolution. Wajdi Sahaweh E
CC Support Team Email: Helpdesk@Progresssoft.com Web:http://194.165.134.179/otrs/customer.pl Closed. Closed. ##ENDOFACTIONBODY
10092 ##ENDOFID 3000CAB-148##ENDOFKEY Active Directory##ENDOFSUMMARY 200704301000002##ENDOFDESCRIPTION We need to test active directory issue for users definition.Assigned to Tanfiq Khalili at
11:00 am. Dear Omar Yacoup, Please note we have disregarded this ticket as it's a duplicate of Ticket #2007043010000011 Lana Al-Basmawi PS-Helpdesk Team Email: Helpdesk@Progresssoft.com Web:http://194.165.13
4.179/otrs/customer.pl Dear Omar Yacoup, Please note we have disregarded this ticket as it's a duplicate of Ticket #2007043010000011 Lana Al-Basmawi PS-Helpdesk Team Email: Helpdesk@Progresssoft.com Web:ht
tp://194.165.134.179/otrs/customer.pl ##ENDOFACTIONBODY

```

Figure 12. Extracted records in an HDFS file

Each line in the HDFS file represents the textual content of an HD ticket which is split into a number of sections that determine every part of the ticket.

3.2.4 Sprint 1 Backlog

The database dump was taken from the production database of the PS helpdesk system on 16th September 2014.

Fetching the ticket text phase took 18 days to complete and after this sprint, the product backlog was updated, as shown in Table 2.

Feature No.	Feature Name	Priority	Sprint	Status	Note
1	Fetching the text from helpdesk tickets	1	1	Completed	Extract the text from all HD tickets and store them on HDFS.
2	Analysing and indexing text	2	2	In-progress	Perform text analysis on the text contents of the tickets and construct the inverted index.
3	Building TF-IDF matrix	3	3	Planned	Generate TF-IDF matrix using the inverted index data.
4	Developing the search API	4	4	Planned	Develop a Java API to retrieve and score the relevant tickets.

Table 2. Product backlog after sprint 1

3.3 Sprint 2: Text Analysis and Indexing

Now the input file contains the text of all the tickets which are represented in string lines, each of which contains the ticket ID and the complete textual part of the ticket. The ticket sections (summary, description and comments) are segregated by specific notations. The boundary of this sprint is to read the lines from the HDFS file and build an inverted index that maps all the words (terms) in the collection to their postings list. The posting includes the identifier of the appropriate ticket, the frequency of that term in this ticket, and other information related to every occurrence of that term in the ticket.

3.3.1 Gathering Requirements and Analysis

The process of building an inverted index consists of the following major steps:

1. Collect the documents (tickets) to be indexed. This step has been already completed in sprint 1, where each ticket is represented as a lines of text.
2. Text tokenisation. This step comprises the process of converting the text of each document into tokens. This technique is explained in detail in the Text Analysis section (Page 29).
3. Perform linguistic pre-processing. This step comprises the process of normalising the tokens generated in the previous step to produce the terms that will be added to the inverted index. This technique is explained in detail in the Text Analysis section (Page 29). The second and third steps together create the process of text analysis. Whenever the text needs to be transformed into terms, text analysis is needed. In our situation, the text analysis is required during the phases of indexing and searching.
4. Build the inverted index. This step involves mapping each term to its postings list. The structure of the postings list is represented in JSON format (as shown in Figure 13) that contains the following elements:
 - A. DocFreq: The total number of tickets in the data set that contain the term.
 - B. PostingLists: An array of the term's postings. The element of the array is structured as follows:
 - a. docID: The ticket ID.
 - b. termFreq: The number of occurrences of the term in this ticket.
 - c. TermPositions: An array that has its elements carries information about the occurrences of the term in the text, i.e. each element contains the attributes of the term for a particular occurrence of that term in the ticket. The structure of the elements contains:
 - i. position: An integer number that represents the position of the term in the ticket. The purpose of capturing this attribute is for future enhancements in weighing equations.
 - ii. termLocation: An integer number that ranges between 1 and 3. It is assigned a 3 if the occurrence of the term is in the ticket summary, a 2 for the ticket description and a 1 for the ticket comments. When calculating the weight of each term in the ticket, the value of this

- element is considered in the weighting equation to give a bonus for the terms based on its location in the ticket, whether it appeared in the summary field, description field or comments.
- iii. startOffset: The character position in the original text where the term text begins (McCandless et al, 2010).
 - iv. endOffset: The position just after the last character of the term text. These offset attributes are captured for a future highlighting feature, where the matched terms are highlighted in search results (McCandless et al, 2010).
 - v. termType: The term type. For example, WORD, NUMBER or HOST. This attribute is captured for future enhancements to add a filter that only allows specific types of terms to be included in the index.

```

1  {"DocFreq":2,
2   "PostingLists":
3   [
4     {"docID":"J000JDIB-9",
5      "termFreq":2,
6      "TermPositions":[
7        {"position":"113",
8         "termLocation":"2",
9         "startOffset":"679",
10        "endOffset":"683",
11        "termType":"<WORD>"},
12        {"position":"200",
13         "termLocation":"1",
14         "startOffset":"850",
15         "endOffset":"854",
16         "termType":"<WORD>"},
17      ]
18    },
19    {"docID":"J000JKB-480",
20     "termFreq":1,
21     "TermPositions":[
22       {"position":"381",
23        "termLocation":"1",
24        "startOffset":"3033",
25        "endOffset":"3037",
26        "termType":"<WORD>"},
27     ]
28   }
29 ]
30 }

```

Figure 13. The structure of the term's postings list in JSON format.

A secondary file is required to store an index to the data stored in the inverted file. The secondary file is called a Map-File because it contains a list of all terms that are available in the inverted file, where each term is mapped to the byte offset to its record in the inverted file. This file is very useful at the retrieval phase when it is needed to randomly access records stored in a large partitioned inverted file.

Text Analysis

Text analysis is the process of converting field text into its most fundamental indexed representation, terms (McCandless et al, 2010). These terms are used to determine what documents match a query during a search. These terms are the primitive building blocks for searching (McCandless et al, 2010). The element that performs the text analysis process is the text analyser. It tokenises the text by extracting words, discarding punctuation,

removing accents from characters, lower casing (also called normalising), removing common words, reducing words to a root form (stemming), or changing words into the basic form (lemmatisation) (McCandless et al, 2010).

Analysers are used when the document is indexed; its full-text fields are analysed into terms that are used to create the inverted index. However, a search is required on a full-text field, we need to pass the query string through the same analysis process to ensure we are searching for terms in the same form as those that exist in the index (Elasticsearch.org, 2015).

Choosing the Text Analyser API

According to the search conducted to find a text analyser API that suits the requirements for the case under study, Lucene API, which is a free open-source information retrieval software library, was found to provide a number of built-in analysers, among which is the StandardAnalyzer which meets all of our requirements. Lucene API is written in Java and is supported by Apache (<http://lucene.apache.org/core/>).

The main characteristics of Lucene's StandardAnalyzer are that it (McCandless et al, 2010):

- Maintains the token position, token start and end character offsets in the original text;
- Performs word, line, sentence and character boundary analysis;
- Has quite a bit of logic to identify certain kinds of tokens, such as company names, email addresses and hostnames;
- Lower cases each token and removes stop words and punctuation;
- Is designed to work with text in almost any Western (European-based) language.

The StandardAnalyzer was adopted as its characteristics were found to meet the requirements for the proposed solution.

3.3.2 Designing Sprint 2's Process

The computation process of text analysing and indexing is coded within one MapReduce job. Therefore, it is represented in the use-case diagram as one use-case called “Text analysis and indexing job” as shown in Figure 14.

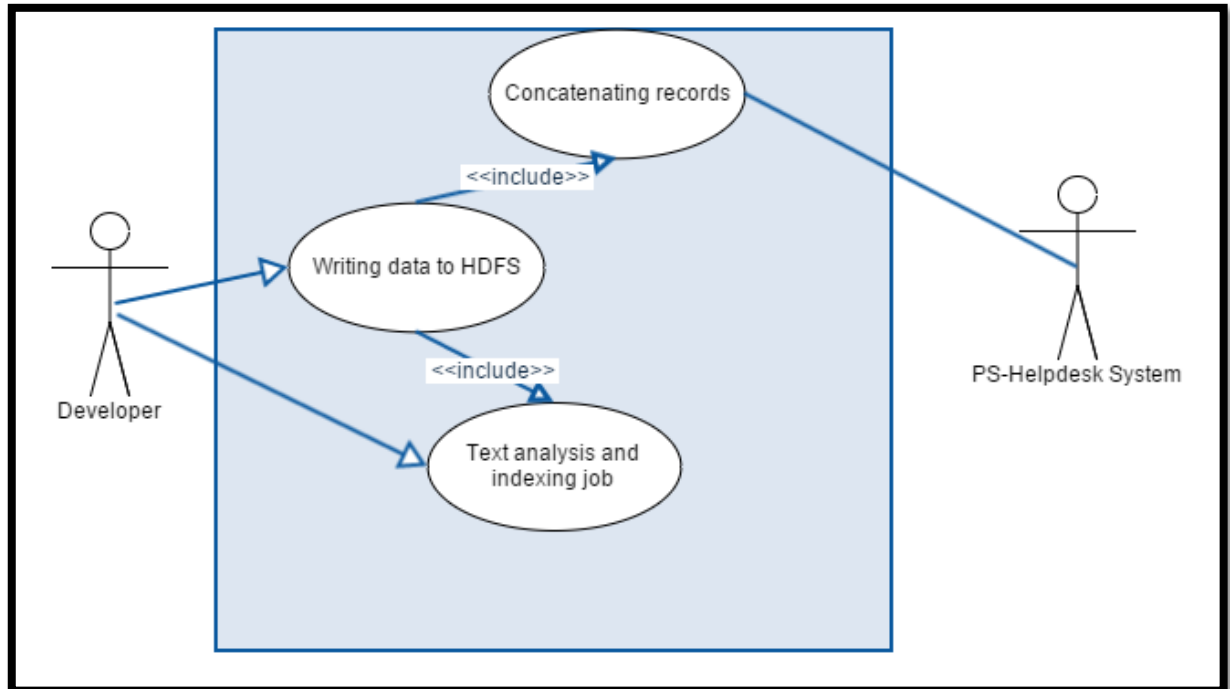


Figure 14. Use-case diagram of text analysis and indexing

Upon executing the job, the following logical actions are performed in order as shown in Figure 17:

1. The job parses each line in the input file by using a custom Hadoop InputFormat. The following values are captured as a result of parsing the line successfully:
 - i. Ticket ID
 - ii. Ticket Summary
 - iii. Ticket Description
 - iv. Ticket Comment
2. The parsed line is then passed to the map function in the form <Key, Value>. The key and the value are instances of Hadoop text data type. The key is assigned to ticket ID. The value holds concatenated strings that represent the ticket summary, description and comments. The text of the summary and the description are segregated by the notations #ETSM# and description and the comments by #ETDS#.
3. The map function passes the string fields to the Lucene StandardAnalyzer. The analyser converts the strings into a stream of terms (perform text analysing).
4. The map function iterates through all the generated terms and outputs each of them in the form <Key, Value>, where the output key is an instance of text data type and the output value is an instance of a custom Hadoop writable data type called TermWritable. The key is assigned to the term text (word). The instance has a number of the fields that hold the following term's attributes:

- i. Term type
 - ii. Term start offset
 - iii. Term end offset
 - iv. Term position
 - v. Ticket ID that the term occurs in
 - vi. Term location (to distinguish whether the term occurs in the ticket summary, description or comments. The value of this field is used during term weighting calculation to give a bonus to the query terms based on its location in the ticket)
5. The reduce function receives a list of TermWritable instances as an input value and an instance of text that contains the term's text as an input key. The instances of TermWritable carry all ticket IDs that the term (key) appears in, along with the attributes of each occurrence. The reduce function performs the following logic:
 - i. Calculates the term frequency in each ticket
 - ii. Calculates the number of tickets in the collection that contain the term
 - iii. Organises each ticket ID, the term's attributes for every occurrence in that ticket
 - iv. Outputs a key of text instance that holds the term's text and value of text instance that holds the term's postings list, as shown earlier in Figure 13.

Upon completing the computation of the reduce function, an output file (inverted index file) has all the terms in the collection which are listed against their postings lists, as shown in Figure 15. Finally, the job proceeds by creating an index file for the inverted index. This index file is called a Map-File which contains a list of all terms mapped to byte offset of their postings lists in the inverted index file.

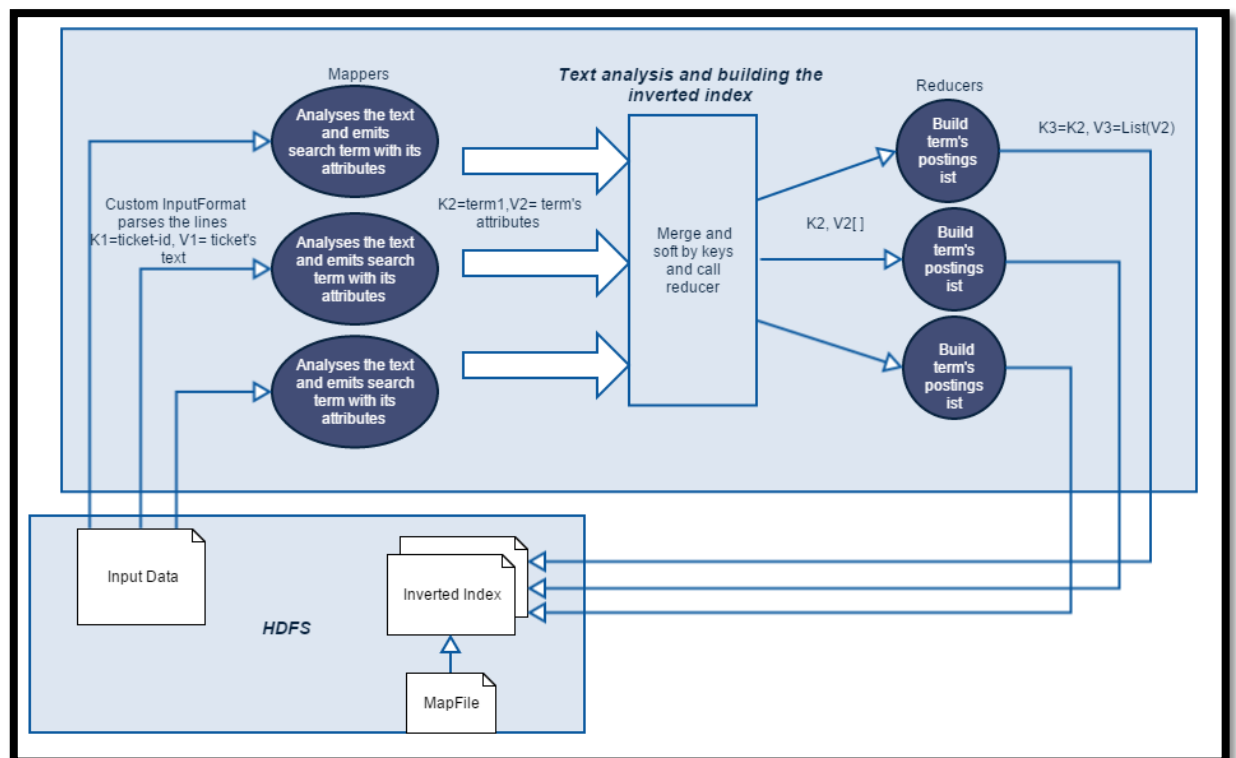


Figure 15. Text analysis and inverted index MapReduce job execution

3.3.3 Implementing Sprint 2's Process

The text analysis and indexing are performed within an implementation of the Hadoop MapReduce job. Four components have to be coded in order to build the Job:

1. Hadoop custom Inputformat "TicketFileInputFormat"
2. Mapper function
3. Reducer function
4. Job driver

The job accepts an input directory, a mapper function and a reducer function as inputs. It uses the mapper function to process the data in parallel, and it uses the reducer function to collect results of the mapper and to produce the final results. The mapper sends its results to the reducer using a key/value model.

In spite the fact that Hadoop API is built on Java, it allows the programmer to write the MapReduce programs in languages other than Java such as Ruby, Python or C++. Java was chosen as the programming language for this project as the author's strongest skills are in this language.

The Java classes for the text analysis and indexed index MapReduce job are implemented within the package "org.greenwich.invertedIndex".

Hadoop Custom InputFormat (formatter)

Hadoop supports the processing of many different formats and types of data through InputFormat. The InputFormat of a Hadoop MapReduce computation generates the key/value pair inputs for the mappers by parsing the input data. Hadoop provides some of the InputFormat implementations to support several common data formats such as TextInputFormat and KeyValueTextInputFormat (Perera and Gunarathne, 2013). However, none of these built-in InputFormats is suitable to parse the input file that contains the ticket text, therefore it is required to invest in creating a custom Hadoop InputFormat called "TicketFileInputFormat".

The formatter creates a record reader "TicketRecordReader", and the record reader will do the bulk of the real work. When the Hadoop job is executed, it will find the formatter, create a new record reader passing each input file, read records from record readers, and pass those records to the map tasks.

The record reader reads the input file line by line. Each line in the input file comprises all the text of a ticket structured in the following format:

```
ID      ##ENDOFID      Ticket's      ID      ##ENDOFKEY      Ticket's      Summary
##ENDOFSUMMARY Ticket's Description ##ENDOFDESCRIPTION Ticket's
Comments ##ENDOFACITIONBODY
```

Upon reading the line, the reader passes the following key/value to the mapper:

Key = Ticket ID

Value = Ticket Summary #ETDS# Ticket's Description #ETDS# Ticket Comments

Map Function

The map function is implemented in “GenerateTermsMapper” class that extends the Hadoop mapper class. The map function receives <Text key, Text value>. The key holds the ticket ID. The value holds the concatenated strings that represent the fields of Ticket Summary, Ticket Description and Ticket Comment. The text of the summary and the description are segregated from each other by the notation “#ETSM#” and, description and the comments by “#ETDS#”.

The map function uses an external library (Lucene’s package org.apache.lucene.analysis) to analyse the string fields of the value. It passes the text of these fields to Lucene’s StandardAnalyzer which in turn converts the text into a stream of terms.

The map function iterates through all the streamed terms and outputs each of them in the form <Key, Value>, where the output key is an instance of text data type which holds the term text (word). The output value is an instance of a custom Hadoop writable data type called TermWritable.

The instance has a number of fields that hold the following term’s attributes:

- Term type
- Term start offset
- Term end offset
- Term position
- Ticket ID that the term occurs in
- Term location

Reduce Function

The reduce function is implemented in “GeneratePositingListsReducer” class that extends the Hadoop reducer class. The reduce function receives <Text key, Iterable<TermWritable> value>. The key holds the term’s text. The instances of TermWritable carry all ticket IDs that the term (key) appears in, along with the term’s attributes for each occurrence. The reduce function performs the following logic:

1. Calculates the term frequency in each ticket.
2. Calculates the number of tickets in the collection that contain the term.
3. Organises for each ticket ID, the term’s attributes for every occurrence in that ticket.
4. Outputs a key of text instance that holds the term’s text and value of text instance that holds the term’s postings list and number of tickets that contain the term “DocFreq”. The value’s elements are coded into JSON format, which is “a lightweight data-interchange format” (www.JSON.org, 2015). The following example shows output key and value:

Key	Value
Jasper	{ "DocFreq":1,"PostingLists":[{"docID":"JO00SCB-585","termFreq":1,"TermPositions":[{"position":"1","termLocation":"2","endOffset":"17","termType":"<ALPHANUM>","startOffset":"1"}]}}

MapFile

The output of the MapReduce computation will be consumed by the TF-IDF MapReduce job and searching API; therefore, it is important to store the result of a MapReduce computation in a format that can be consumed efficiently by another MapReduce job. The results of the reduce function are stored in a file-based data structure called Hadoop's SequenceFile which provides a persistent data structure for binary key/value pairs. The SequenceFile is generated by using Hadoop's OutputFormat called SequenceFileOutputFormat which serialises the data to Hadoop's sequence files. Figure 16 displays a textual form of the SequenceFile which contains the records of the inverted index. The records are sorted alphabetically by the keys (terms).

```
anawidera {"DocFreq":1,"PostingLists":[{"docID":"J000AJ1B-2056","termFreq":1,"TermPositions":[{"position":92,"termLocation":1,"endOffset":579,"termType":"ALPHANUM","startOffset":569}]]}]
anawifeb {"DocFreq":1,"PostingLists":[{"docID":"AE00NBAD-2014","termFreq":1,"TermPositions":[{"position":520,"termLocation":1,"endOffset":3074,"termType":"ALPHANUM","startOffset":3066}]]}]
anawifind {"DocFreq":1,"PostingLists":[{"docID":"J000JKB-2080","termFreq":1,"TermPositions":[{"position":246,"termLocation":1,"endOffset":1420,"termType":"ALPHANUM","startOffset":1411}]]}]
anawil {"DocFreq":1,"PostingLists":[{"docID":"J000JKB-2019","termFreq":1,"TermPositions":[{"position":16,"termLocation":1,"endOffset":108,"termType":"ALPHANUM","startOffset":102}]]}]
anawil {"DocFreq":1,"PostingLists":[{"docID":"J000AJ1B-2056","termFreq":1,"TermPositions":[{"position":27,"termLocation":1,"endOffset":163,"termType":"APOSTROPHES","startOffset":154}]]}]
anawilog {"DocFreq":1,"PostingLists":[{"docID":"J000AJ1B-2039","termFreq":1,"TermPositions":[{"position":13,"termLocation":1,"endOffset":89,"termType":"ALPHANUM","startOffset":80}]]}]
anawileas {"DocFreq":2,"PostingLists":[{"docID":"J000JKB-2053","termFreq":1,"TermPositions":[{"position":51,"termLocation":1,"endOffset":305,"termType":"ALPHANUM","startOffset":294}],["docID":"J000JKB-2174","termFreq":1,"TermPositions":[{"position":288,"termLocation":1,"endOffset":1753,"termType":"ALPHANUM","startOffset":1744}]]}]
anawiregard {"DocFreq":1,"PostingLists":[{"docID":"J000JKB-2017","termFreq":1,"TermPositions":[{"position":21,"termLocation":1,"endOffset":140,"termType":"ALPHANUM","startOffset":126}]]}]
anawiresolv {"DocFreq":1,"PostingLists":[{"docID":"J000JKB-2055","termFreq":1,"TermPositions":[{"position":38,"termLocation":1,"endOffset":263,"termType":"ALPHANUM","startOffset":252}]]}]
anawisolv {"DocFreq":1,"PostingLists":[{"docID":"J001ACC-2012","termFreq":1,"TermPositions":[{"position":9,"termLocation":1,"endOffset":67,"termType":"ALPHANUM","startOffset":56}]]}]
anawith {"DocFreq":2,"PostingLists":[{"docID":"AE00NBAD-2049","termFreq":1,"TermPositions":[{"position":10,"termLocation":1,"endOffset":73,"termType":"ALPHANUM","startOffset":65}],["docID":"J000CAI837","termFreq":1,"TermPositions":[{"position":227,"termLocation":1,"endOffset":1508,"termType":"ALPHANUM","startOffset":1500}]]}]
anawitait {"DocFreq":2,"PostingLists":[{"docID":"J000JKB-2019","termFreq":1,"TermPositions":[{"position":39,"termLocation":1,"endOffset":264,"termType":"ALPHANUM","startOffset":252}],["docID":
```

Figure 16. Inverted index stored in Hadoop SequenceFile

A search for terms in an inverted file might involve a full scan operation to find the entries of those terms. As a result this process will involve an expensive disk operation, especially when the inverted file is large. Thus, this structure might have a significant impact on the system's performance. To solve this issue, an index file is created which contains a list or dictionary of all terms, and contains a map from the term to that term's offset in the inverted index file. This can be implemented using Hadoop's MapFile, which can store an index in the SequenceFiles, making it useful to access random records stored in a large SequenceFile (Perera and Gunarathne, 2013).

A Java function called "convertSeqToMapFile" is implemented in org.greenwich.mapFile.MapFileFixer class to create a MapFile from the SequenceFile that contains the inverted index entries.

The size of the SequenceFile (inverted file) that holds the all terms and their postings lists is 14.6 Mb, compared to the index file is 210.2 Kb. A portion of the index file is displayed in Figure 17.

```

agin      1942678
aginst    1942883
agm       1943041
agnt      1943237
ago       1943418
agoco     1944434
agre      1944672
agreeagre 1950370
agreedear 1950554
agreedwait 1950750
agreement 1950912
agrement  1952081
agrr      1952297
agt       1952460
agt_app09 1953073
agt_mas09 1953258
agt_sec09 1953440
agtdump.zip 1953624
agtfile.rar 1953789
agtfixed2 1953953
ah        1954117
ahade     1954321
aham      1954530
ahamad    1954744

```

Figure 17. Index file contains a dictionary of all terms in inverted index

JobDriver and Job Execution

Figure 18 shows the code of the main method of “InvertedIndexDriver.class” class that will invoke the job. It configures mapper, reducer, input and output formats, and input and output directories.

```

29 public class InvertedIndexDriver extends Configured implements Tool{
30
31
32     public String InvertedIndexInFolder = "hdfs://localhost/user/admin/input";
33     public static String InvertedIndexOutFolder = "hdfs://localhost/user/admin/InvertedIndexOutFolder";
34
35     public int run(String [] args) throws IOException, InterruptedException , Exception{
36         Configuration conf =getConf();
37
38         Job job =new Job(conf,"InvertedIndexJob");
39         job.setJarByClass(InvertedIndexDriver.class);
40         job.setInputFormatClass(TicketFileInputFormat.class);
41         job.setOutputFormatClass(SequenceFileOutputFormat.class);
42         job.setMapperClass(GenerateTermsMapper.class);
43         job.setReducerClass(GeneratePositingListsReducer.class);
44         job.setMapOutputKeyClass(Text.class);
45         job.setMapOutputValueClass(TermWritable.class);
46         job.setOutputKeyClass(Text.class);
47         job.setOutputValueClass(Text.class);
48         FileInputFormat.addInputPath(job, new Path(InvertedIndexInFolder));
49         FileOutputFormat.setOutputPath(job, new Path(InvertedIndexOutFolder));
50         return job.waitForCompletion(true)? 0:1;
51     }
52
53     public static void main(String [] args) throws IOException, InterruptedException , Exception {
54         int exitCode = ToolRunner.run(new InvertedIndexDriver(), args);
55
56         MapFileFixer mapFileFixer = new MapFileFixer();
57         mapFileFixer.convertSeqToMapFile(new Configuration(), InvertedIndexOutFolder );
58         System.exit(exitCode);
59
60     }

```

Figure 18. The code of main method of InvertedIndexDriver

The full code of the classes implemented in this sprint are provided in Appendix B.

3.3.4 Sprint 2 Backlog

To complete sprint 2, the work took 42 days and after this sprint, the product backlog was updated, as shown in Table 3.

Feature No.	Feature Name	Priority	Sprint	Status	Note
1	Fetching the text from helpdesk tickets	1	1	Completed	Extract the text from all HD tickets and store them on HDFS.
2	Analysing and indexing text	2	2	Completed	Perform text analysis on the text contents of the tickets and construct the inverted index.
3	Building TF-IDF matrix	3	3	In-progress	Generate TF-IDF matrix by using the inverted index data.
4	Developing the search API	4	4	Planned	Develop a Java API to retrieve and score the relevant tickets.

Table 3. Product backlog after sprint 2

3.4 Sprint 3: Building the TF-IDF Matrix

This sprint includes building a distributed TD-IDF matrix based on the results of the inverted file. The purpose of building this matrix is to speed up the calculation of the scores of the relevant tickets during the retrieval phase.

3.4.1 Gathering Requirements and Analysis

Weighting Scheme

The vector space model can be used to represent text data as a set of vectors (Perera and Gunarathne, 2013). For the case-study, a vector space model can be built by taking the set of all terms that appear in the helpdesk dataset. The number of terms in the term set is the dimensionality of the resulting vectors and each dimension of the vector corresponds to a term. For each ticket, the vector contains the weight of the importance of each term in a ticket based on the frequency of the term's occurrence in the ticket.

Since not all the words in a ticket are equally important, building vectors using the former term count model gives much more weight to the terms that occur frequently across many tickets (Perera and Gunarathne, 2013). However, frequent terms are less informative than

rare terms but a ticket containing such a term is more likely to be relevant than a ticket that does not, though it is not a sure indicator of relevance. This means that high positive weights are required for frequent terms but are still lower than the weights of rare terms (Raghavan and Nayak, 2014).

The TF-IDF model can be used to translate these requirements, as it utilises the inverted document frequencies (IDF) to scale the term frequencies (TF). IDF is typically calculated by counting the number of documents (DF) the term appears in, inverting it (1/DF), and normalising it by multiplying it with the number of documents and using the logarithm of the resultant value, as shown roughly by the following equation (Perera and Gunarathne, 2013):

$$(3.4.1) \quad \text{TF-IDF} = \log(\text{TF}) \times \log(\text{N/DF})$$

The TF-IDF weight of a term is influenced by the following factors:

1. The weight increases when the term occurs many times within a small number of tickets.
2. The weight decreases when the term appears fewer times in a ticket, or appears in many tickets.
3. Zero or very low weight is given when the term occurs in almost all tickets.

Another variation that needs to be considered when calculating the term's weight is the term's location within a ticket. Terms appearing in the ticket summary can be more important than one in the ticket's comments. This can be translated into a bonus to be given to the term's weight based on its location. Thus, terms that occur in the ticket summary are given a bonus of 3, the ones in the ticket description are given 2 and those in the comments get 1. The final form of the TF-IDF equation:

$$(3.4.2) \quad \text{TF-IDF} = \log(\text{TF} + \text{TL}) \times \log(\text{N/DF})$$

N is the total number of tickets in the collection. TL is the sum of the term's bonuses in a ticket. For example: if the term "memory" appears once in all ticket sections (summary, description and comments), then its $\text{TL} = 3 + 2 + 1 = 6$.

Building the TF-IDF Matrix from the Inverted File

To calculate the weight of a term, its three elements (TF, TL and DF) have to be available. These data have already been gathered for every term in the collection during the execution of the preceding phase and are available in the inverted file. The matrix can be built within two MapReduce jobs as follows:

1. TF-IDF Job 1: to dismantle the postings lists in the inverted file and produce key = ticket ID and value = list of all the ticket's terms where each of them are associated with TF, TL and DF. The expected output from this job would be in the following form:

Key	Value
AE00AGT-2002	[{"DocFreq":867,"colFreq":1522,"term":"on","termLocationCount":1,"termFreq":1},{ "DocFreq":793,"colFreq":3344,"term":"test","termLocationCount":2,"termFreq":2},...]

- a. TF-IDF Job 2: to calculate the weights for each term according to the equation 3.4.2. The expected output from this job would be in the following form:

Key	Value
AE00AGT-2023	[[{"term": "cbuae", "termWeight": 23.03647191567928}, {"term": "desk", "termWeight": 7.633940075514599}, {"term": "kindli", "termWeight": 3.1126050015345745}, ...]

The output of Job 2 is a TF-IDF matrix stored in a Hadoop SequenceFile file-based structure which has a list of all ticket IDs (keys) mapped to their real-value vectors of TF-IDF weights as shown in the previous example. A Map-File is then generated as an index which contains a list or dictionary of all ticket IDs mapped to the byte offset of their vectors in the TF-IDF matrix file. The purpose of generating this file is to enhance the process that involves random access to the lines in the TF-IDF matrix.

3.4.2 Designing Sprint 3's Process

The computation process of the two MapReduce jobs are executed sequentially, therefore they are represented in the use-case diagram as one use-case called “Building TF-IDF matrix”, as shown in Figure 19. Since the process of building a TF-IDF matrix depends on the outputs of the execution of the “Text analysis and indexing job”, the former is considered as an “include” to the latter.

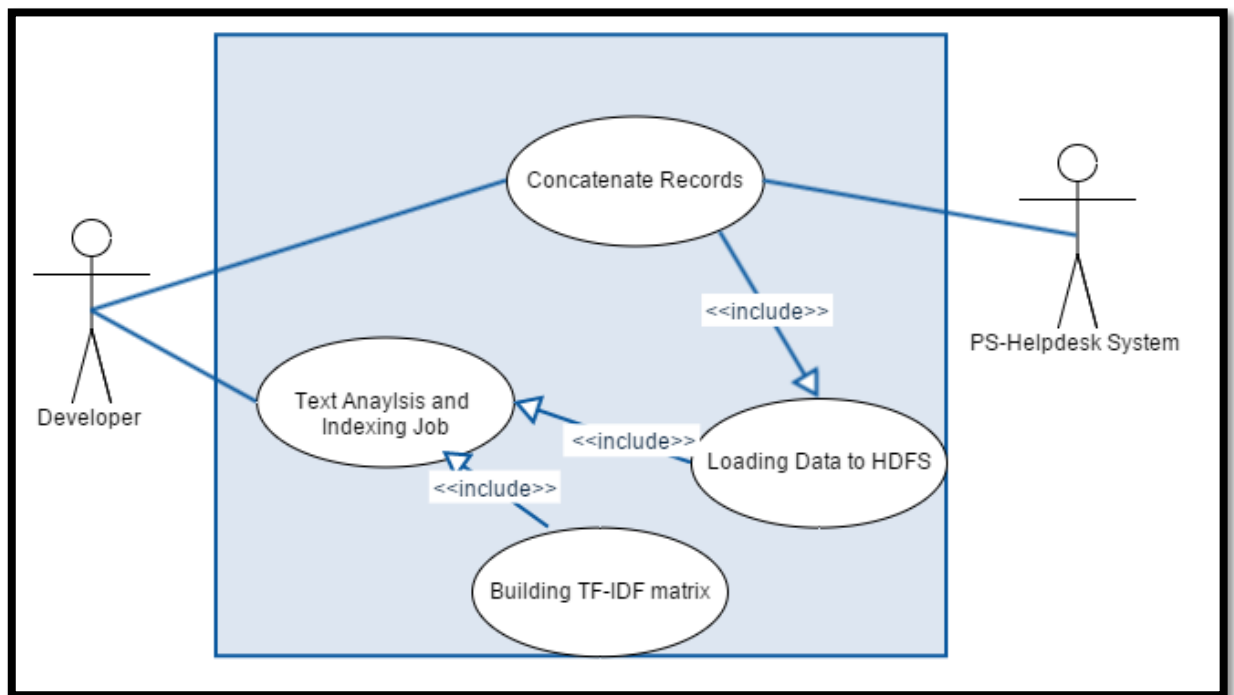


Figure 19. Use-case diagram including “Building TF-IDF matrix”

Figure 20 illustrates the workflow of the execution of the two MapReduce jobs.

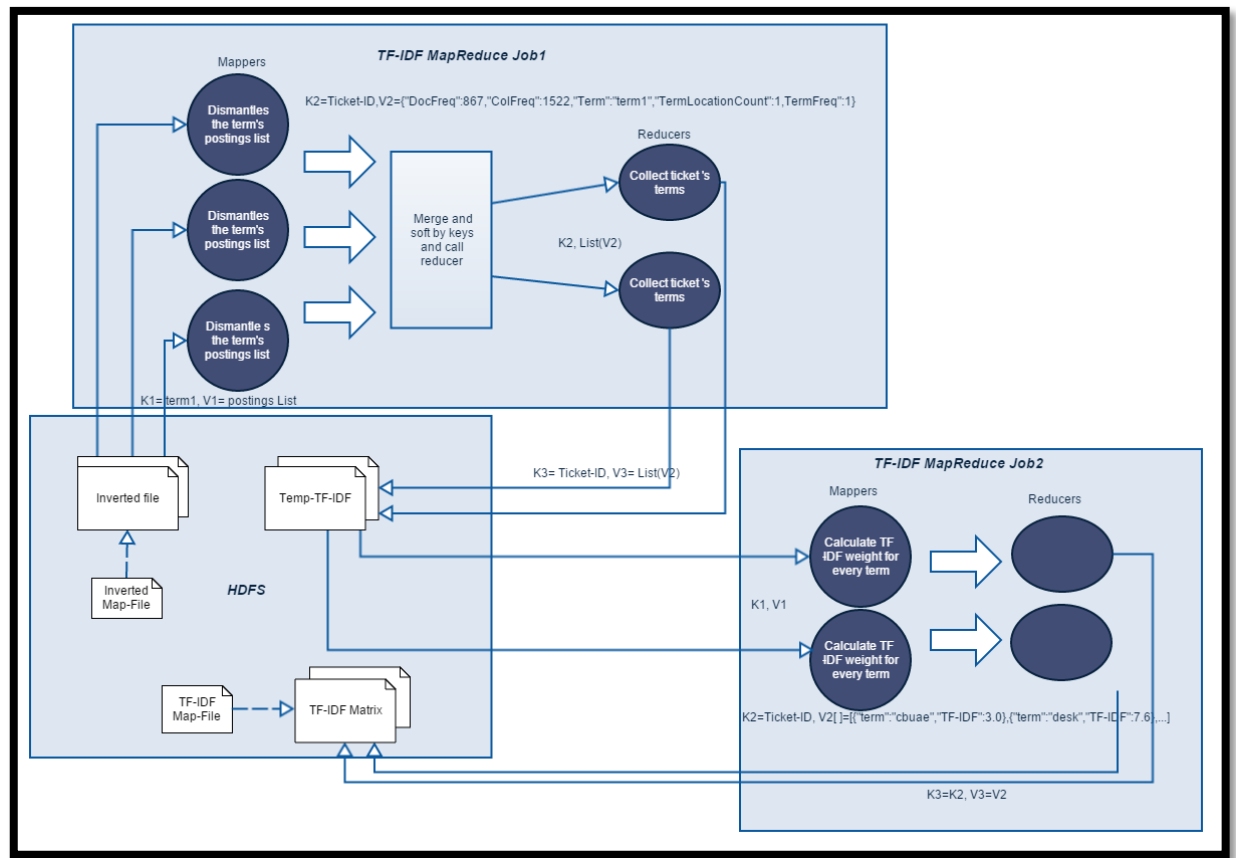


Figure: 20. Generating TF-IDF matrix

3.4.3 Implementation of Sprint 3's Process

The two TF-IDF MapReduce jobs are implemented within the Java package “org.greenwich.tfidf” and are explained as follows:

1. TF-IDF MapReduce Job 1:

The logic of this job is explained as follows:

- Hadoop's SequenceFileInputFormat is used to parse the inverted file and generate from each line <key1, value1> record, where key1 is a term and value1 is the postings list of the term.

```
Key1      Value1
backup    {"DocFreq":1,"PostingLists":[{"docID":"JO00SCB-585","termFreq":1,"TermPositions":[{"position":"1",
"termLocation":"2","endOffset":"17","termType":"<ALPHANUM>","startOffset":"1"}]}]}
```

- The Mapper function is implemented in class “tfidfMapper.class”. It receives <key1, value1> and dismantles the value by decoding it in JSONArray. It loops through the elements of this array and emits every ticket ID as an output key with (TF, DF and TL) for the term in this ticket ID as an output value.

Key2	Value2
AE00AGT-2002	{"DocFreq":867,"colFreq":1522,"term":"on","termLocationCount":1,"termFreq":1}

- The Reducer is implemented in class “tfidfReducer.class”. It receives <key2, List (value2)> and iterates through the elements of the list to compose them in a JSONArray. It outputs key3 = key2 and value3 = JSONArray, as follows:

Key3	Value3
AE00AGT-2002	[{"DocFreq":867,"colFreq":1522,"term":"on","termLocationCount":1,"termFreq":1},{
	"DocFreq":793,"colFreq":3344,"term":"test","termLocationCount":2,"termFreq":2},...]

2. TF-IDF MapReduce Job 2:

The logic of this job is explained as follows:

- The output file of the preceding job is parsed by Hadoop’s KeyValueTextInputFormat to generate from each line <key1, value1> record equals to <key3, value3>.
- The Mapper function is implemented in class “tfidfMapper2.class”. It receives <key1, value1> and decodes value1 to JSONArray to iterate through its elements and calculates the weight for each term. It opens another JSONArray to store the terms and their weights. It then outputs <key2,value2> as follows:

Key	Value
AE00AGT-2023	[{"term":"cbuae","termWeight":23.03647191567928},{ "term":"desk","termWeight":7.63394007551 4599},{ "term":"kindli","termWeight":3.1126050015345745},...]

- The Reducer function used here is the identity Reducer which only takes the input and passes it as output without doing any logic.

The Map-File of the TF-IDF matrix is produced using the same Java function “convertSeqToMapFile” used to generate the Map-File of the inverted index. This function is implemented in org.greenwich.mapFile.MapFileFixer class. These MapReduce jobs are executed by the same main method of the “InvertedIndexDriver.class”.

The full code of the classes implemented in this sprint are provided in Appendix B.

3.4.4 Sprint 3 Backlog

To complete sprint 3 the work took 25 days and after this sprint, the product backlog was updated as shown in Table 4.

Feature No.	Feature Name	Priority	Sprint	Status	Note
1	Fetching the text from helpdesk tickets	1	1	Completed	Extract the text from all HD tickets and store them on HDFS.
2	Analysing and indexing text	2	2	Completed	Perform text analysis on the text contents of the tickets and construct the inverted index.
3	Building TF-IDF matrix	3	3	Completed	Generate TF-IDF matrix by using the inverted index data.
4	Developing the search API	4	4	In-progress	Develop a Java API to retrieve and score the relevant tickets.

Table 4. Product backlog after sprint 3

3.5 Sprint 4: Building the Retrieval API

3.5.1 Gathering Requirements and Analysis

Vector Space Model for scoring the tickets

For the case study, the subject or the summary of the new ticket represents the free-text query for which we need to find the rank-order tickets that match it. There are three requirements:

1. Find the relevant tickets to a query.
2. Compute a score that represents the similarity of each relevant ticket to this query.
3. Rank the tickets based on their scores from the highest to lowest.

In the first requirement, “relevant” means any ticket that contains at least one term of the query’s terms. Since all the tickets have been represented within the vector space model as real-valued vectors of TF-IDF weights, Manning et al (2008) suggest that if the query is also represented as a real-valued vector of TF-IDF weights of its terms, then the similarity between the query and a ticket can be computed by using the equation of the cosine similarity (3.5.1).

$$(3.5.1) \quad score(q, t) = \cos(\vec{q}, \vec{t}) = \frac{\vec{q} \bullet \vec{t}}{\|\vec{q}\| \|\vec{t}\|} = \frac{\vec{q}}{\|\vec{q}\|} \bullet \frac{\vec{t}}{\|\vec{t}\|} = \frac{\sum_{i=1}^{|V|} q_i t_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} t_i^2}}$$

According to Manning et al (2008), using cosine similarity also has the advantage of compensating the effect of ticket length as when two tickets with very similar content can have a significant vector difference simply because one is much longer than the other.

The numerator of the equation (3.5.1) represents the dot product of the query vector \vec{q} and ticket vector \vec{t} , while the denominator is the product of their Euclidean lengths. The Euclidean length of a vector x can be computed as follows (Manning et al, 2008):

$$(3.5.2) \quad \|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$$

Raghavan and Nayak (2014) justify that dividing a vector by its Euclidean length makes it a unit (length) vector and thus the query and ticket can have comparable weights.

By computing the cosine similarity between the query and each relevant ticket, the resulting scores can be used to select the top-scoring tickets for a query. To do so, an algorithm is given in Figure 21 (Manning et al, 2008):

```

COSINESCORE( $q$ )
1  float  $Scores[N] = 0$ 
2  Initialize  $Length[N]$ 
3  for each query term  $t$ 
4  do calculate  $w_{t,q}$  and fetch postings list for  $t$ 
5    for each pair( $d, tf_{t,d}$ ) in postings list
6    do  $Scores[d] += w_{t,d} \times w_{t,q}$ 
7  Read the array  $Length[d]$ 
8  for each  $d$ 
9  do  $Scores[d] = Scores[d] / Length[d]$ 
10 return Top  $K$  components of  $Scores[]$ 

```

Figure 21. The algorithm for computing vector space scores.

Pre-calculate Euclidean lengths

Computing the cosine similarity between a query vector and a number of vectors of the relevant tickets can be an expensive process. This is because the process might entail thousands of arithmetic operations, especially when the query has more than 10 terms. Therefore to improve the performance, it is required to develop a MapReduce job to calculate in advance the Euclidean lengths for all tickets in the TF-IDF matrix.

3.5.2 Designing Sprint 4's Process

The use-case diagram at this step has introduced a new actor, the “helpdesk user”, who will be presented with a list of the top relevant-ranked tickets upon executing the “Search for the relevant tickets” use-case which has three use-cases: “Analysing the query text”, “Retrieving the relevant tickets” and “Calculating the relevant tickets Scores” (see Figure 22). Table 5 provides a description for each use-case of the retrieval phase.

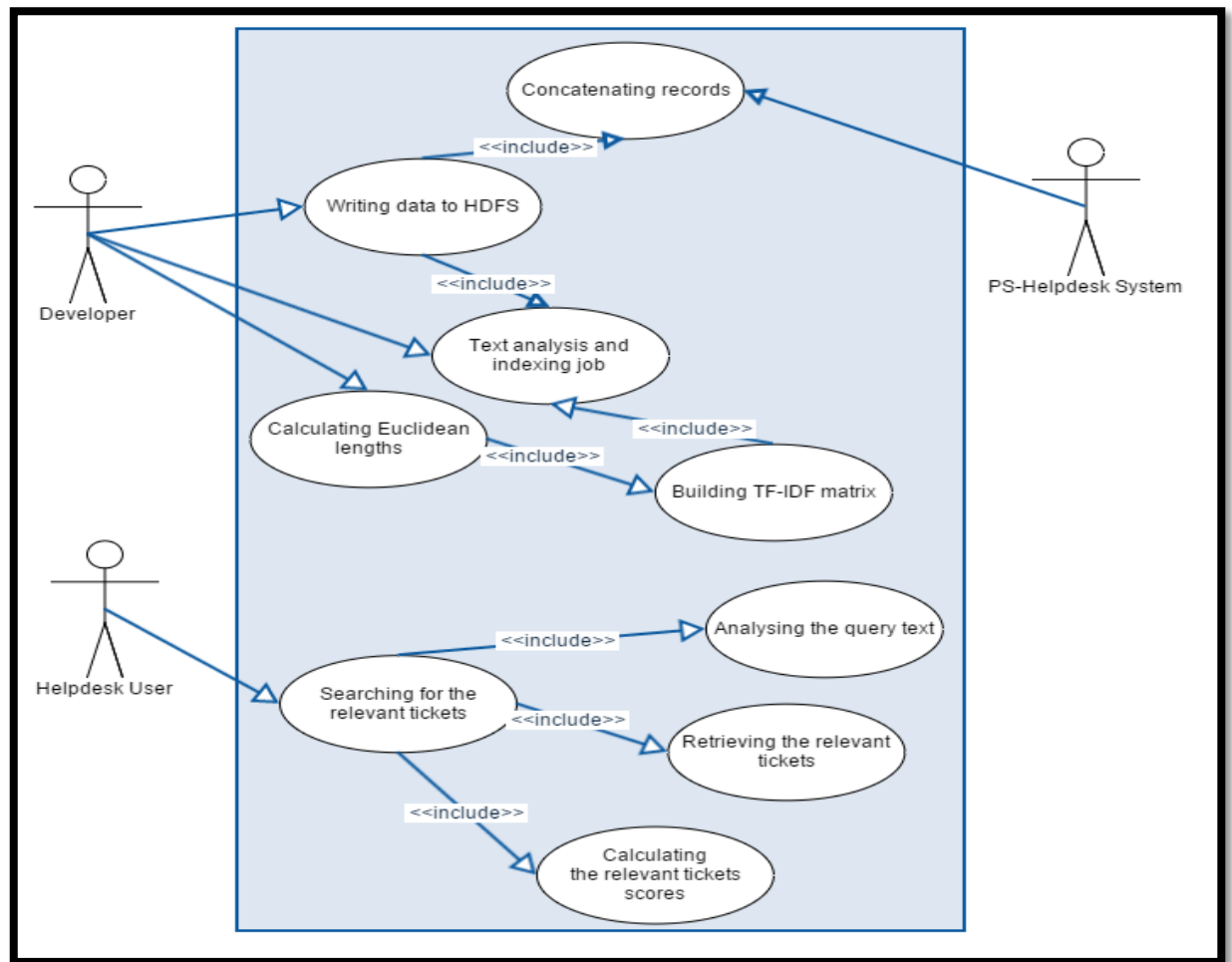


Figure 22. Use-case diagram for retrieving the relevant tickets

Use-case	Description
Searching for the relevant tickets	This use-case gives the same results whether it is being invoked by the PS helpdesk system to retrieve the relevant tickets when a new ticket is opened, or invoked by the helpdesk user to find the relevant tickets for a query. Upon executing this use-case the other three will be invoked.
Analysing the query text	This use-case involves performing text analysis on the query text.
Retrieving the relevant tickets	This use-case involves retrieving the ticket IDs that contain at least one query term.
Calculating the relevant ticket scores	This use-case involves calculating the scores of each relevant ticket and returning a list of ordered ticket IDs based on their score value.

Table 5. Description of retrieval phase use-cases

The process flow diagram for the retrieval phase is illustrated in Figure 23.

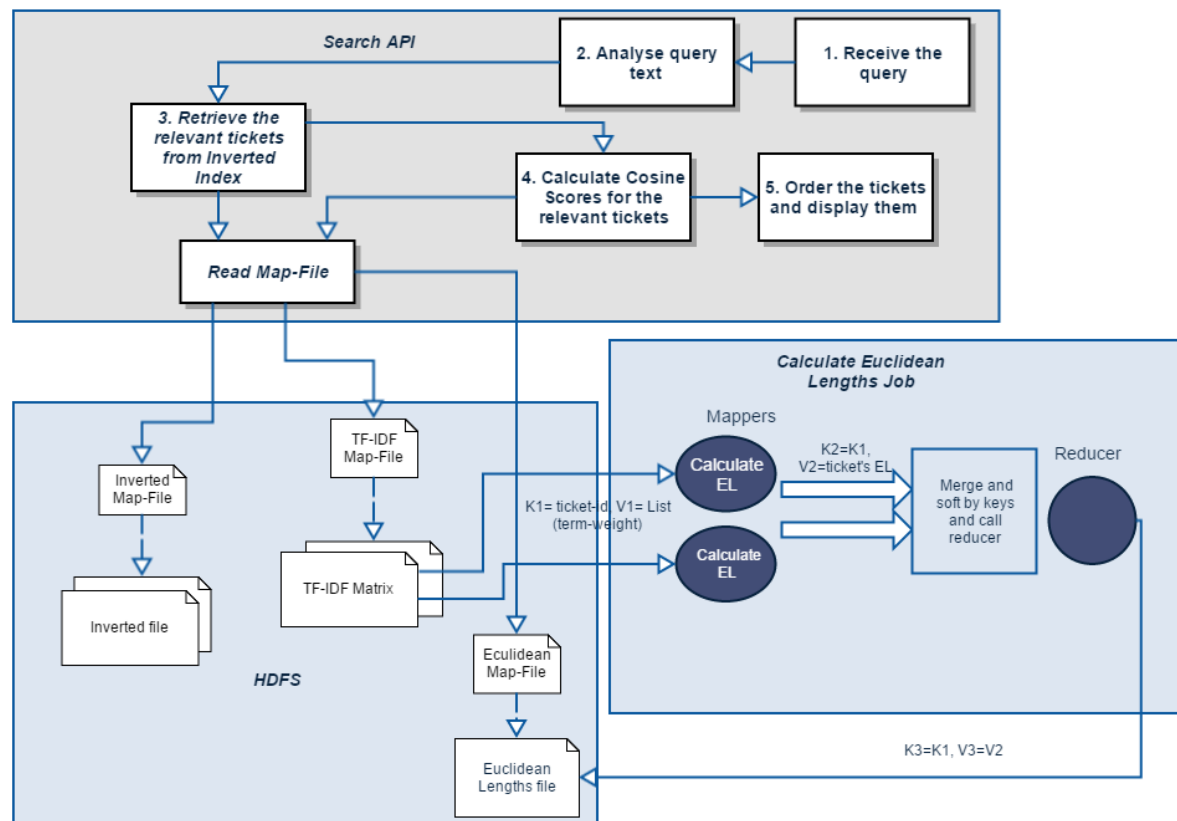


Figure 23. The retrieval and scoring process and the Euclidean lengths MapReduce job

3.5.3 Implementation of Sprint 4's Process

Two components are implemented within this phase: a MapReduce job that calculates the Euclidean lengths for all the tickets in the collection; and the retrieval application.

Euclidean Length MapReduce Job

The Java classes for this job are implemented in the package “org.greenwich.tfidf”. They involve the following logic:

1. Hadoop's SequenceFileInputFormat is used to read TF-IDF file line by line.
2. Each line is passed to the mapper function in the form <Key Text, Value Text>.
 - Key: ticket ID
 - Value: contains a list of all its terms along with their weights as shown in the following example

Key Value

AE00AGT-2104 [{"term": "dear", "termWeight": 1.204}, {"term": "believe", "termWeight": 4.386}]

3. The mapper function calculates the Euclidean lengths for each ticket using the formula (3.5.2).
4. The reducer used here is the identity reducer which does not do any logic; it only takes the input and passes it as output.
5. The results are stored in the HDFS file as a key/value pair using Hadoop “SequenceFileOutputFormat” and a Map-File is generated as an index.

Key	Value
AE00AGT-2071	88.969
AE00AGT-2079	158.91
AE00AGT-2095	63.374
AE00AGT-2097	48.727
AE00AGT-2098	22.096
AE00AGT-2099	58.992

Retrieval application

The retrieval component is also coded using Java and its classes are implemented within the package “org.greenwich.searchEngine”.

The full code of the classes implemented in this sprint are provided in Appendix B.

3.5.4 Sprint 4 Backlog

To complete sprint 4 the work took 27 days and after this sprint, the product backlog was updated as shown in table 5.

Feature No.	Feature Name	Priority	Sprint	Status	Note
1	Fetching the text from helpdesk tickets	1	1	Completed	Extract the text from all HD tickets and store them on HDFS.
2	Analysing and indexing text	2	2	Completed	Perform text analysis on the text contents of the tickets and construct the inverted index.
3	Building TF-IDF matrix	3	3	Completed	Generate TF-IDF matrix by using the inverted index data.
4	Developing the search API	4	4	Completed	Develop a Java API to retrieve and score the relevant tickets.

Table 6. Product backlog after sprint 4

4. Results and Evaluation

In this chapter, the results of a number of executed queries are presented with an assessment of each retrieval experiment. The chapter begins with a brief discussion about what is required to measure the effectiveness of retrieving the relevant tickets. It then explains the relevant measurements required to evaluate ranked retrieval results. Then an analysis of the data set used for testing the implementation is presented before showing the outcomes of the experiments and evaluating their results. Researching time is evaluated, followed by an evaluation of indexing performance. Finally, a discussion of the factors of user happiness (speed of response, size of index and the most important factor which is the relevance of results) is included.

4.1 Evaluation of Retrieval Results

According to Manning et al (2008), in order to gauge the effectiveness of retrieving the relevant tickets, three elements are required:

1. A document collection.
2. A test suite of information needs, expressible as queries.
3. An assessment of the relevance of each document-query pair.

The document collection used for the testing is a data set taken from PS's helpdesk database which contains a sufficient and reasonable number of tickets. Analysis of the contents of this data set is provided in the following subsection. While the success of information retrieval systems is frequently measured using standard test collections (such as Cranfield, TREC and GOV2) (Schütze and Lioma, 2010), these are currently not applicable for examining the proposed solution since it has been developed to deal with a specific kind of document (i.e. ProgressSoft's helpdesk ticket). However, a feasibility study found that to test the application with standard collections required an index-adaptor to be developed along with core-code modifications in the index and retrieval classes.

For the second requirement, five information needs (queries) have been prepared to perform the test. To measure the effectiveness of the results for each query, the number of relevant tickets in the collection for these queries was identified in advance.

A binary classification (relevant or non-relevant) was chosen as a scale of relevance for the ticket. Relevance of a ticket is assessed relative to an information need, not a query (Manning et al, 2008). For example, the query could be "PS-ECC application runs out of memory", but the information need would be "finding the root cause for out of memory issues". This means that a ticket is considered to be relevant, not because it contains some of the query terms but rather because it addresses the information need.

Relevance Measurements

The evaluation of information retrieval results is usually measured by calculating precision and recall. Precision is the fraction of the returned results that are relevant to the information need, and recall is the fraction of the relevant documents in the collection that were returned

by the system. These two fractions are calculated using the following formulae (Manning et al, 2008):

$$\text{Precision} = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} = P(\text{relevant}|\text{retrieved})$$

Figure 24. Precision formula (Manning et al, 2008)

$$\text{Recall} = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} = P(\text{retrieved}|\text{relevant})$$

Figure 25. Recall formula (Manning et al, 2008)

	Relevant	Nonrelevant
Retrieved	true positives (tp)	false positives (fp)
Not retrieved	false negatives (fn)	true negatives (tn)

Then:

$$P = tp / (tp + fp)$$

$$R = tp / (tp + fn)$$

Figure 26. Precision and recall (Manning et al, 2008)

However, to evaluate ranked retrieval results these quantities need to be extended. In a ranked retrieval context, appropriate sets of retrieved documents are given by the top k retrieved documents (Manning et al, 2008). The most common measurements used with standard test collections are the 11-point Interpolated Average Precision (IAP) and the Mean Average Precession (MAP) (Manning et al, 2008). These two measurements are calculated after presenting the results of each information need.

Analysing the Data Set

A data set from PS's helpdesk system contains 19,433 tickets (200 MB). It was made sure that the size of data set is considerable and varied enough to be delegate of system efficiency across different queries. The tickets in the system are classified according to the following hierarchy:

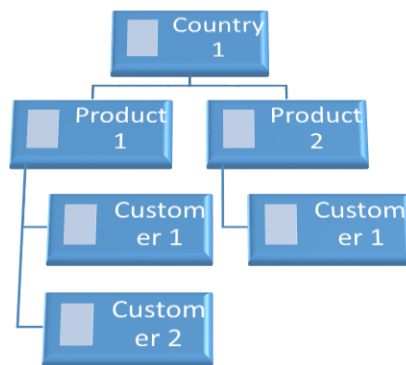


Figure 27. PS-Helpdesk tickets' hierarchy

This classification hierarchy was not considered during indexing or retrieving. This means that searching for relevant tickets was conducted across all the tickets in the system regardless of the country, product and customer. According to the analysis:

1. There are nuances between the products deployed in different countries. Many issues reported for different versions were found to have the same solution.
2. Obviously, different products have different architectures and may use different technology, however, there is still a small probability to find a ticket with the same issue reported for a different product. For example, “out of memory” was reported in a ticket for a product deployed on Oracle WebLogic; the new ticket has the same topic but for a different product also deployed on Oracle WebLogic.

The majority of the records in the data set are in English but some tickets are written in Arabic. It was estimated that the number of these tickets is less than 1% of the total. Since the text analyser used to normalise the text can only identify English words, the Arabic tickets are ignored.

Preparing Information Needs and Identifying Relevant Tickets

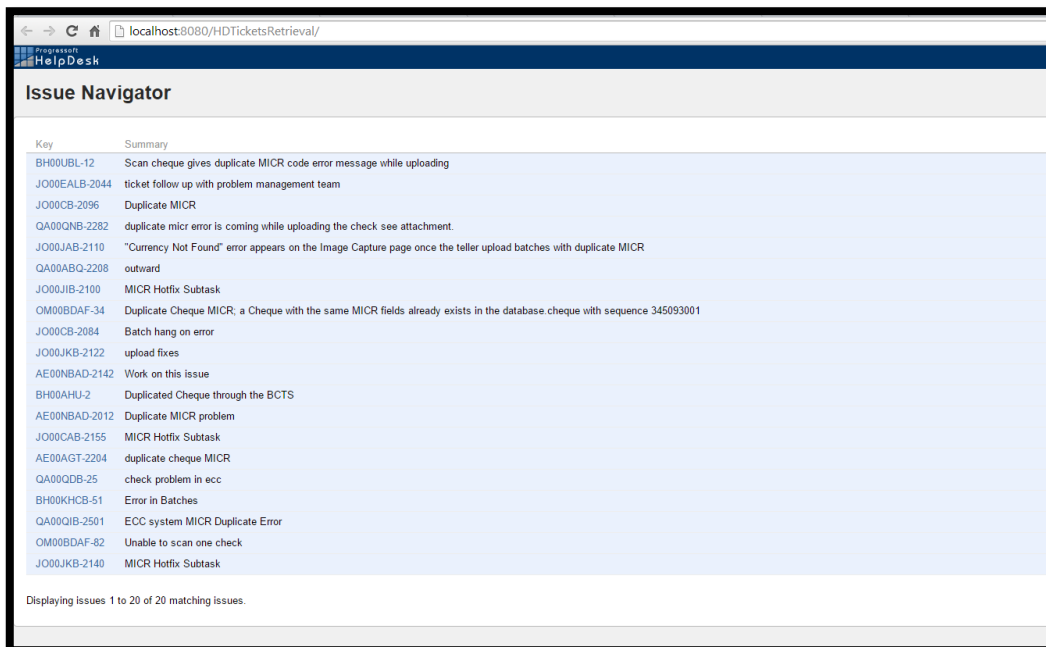
To calculate the precision and recall for each information need, we must identify in advance the set of relevant tickets in the collection for each information need. Table 7 lists five randomly selected information needs that will be used to evaluate the efficiency of the retrieving function. The relevant tickets in the data set were identified manually by going through all the records found from a number of SQL statements that use “like” with “where” clauses. The judgement of the relevance of each ticket was based on the author’s previous knowledge of working as technical support on ProgressSoft’s helpdesk systems for six years. The author’s judgment complied with the feedback received from ProgressSoft’s helpdesk manager.

Query no.	Query	Information need	No. relevant tickets	Ticket-ids of relevant tickets
1	duplicate MICR error	Reasons for duplicate MICR error that occur in scan page of PS-ECC system	15	BH00UBL-12,JO00CB-2096,QA00QNB-2282
2	timeout cheques	Reasons for the cheques being a timeout	26	QA00HSBC-2077,QA00DB-2023,QA00QNB-2085
3	file not found in EDMS	Finding the cause of file not found error in PS-EDM system	13	QA01QNB-2039,QA01QNB-2129,JO01CAB-2009
4	Unreachable destination cheque	Reasons for the cheque being unreachable	61	QA00AB-2198,QA00QNB-2155,QA00KHAL-2123
5	ldap authentication issue in PS-ECC	Reason for failing to authenticate PS-ECC system with ldap	18	JO00CAB-2289,AE00AGT-2255,JO00ARAB-2252,JO00CAB-2381

Table 7. Test queries and their information needs

1. Query 1 “duplicate MICR error”

The aim of this query is to obtain the tickets that mention in their body the root causes of this error. The following figure shows that 20 tickets are returned in response to this query.



Key	Summary
BH00UBL-12	Scan cheque gives duplicate MICR code error message while uploading
JO00EALB-2044	ticket follow up with problem management team
JO00CB-2096	Duplicate MICR
QA00QNB-2282	duplicate micr error is coming while uploading the check see attachment.
JO00JAB-2110	"Currency Not Found" error appears on the Image Capture page once the teller upload batches with duplicate MICR
QA00ABQ-2208	outward
JO00JIB-2100	MICR Hotfix Subtask
OM00BDAF-34	Duplicate Cheque MICR; a Cheque with the same MICR fields already exists in the database cheque with sequence 345093001
JO00CB-2084	Batch hang on error
JO00JKB-2122	upload fixes
AE00NBAD-2142	Work on this issue
BH00AHU-2	Duplicated Cheque through the BCTS
AE00NBAD-2012	Duplicate MICR problem
JO00CAB-2155	MICR Hotfix Subtask
AE00AGT-2204	duplicate cheque MICR
QA00QDB-25	check problem in ecc
BH00KHC-51	Error in Batches
QA00QIB-2501	ECC system MICR Duplicate Error
OM00BDAF-82	Unable to scan one check
JO00JKB-2140	MICR Hotfix Subtask

Displaying Issues 1 to 20 of 20 matching issues.

Figure 28. The results of query “Duplicate MICR error”

The precision and recall are computed for the top k ranked hits ($k = 1, 2, 3, 4, \dots, 20$) (see Table 8) and then plotted to obtain a precision-recall curve, as shown in Figure 29.

Key	No. relevant tickets	Precision	Recall
1	1	1	0.07
2	1	0.5	0.07
3	2	0.67	0.13
4	3	0.75	0.2
5	4	0.8	0.27
6	5	0.83	0.33
7	5	0.71	0.33
8	6	0.75	0.4
9	7	0.77	0.47
10	7	0.7	0.47
11	7	0.64	0.47
12	8	0.67	0.53
13	9	0.69	0.6
14	9	0.64	0.6
15	10	0.67	0.67
16	11	0.68	0.73
17	12	0.71	0.8
18	13	0.72	0.87
19	14	0.73	0.93
20	14	0.7	0.93
Avg.		0.72	0.49

Table 8. Precision-Recall values of query 1

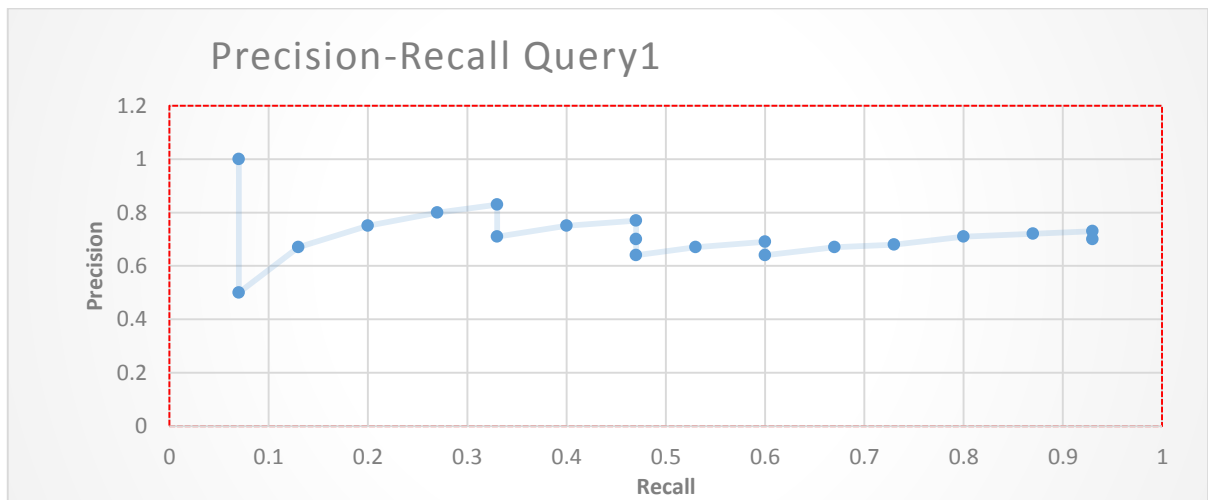


Figure 29. Precision-Recall curve query 1

It is noticed from the table 8 that the recall at k set of tickets is equal to the recall at $k + 1$ if the latter does not retrieve a new relevant ticket, but precision has been reduced. However, if it is relevant, then both precision and recall grow, and the curve climbs up and to the right.

Taking the statistics obtained with ($k=10$), precision (70%) is higher than recall (47%). This is in line with the circumstances of the helpdesk user who wants most of results on the first page to be relevant rather than having to look at every relevant ticket in the collection.

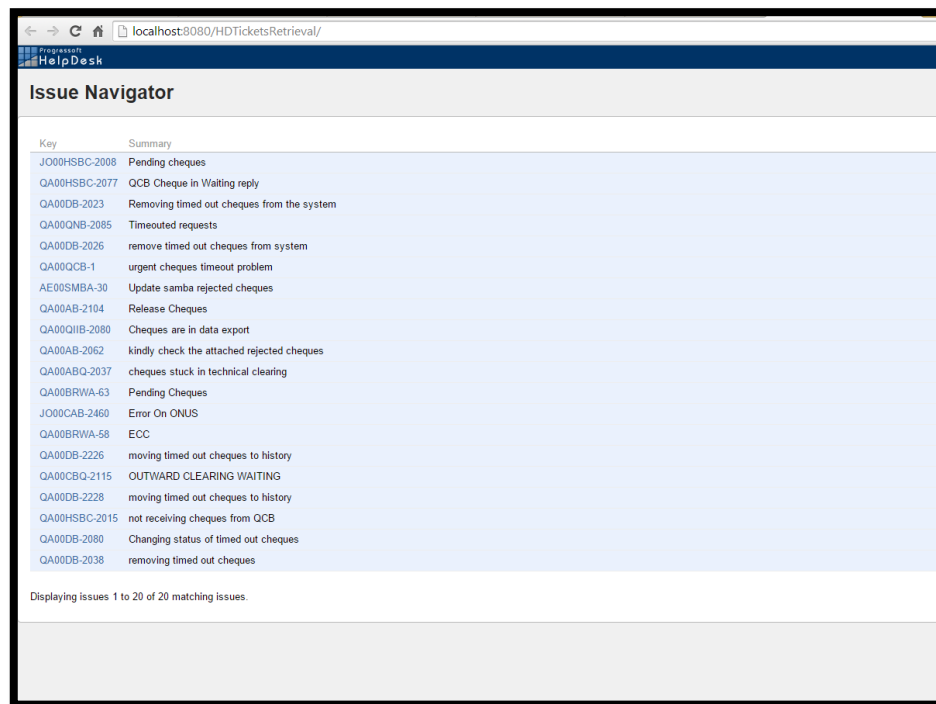
This leads into finding the interpolated precision which is the precision at a certain recall level r , and is defined as the highest precision found for any recall level $r' \geq r$. The rationale for interpolation precision is that the user is willing to look at more results if both precision and recall are stronger (Schütze and Lioma, 2010). According to Figure 29, the interpolated precision is measured at the 11-recall level as shown in the following table.

Recall	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Pinterp	1	0.83	0.83	0.83	0.77	0.73	0.73	0.73	0.73	0.73	0.3

Table 9. Interpolated precision and 11-recall levels

2. Query 2 “timeout cheques”

The aim of this query is to obtain the tickets that discuss in their bodies the issue of the timeout cheques. The following figure shows that 10 tickets are returned as a response to this query.



Key	Summary
JO00HSBC-2008	Pending cheques
QA00HSBC-2077	QCB Cheque in Waiting reply
QA00DB-2023	Removing timed out cheques from the system
QA00QNB-2085	Timeouted requests
QA00DB-2026	remove timed out cheques from system
QA00QCB-1	urgent cheques timeout problem
AE00SMB-30	Update samba rejected cheques
QA00AB-2104	Release Cheques
QA00QIB-2080	Cheques are in data export
QA00AB-2062	kindly check the attached rejected cheques
QA00ABQ-2037	cheques stuck in technical clearing
QA00BRWA-63	Pending Cheques
JO00CAB-2460	Error On ONUS
QA00BRWA-58	ECC
QA00DB-2226	moving timed out cheques to history
QA00CBQ-2115	OUTWARD CLEARING WAITING
QA00DB-2228	moving timed out cheques to history
QA00HSBC-2015	not receiving cheques from QCB
QA00DB-2080	Changing status of timed out cheques
QA00DB-2038	removing timed out cheques

Displaying issues 1 to 20 of 20 matching issues.

Figure 30. The results of query “timeout cheques”

The precision and recall are computed for the top k ranked hits ($k = 1, 2, 3, 4, \dots, 20$) (see Table 9) and then plotted to obtain a precision-recall curve, as shown in Figure 31.

Key	No. relevant tickets	Precision	Recall
1	0	0.00	0.00
2	1	0.50	0.04
3	2	0.67	0.08
4	3	0.75	0.12
5	4	0.80	0.15
6	5	0.83	0.19
7	5	0.71	0.19
8	6	0.75	0.23
9	6	0.67	0.23
10	6	0.60	0.23
11	6	0.55	0.23
12	6	0.50	0.23
13	6	0.46	0.23
14	6	0.43	0.23
15	7	0.47	0.27
16	7	0.44	0.27
17	8	0.47	0.31
18	8	0.44	0.31
19	9	0.47	0.35
20	10	0.50	0.38
Avg.		0.55	0.21

Table 9. Precision-Recall values query 2

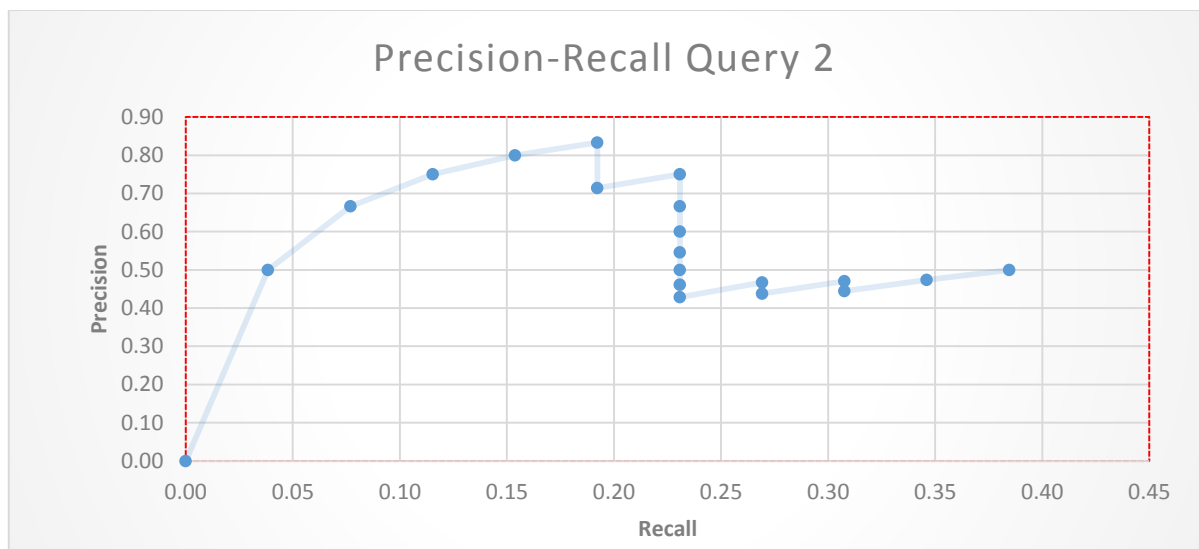


Figure 31. Precision-Recall curve query 2

The second test gives a very low recall compared to the first. However a reasonable precision (60%) value is obtained with k=10. These six relevant tickets carry enough information to answer the information need of this query.

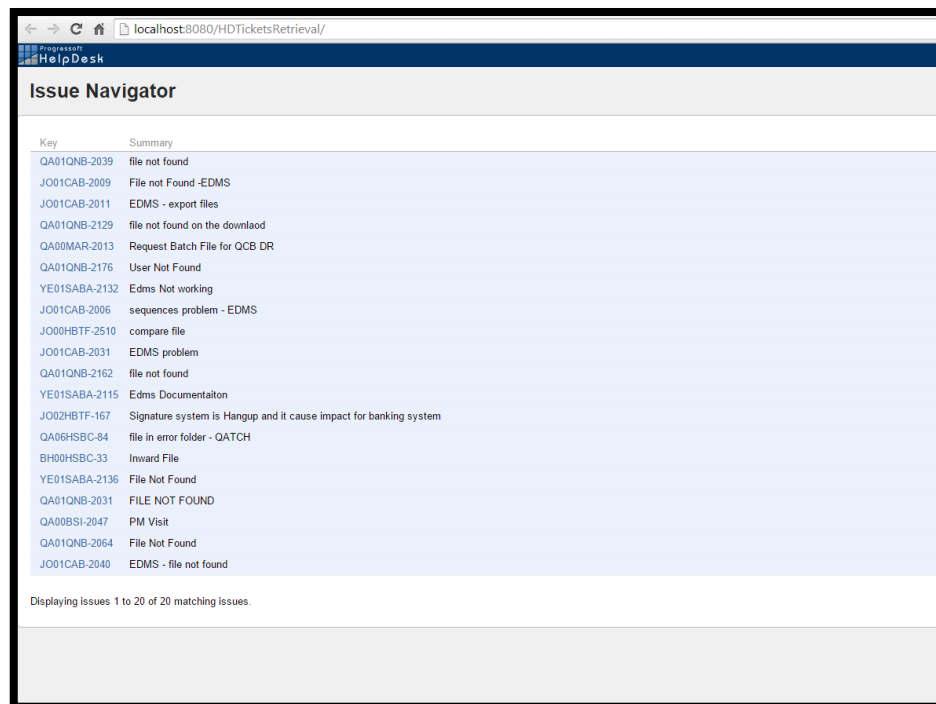
According to the statistics in Figure 31, the interpolated precision is measured at the 11-recall level as shown in the following table.

Recall	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Pinterp	0.83	0.83	0.83	0.5	0.5	0.4	0.32	0.2	0.13	0.07	0.02

Table 10. Interpolated precision – 11-recall levels query 2

3. Query 3 “file not found in EDMS”

The aim of this query is to obtain the tickets that discuss the resolutions of the error “file not found” in the EDMS system. Figure 32 shows that 10 tickets are returned as a response to this query.



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/HDTicketsRetrieval/'. The page title is 'HelpDesk'. Below the title is a section titled 'Issue Navigator'. It contains a table with two columns: 'Key' and 'Summary'. The table lists 20 issues, with the first 10 matching the query 'file not found in EDMS'. The issues are as follows:

Key	Summary
QA01QNB-2039	file not found
JO01CAB-2009	File not Found -EDMS
JO01CAB-2011	EDMS - export files
QA01QNB-2129	file not found on the download
QA00MAR-2013	Request Batch File for QCB DR
QA01QNB-2176	User Not Found
YE01SABA-2132	Edms Not working
JO01CAB-2006	sequences problem - EDMS
JO00HBTf-2510	compare file
JO01CAB-2031	EDMS problem
QA01QNB-2162	file not found
YE01SABA-2115	Edms Documentaiton
JO02HBTf-167	Signature system is Hangup and it cause impact for banking system
QA06HSBC-84	file in error folder - QATCH
BH00HSBC-33	Inward File
YE01SABA-2136	File Not Found
QA01QNB-2031	FILE NOT FOUND
QA00BSI-2047	PM Visit
QA01QNB-2064	File Not Found
JO01CAB-2040	EDMS - file not found

At the bottom of the table, it says 'Displaying issues 1 to 20 of 20 matching issues.'

Figure 32. The results of query “file not found in EDMS”

The precision and recall are computed for the top k ranked hits ($k = 1, 2, 3, 4, \dots, 20$) (see Table 11) and then plotted to obtain a precision-recall curve, as shown in Figure 33.

Key	No. relevant tickets	Precision	Recall
1	1	1.00	0.08
2	1	0.50	0.08
3	2	0.67	0.15
4	2	0.50	0.15
5	3	0.60	0.23
6	3	0.50	0.23
7	4	0.57	0.31
8	4	0.50	0.31
9	5	0.56	0.38
10	6	0.60	0.46
11	6	0.55	0.46
12	6	0.50	0.46
13	7	0.54	0.54
14	8	0.57	0.62
15	9	0.60	0.69
16	10	0.63	0.77
17	10	0.59	0.77
18	10	0.56	0.77
19	10	0.53	0.77
20	10	0.50	0.77
Avg.		0.58	0.45

Table 11. Precision-Recall values query 3

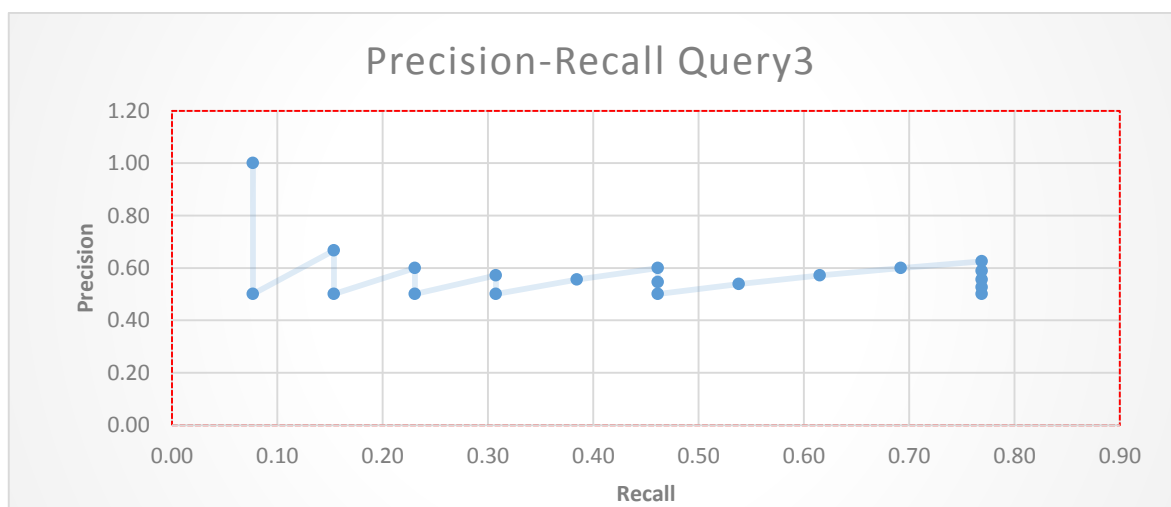


Figure 33. Precision-Recall curve query 3

The third experiment at $k=10$ gives the same precision value as the second experiment but with better recall (46%), bearing in mind that the third information need has a higher number of relevant tickets in the collection (26) than the second information need (13). Also, the six relevant tickets returned at $k=10$ carry enough information to answer the information need of this query.

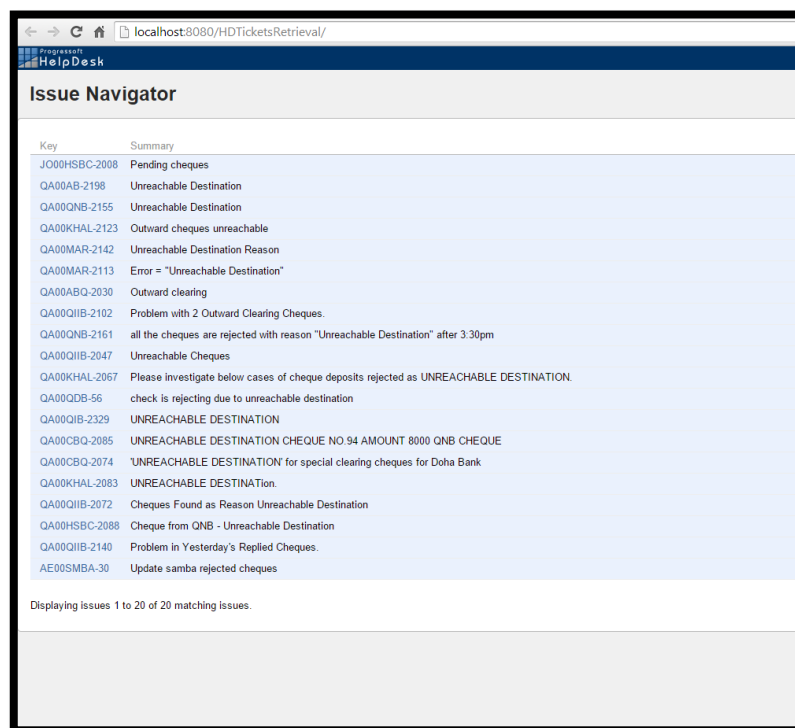
According to the statistics in the Figure 33, the interpolated precision is measured at the 11-recall level as shown in the following table.

Recall	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Pinterp	1	0.67	0.6	0.63	0.63	0.63	0.63	0.63	0.4	0.2	0.01

Table 12. Interpolated precision – 11 recall levels query 3

4. Query 4 “Unreachable destination cheques”

The aim of this query is to obtain the tickets that mention the causes of the cheques that have the status “Unreachable destination” in the PS-ECC system. Figure 34 shows that 10 tickets were returned as a response to this query.



Key	Summary
JO00HSBC-2008	Pending cheques
QA00AB-2198	Unreachable Destination
QA00QNB-2155	Unreachable Destination
QA00KHAL-2123	Outward cheques unreachable
QA00MAR-2142	Unreachable Destination Reason
QA00MAR-2113	Error = "Unreachable Destination"
QA00ABQ-2030	Outward clearing
QA00QIB-2102	Problem with 2 Outward Clearing Cheques.
QA00QNB-2161	all the cheques are rejected with reason "Unreachable Destination" after 3:30pm
QA00QIB-2047	Unreachable Cheques
QA00KHAL-2067	Please investigate below cases of cheque deposits rejected as UNREACHABLE DESTINATION.
QA00QDB-56	check is rejecting due to unreachable destination
QA00QIB-2329	UNREACHABLE DESTINATION
QA00CBQ-2085	UNREACHABLE DESTINATION CHEQUE NO.94 AMOUNT 8000 QNB CHEQUE
QA00CBQ-2074	'UNREACHABLE DESTINATION' for special clearing cheques for Doha Bank
QA00KHAL-2083	UNREACHABLE DESTINATION.
QA00QIB-2072	Cheques Found as Reason Unreachable Destination
QA00HSBC-2088	Cheque from QNB - Unreachable Destination
QA00QIB-2140	Problem in Yesterday's Replied Cheques.
AE00SMB-30	Update samba rejected cheques

Displaying issues 1 to 20 of 20 matching issues.

Figure 34. The results of query “Unreachable destination cheques”

The precision and recall are computed for the top k ranked hits ($k = 1, 2, 3, 4, \dots, 20$) (see Table 13) and then plotted to obtain a precision-recall curve, as shown in Figure 35.

Key	No. relevant tickets	Precision	Recall
1	0	0.00	0.00
2	1	0.50	0.02
3	2	0.67	0.03
4	3	0.75	0.05
5	4	0.80	0.07
6	5	0.83	0.08
7	6	0.86	0.10
8	7	0.88	0.11
9	8	0.89	0.13
10	9	0.90	0.15
11	9	0.82	0.15
12	10	0.83	0.16
13	11	0.85	0.18
14	12	0.86	0.20
15	13	0.87	0.21
16	14	0.88	0.23
17	15	0.88	0.25
18	16	0.89	0.26
19	17	0.89	0.28
20	17	0.85	0.28
Avg.		0.78	0.15

Table 13. Precision-Recall values query 4

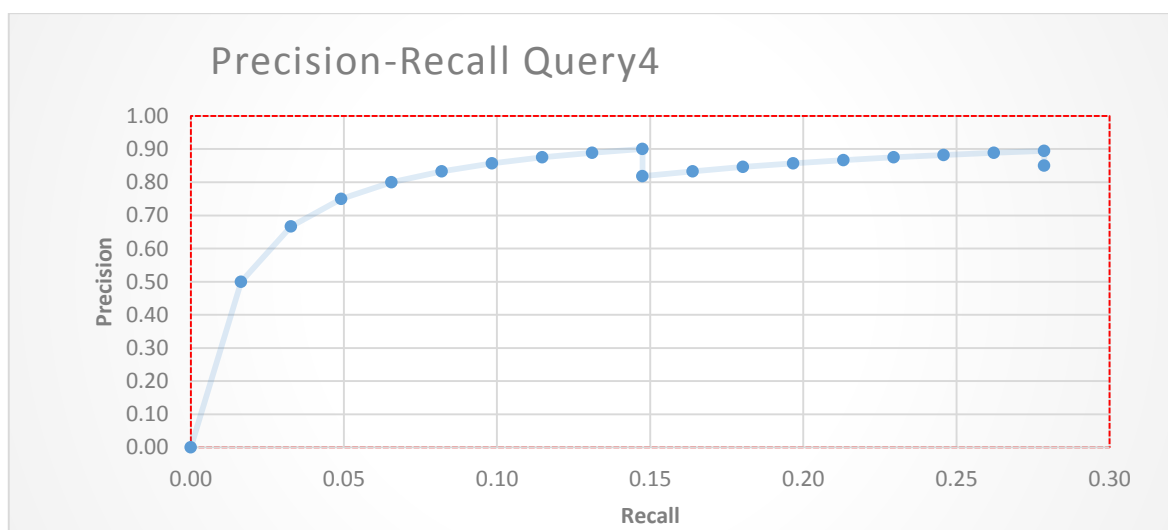


Figure 35. Precision-Recall curve query 4

This experiment gives a very high precision value, for example at $k=10$, the system retrieved 9 out of 10 as relevant tickets. The recall value is low as this information need has a high number of relevant tickets in the collection (61).

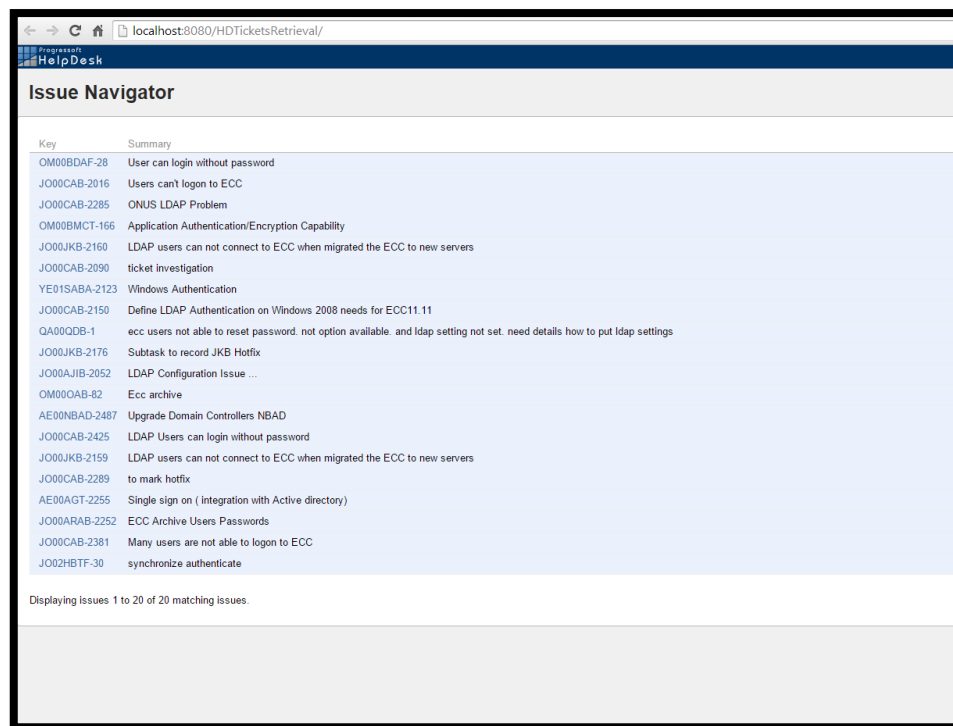
According to the statistics in Figure 35, the interpolated precision is measured at the 11 recall level as shown in the following table.

Recall	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Pinterp	0.9	0.85	0.8	0.8	0.65	0.65	0.5	0.33	0.21	0.08	0.06

Table 14. Interpolated precision – 11 recall levels query 4

5. Query 5 “ldap authentication issue”

The aim of this query is to obtain the tickets that mention the causes of the failures to authenticate the PS-ECC system with ldap. The following figure (36) shows that 10 tickets returned a response to this query.



Key	Summary
OM00BDAF-28	User can login without password
JO00CAB-2016	Users can't login to ECC
JO00CAB-2285	ONUS LDAP Problem
OM00BMCT-166	Application Authentication/Encryption Capability
JO00JKB-2160	LDAP users can not connect to ECC when migrated the ECC to new servers
JO00CAB-2090	ticket investigation
YE01SABA-2123	Windows Authentication
JO00CAB-2150	Define LDAP Authentication on Windows 2008 needs for ECC11.11
QA00QDB-1	ecc users not able to reset password. not option available. and ldap setting not set. need details how to put ldap settings
JO00JKB-2176	Subtask to record JKB Hotfix
JO00AJIB-2052	LDAP Configuration Issue ...
OM00OAB-82	Ecc archive
AE00NBAD-2487	Upgrade Domain Controllers NBAD
JO00CAB-2425	LDAP Users can login without password
JO00JKB-2159	LDAP users can not connect to ECC when migrated the ECC to new servers
JO00CAB-2289	to mark hotfix
AE00AGT-2255	Single sign on (integration with Active directory)
JO00ARAB-2252	ECC Archive Users Passwords
JO00CAB-2381	Many users are not able to login to ECC
JO02HBTf-30	synchronize authenticate

Displaying issues 1 to 20 of 20 matching issues.

Figure 36. The results of query “ldap authentication issue”

The precision and recall are computed for the top k ranked hits ($k = 1, 2, 3, 4, \dots, 20$) (see Table 15) and then plotted to obtain a precision-recall curve, as shown in Figure 37.

Key	No. relevant tickets	Precision	Recall
1	1	1.00	0.06
2	2	1.00	0.11
3	3	1.00	0.17
4	3	0.75	0.17
5	4	0.80	0.22
6	5	0.83	0.28
7	6	0.86	0.33
8	7	0.88	0.39
9	8	0.89	0.44
10	8	0.80	0.44
11	9	0.82	0.50
12	10	0.83	0.56
13	11	0.85	0.61
14	12	0.86	0.67
15	13	0.87	0.72
16	14	0.88	0.78
17	15	0.88	0.83
18	16	0.89	0.89
19	17	0.89	0.94
20	18	0.90	1.00
Avg.		0.87	0.51

Table 15. Precision-Recall values query 5

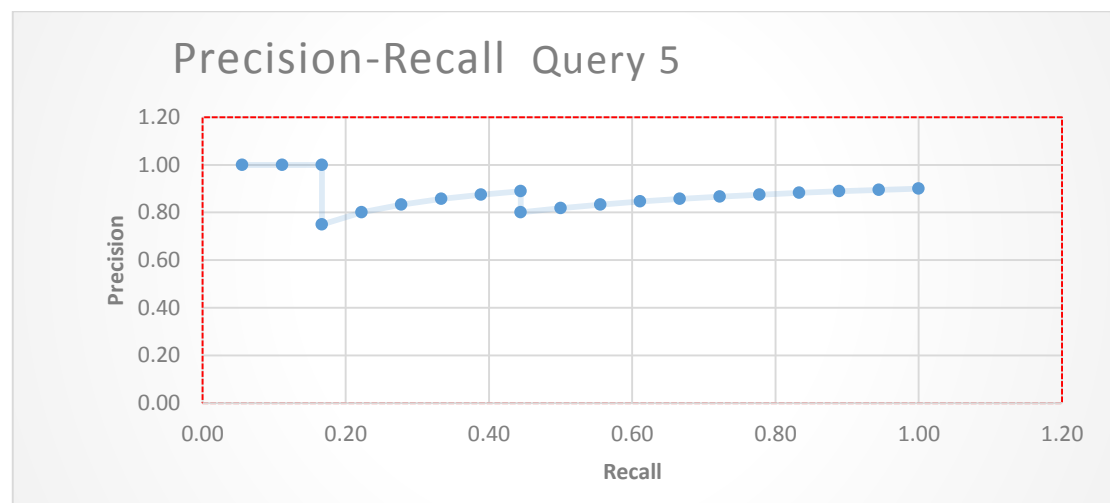


Figure 37. Precision-Recall curve query 5

The results obtained from this experiment are fascinating. The precision fraction reached 80% with $k=10$ and the system achieved 100% recall by returning 20 tickets, knowing that the total number of relevant tickets in the collection for this information need is 18.

According to the statistics in Figure 37, the interpolated precision is measured at the 11-recall level as shown in the following table.

Recall	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Pinterp	1	1	0.89	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9

Table 16. Interpolated Precision – 11 recall levels query 5

The system’s ability to retrieve the relevant tickets is high, based on the precision recall figures over five tests. The recall is also reasonable and acceptable as only the top 20 retrieved tickets were examined while some of the queries may have had more than 40 relevant tickets.

Examining the precision-recall curve for each information need is very useful in terms of providing a clue about the system behaviour against different queries. However, there is a desire to utilize these information towards having a few figures, or even a single figure to obtain the “Big Picture” of system’s efficiency. To do so, the data obtained from the five experiments can be translated into two precision measurements:

1. 11-point Interpolated Average Precision. Used in the first 8 TREC ad doc evaluations, the interpolated precision is measured at the 11 recall levels of 0.0, 0.1, 0.2, . . . , 1.0. The arithmetic mean of the interpolated precision is calculated at that recall level for each information need in the test collection (Manning et al, 2008).
2. Mean Average Precision. This provides a single-figure measure of quality across recall levels. Among evaluation measures, MAP has good discrimination and stability. For a single information need, average precision is the average of the precision value obtained for the set of top k documents existing after each relevant document is retrieved, and this value is then averaged over the information needs (Manning et al, 2008).

11-Point Interpolated Average Precision

The following table lists all precision interpolated values for all information needs along with the 11 recall levels. The last column is the calculated arithmetic mean of the interpolated precisions for all information needs at that recall level. Figure 38 shows the averaged 11-point precision/recall graph.

Recall Levels	<i>Pinterp1</i>	<i>Pinterp2</i>	<i>Pinterp3</i>	<i>Pinterp4</i>	<i>Pinterp5</i>	<i>Pinterp-avg</i>
0	1	0.83	1	0.9	1	0.946
0.1	0.83	0.83	0.67	0.85	1	0.836
0.2	0.83	0.83	0.6	0.8	0.89	0.79
0.3	0.83	0.5	0.63	0.8	0.9	0.732
0.4	0.77	0.5	0.63	0.65	0.9	0.69
0.5	0.73	0.4	0.63	0.65	0.9	0.662
0.6	0.73	0.32	0.63	0.5	0.9	0.616
0.7	0.73	0.2	0.63	0.33	0.9	0.558
0.8	0.73	0.13	0.4	0.21	0.9	0.474
0.9	0.73	0.07	0.2	0.08	0.9	0.396
1	0.3	0.02	0.01	0.06	0.9	0.258

Table 17. Computing the average of 11-point interpolated precision for five queries

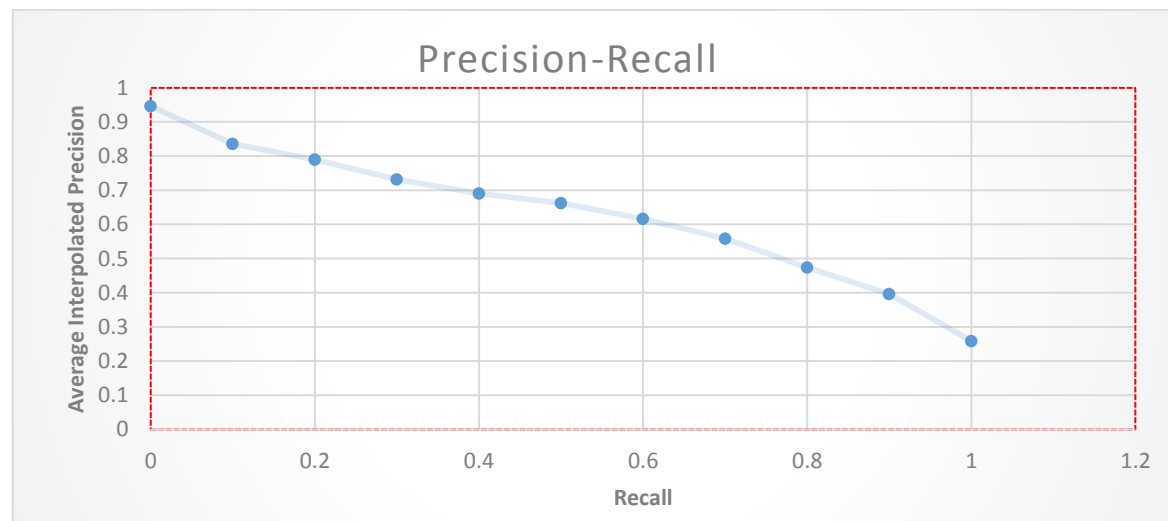


Figure 38. Averaged 11-point interpolated precision-recall graph across 5 queries

Overall, the two quantities seem to balance against one another: it is possible to get a recall of 1 by retrieving all tickets for all queries, however this will lead to a very low precision. The curve presents good system behaviour for recall and precision. These results are in line with what Manning et al (2008) said about a useful retrieval system, precision decreases with the increasing number of documents retrieved. A certain amount of recall is needed while tolerating only a small percentage of false positives (Manning et al, 2008).

Calculate Mean Average Precision

In Table 18 the precision is calculated for each information need if the k set of tickets returns a relevant ticket. The arithmetic mean of these precisions is calculated for each information need. These means are then averaged to obtain MAP.

Key	No. RT	P1	No. RT	P2	No. RT	P3	No. RT	P4	No. RT	P5
	1		2		3		4		5	
1	1	1	0	0.00	1	1.00	0	0.00	1	1.00
2	1		1	0.50	1		1	0.50	2	1.00
3	2	0.67	2	0.67	2	0.67	2	0.67	3	1.00
4	3	0.75	3	0.75	2		3	0.75	3	
5	4	0.8	4	0.80	3	0.60	4	0.80	4	0.80
6	5	0.83	5	0.83	3		5	0.83	5	0.83
7	5		5		4	0.57	6	0.86	6	0.86
8	6	0.75	6	0.75	4		7	0.88	7	0.88
9	7	0.77	6		5	0.56	8	0.89	8	0.89
10	7		6		6	0.60	9	0.90	8	
11	7		6		6		9		9	0.82
12	8	0.67	6		6		10	0.83	10	0.83
13	9	0.69	6		7	0.54	11	0.85	11	0.85
14	9		6		8	0.57	12	0.86	12	0.86
15	10	0.67	7	0.47	9	0.60	13	0.87	13	0.87
16	11	0.68	7		10	0.63	14	0.88	14	0.88
17	12	0.71	8	0.47	10		15	0.88	15	0.88
18	13	0.72	8		10		16	0.89	16	0.89
19	14	0.73	9	0.47	10		17	0.89	17	0.89
20	14		10	0.50	10		17		18	0.90
Precision Avg.		0.745714		0.56		0.63		0.78		0.88
MAP				0.72						

Table 18. Calculating the Mean Average Precision for five queries

The value of MAP (0.72) weights each information need equally, even if many tickets are relevant to some queries and very few are relevant to others.

The averaged 11-point precision/recall curve and MAP value demonstrate the effectiveness of the proposed system in finding the similarity between the ticket queries. The good performance of the proposed system is proven to benefit from using the vector space scoring model. The relevance can be further improved by introducing the concept of token normalisation, meaning to “canonicalise” tokens so that matches occur regardless of artificial differences in the character sequences of the tokens (Manning et al, 2008). For example, a search for “EDMS” might also need to match tickets containing “PS-EDMS”. The relevance can be enhanced by integrating a dictionary of synonyms to maintain the relationship between unnormalised tokens, such as “car” and “automobile”, so that seeking for one term will retrieve tickets that contain any of them. Furthermore, introducing approach for correcting spelling mistakes at the indexing phase and in queries will also help to boost relevance. As a result of having misspelled terms, tickets that contain those terms will not be returned because the correctly spelled query term doesn’t match with the misspelled index term and vice versa.

4.2 Evaluation of Researching Time

The researching time is evaluated in order to measure the system's performance and to give an idea of quality of processing speed. For this research time was calculated for each query executed by increasing the size of the collection (number of tickets). The experiments were performed on a Linux server with the following hardware resources:

Model name: Intel® Core™ i7-4700MQ CPU @ 2.40GHz

Mem-Total: 5969860 KB

Operating system: Red Hat Enterprise Linux Server release 6.5

The results are as follows:

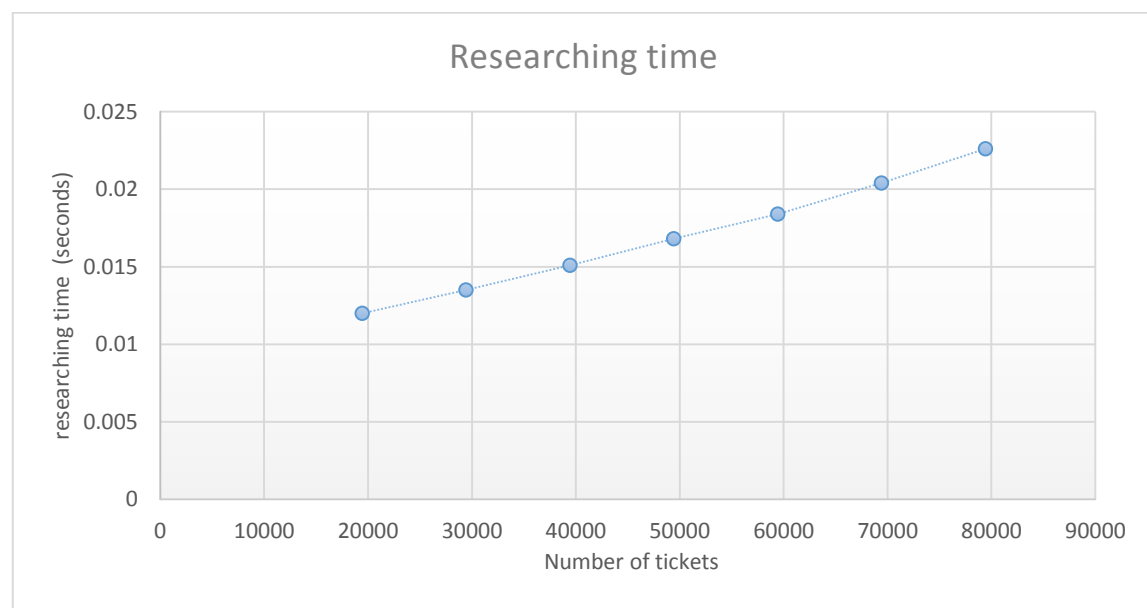


Figure 39. Researching time vs number of tickets

The curve in Figure 39 shows the evolution of researching time depending on the number of the tickets in the collection. The time of research was calculated by increasing the number of tickets each time. This increase was conducted by modifying the ticket IDs for 10,000 tickets and adding them to the collection by amending each word with a special character in order to create new, unique terms.

Although the time increases linearly (with an average 0.001766667 seconds per 10000 tickets) with respect to the increased number of the tickets, the researching time of the system is very rapid. This is demonstrated by finding the time difference between 79,433 tickets (800 MB) and 19,433 tickets (200 MB), which is 10.6 milliseconds. These results was compared with the outcomes of the test conducted by Ali et al (2011) to measure the researching time of a retrieval function that takes into account the vicinity of the terms in the query to find the relevant documents. Their evaluation shows that the researching time takes 20 milliseconds. Knowing that their experiment included only 340 document (104 MB) with 35,700 words in each.

Storing the terms' weighted values in the inverted file within their postings list significantly increases the speed of researching. Currently, when the cosine score is calculated for the tickets that contain the query's terms, the inverted file is opened to seek the postings list for each term and then the TF-IDF file is opened to obtain the weights in each ticket they appear in. Thus, by storing the values in one file, only one seek operation is needed for each term to obtain the postings list and their weights.

Another element that has to be looked at is the Hadoop MapFile used by the search API to lookup the query's terms in the inverted file. Currently, the index of the MapFile stores all the terms against the byte offsets of their postings lists in the inverted file. For instance, its index has the following structure:

Term	Byte Offset
acnum	8972455
account	8972632
acord	8973041
acprf	8974991
acquiretask	8977220
acquisit	8977410
acr	8977698
acrchiv	8977930

The Hadoop MapFile has the ability to store fractions of these terms in its index. For instance, if it is assumed that the inverted file only contains the above eight terms with their postings lists, and the MapFile is configured with an interval of "2" then only the following terms will be included in the index of the MapFile (partial index of terms):

Term	Byte Offset
acnum	8972455
acord	8973041
acquiretask	8977220
acr	8977698

In this case, if the MapFile is contacted to get the postings list for a particular term (not available in its index "acprf"), then its reader performs a binary search on the in-memory index to find the term "acord" in the index that is less than or equal to the search term "acprf" (White, 2012). Next, the reader seeks the offset of "acord" in the inverted file and reads entries until the term is greater than or equal to the search term "acprf". Increasing the index interval would decrease the amount of memory that the MapFile needs to store the index (White, 2012).

4.3 Evaluation of Index Performance

An experiment concerns the evaluation of indexing time. It includes 7 rounds and the number of tickets in the collection is increased every new round. The experiment was conducted on a Hadoop cluster environment which consists of one master node and two slave data nodes with the following resources:

Master node:

Model name: Intel® Core™ i7-4700MQ CPU @ 2.40GHz

Mem-Total: 5.9 GB

Operating system: Red Hat Enterprise Linux Server release 6.5

Slave nodes:

Model name: Intel® Core™ i7-4700MQ CPU @ 2.40GHz

Mem-Total: 1.9 GB

Operating system: Red Hat Enterprise Linux Server release 6.5

As mentioned in building search API section, in order to prepare the data needed for searching, three files have to be produced in advance:

1. Inverted file: demands one MapReduce job.
2. TF-IDF file: demands two MapReduce jobs.
3. Euclidean lengths file: demands one MapReduce job.

These jobs are executed sequentially, and the completion of one event invokes the next job in the chain. Thus, the total time for the entire indexing process is considered to be the sum of the execution time of these four jobs. To perform this experiment the number of tickets in the collection is varied for each round. At each round, the execution times for the four jobs are recorded and then summed up, as shown in Table 19.

		Job1			Job2			Job3			Job4			Total Time
Round	Number of Tickets	Start Time	End Time	Exec. Time	Start Time	End Time	Exec. Time	Start Time	End Time	Exec. Time	Start Time	End Time	Exec. Time	
Round 1	19433	05:07:50	05:09:56	00:02:06	05:09:56	05:10:46	00:00:50	05:10:46	05:10:59	00:00:13	05:10:59	05:11:04	00:00:05	00:03:14
Round 2	29433	05:30:35	05:33:49	00:03:14	05:33:49	05:34:56	00:01:07	05:34:56	05:35:18	00:00:22	05:35:18	05:35:24	00:00:06	00:04:49
Round 3	39433	06:20:03	06:24:17	00:04:14	06:24:17	06:25:49	00:01:32	06:25:49	06:26:12	00:00:23	06:26:12	06:26:20	00:00:08	00:06:17
Round 4	49433	07:10:30	07:15:47	00:05:17	07:15:47	07:17:40	00:01:53	07:17:40	07:18:08	00:00:28	07:18:08	07:18:18	00:00:10	00:07:48
Round 5	59433	08:06:39	08:13:25	00:06:46	08:13:25	08:15:48	00:02:23	08:15:48	08:16:25	00:00:37	08:16:25	08:16:39	00:00:14	00:10:00
Round 6	69433	09:23:12	09:31:16	00:08:04	09:31:16	09:34:52	00:03:36	09:34:52	09:35:40	00:00:48	09:35:40	09:36:01	00:00:21	00:12:49
Round 7	79433	10:15:07	10:25:03	00:09:56	10:25:03	10:29:59	00:04:56	10:29:59	10:31:27	00:01:28	10:31:27	10:32:01	00:00:34	00:16:54

Table 19. The execution time for each job

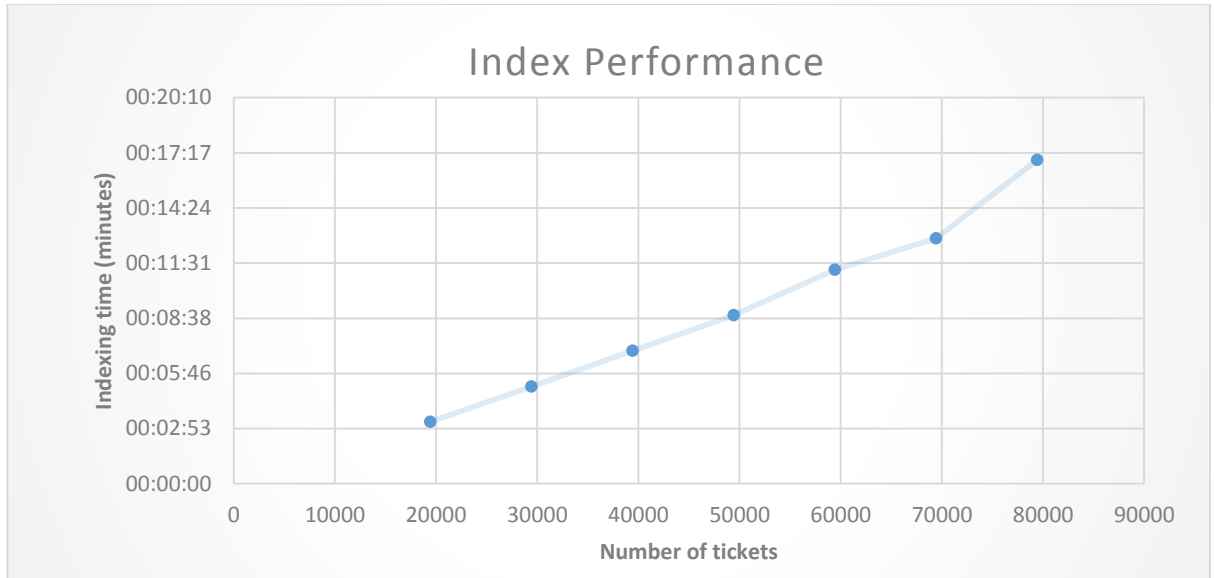


Figure 40. Evaluation of indexing time depending on the number of tickets

The results show that indexing time increases linearly with the number of the tickets (Figure 40), which gives an passable average indexing time of 0.212 milliseconds for each ticket (Table 20 and Figure 41).

Round	Number of Tickets	Total Time	Ticket Indexing Time (millisecond)
Round 1	19433	00:03:14	0.166383643
Round 2	29433	00:05:04	0.17214238
Round 3	39433	00:06:57	0.17624832
Round 4	49433	00:08:48	0.178018732
Round 5	59433	00:11:10	0.18788664
Round 6	69433	00:14:09	0.203793585
Round 7	79433	00:16:54	0.212757922
Average Ticket indexing time		0.185318746	

Table 20. Ticket indexing time per round

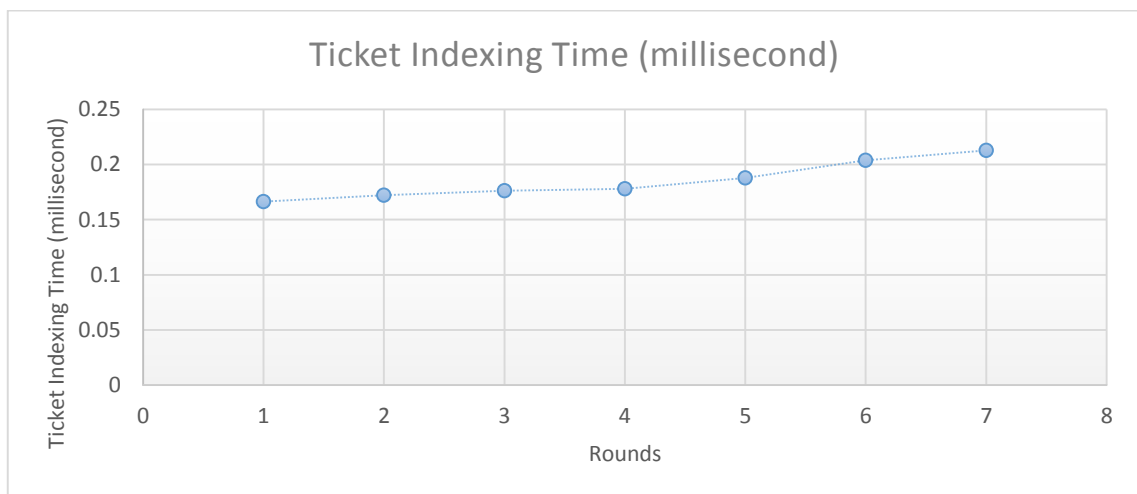


Figure 41. Ticket indexing time for each round

The results show that the distributed indexing has achieved good speed compared to an indexing process running on a single machine. For example, the indexing solution proposed by Ali et al (2011) was able to index a collection of documents that has a size of (104 MB) within 2 minutes. However, the distributed index proved to index a collection of 800 MB within 10 minutes. Knowing that the index computation includes ticket parsing which adds more processing overhead.

The size of the inverted file, produced from 19,433 tickets (700 MB) is 77.6 MB which can be reduced to less than 30 MB if compression is considered using Hadoop's SequenceFileOutputFormat. In addition, it was estimated that 5% of terms in the index file need to be added to a stop-word list as they are extremely common and have little value in helping to select matching tickets. These words included employees' names, dates and numbers. Furthermore, introducing spelling corrections (during text analysis) before passing the words to the index process will significantly reduce the number of terms in the index, thus reducing its size. However, neither the storage nor the index processing are big concerns as the cluster environment used to execute the index code and then store its data is built with inexpensive commodity machines. This means that the indexing computation can simply be scaled up by adding more commodity computers. Finally, using this distributed indexing approach gives the advantage of using a free software which is deployed on a cheap hardware and provide intrinsic distribution.

5. Conclusion

In this report a relevant-ticket retrieval solution has been presented. The solution adopts a distributable approach for indexing the tickets and storing data. It has a retrieval interface that exploits the vector space model for scoring the relevant tickets according to their similarities to a new ticket. The implementation has been tested using a data set from PS's helpdesk system. The results of retrieval experiments showed a great percentage of relevance and the performance of indexing components was fascinating.

The outcome of the five query experiments demonstrates that the proposed approach of using vector space scoring has the ability to achieve a precision of 70% and a recall of 30%. Overall, the system's ability to retrieve the relevant tickets is high as it was found that in average 7 out of top ten retrieved tickets contain the information need for a query. The recall is also reasonable and acceptable as only the top 20 retrieved tickets were examined even though some of the queries may have more than 40 relevant tickets. The percentage of relevance can be further improved by introducing spelling correction and using token normalisation techniques.

The research time was very fast compared to the results obtained by Ali et al (2011) in their experiments. The proposed system has a researching time equal to 22.6 milliseconds for a collection of 79,433 tickets (800 MB). However, suggestions to increase the research speed were highlighted. The distributed indexing process is also powerful, as 10 minutes were needed to completely indexing (800 MB) a collection of tickets. Suggestions were also highlighted for speeding-up the indexing process and expanding storage to cope with the growing number of tickets.

In summary, the proposed system has demonstrated good precision and acceptable recall values, in addition to providing a powerful indexing capability.

5.2 Challenges and Future Recommendations

Inverted Index and Data Structure

An earlier objective was to build the inverted index with a real-time property. Thus, the index will be automatically updated once a new ticket is added to the history. However, having a real-time updating index is a challenge as the structure has to include an auxiliary inverted index and a content filtering index besides the main inverted index. Furthermore, building the real-time structure in a distributable approach is another added problem. For these reasons, it was decided for this version of the project to stick to a single offline model of a traditional inverted index with a distributable architecture and to focus on adding a real-time property in another version.

The structure of the inverted index includes a list of the terms against their postings lists. The postings list of a term contains a number of posting elements which is equal to the number of the tickets that term occurs in. Each posting element contains another list that stores the term's attributes associated with every occurrence in the document. The complexity of these requirements made it difficult to answer which data structure had to be used for coding and storing a postings list as we also needed to consider dismantling the

postings list structure at the retrieval phase in order to extract the required values. The solution for this challenge was to use JSON, which significantly simplifies the process of building and parsing the postings list.

Another limitation to be addressed is the file-based data structure used to store the inverted index entries. Hadoop's SequenceFile is only readable and writable with Java. To overcome this limitation one could use Avro data file which works with large-scale data processing in Hadoop and can be processed in many languages (currently C, C++, C#, Java, Python, and Ruby), making it easier to share datasets with a larger audience (White, 2012).

Further, a strongly recommended modification is to incorporate the values of a term's weighting in the postings list, thus at the stage of calculating the cosine score for the relevant tickets only the inverted file will be sought with no need for the TF-IDF file. This change can significantly increase the speed of researching.

Text Analysing

The text analyser (Lucene StandardAnalyzer) used to analyse the textual content of tickets can only recognise English words, limiting the system. Future work will integrate a multi-language analyser.

Searching for Relevant Tickets

The search interface does not recognise the semantic meaning of terms. For example, searching for the tickets that contain the word "car" will not include the tickets that contain "automobile". This issue has an impact on the recall value. There are two solutions for this problem:

1. Manually refine a query.
2. Build a dictionary of synonyms to maintain the relationship between unnormalised tokens.

Another recommended enhancement that can boost relevance is to introduce an approach for rectifying spelling errors at the indexing phase and in queries. As a result of having misspelled terms, tickets that contain those terms will not be returned because the correctly spelled query term doesn't match the misspelled index term and vice versa.

6. References

- Agilemanifesto.org (2001) *Manifesto for Agile software development*. Available at: <http://agilemanifesto.org/> (Accessed: 11 January 2015).
- Agrawal, J., Sharma, N., Kumar, P., Parshav, V. and Goudar, R. (2013) 'Ranking of searched documents using semantic technology'. *Procedia Engineering*, 64, 1-7.
- Ali, B., Abdelkrim, B. and Mebarek, S. (2011) 'Term Proximity in document retrieval systems'. *Computer Science and Automation Engineering (CSAE)*, 4, 267-271.
- Arroyuelo, D., Bonacic, C., Gil-Costa, V., Marin, M. and Navarro, G. (2014) 'Distributed text search using suffix arrays'. *Parallel Computing*, 40(9), 471-495.
- Atlassian.com (2015) *JIRA - Issue & project tracking software*. Available at: <https://www.atlassian.com/software/jira> (Accessed: 18 January 2015).
- Baeza-Yates, R. and Ribeiro-Neto, B. (2011) *Modern information retrieval*. New York: Addison Wesley.
- Bridge, D., Goker, M., McGinty, L. and Smyth, B. (2005) 'Case-based recommender systems'. *Knowledge Engineering Review*, 20(3), 315.
- Chang, K., Raman, P., Carlisle, W. and Cross, J. (1996) 'A self-improving helpdesk service using case-based reasoning techniques', *Computers in Industry*, 30(2), 113-125.
- Chen, W., Tseng, S., Chang, L., Hong, T. and Jiang, M. (2002) 'A parallelized indexing method for large-scale case-based reasoning'. *Expert Systems with Applications*, 23(2), 95-102.
- Customerservicetrainingcenter.com (2015) *Customer service training programs & classes*. Available at: <http://www.customerservicetrainingcenter.com> (Accessed 7 January 2015).
- Dean, J. and Ghemawat, S. (2008) MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1), 107.
- Docs.oracle.com (2015) *Datatype limits*. Available at: http://docs.oracle.com/cd/B28359_01/server.111/b28320/limits001.htm#i287903 (Accessed 2 January 2015).
- Elasticsearch.org (2015) *Analysis and analyzers*. Available at: <http://www.elasticsearch.org/guide/en/elasticsearch/guide/current/analysis-intro.html> (Accessed 12 January 2015).
- Gog, I., Ionescu, V., Bercea, T., Musielak, T. and Yao, C. (2011) *Thorn, a replacement for Java - Implementing MapReduce for the Thorn language*. Imperial College of Science, Technology and Medicine Department of Computing.
- Hd.progresssoft.com (2015) *PS - Help Desk Customer Manual*. Available at: <http://hd.progresssoft.com/guides/download/attachments/12059143/PS->

HelpDesk+Customer+Manual.pdf?version=1&modificationDate=1291198272000
(Accessed 1 January 2015).

Istqbexamcertification.com (2015) *What are the Software Development Life Cycle (SDLC) phases?* Available at: <http://istqbexamcertification.com/what-are-the-software-development-life-cycle-sdlc-phases/> (Accessed 11 January 2015).

Iyaniwura, J. (2012) *Agile software development and UML*. [Presentation] Available at: <http://www.slideshare.net/johnnoi/agile-software-developmentanduml> (Accessed 21 January 2015).

Json.org (2015) *JSON*. Available at: <http://www.json.org/> (Accessed 13 January 2015).

Knuth, D. (ed.) (1968) 'Retrieval on secondary keys'. In: *The art of computer programming*, 2nd ed. Boston, MA: Pearson Education, 561-569.

Kyriakidou, A. (2014). Contemporary Methodologies.

Lin, J. and Dyer, C. (2010) 'Data-intensive text processing with MapReduce'. *Synthesis Lectures on Human Language Technologies*, 3(1), 1-177.

Lucene.apache.org (2015) *Apache Lucene - Apache Lucene Core*. Available at: <http://lucene.apache.org/core> (Accessed 6 January 2015).

Luk, R. and Lam, W. (2007) 'Efficient in-memory extensible inverted file'. *Information Systems*, 32(5), 733-754.

Manning, C., Raghavan, P. and Schütze, H. (2008) *Introduction to information retrieval*. New York: Cambridge University Press.

McCandless, M., Hatcher, E. and Gospodnetic, O. (2010) *Lucene in action*, 2nd edition. Stamford, CT: Manning Publications.

Oracle.com (2015) *Oracle 12c & Hadoop: Optimal store and process of Big Data*. Available at: <http://www.oracle.com/technetwork/articles/bigdata/hadoop-optimal-store-big-data-2188939.html> (Accessed 2 January 2015).

Perera, S. (2013) *Instant MapReduce patterns - Hadoop essentials how-to*. Birmingham: Packt Publishing.

Perera, S. and Gunarathne, T. (2013). *Hadoop MapReduce cookbook*. Birmingham: Packt Publishing.

Perez-Goytia, B. (2015). *Importing data from SQL databases into Hadoop with Sqoop and Oracle Data Integrator (ODI)*. Available at: <http://www.ateam-oracle.com/importing-data-from-sql-databases-into-hadoop-with-sqoop-and-oracle-data-integrator-odi/> (Accessed 2 January 2015).

Picek, R. (2009) Preliminary Communication. *Journal of Information and Organizational Sciences*, 33(2), 285-295.

Raghavan, P. and Nayak, P. (2014) *Lecture 6: Scoring, Term Weighting and the Vector Space Model*. Stanford University

Scrum Alliance (2015) *Home*. Available at: <https://www.scrumalliance.org/> (Accessed 11 January 2015).

Scrumguides.org (2014) *Scrum Guide*. Available at: <http://www.scrumguides.org/scrum-guide.html#artifacts-productbacklog> (Accessed 30 December 2014).

Scrumtrainingseries.com (2015) *Scrum Training series: free Scrum master training*. Available at: <http://Scrumtrainingseries.com> (Accessed 11 January 2015).

CMS (2008) Selecting a development approach. Available at: <http://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf> (Accessed 11 January 2015).

Smyth, B. (2007) 'Case-based recommendation'. In: Brusilovsky, P., Kobsa, A. and Nejdl, W. (eds.) *The Adaptive Web*. Berlin: Springer 342-376. Available at: <https://www.cs.auckland.ac.nz/~ian/CBR/recommenders.pdf> (Accessed 16 January 2015).

Softwaretestinghelp.com (2015) *How to deliver high value software features in a short time period using Agile Scrum process*. Available at: <http://www.softwaretestinghelp.com/how-to-deliver-high-value-software-features-in-a-short-time-period-using-agile-scrum-process/> (Accessed 11 January 2015).

Sqoop.apache.org (2015) *Sqoop User Guide (v1.4.4)*. Available at: <http://sqoop.apache.org/docs/1.4.4/SqoopUserGuide.html> (Accessed 2 January 2015).

Wang, D., Li, T., Zhu, S. and Gong, Y. (2011) iHelp: an intelligent online helpdesk system. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 41(1), 173-182.

Weiss, S., White, B., Apte, C. and Damerau, F. (2000) Lightweight document matching for help-desk applications. *IEEE Intelligent Systems*, 15(2), 57-61.

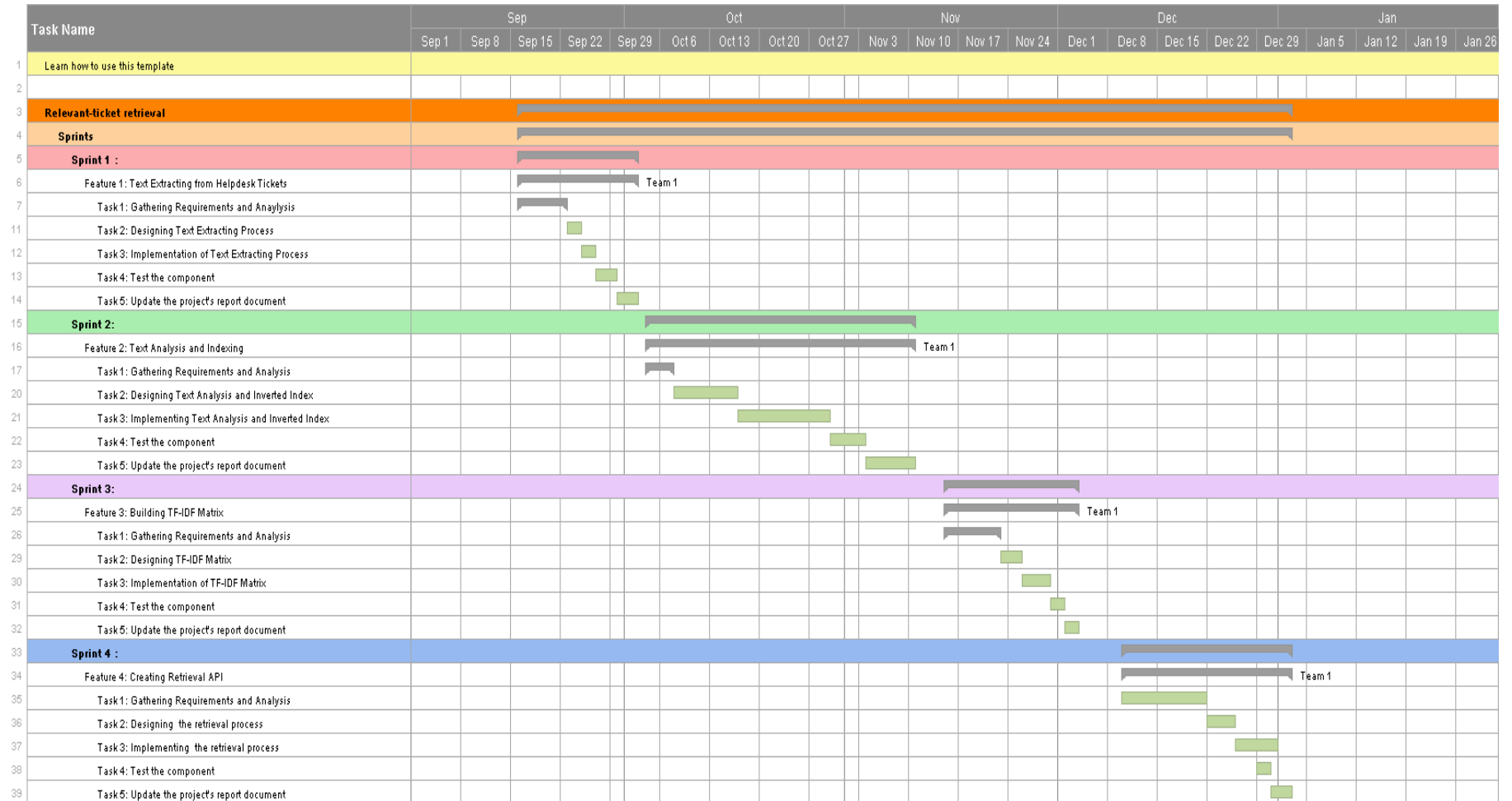
White, T. (2012) *Hadoop: the definitive guide*. Sebastopol, CA: O'Reilly & Associates.

Worms, D. (2013) *Options to connect and integrate Hadoop with Oracle - Adaltas*. Available at: <http://www.adaltas.com/blog/2013/05/15/hadoop-oracle-integration-tools/> (Accessed 2 January 2015).

Schütze, H. and Lioma, C. (2010). Lecture 8: Evaluation & Result Summaries. University of Stuttgart

7. Appendix

Appendix A: Gantt Diagram for the Development Process



Appendix B: Code

1. package org.greenwich.invertedIndex

```
package org.greenwich.invertedIndex;
import java.io.IOException;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.SequenceFileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
import org.greenwich.mapFile.MapFileFixer;
import org.greenwich.tfidf.EuclideanLengthsDriver;
import org.greenwich.tfidf.EuclideanLengthsMapper;
import org.greenwich.tfidf.tfidfMapper;
import org.greenwich.tfidf.tfidfMapper2;
import org.greenwich.tfidf.tfidfReducer;

public class InvertedIndexDriver extends Configured implements Tool{

    public String InvertedIndexInFolder ;
    public static String InvertedIndexOutFolder = "hdfs://localhost/user/admin/InvertedIndexOutFolder1";
    public String tfidf1InFolder = InvertedIndexOutFolder;
    public String tfidf1OutFolder = "hdfs://localhost/user/admin/tfidf1OutFolder1";
    public String tfidf2InFolder = tfidf1OutFolder;
    public static String tfidf2OutFolder = "hdfs://localhost/user/admin/tfidf2OutFolder1";
    public String EuclideanLengthsInFolder = tfidf2OutFolder;
    public static String EuclideanLengthsOutFolder = "hdfs://localhost/user/admin/EuclideanLengthsOutFolder1";

    public int run(String [] args) throws IOException, InterruptedException , Exception{

        if(args.length != 2) {
            System.err.printf("Usage:      %s      [generic-options]      <INPUT>      <OUTPUT>",
getClass().getSimpleName());
            ToolRunner.printGenericCommandUsage(System.err);
            return -1;
        }

        InvertedIndexInFolder = "hdfs://localhost/user/admin/input";

        Configuration conf =getConf();
        conf.set("fs.default.name", "hdfs://localhost:8020");
        conf.set("fs.file.impl", "com.conga.services.hadoop.patch.HADOOP_7682.WinLocalFileSystem");
        conf.set("mapreduce.map.log.level", "ALL");

        Path InvertedIndexOutPath = new Path(InvertedIndexOutFolder);
        FileSystem fs = InvertedIndexOutPath.getFileSystem(conf);
        if (fs.exists(InvertedIndexOutPath)){
            fs.delete(InvertedIndexOutPath,true);
        }

        Path tfidf1OutPath = new Path(tfidf1OutFolder);
        if (fs.exists(tfidf1OutPath)){
            fs.delete(tfidf1OutPath,true);
        }
    }
}
```

```

        Path tfidf2OutPath = new Path(tfidf2OutFolder);
        if (fs.exists(tfidf2OutPath)){
            fs.delete(tfidf2OutPath,true);
        }

        Path EuclideanLengthsOutPath = new Path(EuclideanLengthsOutFolder);
        if (fs.exists(EuclideanLengthsOutPath)){
            fs.delete(EuclideanLengthsOutPath,true);
        }

        Job job =new Job(conf,"InvertedIndex.Job");
        job.setJarByClass(InvertedIndexDriver.class);

        job.setInputFormatClass(TicketFileInputFormat.class);
        job.setOutputFormatClass(SequenceFileOutputFormat.class);

        job.setMapperClass(GenerateTermsMapper.class);
        job.setReducerClass(GeneratePositingListsReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(TermWritable.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job, new Path(InvertedIndexInFolder));
        FileOutputFormat.setOutputPath(job, new Path(InvertedIndexOutFolder));

        job.waitForCompletion(true);

        Job job2 =new Job(conf,"tfidf1.Job");
        job2.setJarByClass(org.greenwich.tfidf.tfidfDriver.class);

        job2.setInputFormatClass(SequenceFileInputFormat.class);
        job2.setOutputFormatClass(TextOutputFormat.class);

        job2.setMapperClass(tfidfMapper.class);
        job2.setReducerClass(tfidfReducer.class);

        job2.setMapOutputKeyClass(Text.class);
        job2.setMapOutputValueClass(Text.class);

        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job2, new Path(InvertedIndexOutFolder));
        FileOutputFormat.setOutputPath(job2, new Path(tfidf1OutFolder));

        job2.waitForCompletion(true);

        Job job3 =new Job(conf,"tfidf1.Job");
        job3.setJarByClass(org.greenwich.tfidf.tfidfDriver2.class);

        job3.setInputFormatClass(KeyValueTextInputFormat.class);
        job3.setOutputFormatClass(SequenceFileOutputFormat.class);

        job3.setMapperClass(tfidfMapper2.class);
        job3.setReducerClass(Reducer.class);

        job3.setMapOutputKeyClass(Text.class);
        job3.setMapOutputValueClass(Text.class);

        job3.setOutputKeyClass(Text.class);
        job3.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job3, new Path(tfidf2InFolder));
        FileOutputFormat.setOutputPath(job3, new Path(tfidf2OutFolder));

        job3.waitForCompletion(true);

```

```

        Job job4 =new Job(conf,"EuclideanLengthsJob");
        job4.setJarByClass(EuclideanLengthsDriver.class);

        job4.setInputFormatClass(SequenceFileInputFormat.class);
        job4.setOutputFormatClass(SequenceFileOutputFormat.class);

        job4.setMapperClass(EuclideanLengthsMapper.class);
        job4.setReducerClass(Reducer.class);

        job4.setMapOutputKeyClass(Text.class);
        job4.setMapOutputValueClass(Text.class);

        job4.setOutputKeyClass(Text.class);
        job4.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job4, new Path(EuclideanLengthsInFolder));
        FileOutputFormat.setOutputPath(job4, new Path(EuclideanLengthsOutFolder));

        return job4.waitForCompletion(true)? 0:1;

    }

    public static void main(String [] args) throws IOException, InterruptedException , Exception {
        int exitCode = ToolRunner.run(new InvertedIndexDriver(), args);

        MapFileFixer mapFileFixer = new MapFileFixer();
        mapFileFixer.convertSeqToMapFile(new Configuration(), InvertedIndexOutFolder );

        mapFileFixer.convertSeqToMapFile(new Configuration(), tfidf2OutFolder );

        mapFileFixer.convertSeqToMapFile(new Configuration(), EuclideanLengthsOutFolder );
        System.exit(exitCode);

    }
}

package org.greenwich.invertedIndex;

import java.io.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.TaskAttemptContext;

public class TicketFileInputFormat extends FileInputFormat<Text,Text>{
    private TicketRecordReader ticketRecordReader = null;

    public RecordReader<Text,Text> createRecordReader(
        InputSplit inputsplit, TaskAttemptContext context)
        throws IOException,InterruptedException {
        ticketRecordReader = new TicketRecordReader();
        ticketRecordReader.initialize(inputsplit, context);
        return ticketRecordReader;
    }
}

package org.greenwich.invertedIndex;
//import java.io.*;
import java.lang.StringBuilder;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.apache.hadoop.conf.Configuration;

```

```

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;

public class TicketRecordReader extends RecordReader<Text,Text>{
    private FileSplit filesplit;
    private Configuration conf;
    private Text value ;

    private static final Pattern regexPattern1 = Pattern.compile("([\\d]+)\\s+##ENDOFID\\s+([a-zA-Z0-9]+)-[0-9]+##ENDOFKEY\\s+(.*)##ENDOFSUMMARY\\s+(.*)##ENDOFDESCRIPTION\\s+(.*)##ENDOFACTIONBODY$");
    private Text currentKey;

    private BufferedReader bufferedreader;
    private int inputCountSofar=0;
    private StringBuilder issueID ;
    private StringBuilder ticketID ;
    private StringBuilder ticketSummary;
    private StringBuilder ticketDescription;
    private StringBuilder ticketComment;
    private String line = null;

    @Override
    public void initialize(InputSplit inputsplit,TaskAttemptContext context) throws IOException, InterruptedException{
        this.filesplit = (FileSplit) inputsplit;
        this.conf = context.getConfiguration();
        Path file = this.filesplit.getPath();
        FileSystem filesystem = FileSystem.get(conf);
        FSDataInputStream inputstream =filesystem.open(file);
        bufferedreader = new BufferedReader ( new InputStreamReader(inputstream),1024*100);
        while ((line = bufferedreader.readLine())!= null) {
            Matcher matcher = regexPattern1.matcher(line);
            if(matcher.matches()){
                break;
            }
        }
    }

    @Override
    public boolean nextKeyValue() throws IOException, InterruptedException {
        inputCountSofar++;
        issueID = new StringBuilder();
        ticketID = new StringBuilder();
        ticketSummary = new StringBuilder();
        ticketDescription =new StringBuilder();
        ticketComment = new StringBuilder();

        while ( (line = bufferedreader.readLine() ) != null) {
            Matcher matcher1 = regexPattern1.matcher(line);
            if (matcher1.matches()) {
                issueID = issueID.append(matcher1.group(1));
                ticketID = ticketID.append(matcher1.group(2));
                ticketSummary = ticketSummary.append(matcher1.group(3));
                ticketDescription = ticketDescription.append(matcher1.group(4));
                ticketComment = ticketComment.append(matcher1.group(5));
                currentKey = new Text(ticketID.toString());

                // the reason behind having the summary, description and comment values segregated, it is the anticipation of
                // applying different text analyzing during indexing.
                value= new Text(ticketSummary.toString() + " #ETSM# " + ticketDescription.toString() + " #ETDS# " +
                ticketComment.toString());
            }else {
                // value= new Text(ticketSummary.toString() + " #ETSM# " + ticketDescription.toString());
                //System.out.println(line);
                continue;
            }
        }

        return true;
    }
}

```

```

    }
    System.out.println("Line processing ends halfway through");
    return false;
    }

    @Override
    public Text getCurrentKey() {
        return currentKey;
    }

    @Override
    public Text getCurrentValue() {
        return value;
    }

    @Override
    public float getProgress() {
        return inputCountSofar;
    }

    @Override
    public void close() throws IOException{
        bufferedreader.close();
    }
}

}

package org.greenwich.invertedIndex;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;
import org.apache.hadoop.io.WritableUtils;

public class TermWritable implements WritableComparable<TermWritable> {

    Text    term;
    Text    type;
    IntWritable startOffset;
    IntWritable endOffset;
    IntWritable position;
    Text docID;
    IntWritable location;

    //All Writable implementations must have a default constructor so that the MapReduce framework can instantiate them
    public TermWritable() {
        set(new Text(), new Text(), new IntWritable(), new IntWritable(), new IntWritable(), new Text(), new IntWritable());
    }
    public TermWritable(String term, String type, int startOffset, int endOffset, int position, String docID, int location ) {
        set(new Text(term), new Text(type), new IntWritable(startOffset), new IntWritable(endOffset), new IntWritable(position),
new Text(docID), new IntWritable(location));
    }
    public TermWritable(Text term, Text type, IntWritable startOffset, IntWritable endOffset, IntWritable position, Text docID,
IntWritable location) {
        set(term, type, startOffset, endOffset, position, docID, location);
    }

    public void set(Text term, Text type, IntWritable startOffset, IntWritable endOffset, IntWritable position, Text docID,
IntWritable location){
        this.term = term;
        this.type = type;
        this.startOffset = startOffset;
        this.endOffset = endOffset;

```

```

        this.position = position;
        this.docID = docID;
        this.location =location;
    }

    public void setTerm(String term){
        this.term = new Text(term);
    }
    public void setTerm(Text term){
        this.term = term;
    }
    public void setType(String type){
        this.type= new Text(type);
    }
    public void setType(Text type){
        this.type= type;
    }
    public void setStartOffset(int startOffset){
        this.startOffset = new IntWritable(startOffset);
    }
    public void setStartOffset(IntWritable startOffset){
        this.startOffset= startOffset;
    }
    public void setEndOffset(int endOffset){
        this.endOffset= new IntWritable(endOffset);
    }
    public void setEndOffset(IntWritable endOffset){
        this.endOffset = endOffset;
    }
    public void setPosition(int position){
        this.position= new IntWritable(position);
    }
    public void setPosition(IntWritable position){
        this.position= position;
    }
    public void setDocID(String docID){
        this.docID = new Text(docID);
    }
    public void setDocID(Text docID){
        this.docID= docID;
    }
    public void setLocation(int location){
        this.location = new IntWritable(location);
    }
    public void setLocation(IntWritable location){
        this.location= location;
    }
    public Text getTerm() {
        return term;
    }
    public Text getType() {
        return type;
    }
    public IntWritable getStartOffset() {
        return startOffset;
    }
    public IntWritable getEndtOffset() {
        return endOffset;
    }
    public IntWritable getPosition() {
        return position;
    }
    public Text getDocID() {
        return docID;
    }
    public IntWritable getLocation() {
        return location;
    }
}

```

// TermWritable's write() method serializes each Text and IntWritable objects in turn to the output stream, by delegating to the Text and IntWritable objects themselves

```

@Override
public void write(DataOutput out) throws IOException {

```



```

term.write(out);
docID.write(out);
position.write(out);
startOffset.write(out);
endOffset.write(out);
type.write(out);
location.write(out);

}
//Similarly, readFields() deserializes the bytes from the input stream by delegating to each Text and IntWritable objects.
@Override
public void readFields(DataInput in) throws IOException {
term.readFields(in);
docID.readFields(in);
position.readFields(in);
startOffset.readFields(in);
endOffset.readFields(in);
type.readFields(in);
location.readFields(in);

}

/* The hash Code() method is used by the HashPartitioner (the default partitioner in MapReduce) to choose a reduce partition,
so you should make sure that you write a good hash
function that mixes well to ensure reduce partitions are of a similar size.*/
@Override
public int hashCode() {
//return term.hashCode() * 163 + type.hashCode();
return term.hashCode() ;
}
@Override
public boolean equals(Object o) {
if (o instanceof TermWritable) {
TermWritable tw = (TermWritable) o;
return term.equals(tw.term) && docID.equals(tw.docID) && position.equals(tw.position);
}
return false;
}
//TextOutputFormat calls toString() on keys and values for their output representation. For Text Pair, we write the underlying
Text objects as strings separated by a tab character.
@Override
public String toString() {
return docID + "[" + type + "," + position + "," + startOffset + "," + endOffset + "," + location + "$]";
//return docID + "\t";
}

//TextPair is an implementation of WritableComparable, so it provides an implementation of the compareTo() method that
imposes the ordering you would expect:
@Override
public int compareTo(TermWritable tp) {
int cmp = term.compareTo(tp.term);
if (cmp != 0) {
return cmp;
}
int cmp1 = docID.compareTo(tp.docID);
if (cmp1 != 0) {
return cmp1;
}
return position.compareTo(tp.position);
// return cmp;
}

/*
compare two TermWritable objects just by looking at their serialized representations.
It turns out that we can do this, since TermWritable is the concatenation of three Text objects and three IntWritable objects,
and the binary representation of a Text object is a variable-length integer containing
the number of bytes in the UTF-8 representation of the string, followed by the UTF-8
bytes themselves. The trick is to read the initial length, so we know how long the "term" Text object's byte representation is;
*/

public static class Comparator extends WritableComparator {

```

```

private static final Text.Comparator Text_Comparator = new Text.Comparator();
private static final IntWritable.Comparator IntWritable_Comparator = new IntWritable.Comparator();
public Comparator() {
    super(TermWritable.class);
}

public int compare(byte[] b1, int s1, int l1,
    byte[] b2, int s2, int l2) {
    try {

        /* The subtle part of this code is calculating firstL1 and firstL2, the lengths of the term
        Text field in each byte stream. Each is made up of the length of the variable-length integer
        (returned by decodeVIntSize() on WritableUtils) and the value it is encoding (returned by readVInt()). */

        int termL1 = WritableUtils.decodeVIntSize(b1[s1]) + readVInt(b1,s1) ;
        int termL2 = WritableUtils.decodeVIntSize(b2[s2]) + readVInt(b2,s2) ;
        int cmp = Text_Comparator.compare(b1, s1, termL1, b2, s2, termL2);
        if ( cmp != 0){
            return cmp;
        }
        // Since the representation binary on IntWritable is 4 bytes then the length of the second field docID is
s1+firstL1,firstL1+4.
        // firstL1 is the length of the binary representation of the first field term.

        int docIDL1 = WritableUtils.decodeVIntSize(b1[s1+termL1]) + readVInt(b1,s1+termL1) ;
        int docIDL2 = WritableUtils.decodeVIntSize(b2[s2+termL2]) + readVInt(b2,s2+termL2) ;
        int cmp1 = Text_Comparator.compare(b1, s1+termL1, docIDL1, b2, s2+termL2, docIDL2);
        if ( cmp1 != 0){
            return cmp1;
        }
        return IntWritable_Comparator.compare(b1, docIDL1, docIDL1+4, b2, docIDL2, docIDL2+4);
    } catch (IOException e) {
        throw new IllegalArgumentException(e);
    }
}

@Override
public int compare(WritableComparable a, WritableComparable b) {
    if (a instanceof TermWritable && b instanceof TermWritable) {
        int cmp = ((TermWritable) a).term.compareTo(((TermWritable)b).term);
        if (cmp != 0) {
            return cmp;
        }
        int cmp1 = ((TermWritable) a).docID.compareTo(((TermWritable)b).docID);
        if (cmp1 != 0) {
            return cmp1;
        }
        return ((TermWritable) a).position.compareTo(((TermWritable)b).position);
    }
    return super.compare(a, b);
}
}
//MapReduce sees the TermAttribute class, it knows to use the raw comparator as its default comparator.

static{
    WritableComparator.define(TermWritable.class, new Comparator());
}
}

package org.greenwich.invertedIndex;

import java.io.IOException;
import java.io.StringReader;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.lucene.analysis.TokenStream;
import org.apache.lucene.analysis.tokenattributes.OffsetAttribute;
import org.apache.lucene.analysis.tokenattributes.PositionIncrementAttribute;
import org.apache.lucene.analysis.tokenattributes.TermAttribute;
import org.apache.lucene.analysis.tokenattributes.TypeAttribute;
import org.apache.lucene.util.Version;

```

```

import org.greenwich.analyzers.TicketStandardAnalyzer;

@SuppressWarnings("deprecation")
public class GenerateTermsMapper extends Mapper<Text, Text, Text, TermWritable>{

    private TicketStandardAnalyzer ticketstandardanalyzer =new TicketStandardAnalyzer(Version.LUCENE_30); ;

    public void map(Text key, Text value, Context context) throws IOException, InterruptedException {

        String[] value0 = value.toString().split("#ETSM#");
        String ticketSummary = value0[0];
        String[] value1 = value0[1].split("#ETDS#");

        String ticketDescription =value1[0];
        String ticketComment = value1[1];

        String[] Values = {ticketSummary,ticketDescription,ticketComment};

        for(int i=0 ; i< Values.length; i++) {

            TokenStream standardtokenstream = ticketstandardanalyzer.tokenStream("contents", new
StringReader(Values[i].toString()));
            TermAttribute term = standardtokenstream.addAttribute(TermAttribute.class);
            PositionIncrementAttribute posIncr =
standardtokenstream.addAttribute(PositionIncrementAttribute.class);
            OffsetAttribute offset = standardtokenstream.addAttribute(OffsetAttribute.class);
            TypeAttribute type = standardtokenstream.addAttribute(TypeAttribute.class);

            int position =0;

            while (standardtokenstream.incrementToken()) {
                int increment = posIncr.getPositionIncrement();
                if ( increment > 0) {
                    position = position + increment;
                }
            }

            posIncr.setPositionIncrement(position);

            TermWritable tw = new TermWritable();
            tw.setTerm(term.term());
            tw.setType(type.type());
            tw.setStartOffset(offset.startOffset());
            tw.setEndOffset(offset.endOffset());
            tw.setPosition(posIncr.getPositionIncrement());
            tw.setDocID(key.toString());
            tw.setLocation(Values.length - i);
            context.write(tw.getTerm(),tw);

        }

    }

}

package org.greenwich.invertedIndex;

```

```

import java.io.IOException;
import java.util.*;

import org.json.*;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class GeneratePositingListsReducer extends Reducer<Text,TermWritable,Text,Text>{

    public void reduce(Text key, Iterable<TermWritable> values, Context context) throws IOException,
    InterruptedException {

        try{

            TreeMap<String,JSONArray> DocPosMap = new TreeMap<String,JSONArray>();

            for(TermWritable value: values) {

                TermWritable termWritable = value;
                String [] docIDPos = new String[6];
                docIDPos[0] = termWritable.getDocID().toString();
                docIDPos[1] = termWritable.getPosition().toString();
                docIDPos[2] = termWritable.getStartOffset().toString();
                docIDPos[3] = termWritable.getEndtOffset().toString();
                docIDPos[4] = termWritable.getType().toString();
                docIDPos[5] = termWritable.getLocation().toString();

                JSONObject obj= new JSONObject();
                obj.put("position",docIDPos[1]);
                obj.put("startOffset",docIDPos[2]);
                obj.put("endOffset",docIDPos[3]);
                obj.put("termType",docIDPos[4]);
                obj.put("termLocation",docIDPos[5]);

                if (DocPosMap.containsKey(docIDPos[0])){

                    JSONArray array = (JSONArray)DocPosMap.get(docIDPos[0]);
                    array.put(obj);

                    DocPosMap.put(docIDPos[0],array);

                }

                else {

                    JSONArray array = new JSONArray();

                    array.put(obj);
                    DocPosMap.put(docIDPos[0],array);

                }

            }

            JSONArray PostingListsArray = new JSONArray();

            Set<Map.Entry<String, JSONArray>> setDocPosMap = DocPosMap.entrySet();
            Iterator<Map.Entry<String, JSONArray>> IT = setDocPosMap.iterator();

            while (IT.hasNext()) {

                Map.Entry<String, JSONArray> ME = (Map.Entry<String, JSONArray>) IT.next();

                JSONObject obj= new JSONObject();
                obj.put("docID", ME.getKey());
                obj.put("termFreq", ME.getValue().length());
                obj.put("TermPositions", ME.getValue());
            }
        }
    }
}

```

```

        PostingListsArray.put(obj);
    }

    JSONObject obj= new JSONObject();
    obj.put("DocFreq", PostingListsArray.length());
    obj.put("PostingLists", PostingListsArray);

    context.write(key, new Text(obj.toString()));
}
catch (JSONException e){
}

}

}

```

2. package org.greenwich.analyzers;

```
package org.greenwich.analyzers;
```

```
import java.io.*;
import java.util.*;
```

```
import org.apache.lucene.analysis.*;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.util.*;
```

```
public class TicketStandardAnalyzer extends Analyzer{
```

```
    private StandardAnalyzer standardanalyzer;
    private CharArraySet stopWords ;
```

```
    public TicketStandardAnalyzer(Version version) {
```

```
        this.stopWords = new CharArraySet(Version.LUCENE_30, 30, true);
        stopWords.add("issue");
        stopWords.add("helpdesk");
        stopWords.add(standardanalyzer.STOP_WORDS_SET);
```

```
        System.out.println(stopWords);
        this.standardanalyzer = new StandardAnalyzer(version, this.stopWords);
```

```
    }
    public TokenStream tokenStream( String fieldName,Reader reader) {
```

```
        TokenStream standardtokenstream = standardanalyzer.tokenStream(fieldName,reader);
        return new PorterStemFilter(standardtokenstream);
```

```
    }
```

```
}
```

```
package org.greenwich.analyzers;
```

```
import java.io.StringReader;
import java.util.*;
import java.util.List;
import java.io.*;
```

```
import org.apache.lucene.analysis.TokenStream;
import org.apache.lucene.analysis.tokenattributes.TermAttribute;
import org.apache.lucene.util.Version;
import org.greenwich.invertedIndex.TermWritable;
```

```
@SuppressWarnings("deprecation")
public class QueryAnalyzer {
```

```
    private String query;
```

```
    public QueryAnalyzer(String query) {
        this.query = query;
    }
```

```
    public List<TermWritable> analyzeQuery() throws IOException {
        List<TermWritable> queryTerms = new LinkedList<TermWritable>();
        TicketStandardAnalyzer ticketstandardanalyzer = new TicketStandardAnalyzer(Version.LUCENE_30);
```

```

        TokenStream standardtokenstream = ticketstandardanalyzer.tokenStream("contents", new
StringReader(query.toString()));
        TermAttribute term = standardtokenstream.addAttribute(TermAttribute.class);

        while (standardtokenstream.incrementToken()) {
            TermWritable termWritable = new TermWritable();
            termWritable.setTerm(term.term());
            queryTerms.add(termWritable);
        }
        ticketstandardanalyzer.close();
        return queryTerms;
    }
}

```

3. package org.greenwich.mapFile

```
package org.greenwich.mapFile;

import java.io.IOException;
import java.net.URI;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.util.ReflectionUtils;
import org.apache.hadoop.io.MapFile;

import org.apache.hadoop.io.SequenceFile;

public class MapFileFixer {
    String mapUri ;
    Configuration conf ;
    FileSystem fs;
    Path map ;
    Path mapData ;

    public MapFileFixer() {

    }

    public void convertSeqToMapFile(Configuration conf,String mapUri) throws IOException, Exception{
        this.mapUri = mapUri;
        this.conf =conf;

        conf.set("fs.default.name", "hdfs://localhost:8020");
        conf.set("fs.file.impl", "com.conga.services.hadoop.patch.HADOOP_7682.WinLocalFileSystem");
        conf.set("mapreduce.map.log.level", "ALL");

        this.fs = FileSystem.get(URI.create(mapUri), conf);
        this.map = new Path(mapUri+"/part-r-00000");

        SequenceFile.Reader reader = null;
        MapFile.Writer writer = null;
        try {
            reader = new SequenceFile.Reader(fs, map, conf);
            WritableComparable key = ReflectionUtils.newInstance(reader.getKeyClass(),conf);
            WritableComparable value = (WritableComparable) ReflectionUtils.newInstance(reader.getValueClass(),conf);
            writer = new MapFile.Writer(conf, fs,this.mapUri, key.getClass(),value.getClass());
            writer.setIndexInterval(1);
            while (reader.next(key, value)) {
                writer.append(key, value);
            }
        } finally {
            IOUtils.closeStream(reader);
            IOUtils.closeStream(writer);
        }
    }
}
```



```

}

package org.greenwich.mapFile;

import java.net.URI;
import java.io.*;
import java.util.*;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
//import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.io.MapFile;
import org.apache.hadoop.io.Text;
import org.json.*;
public class ReadingMapFile {

    private String mapUri;

    private Configuration conf;
    private FileSystem filesystem;
    private MapFile.Reader reader = null ;

    public ReadingMapFile(String mapUri)throws IOException{

        this.conf = new Configuration();
        conf.set("fs.default.name", "hdfs://localhost:8020");
        conf.set("fs.file.impl", "com.conga.services.hadoop.patch.HADOOP_7682.WinLocalFileSystem");
        conf.set("mapreduce.map.log.level", "ALL");
        this.mapUri= mapUri;
        this.filesystem = FileSystem.get(URI.create(mapUri), conf);
    }

    public JSONObject getTermPostingsList(String term) throws IOException, JSONException{

        Text key ;
        Text value ;

        try {
            reader = new MapFile.Reader(filesystem, mapUri, conf);
            key = new Text(term);
            value = new Text();

            if (reader.get(key, value) == null){
                return new JSONObject();
            }
            else {
                reader.get(key, value);
                return new JSONObject(value.toString());
            }

        }
        finally {
            IOUtils.closeStream(reader);
        }

    }

    public Map<String, Float> getDocEuclideanLengths(Set<String> DocsSet)throws IOException{

        Map<String, Float> DocsEuclideanLengthsMap = new TreeMap<String, Float> ();
        Text key ;
        Text value ;

        try {
            reader = new MapFile.Reader(filesystem, mapUri, conf);
            for(String docID:DocsSet) {
                key = new Text(docID);
                value = new Text();
            }
        }
    }

```

```

        reader.get(key, value);
        DocsEuclideanLengthsMap.put(docID, Float.parseFloat(value.toString()));
    }
    return DocsEuclideanLengthsMap;
}finally {
    IOUtils.closeStream(reader);
}

}

public JSONObject getDocVector(String DocID) throws IOException, JSONException{

    Text key ;
    Text value ;

    try {
        reader = new MapFile.Reader(filesystem, mapUri, conf);
        key = new Text(DocID);
        value = new Text();

        if (reader.get(key, value) == null){
            return new JSONObject();
        }
        else {
            reader.get(key, value);
            return new JSONObject(value.toString());
        }
    }
    finally {
        IOUtils.closeStream(reader);
    }

}

}

```

4. package org.greenwich.searchEngine;

```
package org.greenwich.searchEngine;
import java.util.*;
import java.io.*;
```

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.MapFile;
import org.greenwich.invertedIndex.TermWritable;
import org.greenwich.mapFile.ReadingMapFile;
import org.greenwich.termWeighting.TermWeighting;
import org.json.*;
```

```
public class Ranking {
```

```
    public long N;
    private String mapUriInvertedIndex = "hdfs://localhost/user/admin/InvertedIndexOutFolder/";
    private String mapUriEuclideanLengths = "hdfs://localhost/user/admin/EuclideanLengthsOutFolder/";
    private String mapUriTFIDF = "hdfs://localhost/user/admin/tfidf2OutFolder/";
    private ReadingMapFile readingMapFileInvertedIndex ;
    private ReadingMapFile readingMapFileEuclideanLengths ;
    private ReadingMapFile readingMapFileTFIDF;
```

```
    public Ranking(long N) throws IOException{
        this.N = N;
    }
```

```
    public List<ReturnedDocument> calCosineScore(List<TermWritable> QueryTermsList)throws IOException,
    JSONException {
```

```
        Map<String,Float> DocsScoresMap = new TreeMap<String,Float>();
```

```
        // calculate the term frequencies for query terms
```

```
        Map<String,Integer> termFreqsMap = calcTermQeuryFreqs(QueryTermsList);
```

```
        TermWeighting termWeighting = new TermWeighting();
```

```
        Float queryNormalizationFactor= new Float(0);
```

```
        readingMapFileInvertedIndex = new ReadingMapFile(mapUriInvertedIndex);
```

```
        for(TermWritable queyTerm:QueryTermsList){
```

```
            //calculate term query weighting
```

```
            String Term = queyTerm.getTerm().toString();
```

```
            Integer termQueryFreq = termFreqsMap.get(Term);
```

```
            JSONObject TermPostingList;
```

```
            if ( readingMapFileInvertedIndex.getTermPostingsList(Term).length() != 0)
```

```
            {
```

```
                TermPostingList = readingMapFileInvertedIndex.getTermPostingsList(Term);
```

```
                long termDocFreq = getTermDocFreq(TermPostingList);
```

```
                Float TermQueryWeighting = termWeighting.calTermQueryWeight(N,
termQueryFreq, termDocFreq);
```

```
                queryNormalizationFactor += (float) Math.pow(TermQueryWeighting, 2);
```

```
                JSONArray TermPostingListArray =
TermPostingList.getJSONArray("PostingLists");
```

```
                for(int i=0; i <TermPostingListArray.length(); i++){
```

```
                    JSONObject Posting = TermPostingListArray.getJSONObject(i);
```

```
                    String docID = Posting.getString("docID");
```

```
                    Integer termFreq = new Integer(Posting.getInt("termFreq"));
```

```
                    JSONArray TermPositionsArray =
Posting.getJSONArray("TermPositions");
```

```
                    int termLocationCount=0;
```

```

        for(int j=0; j<TermPositionsArray.length(); j++) {
            JSONObject TermPosition
=TermPositionsArray.getJSONObject(j);
            termLocationCount +=
TermPosition.getInt("termLocation");
        }

        Float TermDocWeighting = new
TermWeighting().calTermDocWeight(N, termFreq, termDocFreq, termLocationCount);
        if (DocsScoresMap.containsKey(docID)){
            DocsScoresMap.put(docID,DocsScoresMap.get(docID) +
(TermDocWeighting*TermQueryWeighting));
        }
        else {
DocsScoresMap.put(docID,TermDocWeighting*TermQueryWeighting);
        }
    }
    else {
        continue;
    }
}

if ( DocsScoresMap.size() == 0){
    List<ReturnedDocument> DocumentsList = new ArrayList();
    return DocumentsList;
}

else {
    //The Map of DocID - Length holds the lengths (normalization factors) for each of document in
DocumentsList
    readingMapFileEuclideanLengths = new ReadingMapFile(mapUriEuclideanLengths);
    Map<String, Float> DocsEuclideanLengthsMap =
    readingMapFileEuclideanLengths.getDocEuclideanLengths(DocsScoresMap.keySet());

    //divide the score of each document by its normalization factor
    for(String docID:DocsScoresMap.keySet()){
        Float score = DocsScoresMap.get(docID);
        score = score / (DocsEuclideanLengthsMap.get(docID)*queryNormalizationFactor);
        DocsScoresMap.put(docID,score );
    }

    //Sort the documents based on their scores.
    List<ReturnedDocument> DocumentsList = sortDocumentScores(DocsScoresMap);

    return DocumentsList;
}

}

public List<ReturnedDocument> calCosineScoreTFIDF(List<TermWritable> QueryTermsList)throws IOException,
JSONException {
    Map<String,Float> DocsScoresMap = new TreeMap<String,Float>();

    // calculate the term frequencies for query terms
    Map<String,Integer> termFreqsMap = calcTermQueryFreqs(QueryTermsList);

    TermWeighting termWeighting = new TermWeighting();

    Float queryNormalizationFactor= new Float(0);
    readingMapFileInvertedIndex = new ReadingMapFile(mapUriInvertedIndex);
    readingMapFileTFIDF = new ReadingMapFile(mapUriTFIDF);

```

```

Map<String, JSONObject> TFIDFMap = new TreeMap<String, JSONObject>();
for(TermWritable queyTerm:QueryTermsList){

    //calculate term query weighting
    String Term = queyTerm.getTerm().toString();
    Integer termQueryFreq = termFreqsMap.get(Term);
    JSONObject TermPostingList;
    if ( readingMapFileInvertedIndex.getTermPostingsList(Term).length() != 0)
    {
        TermPostingList = readingMapFileInvertedIndex.getTermPostingsList(Term);

        long termDocFreq = getTermDocFreq(TermPostingList);

        Float TermQueryWeighting = termWeighting.calTermQueryWeight(N,
termQueryFreq, termDocFreq);

        queryNormalizationFactor += (float) Math.pow(TermQueryWeighting, 2);

        JSONArray TermPostingListArray =
TermPostingList.getJSONArray("PostingLists");
        for(int i=0; i <TermPostingListArray.length(); i++){
            JSONObject Posting = TermPostingListArray.getJSONObject(i);
            String docID = Posting.getString("docID");
            Float TermDocWeighting;
            if (TFIDFMap.containsKey(docID) ) {
                TermDocWeighting = (float)
TFIDFMap.get(docID).getDouble(queyTerm.getTerm().toString());
            }
            else {
                TFIDFMap.put(docID,
TermDocWeighting = (float)
TFIDFMap.get(docID).getDouble(queyTerm.getTerm().toString());
            }

            if (DocsScoresMap.containsKey(docID)){
                DocsScoresMap.put(docID,DocsScoresMap.get(docID) +
(TermDocWeighting*TermQueryWeighting));
            }
            else {
                DocsScoresMap.put(docID,TermDocWeighting*TermQueryWeighting);
            }
        }
    }
    else {
        continue;
    }
}

if ( DocsScoresMap.size() == 0){
    List<ReturnedDocument> DocumentsList = new ArrayList();
    return DocumentsList;
}

else {

    //The Map of DocID - Length holds the lengths (normalization factors) for each of document in
DocumentsList
    readingMapFileEuclideanLengths = new ReadingMapFile(mapUriEuclideanLengths);
    Map<String, Float> DocsEuclideanLengthsMap =
readingMapFileEuclideanLengths.getDocEuclideanLengths(DocsScoresMap.keySet());

    //divide the score of each document by its normalization factor

```

```

        for(String docID:DocsScoresMap.keySet()){
            Float score = DocsScoresMap.get(docID);
            score = score / (DocsEuclideanLengthsMap.get(docID)*queryNormalizationFactor);
            DocsScoresMap.put(docID,score );
        }

        //Sort the documents based on their scores.
        List<ReturnedDocument> DocumentsList = sortDocumentScores(DocsScoresMap);

        return DocumentsList;
    }
}

public long getTermDocFreq(JSONObject TermPostingList) throws JSONException{

    long TermDocFreq = (long)TermPostingList.getLong("DocFreq");

    return TermDocFreq;
}

public Map<String,Integer> calcTermQeuryFreqs(List<TermWritable> QueryTermsList) {

    Map<String,Integer> termFreqsMap = new TreeMap<String,Integer>();

    for(TermWritable queyTerm:QueryTermsList){

        if(termFreqsMap.containsKey(queyTerm.getTerm().toString())){
            Integer termFreq = termFreqsMap.get(queyTerm.getTerm().toString());
            termFreq++;
            termFreqsMap.put(queyTerm.getTerm().toString(), termFreq);
        }
        else{
            termFreqsMap.put(queyTerm.getTerm().toString(), new Integer(1));
        }
    }

    return termFreqsMap;
}

public List<ReturnedDocument> sortDocumentScores(Map<String, Float> DocsScoresMap ){
    List<ReturnedDocument> DocumentsList = new ArrayList<ReturnedDocument>();
    for(String docID:DocsScoresMap.keySet()){
        ReturnedDocument returnedDocument = new ReturnedDocument();
        returnedDocument.setDocID(docID);
        returnedDocument.setDocScore(DocsScoresMap.get(docID));
        DocumentsList.add(returnedDocument);
    }

    Collections.sort(DocumentsList, new ReturnedDocument());

    return DocumentsList;
}

}

package org.greenwich.searchEngine;
import java.util.*;
public class ReturnedDocument implements Comparator<ReturnedDocument>{

    private String docID;
    private Float docScore;

    public ReturnedDocument() {

    }
}

```

```

    public ReturnedDocument(String docID, Float docScore) {
        this.docID = docID;
        this.docScore = docScore;
    }

    public void setDocID(String docID){
        this.docID = docID;
    }
    public void setDocScore(Float docScore){
        this.docScore = docScore;
    }
    public String getDocID(){
        return docID;
    }
    public Float getDocScore(){
        return docScore;
    }
    public String toString(){
        return docID.toString() + ":" + docScore.toString();
    }
    public int compare(ReturnedDocument returnedDocument1,ReturnedDocument returnedDocument2){

        int cmp = returnedDocument1.docScore.compareTo(returnedDocument2.docScore);
        return cmp*-1;
    }

}

package org.greenwich.searchEngine;

import java.io.*;
import java.util.*;
import java.text.*;

import org.greenwich.analyzers.*;
import org.greenwich.invertedIndex.TermWritable;
import org.json.*;

public class SearchEngine {

    public static void main (String[] args) throws IOException, JSONException{

        String query =" duplicate MICR ";
        Date date = new Date();
        System.out.println(date.toString());

        long N = 19542; // number of documents in the collection
        QueryAnalyzer queryAnalyzer = new QueryAnalyzer(query);

        List<TermWritable> QueryTermsList = queryAnalyzer.analyzeQuery();

        Ranking ranking = new Ranking(N);
        // List<ReturnedDocument> returnedDocument = ranking.calCosineScore(QueryTermsList);
        List<ReturnedDocument> returnedDocument = ranking.calCosineScoreTFIDF(QueryTermsList);

        if (returnedDocument.size() == 0){
            System.out.println("No matching issues found.");
        }
        else {
            System.out.println(returnedDocument.size());
            for (ReturnedDocument x:returnedDocument.subList(0,5))
            {
                System.out.println(x);
            }
        }
    }
}

```

```
        }  
    }  
    Date date1 = new Date();  
    System.out.println(date1.toString());  
}  
  
}
```


5. package org.greenwich.termWeighting;

```
package org.greenwich.termWeighting;
```

```
public class TermWeighting {
```

```
    public TermWeighting(){
```

```
    }
```

```
    public Float calTermDocWeight(long N, int termFreq, long termDocFreq, int termLocationCount){
```

```
        Float weightTermFreq =(float) Math.log(termFreq+termLocationCount);
```

```
        Float weightInverseDocFreq = (float) Math.log(N+1/termDocFreq);
```

```
        return weightTermFreq*weightInverseDocFreq;
```

```
    }
```

```
    public Float calTermQueryWeight(long N, int termFreq, long termDocFreq){
```

```
        Float weightTermFreq = (float) Math.log(termFreq+3);
```

```
        Float weightInverseDocFreq = (float) Math.log(N+1/termDocFreq);
```

```
        return weightTermFreq*weightInverseDocFreq;
```

```
    }
```

```
}
```

6. package org.greenwich.tfidf;

```
package org.greenwich.tfidf;
```

```
import java.io.IOException;
```

```
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.SequenceFileInputFormat;
import org.greenwich.mapFile.MapFileFixer;
```

```
public class EuclideanLengthsDriver extends Configured implements Tool{
```

```
    public int run(String [] args) throws IOException, InterruptedException , Exception{

        /*if(args.length != 2) {
            System.err.printf("Usage:      %s      [generic-options]      <INPUT>      <OUTPUT>",
getClass().getSimpleName());
            ToolRunner.printGenericCommandUsage(System.err);
            return -1;
        }*/

        Configuration conf =getConf();
        conf.set("fs.default.name", "hdfs://localhost:8020");
        conf.set("fs.file.impl", "com.conga.services.hadoop.patch.HADOOP_7682.WinLocalFileSystem");
        conf.set("mapreduce.map.log.level", "ALL");

        Path path = new Path(args[1]);
        FileSystem fs = path.getFileSystem(conf);
        if (fs.exists(path)){
            fs.delete(path,true);
        }

        Job job =new Job(conf,"EuclideanLengthsJob");
        job.setJarByClass(getClass());

        job.setInputFormatClass(SequenceFileInputFormat.class);
        job.setOutputFormatClass(SequenceFileOutputFormat.class);

        job.setMapperClass(EuclideanLengthsMapper.class);
        job.setReducerClass(Reducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job, new Path("hdfs://localhost/user/admin/tfidf2OutFolder"));
        FileOutputFormat.setOutputPath(job, new Path("hdfs://localhost/user/admin/EuclideanLengthsOutFolder"));

        return job.waitForCompletion(true)? 0:1;
    }

    public static void main(String [] args) throws IOException, InterruptedException , Exception {
        int exitCode = ToolRunner.run(new EuclideanLengthsDriver(), args);
        MapFileFixer mapFileFixer = new MapFileFixer();
```

```

        mapFileFixer.convertSeqToMapFile(new
"hdofs://localhost/user/admin/EuclideanLengthsOutFolder" );
        System.exit(exitCode);

    }

}

package org.greenwich.tfddf;

import java.io.IOException;
import java.util.*;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.json.*;

public class EuclideanLengthsMapper extends Mapper<Text, Text, Text, Text>{

    //Key          Value
    //10098
    [{"term":"dear","termWeight":0},{"term":"delivered","termWeight":0.9030899869919435},{"term":"dispatched","termW
eight":0.4259687452152651}]

    public void map(Text key, Text value, Context context)throws IOException, InterruptedException {

        //
        try{
            JSONObject docVector = new JSONObject(value.toString());
            Iterator<String> terms = docVector.keys();
            Float docNormalizationFactor= new Float(0);
            while (terms.hasNext()) {

                Float termWeight =(float) docVector.getDouble(terms.next());
                docNormalizationFactor += (float) Math.pow(termWeight, 2);

            }

            docNormalizationFactor= (float) Math.sqrt(docNormalizationFactor);
            context.write(key, new Text(docNormalizationFactor.toString()));

        }catch(JSONException e) {
            System.err.println(e);
        }
    }

}

package org.greenwich.tfddf;

import java.io.IOException;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;

public class tfddfDriver extends Configured implements Tool{

```

```

    public int run(String [] args) throws IOException, InterruptedException , Exception{

        if(args.length != 2) {
            System.err.printf("Usage:      %s      [generic-options]      <INPUT>      <OUTPUT>",
getClass().getSimpleName());
            ToolRunner.printGenericCommandUsage(System.err);
            return -1;
        }

        Configuration conf =getConf();
        // conf.set("fs.default.name", "hdfs://localhost:8020");
        //conf.set("fs.file.impl", "com.conga.services.hadoop.patch.HADOOP_7682.WinLocalFileSystem");
        //conf.set("mapreduce.map.log.level", "ALL");

        Path path = new Path(args[1]);
        FileSystem fs = path.getFileSystem(conf);
        if (fs.exists(path)){
            fs.delete(path,true);
        }

        Job job =new Job(conf,"tfidf");
        job.setJarByClass(getClass());

        job.setInputFormatClass(KeyValueTextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        job.setMapperClass(tfidfMapper.class);
        job.setReducerClass(tfidfReducer.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        return job.waitForCompletion(true)? 0:1;
    }

    public static void main(String [] args) throws  IOException, InterruptedException , Exception {
        int exitCode = ToolRunner.run(new tfidfDriver(), args);
        System.exit(exitCode);
    }
}

package org.greenwich.tfidf;

import java.io.IOException;

import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;

public class tfidfDriver2 extends Configured implements Tool{

```

```

        public int run(String [] args) throws IOException, InterruptedException , Exception{

            if(args.length != 2) {
                System.err.printf("Usage:          %s          [generic-options]          <INPUT>          <OUTPUT>",
getClass().getSimpleName());
                ToolRunner.printGenericCommandUsage(System.err);
                return -1;
            }

            Configuration conf =getConf();
            // conf.set("fs.default.name", "hdfs://localhost:8020");
            //      conf.set("fs.file.impl", "com.conga.services.hadoop.patch.HADOOP_7682.WinLocalFileSystem");
            //      conf.set("mapreduce.map.log.level", "ALL");

            Path path = new Path(args[1]);
            FileSystem fs = path.getFileSystem(conf);
            if (fs.exists(path)){
                fs.delete(path,true);
            }

            Job job =new Job(conf,"tfidf2");
            job.setJarByClass(getClass());

            job.setInputFormatClass(KeyValueTextInputFormat.class);
            job.setOutputFormatClass(TextOutputFormat.class);

            job.setMapperClass(tfidfMapper2.class);
            job.setReducerClass(Reducer.class);

            job.setMapOutputKeyClass(Text.class);
            job.setMapOutputValueClass(Text.class);

            job.setOutputKeyClass(Text.class);
            job.setOutputValueClass(Text.class);

            FileInputFormat.addInputPath(job, new Path(args[0]));
            FileOutputFormat.setOutputPath(job, new Path(args[1]));

            return job.waitForCompletion(true)? 0:1;
        }

        public static void main(String [] args) throws  IOException, InterruptedException , Exception {
            int exitCode = ToolRunner.run(new tfidfDriver2(), args);
            System.exit(exitCode);
        }
    }

    package org.greenwich.tfidf;

    import java.io.IOException;
    import org.apache.hadoop.io.Text;
    import org.apache.hadoop.mapreduce.Mapper;
    import org.json.*;

    public class tfidfMapper extends Mapper<Text, Text, Text, Text>{

        public void map(Text key, Text value, Context context)throws IOException, InterruptedException {

            try {

                //calculate collection Frequency for a term
                JSONObject PostingLists = new JSONObject(value.toString());

```

```

JSONArray arrayPostingLists = PostingLists.getJSONArray("PostingLists");
long colFreq = 0;
for ( int i=0; i <arrayPostingLists.length(); i++) {

    JSONObject DocObj = arrayPostingLists.getJSONObject(i);
    colFreq += DocObj.getInt("termFreq");

}

// create JSON Object that contains the following term characteristic per term per document
{"term":term1,"colFreq":4,"DocFreq":2,"termFreq":2} a

for ( int i=0; i <arrayPostingLists.length(); i++) {

    JSONObject obj = arrayPostingLists.getJSONObject(i);

    JSONObject objValue = new JSONObject();
    objValue.put("term", key.toString());
    objValue.put("colFreq", colFreq);
    objValue.put("DocFreq", PostingLists.getInt("DocFreq"));
    objValue.put("termFreq", obj.getInt("termFreq"));

    //count term Significance in a document through its location, terms appear in the ticket's summary
    are given 3, description 2 and comments 1
    JSONArray TermPositionsArray = obj.getJSONArray("TermPositions");
    int termLocationCount=0;
    for(int j=0; j<TermPositionsArray.length(); j++) {
        JSONObject TermPosition =TermPositionsArray.getJSONObject(j);
        termLocationCount += TermPosition.getInt("termLocation");
    }
    objValue.put("termLocationCount", termLocationCount);

    context.write(new Text(obj.getString("docID")), new Text(objValue.toString()));

}
} catch (JSONException e) {
    System.err.println("Caught IOException: " + e.getMessage());
}
}

}

package org.greenwich.tfidf;

import java.io.IOException;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.json.*;

public class tfidfMapper2 extends Mapper<Text, Text, Text, Text>{

    public void map(Text key, Text value, Context context)throws IOException, InterruptedException {

        // Key          Value
        //10051
        [{"DocFreq":1,"colFreq":1,"term":"tll","termFreq":1},{ "DocFreq":8,"colFreq":19,"term":"email","termFreq":2},{ "
DocFreq":8,"colFreq":17,"term":"ecc","termFreq":1},{ "DocFreq":1,"colFreq":1,"term":"done","termFreq":1},{ "DocFre
q":8,"colFreq":21,"term":"dear","termFreq":1},{ "DocFreq":8,"colFreq":19,"term":"team","termFreq":2},{ "DocFreq":8,
"colFreq":15,"term":"support","termFreq":2},{ "DocFreq":3,"colFreq":3,"term":"data","termFreq":1},{ "DocFreq":8,"col
Freq":18,"term":"customer.pl","termFreq":2},{ "DocFreq":6,"colFreq":9,"term":"confirm","termFreq":1},{ "DocFreq":1,
"colFreq":1,"term":"conducting","termFreq":1},{ "DocFreq":4,"colFreq":8,"term":"closed","termFreq":1},{ "DocFreq":5,
"colFreq":6,"term":"believe","termFreq":1},{ "DocFreq":1,"colFreq":1,"term":"stuff","termFreq":1},{ "DocFreq":5,"colF
req":7,"term":"bank","termFreq":1},{ "DocFreq":2,"colFreq":2,"term":"specific","termFreq":1},{ "DocFreq":1,"colFreq"

```

```
:1,"term":"assiged","termFreq":1},{ "DocFreq":1,"colFreq":1,"term":"2007041710000019","termFreq":1},{ "DocFreq":1,"colFreq":1,"term":"1","termFreq":1},{ "DocFreq":1,"colFreq":1,"term":"sheet","termFreq":1},{ "DocFreq":1,"colFreq":2,"term":"session","termFreq":2},{ "DocFreq":1,"colFreq":1,"term":"result","termFreq":1},{ "DocFreq":5,"colFreq":11,"term":"resolved","termFreq":2},{ "DocFreq":6,"colFreq":10,"term":"resolution","termFreq":1},{ "DocFreq":1,"colFreq":1,"term":"request","termFreq":1},{ "DocFreq":2,"colFreq":3,"term":"reports","termFreq":1},{ "DocFreq":1,"colFreq":1,"term":"00","termFreq":1},{ "DocFreq":4,"colFreq":6,"term":"report","termFreq":2},{ "DocFreq":2,"colFreq":2,"term":"rami","termFreq":1},{ "DocFreq":4,"colFreq":6,"term":"yanal","termFreq":1},{ "DocFreq":5,"colFreq":7,"term":"progresssoft","termFreq":1},{ "DocFreq":6,"colFreq":20,"term":"problem","termFreq":1},{ "DocFreq":8,"colFreq":14,"term":"please","termFreq":1},{ "DocFreq":2,"colFreq":2,"term":"pages","termFreq":1},{ "DocFreq":8,"colFreq":20,"term":"web","termFreq":2},{ "DocFreq":1,"colFreq":1,"term":"17/04/2007","termFreq":1},{ "DocFreq":2,"colFreq":3,"term":"about","termFreq":1},{ "DocFreq":8,"colFreq":18,"term":"194.165.134.179/otrs","termFreq":2},{ "DocFreq":4,"colFreq":5,"term":"above","termFreq":1},{ "DocFreq":1,"colFreq":2,"term":"0478","termFreq":2},{ "DocFreq":2,"colFreq":5,"term":"page","termFreq":1},{ "DocFreq":1,"colFreq":1,"term":"orientation","termFreq":1},{ "DocFreq":4,"colFreq":7,"term":"now","termFreq":1},{ "DocFreq":7,"colFreq":8,"term":"we","termFreq":1},{ "DocFreq":1,"colFreq":1,"term":"no.478","termFreq":1},{ "DocFreq":3,"colFreq":3,"term":"need","termFreq":1},{ "DocFreq":2,"colFreq":4,"term":"mohammad","termFreq":2},{ "DocFreq":1,"colFreq":1,"term":"kilani","termFreq":1},{ "DocFreq":2,"colFreq":2,"term":"karram","termFreq":1},{ "DocFreq":8,"colFreq":24,"term":"issue","termFreq":1},{ "DocFreq":2,"colFreq":3,"term":"information","termFreq":1},{ "DocFreq":8,"colFreq":18,"term":"http","termFreq":2},{ "DocFreq":8,"colFreq":18,"term":"helpdesk@progresssoft.com","termFreq":2},{ "DocFreq":7,"colFreq":9,"term":"helpdesk","termFreq":1},{ "DocFreq":4,"colFreq":6,"term":"hadad","termFreq":1},{ "DocFreq":1,"colFreq":1,"term":"alkaram","termFreq":1},{ "DocFreq":1,"colFreq":3,"term":"visit","termFreq":3},{ "DocFreq":2,"colFreq":3,"term":"generating","termFreq":1},{ "DocFreq":6,"colFreq":17,"term":"from","termFreq":1},{ "DocFreq":1,"colFreq":3,"term":"form","termFreq":3},{ "DocFreq":8,"colFreq":8,"term":"etsm","termFreq":1},{ "DocFreq":8,"colFreq":8,"term":"etds","termFreq":1}]
```

```
try {

    JSONArray PostingEntry =new JSONArray(value.toString());
    JSONArray TermsWeightsArray =new JSONArray();
    JSONObject TermWeight = new JSONObject();
    long docTerms = PostingEntry.length();           //number of terms in document
    long N=19542;                                     // the
total number of documents in a collection

    for (int i= 0; i<PostingEntry.length(); i++)
    {
        String      term;
        long        colFreq;                          // the collection frequency for this
term
        long        DocFreq;                          // the number of
documents in the collection that contain the term
        long        termFreq;                        // number of times the term appears in this
document
        long        termLocationCount;
        float       tf;
        float       idf;
        float       termWeight;
        // a composite weight for each term in each document
        JSONObject Term =PostingEntry.getJSONObject(i);

        term= Term.getString("term");
        colFreq= Term.getLong("colFreq");
        termFreq= Term.getLong("termFreq");
        DocFreq= Term.getLong("DocFreq");
        termLocationCount = Term.getLong("termLocationCount");

        tf= (float) Math.log(termFreq+termLocationCount);

        // term frequency
        idf =(float) Math.log10(N+1/DocFreq);
        //inverse document frequency
        termWeight = tf*idf;

        TermWeight.put(term, termWeight);
    }

    context.write(key,new Text(TermWeight.toString()));

} catch (JSONException e){

    System.err.println("Caught IOException: " + e.getMessage());

}
```

```

    }

}

package org.greenwich.tfidf;

import java.io.IOException;
import java.util.*;
import java.util.regex.Pattern;

import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.SortableWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import org.json.*;

public class tfidfReducer extends Reducer<Text, Text, Text, Text> {

    public void reduce(Text key , Iterable<Text> values,Context context) throws IOException,InterruptedException {
        JSONArray array =new JSONArray();

        try {
            for (Text value:values) {

                JSONObject obj = new JSONObject(value.toString());
                array.put(obj);

            }
            context.write(key, new Text(array.toString()));

        }
        catch (JSONException e) {
            System.err.println("Exception " +e );
        }

    }
}

```