ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/306400561

SinMin - Sinhala Corpus Project

Technical Report · February 2015

DOI: 10.13140/RG.2.2.19037.41443

CITATIONS
0

READS
5,029

4 authors, including:

Chamila Wijayarathna
UNSW - Canberra
12 PUBLICATIONS **50** CITATIONS
SEE PROFILE

Dimuthu Upeksha
University of Moratuwa
4 PUBLICATIONS **17** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Sinhala Natural Language Processing View project

Improving the usability of security APIs View project

All content following this page was uploaded by Chamila Wijayarathna on 24 August 2016.

The user has requested enhancement of the downloaded file.

UNIVERSITY OF MORATUWA

Faculty of Engineering

SinMin - Sinhala Corpus Project

Project Members:

100552T	Upeksha W.D.
100596F	Wijayarathna D.G.C.D
100512X	Siriwardena M. P.
100295G	Lasandun K.H.L.

Supervisor:
Dr. Chinthana Wimalasuriya
Department of Computer Science and Engineering,
University of Moratuwa,
Sri Lanka

Co-supervisors:
Prof. Gihan Dias
Department of Computer Science and Engineering,
University of Moratuwa,
Sri Lanka

Mr. Nisansa De Silva
Department of Computer Science and Engineering,
University of Moratuwa,
Sri Lanka

Abstract

Today, the corpus based approach can be identified as the state of the art methodology in language learning studying for both prominent and less known languages in the world. The corpus based approach mines new knowledge on a language by answering two main questions:

- What particular patterns are associated with lexical or grammatical features of the language?
- How do these patterns differ within varieties and registers?

A language corpus can be identified as a collection of authentic texts that are stored electronically. It contains different language patterns in different genres, time periods and social variants. Most of the major languages in the world have their own corpora. But corpora which have been implemented for Sinhala language have so many limitations.

SinMin is a corpus for Sinhala language which is

- Continuously updating
- Dynamic (Scalable)
- Covers wide range of language (Structured and unstructured)
- Providing a better interface for users to interact with the corpus

This report contains the comprehensive literature review done and the research, and design and implementation details of the SinMin corpus. The implementation details are organized according to the various components of the platform. Testing, and future works have been discussed towards the end of this report.

Acknowledgement

Upon the successful completion of this project, members of the SinMin team would like to acknowledge every person who guided, assisted, encouraged and helped SinMin in numerous ways with greatest gratitude and sincerity.

Our heartfelt gratitude goes to Dr. Chinthana Wimalasuriya, the supervisor of the project, for giving us the guidance, courage, confidence, and support in numerous ways throughout the project. The successful completion of SinMin with achieved deliverables and goals would not have been possible without his assistance and motivation.

We would also like to thank Prof. Gihan Dias and Mr. Nisansa Dilushan De Silva, the co-supervisors of our project for all the advice, help and courage given to us throughout the project.

SinMin is not an idea that just came up as a final year project idea this year. It was started sometimes back and at the time we started the project some of the components were already there. We would like to thank Mr. Nisansa Dilushan De Silva and Mr. Adeesha Wijesiri for initializing this project idea and all the team including them for all the work they did on this project.

We would like to express our sincere gratitude to Dr. Ajith Pasqual, the Head of Department of Electronic & Telecommunication Engineering, and all the members of CITeS (Center for Information Technology Services) who supported to provide us with infrastructure facilities for the project.

We would also like to thank Dr. Malaka Walpola, course coordinator of Final Year Project, for his guidance and support throughout the project.
We are thankful to absolutely everyone who assisted us in every possible means to make SinMin a success.

Table of Content

Table of Figures

Abbreviations

ANC	:	American National Corpus
API	:	Application Programming Interface
ARTFL	:	American and French Research on the Treasury of the French Language
BAM	:	Business Activity Monitor
BNC	:	British National Corpus
BYU	:	Brigham Young University
CDIF	:	Corpus Data Interchange Format
COCA	:	Corpus for Contemporary American National English
CORDE	:	Corpus Diacrónico del Español
CREA	:	Corpus de Referencia del Español Actual
CSV	:	Comma Separated Values
DBCA	:	Database Configuration Assistant
DOM	:	Document Object Model
GUI	:	Graphical User Interface
HTML	:	Hypertext Markup Language
HTTP	:	Hypertext Transfer Protocol
IP	:	Internet Protocol
JAX-RS	:	Java API for XML Restful Web Services
JDBC	:	Java Data Base Connectivity
JSON	:	JavaScript Object Notation
KWIC	:	Keyword in Context
LDAP	:	Lightweight Directory Access Protocol
NLP	:	Natural Language Protocol
NoSQL	:	Not only Structured Query Language
OCR	:	Optical Character Recognition
OS	:	Operating System
OUI	:	Oracle Universal Installer
PL/SQL	:	Procedural Language/Structured Query Language
POS	:	Part of Speech
REST	:	Representational State Transfer
RSS	:	Rich Site Summary

SGML	:	Standard Generalized Mark Up Language
SQL	:	Structured Query Language
SSH	:	Secure SHell
TCP	:	Transmission Control Protocol
UCSC	:	University of Colombo School of Computing
URL	:	Uniform Resource Locator
VCS	:	Version Controlling System
XML	:	Extensible Markup Language

1.0 Introduction

1.1 Overview

A language corpus can be identified as a collection of authentic texts that are stored electronically. It contains different language patterns in different genres, time periods and social variants. The quality of a language corpus depends on the area that has been covered by it. Corpora which cover a wide range of language can be used to discover information about a language that may not have been noticed through intuition alone. This enables us to see a language from a different point of view. Corpus linguistics tries to study the language through corpora to answer two fundamental questions: what particular patterns are associated with lexical or grammatical features of a language and how do these patterns differ within varieties of context.

Most of the major languages in the world have their own corpora. Some languages like English have separate specialized corpora for different types of research work. For Sinhala language there have been a few attempts to develop a corpus and most of them mainly focused on extracting data from Sinhala newspapers since e Sinhala newspapers are easily found and crawled. However we found that those corpora for Sinhala language have the following drawbacks.

- Lack of data sources (most of them were from newspapers)
- Not keeping sufficient meta data
- Only contain old data, not updating with new resources
- Data is stored as raw text files that are less usable in analyzing
- Not having a proper interface where outsiders can make use of them (API or WEB interface)

1.2 SinMin - A corpus for Sinhala Language

Unlike English, the Sinhala language comprises different variations of the language because it has been propagated and developed for more than 2000 years. Sinhala language is handled mainly in two ways while speaking and writing. Those speaking and writing patterns also differ according to time period, region, caste and other contextual parameters. It will be quite interesting to mine similarities and differences that appear in those patterns and identify trends and developments happening in the Sinhala language.

In this project we design and implement a corpus for Sinhala language which is

- Continuously updating
- Dynamic (Scalable)
- Covers a wide range of language (structured and unstructured)
- Provides a better interface for users to interact with the corpus

Instead of sticking to one language source like newspapers, we use all possible kinds of available Sinhala digital data sources like blogs, eBooks, wiki pages, etc. Separate crawlers and parsers for each language source are developed and controlled using a separate centralized management server. The corpus is designed so that it will be automatically updated with new data sources added to the internet (latest newspaper articles or blog feed). This keeps the corpus up to date with the latest data sources.

Keeping data only as raw files would reduce the usability because there is only a limited number of operations that can be performed on them. So, instead of keeping them in raw text format, we store them in a structured manner in databases in order to provide support for searching and updating data. The selection of a proper database system is done through performance and requirement testing on currently available database systems.

A Web based interface is designed with rich data visualization tools, so that anyone can use this corpus to find details of the language. At the same time, features of the corpus are exposed through an Application Programming Interface. So any third party who wishes to consume services of SinMin can directly access to them through it.

2.0 Literature Review

2.1 Introduction to Corpus Linguistics and What is a Corpus

Before going into more inside details, we will first see what corpus linguistics is and what a corpus is.

Corpus linguistics approaches the study of language in use through corpora (singular: corpus). In short, corpus linguistics serves to answer two fundamental research questions:

- What particular patterns are associated with lexical or grammatical features?
- How do these patterns differ within varieties and registers?

In 1991, John Sinclair, in his book "Corpus Concordance Collocation" stated that a word in and of itself does not carry meaning, but that meaning is often made through several words in a sequence. This is the idea that forms the backbone of corpus linguistics.

It's important to not only understand what corpus linguistics is, but also what corpus linguistics is not. Corpus linguistics is not:

- Able to provide negative evidence - corpus can't tell us what is possible or correct or not possible or incorrect in language; it can only tell us what is or is not present in the corpus.
- Able to explain why
- Able to provide all possible languages at one time.

Broadly, corpus linguistics looks to see what patterns are associated with lexical and grammatical features. Searching corpora provides answers to questions like these:

- What are the most frequent words and phrases in English?
- What are the differences between spoken and written English?
- What tenses do people use most frequently?
- What prepositions follow particular verbs?
- How do people use words like can, may, and might?
- Which words are used in more formal situations and which are used in more informal ones?
- How often do people use idiomatic expressions?
- How many words must a learner know to participate in everyday conversation?

- How many different words do native speakers generally use in conversation? [19]

The Corpus Approach for linguistic study [7] is comprised of four major characteristics:

- It is empirical, analyzing the actual patterns of language use in natural texts.
- It utilizes a large and principled collection of natural texts as the basis for analysis.
- It makes extensive use of computers for analysis. - Not only do computers hold corpora, they help analyze the language in a corpus.
- It depends on both quantitative and qualitative analytical techniques.

Nowadays most of the languages implement corpora for their languages and research on extracting linguistic features from them.

A corpus is a principled collection of authentic texts stored electronically that can be used to discover information about language that may not have been noticed through intuition alone. Strictly speaking, any collection of texts can be called a corpus, but normally other conditions are required for a bunch of texts to be considered a corpus: it must be a 'big' collection of 'real' language samples, collected in accordance with certain 'criteria' and 'linguistically' tagged.

There are mainly eight kind of corpuses. They are generalized corpuses, specialized corpuses, learner corpuses, pedagogic corpuses, historical corpuses, parallel corpuses, comparable corpuses, and monitor corpuses. Which type of corpora to be used is depend on the purpose for the corpus.

The broadest type of corpus is a generalized corpus. Generalized corpora are often very large, more than 10 million words, and contain a variety of language so that findings from it may be somewhat generalized. The British National Corpus (BNC) and the American National Corpus (ANC) are examples of large, generalized corpora.

A specialized corpus contains texts of a certain type and aims to be representative of the language of this type. A learner corpus is a kind of specialized corpus that contains written texts and/or spoken transcripts of language used by students who are currently acquiring the language [5].

Since our goal is to cover all types on Sinhala language, ‘SinMin’: the corpus we are developing will be a Generalized corpus.

2.2 Usages of a Corpus

In above section 2.0, we have briefly described how a corpus can be used in language study.

Other than that, there are many more usages of a corpus. Some of them are

1. Implementing translators, spell checkers and grammar checkers.
2. Identifying lexical and grammatical features of a language.
3. Identifying varieties of language of context of usage and time.
4. Retrieving statistical details of a language.
5. Providing backend support for tools like OCR, POS Tagger, etc.

The corpus-based approach to study translation has become popular over the last decade or two, with a wealth of data now emerging from studies using parallel corpora, multilingual corpora and comparable corpora [37]. The use of computer-based bilingual corpora can enhance the speed of translation as well as its quality, for they enable more native-like interpretations and strategies in source and target texts respectively [2]. Parallel corpus is a valuable resource for cross-language information retrieval and data-driven natural language processing systems, especially for Statistical Machine Translation (SMT) [34]. The google translator, the most widely used translator nowadays also uses parallel corpuses [16]. Many other translators also use parallel corpuses for translation [22].

Not only translators, but also spell checkers and grammar checkers heavily depend on corpuses. By using corpus as the collection of correct words and sentences , spell checkers and grammar checkers will show suggestions when a user entered something that is not consistent with the corpus. A few examples for spell checkers can be found at [25], [36], [29], and some grammar checkers that run on top of corpuses can be found at [23], [26], [27].

In a language study, questions about what forms are more common, what examples will best exemplify naturally occurring language, and what words are most frequent with grammatical structures, are not easy to answer using common teaching methodologies. Answers to these kind of questions in recent years have been coming from researches that use the tools and techniques of corpus linguistics to describe English grammar [6]. [32] describes many scenarios where corpuses has been used in language study for extracting lexical and grammatical features. Two of the major usages of a language corpus are identifying varieties of language of context of usage and time and retrieving statistical details of a language. The best example for a corpus which gives this functionality is Google's N-gram viewer [17] . It lets users to discover many statistical details about language use like comparison of word usage over time and context, frequently used words, bigrams, trigrams, etc.,. Other than Google N-gram viewer, many other popular corpuses like British National Corpus (BNC) [8], Corpus of Contemporary American English (COCA) [9] also facilitate users to search through many statistical details of a language.

Other than the major usages mentioned above, corpuses can be used to provide backend support for tools like Optical Character Recognition (OCR) and Part of Speech Taggers. We can use a corpus to increase the accuracy of an OCR tool by predicting best fitting character for a unclear symbol. Further, when creating automated POS taggers, we can use a corpus for generating rules and relationships in the POS Tagger. [20] describes a POS Tagger model for Sinhala language which uses annotated corpus for generating rules. It is based on statistical based approach, in which the tagging process is done by computing the tag sequence probability and the word-likelihood probability from the given corpus, where the linguistic knowledge is automatically extracted from the annotated corpus.

2.3 Existing Corpus Implementations

Currently many of the languages have corpora for them. One of the most popular corpora in the world is British National Corpus which contains about 100 million words [8],[3]. BNC is not the only corpus existing for the English Language. Some other corpuses implemented for English Language are COCA ([9], [15]) the Brown Corpus, Corpus for Spoken Professional American English, etc.

Not only English, most of the popular languages such as Japanese [24], Spanish, German [28], Tamil [30] as well as unpopular languages like Turkish [1], Thai (ORCHID: Thai Part-Of-Speech Tagged Corpus), Tatar [11] have corpuses implemented for those languages.

There is an implemented corpus for the Sinhala language which is known as UCSC Text Corpus of Contemporary Sinhala consisting of 10 million words, but it covers very few Sinhala sources and it is not updated

2.4 Identifying Sinhala Sources and Crawling

One of the most important steps of creating a corpus is selecting sources to add to the corpus.

2.4.1 Composition of the corpus

There were a few opinions expressed about selecting sources for a corpus.

A corpus is a principled collection of authentic texts stored electronically. When creating a corpus, there must be a focus on three factors: the corpus must be principled, it must use authentic texts and it must have the ability to be stored electronically. A corpus should be principled, meaning that the language comprising the corpus cannot be random but chosen according to specific characteristics. Having a principled corpus is especially important for more narrow investigations; for example, if you want your students to look at the use of signal words in academic speech, then it is important that the corpus used is comprised of only academic speech. A principled corpus is also necessary for larger, more general corpora, especially in instances where users may want to make generalizations based on their findings.

Authentic texts are defined as those that are used for a genuine communicative purpose. The main idea behind the authenticity of the corpus is that the language it contains is not made up for the sole purpose of creating the corpus [5].

In this context, we think it is useful to look at composition of some of the popular corpora. The COCA [15] contains more than 385 million words from 1990–2008 (20 million words each year), balanced between spoken, fiction, popular magazines, newspapers, and academic journals. This was developed as an alternative to American National Corpus. COCA addresses issues that were available in ANC in its content. In ANC there were only two magazines in the corpus, one newspaper, two academic journals, and the fiction texts represent only about half a million words of text from a total of 22 million words. Certain genres seem to be overrepresented. For example, nearly fifteen percent of the corpus comes from one single blog, which deals primarily with the teen movie 'Buffy the Vampire Slayer'. The COCA which was the solution for above, contains more than 385 million words of American English from 1990 to 2008. There are 20 million words for each of these nineteen years, and 20 million words will be added to the corpus each year from this point onwards . In addition, for each year the corpus is evenly divided between spoken language , fiction, popular magazines, newspapers, and academic journals. We can look at the composition of the corpus in a more detailed manner. Approximately 10% of the texts come from spoken langage , 16% from fiction, 15% from (popular) magazines, 10% from newspapers, and 15% from academic, with the balance coming from other genres. In the COCA, texts are evenly divided between spoken language (20%), fiction (20%), popular magazines (20%), newspapers (20%) and academic journals (20%). This composition holds for the corpus overall, as well as for each year in the corpus. Having this balance in the corpus allows users to compare data diachronically across the corpus, and be reasonably sure that the equivalent text composition from year to year will accurately show changes in the language. Spoken texts, were based almost entirely on transcripts of unscripted conversation on television and radio programs.

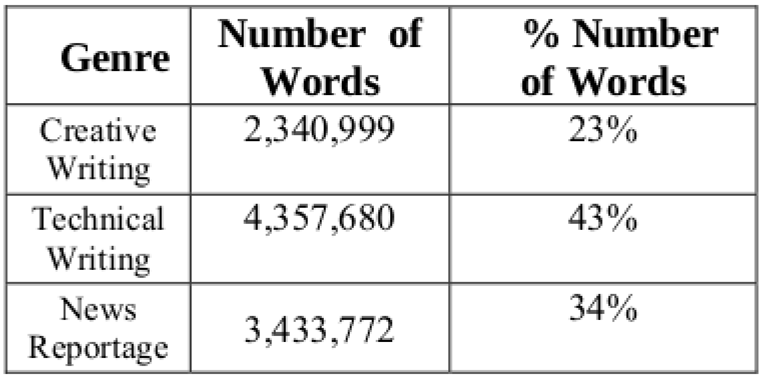
BNC and other corpora available at [13] also adhere to the most of the above mentioned facts such as balance between genres.

Not only the corpora listed there, many different corpora agree with the importance of a balanced corpus. For example let’s consider the following quote from [1]. “The major issue that should be addressed in design of TNC is its representativeness. Representativeness refers to the extent to which a sample includes the full range of variability in a population. In other words, representativeness can be achieved through balance and sampling of language or language variety presented in a corpus. A balanced general corpus contains texts from a wide range of genres, and text chunks for each genre are sampled proportionally for the inclusion in a corpus.”

There are few corpuses where balancing between genres are not considered. For Thai Linguistically Annotated Corpus for Language Processing contains 2,720 articles (1,043,471words) from the entertainment and lifestyle (NE&L) domain and 5,489 articles (3,181,487 words) in the news (NEWS) domain.

[35] describes the composition of UCSC Text Corpus of Contemporary Sinhala. Table 1 shows the number of words each genre has.

Table 1: Distribution of Corpus Text across Genres in UCSC Text Corpus of Contemporary Sinhala [35].



Genre	Number of Words	% Number of Words
Creative Writing	2,340,999	23%
Technical Writing	4,357,680	43%
News Reportage	3,433,772	34%

For constructing SinMin, we have identified 5 main genres in Sinhala sources. Table 2 shows the identified categories and sources belonging to each category.

Table 2: Expected Composition of sources in SinMin Corpus

News	**Academic**	**Creative Writing**	**Spoken**	**Gazette**
News Paper	Text books	Fiction	Subtitle	Gazette
News Items	Religious	Blogs		
	Sinhala Wikipedia	Magazine		

One of the challenges in creating a balanced corpus for the Sinhala language is that the amount of sources available for 'News' category is relatively very large while for 'Academic' and 'Spoken' it can be very much less.

For Sinhala language, the main source of spoken language we could think of is subtitles. So we have identified [4] as the largest source for Sinhala subtitles and we use subtitles available there as the main source of spoken language.

2.4.2 Crawling language sources

Many existing corpus implementations have discussed about various ways of crawling online sources to the corpus. Here we are only considering about online sources because we are only interested in them. Even though they speak about collecting data for their corresponding languages and some of them may not suit our work on creating a corpus for Sinhala language, we have presented them below since they are relevant to the subject.

[5]suggests that one way to gather principled, authentic texts that can be stored electronically is through Internet "alerts." Search engines such as Yahoo and Google gather email updates of the latest relevant results based on a topic or specific query generated by the user. It Also says that another means of gathering principled, authentic texts that can be stored electronically is looking at internet essay sites. Many of the academic essay sites have a disclaimer that their essays should be used for research purposes only, and should not to be downloaded or turned in as one's own work. These sites can be very helpful for creating corpora specific for academic writing with term papers, essays, and reports on subjects such as business, literature, art, history, and science.

[15] describes procedures the researchers used for collecting sources for creating COCA. “While some of the materials were retrieved manually, others were retrieved automatically. Using VB.NET (a programming interface and language), we created a script that would check our database to see what sources to query (a particular magazine, academic journal, newspaper, TV transcript, etc) and how many words we needed from that source for a given year. The script then sent this information to Internet Explorer, which would enter that information into the search form at the text archive, check to see if we already had the articles that would be retrieved by the query, and (if not) then retrieve the new article(s). In so doing, it would store all of the relevant bibliographic information (publication data, title, author, number of words, etc.) in the database. It would continue this process until it reached the desired number of words for a particular source in a particular year.”

For SinMin like in COCA, we implemented crawlers using Java which get each online resource using HTTP connection and parse received HTML files to get required articles and their Meta data.

2.5 Data Storage and Information Retrieval from Corpus

After crawling online sources, corpus should store them in a way so that the information can be used efficiently when required. One of the main objectives of this project is to identify a good database tool where data can be efficiently inserted and information can be efficiently retrieved. In this section we will look at storage mechanisms used by existing corpora and some of the mechanisms we are trying in this project. Other than currently used storage mechanisms we will test the usage of some NoSQL techniques as data storage mechanism in corpus.

2.5.1 Data storage models in existing corpora

When we study existing corpus implementations, we can see two commonly used storage mechanisms.

First let us consider the architecture that has been used in BNC. In BNC data is stored as XML like files which follow a scheme known as the Corpus Data Interchange Format (CDIF)[33]. The purpose of CDIF is to allow the portability of corpora across different types of hardware and software environments. Thje same kind of storage mechanism has been used in other large 100+ million word corpora, such as ARTFL , and CREA and CORDE from the Real Academia Española [31] [14] . This is designed to capture an extensive variety of information. This supports to store a great deal of detail about the structure of each text, that is, its division into sections or chapters, paragraphs, verse lines, headings, etc. for written text, or into speaker turns, conversations, etc. for spoken texts.

In this storage mechanism contextual information common to all texts is described in an initial corpus header. Contextual information specific to a given text is listed in a text header which precedes each text. Detailed structural and descriptive information is marked at appropriate positions within each text. Following text from [3] describes the how details about texts are stored in BNC.

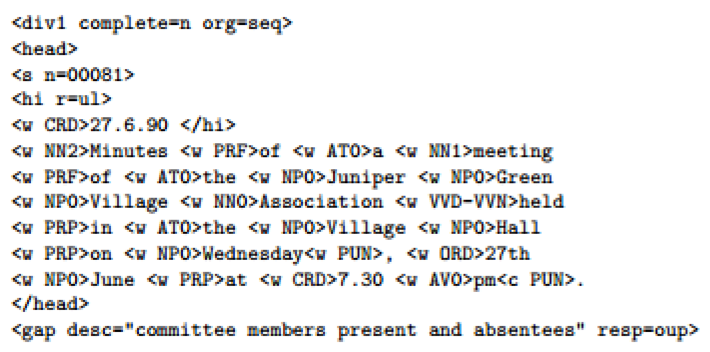
“CDIF uses an international standard known as SGML (ISO 8879: Standard Generalized Mark Up Language), now very widely used in the electronic publishing and information retrieval communities. In SGML, electronic texts are regarded as consisting of named elements, which may bear descriptive attributes and can be combined according to a simple grammar, known as a document type definition. In an SGML document, element occurrences are delimited by the use of tags. There are two forms of tag, a start-tag, marking the beginning of an element, and an end-tag marking its end. Tags are delimited by the characters < and >, and contain the name of the element, preceded by a solidus (/) in the case of an end-tag. For example, a heading or title in a written text will be preceded by a tag of the form <HEAD> and followed by a tag in the form </HEAD>. Everything between these two tags is regarded as the content of an element of type <HEAD>.

End-tags are omitted for the elements <s>, <w> and <c> (i.e., for sentences, words, and punctuation). For all other non-empty elements, every occurrence in the corpus has both a start-tag and an end-tag. In addition, attribute names are omitted for the elements <w> and <c> to save space.”

Figure 1 shows a sample text annotated to store in BNC.

Details about elements used in BNC are available at [10].

COCA uses an architecture based on extensive use of relational databases. [15] describes the architecture used in COCA as follows.



```
<div1 complete=n org=seq>
<head>
<s n=00081>
<hi r=ul>
<w CRD>27.6.90 </hi>
<w NN2>Minutes <w PRF>of <w AT0>a <w NN1>meeting
<w PRF>of <w AT0>the <w NP0>Juniper <w NP0>Green
<w NP0>Village <w NN0>Association <w VVD-VVN>held
<w PRP>in <w AT0>the <w NP0>Village <w NP0>Hall
<w PRP>on <w NP0>Wednesday<w PUN>, <w ORD>27th
<w NP0>June <w PRP>at <w CRD>7.30 <w AV0>pm<c PUN>.
</head>
<gap desc="committee members present and absentees" resp=oup>
```

Figure 1: Sample annotated text in BNC

“The main [seqWords] database contains a table with one row for each token in the corpus in sequential order (i.e.385+ million rows for a 385+ million word corpus, such as COCA). The table contains an [ID] column that shows the sequential position of each word in the corpus (1, 2, 3, ... 385,000,000), a [wordID] column with the integer value for each unique type in the corpus (wordID), and a [textID] number that refers to one of the 150,000+ texts in the corpus.

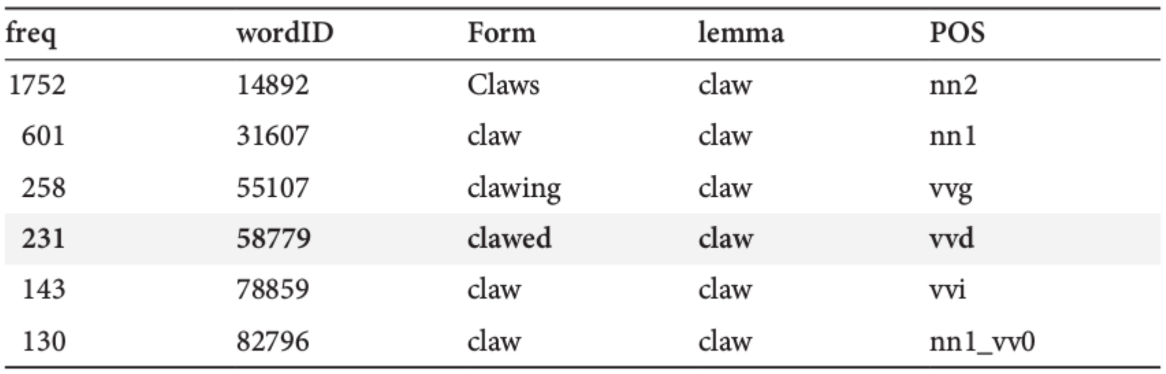
Table 3: part of seqWord Table in COCA [15]

ID	textID	wordID
359653867	1034159	539
359653868	1034159	305
359653869	1034159	12799
359653870	**1034159**	**58779**
359653871	1034159	3
359653872	1034159	2636

The ‘dictionary’ table contains part of speech, lemma, and frequency information for each of the 2.3 million types in the corpus, and the [wordID] value in this table relates to the [wordID] value in the [seqWord] table.

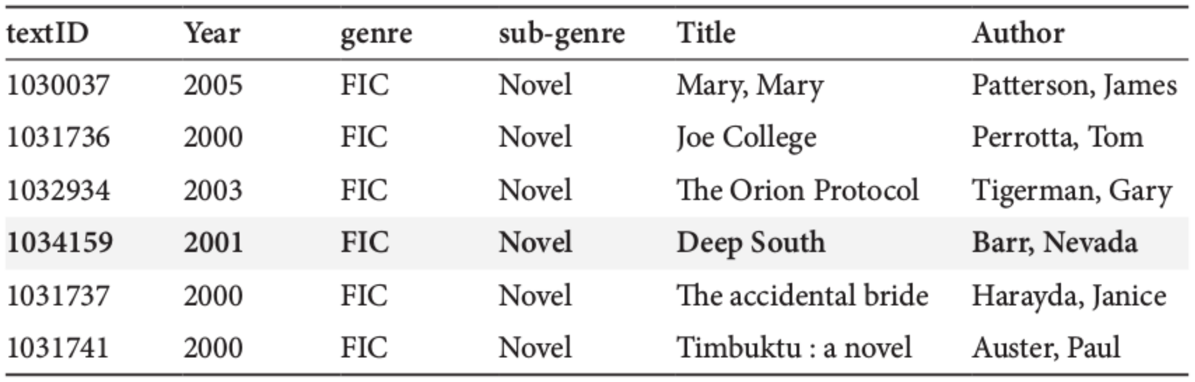
The ‘sources’ table contains metadata on each of the 150,000+ texts in the corpus, and contains information on such things as genre, sub-genre, title, author, source information (e.g. magazine, issue, and pages)

Table 4: part of dictionary Table in COCA [15]



freq	wordID	Form	lemma	POS
1752	14892	Claws	claw	nn2
601	31607	claw	claw	nn1
258	55107	clawing	claw	vvg
231	**58779**	**clawed**	**claw**	**vvd**
143	78859	claw	claw	vvi
130	82796	claw	claw	nn1_vv0

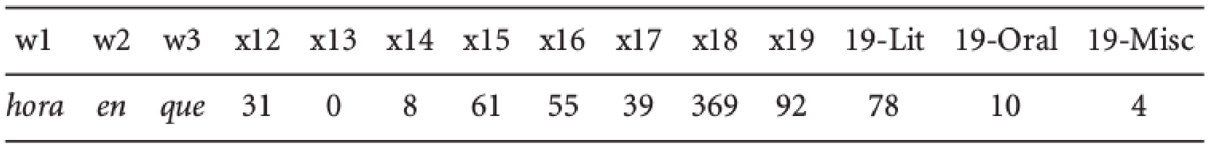
Table 5: part of dictionary Table in COCA [15]



textID	Year	genre	sub-genre	Title	Author
1030037	2005	FIC	Novel	Mary, Mary	Patterson, James
1031736	2000	FIC	Novel	Joe College	Perrotta, Tom
1032934	2003	FIC	Novel	The Orion Protocol	Tigerman, Gary
1034159	**2001**	**FIC**	**Novel**	**Deep South**	**Barr, Nevada**
1031737	2000	FIC	Novel	The accidental bride	Harayda, Janice
1031741	2000	FIC	Novel	Timbuktu : a novel	Auster, Paul

The 100 million word Corpus del Español also uses a relational database model. [14] describes how it’s architecture supports information retrieval and statistics generation. The following figure shown a table it has used to store 3-grams.

Table 6: Part of trigram table in Corpus del Español [14]



w1	w2	w3	x12	x13	x14	x15	x16	x17	x18	x19	19-Lit	19-Oral	19-Misc
hora	*en*	*que*	31	0	8	61	55	39	369	92	78	10	4

The columns x12–x19 refer to the frequency of this 3-gram in the 1200s–1900s; and 19-Lit, 19-Oral, and 19-Misc refer to the frequency in three categories from the 1900s. Because each n-gram relational database table is indexed, including some clustered indices, the queries on the tables are very fast usually just one or two seconds.

By taking ideas from the implementation of above two corpus architectures, we decided on moving forward with our project with a few architectural designs and identifying the best solution. We leave the XML based architecture in BNC since it uses software like SARA, BNCweb and Sketch engine of which source codes are not available free and also since these software are not maintained properly now . Also according to [14] BNC architecture has the following drawbacks. "These corpora make extensive use of separate indexes, which contain pointers to words in the actual textual corpus. For example, the BNC uses more than 22,000 index files (more than 2.0 GB) to speed up the searches.

Even with extensive indexes, many queries are still quite slow. For example, with the current version of the BNC, a query to find the most common collocates with a moderate common word like [way] is quite expensive, and is almost prohibitive with a word like [with] or [had]. More important, due to the limitations of the indexing schema in the current version of the BNC, it is difficult (and sometimes impossible) to directly query part of speech, such as [had always VVD] or [other AJ0 NN1]. Finally, it is difficult to add additional layers of annotation – such as synonyms or user-defined lexical categories – which would allow users to perform more semantically oriented queries."

With the best solution we could get from existing implementations, which is using relational database, we also observed that there have been not many studies carried out about using NoSQL for implementing a corpus. Therefore we will be looking at graph database and in memory database technologies also and will evaluate what will be the best solution.

2.5.2 Relational data base as storage medium

When study the architecture of existing corpus implementations, best database model we can think of uses a relational database model. In this section we will examine the literature that supports the decision of using a database system using relational databases and what are against it.

In literature related to COCA [15], they have mentioned the impact of using relational databases in the implementation of data storage system of the corpus. "The relational database architecture allows a number of significant advantages over competing architectures. The first is speed and size. Because each of the tables is indexed (including the use of clustered indexes), queries of even large corpora are very fast. For example, it takes just about 1.3 seconds to find the top 100 noun collocates after the 23,000 tokens of white in the 100 million word BNC (paper, house, wine), and this increases to just 2.1 seconds for the

168,000 tokens of white in the 385+ million word American Corpus. Another example is that it takes about 1.2 seconds to find the 100 most frequent strings for [end] up [vvg] in the BYU-BNC corpus (end up paying, ended up going), and this is the same amount of time that it takes in the 385 million word American Corpus as well. In other words, the architecture is very scalable, with little or no decrease in speed, even as we move from a 100 million word corpus to a 385+ million word corpus. Even more complicated queries are quite fast."

From what we saw in the last section about existing corpora which use relational databases and from the above description, we can see that we can solve balance size, annotation, and speed, the three fundamental challenges of developing a corpus by using a relational database system. After considering these facts we selected a relational database as one of the options to use as the storage medium.

2.5.3 NoSQL Graph database as storage medium

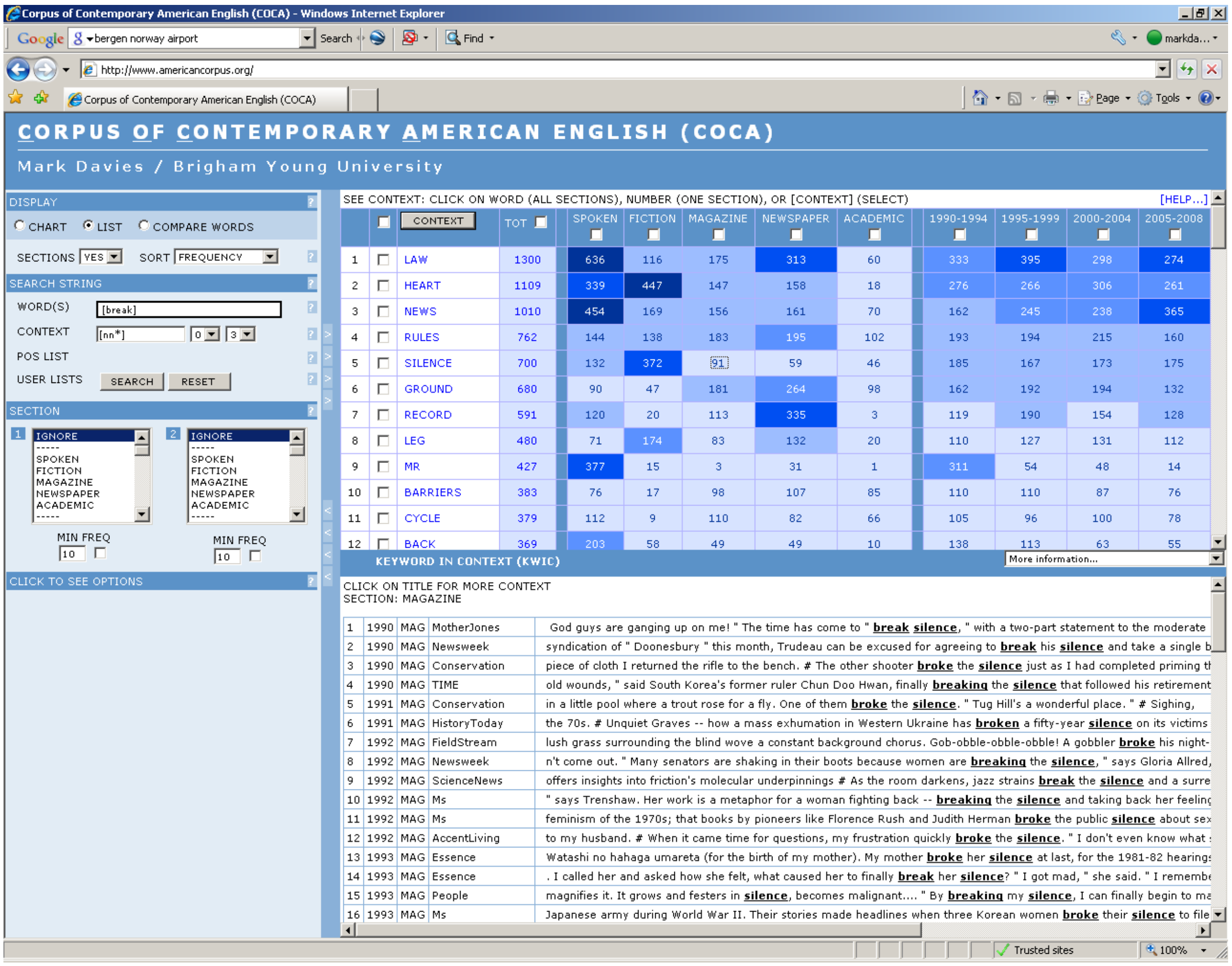
When storing details about words, bigrams, trigrams and sentences, one of the biggest challenges is how to store the relationships between each of these entities. In the relational model, since different relationships are stored in different tables, table joins have to be performed at every information retrieval, which affects performance very much. One of the best ways to represent relationships between entities is to use a graph. A graph database applies graph theory in the storage of information about the relationship between entities. So we are also considering graph databases in our study as a candidate for the optimal storage solution.

Another requirement of a corpus is to find base forms of a particular word and syntactically similar phrases. This needs character level splitting of words in order to find character level differences like that. Graph databases are very good for analyzing how closely things are related, how many steps are required to get from one point to another. So we are considering usage of a graph structure for this requirement and analyzing performance. Currently there are several implementations of graph databases such as Neo4j, OrientDB, Titan and DEX. In our study we use Neo4j as our graph database system since, it has been proven to perform better than other graph databases [21].

2.6 Information Visualization

One of the main ideas to consider when developing a corpus is providing a user interface, so that users can use the corpus to retrieve information , especially statistical details. When considering existing corpus implementations, most of the popular corpora like BNC, COCA, Corpus del Español, Google books corpus [12], etc. use a similar kind of interface.

Figure 2 shows a screenshot of the web interface of COCA, which shows the main parts of its interface.


Corpus of Contemporary American English (COCA) - Windows Internet Explorer
http://www.americancorpus.org/
CORPUS OF CONTEMPORARY AMERICAN ENGLISH (COCA)
Mark Davies / Brigham Young University

DISPLAY: CHART [✓ LIST] COMPARE WORDS
SECTIONS: YES SORT: FREQUENCY
SEARCH STRING
WORD(S): [break]
CONTEXT: [nn*] 0 3
POS LIST
USER LISTS
SECTION: 1 IGNORE / 2 IGNORE (SPOKEN, FICTION, MAGAZINE, NEWSPAPER, ACADEMIC)
MIN FREQ: 10 MIN FREQ: 10
CLICK TO SEE OPTIONS

SEE CONTEXT: CLICK ON WORD (ALL SECTIONS), NUMBER (ONE SECTION), OR [CONTEXT] (SELECT)

	CONTEXT	TOT	SPOKEN	FICTION	MAGAZINE	NEWSPAPER	ACADEMIC	1990-1994	1995-1999	2000-2004	2005-2008
1	LAW	1300	636	116	175	313	60	333	395	298	274
2	HEART	1109	339	447	147	158	18	276	266	306	261
3	NEWS	1010	454	169	156	161	70	162	245	238	365
4	RULES	762	144	138	183	195	102	193	194	215	160
5	SILENCE	700	132	372	91	59	46	185	167	173	175
6	GROUND	680	90	47	181	264	98	162	192	194	132
7	RECORD	591	120	20	113	335	3	119	190	154	128
8	LEG	480	71	174	83	132	20	110	127	131	112
9	MR	427	377	15	3	31	1	311	54	48	14
10	BARRIERS	383	76	17	98	107	85	110	110	87	76
11	CYCLE	379	112	9	110	82	66	105	96	100	78
12	BACK	369	203	58	49	49	10	138	113	63	55

KEYWORD IN CONTEXT (KWIC)

CLICK ON TITLE FOR MORE CONTEXT
SECTION: MAGAZINE

1	1990	MAG	MotherJones	God guys are ganging up on me! " The time has come to " **break** **silence**, " with a two-part statement to the moderate
2	1990	MAG	Newsweek	syndication of " Doonesbury " this month, Trudeau can be excused for agreeing to **break** his **silence** and take a single b
3	1990	MAG	Conservation	piece of cloth I returned the rifle to the bench. # The other shooter **broke** the **silence** just as I had completed priming th
4	1990	MAG	TIME	old wounds, " said South Korea's former ruler Chun Doo Hwan, finally **breaking** the **silence** that followed his retirement
5	1991	MAG	Conservation	in a little pool where a trout rose for a fly. One of them **broke** the **silence**. " Tug Hill's a wonderful place. " # Sighing,
6	1991	MAG	HistoryToday	the 70s. # Unquiet Graves -- how a mass exhumation in Western Ukraine has **broken** a fifty-year **silence** on its victims
7	1992	MAG	FieldStream	lush grass surrounding the blind wove a constant background chorus. Gob-obble-obble-obble! A gobbler **broke** his night-
8	1992	MAG	Newsweek	n't come out. " Many senators are shaking in their boots because women are **breaking** the **silence**, " says Gloria Allred,
9	1992	MAG	ScienceNews	offers insights into friction's molecular underpinnings # As the room darkens, jazz strains **break** the **silence** and a surre
10	1992	MAG	Ms	" says Trenshaw. Her work is a metaphor for a woman fighting back -- **breaking** the **silence** and taking back her feeling
11	1992	MAG	Ms	feminism of the 1970s; that books by pioneers like Florence Rush and Judith Herman **broke** the public **silence** about sex
12	1992	MAG	AccentLiving	to my husband. # When it came time for questions, my frustration quickly **broke** the **silence**. " I don't even know what
13	1993	MAG	Essence	Watashi no hahaga umareta (for the birth of my mother). My mother **broke** her **silence** at last, for the 1981-82 hearings
14	1993	MAG	Essence	. I called her and asked how she felt, what caused her to finally **break** her **silence**? " I got mad, " she said. " I remembe
15	1993	MAG	People	magnifies it. It grows and festers in **silence**, becomes malignant.... " By **breaking** my **silence**, I can finally begin to ma
16	1993	MAG	Ms	Japanese army during World War II. Their stories made headlines when three Korean women **broke** their **silence** to file

Figure 2: Web interface of COCA [15]

[15] describes more details about the features of the interface.

Users fill out the search form in the left frame of the window, they see the frequency listings or charts in the upper right-hand frame, and they can then click on any of the entries from the frequency lists to see the Keyword in Context (KWIC) display in the lower right-hand frame. They can also click on any of the KWIC entries to see even more context (approximately one paragraph) in this lower right-hand frame. Note also that there is a drop-down box (between the upper and lower right-hand frames) which provides help for many different topics.

Figure 3 shows the frame where the user has to enter the query to retrieve information.

Since most of the popular corpora follows this template for the interface and it contains almost every functionality required, we are also following a similar approach when designing the corpus.

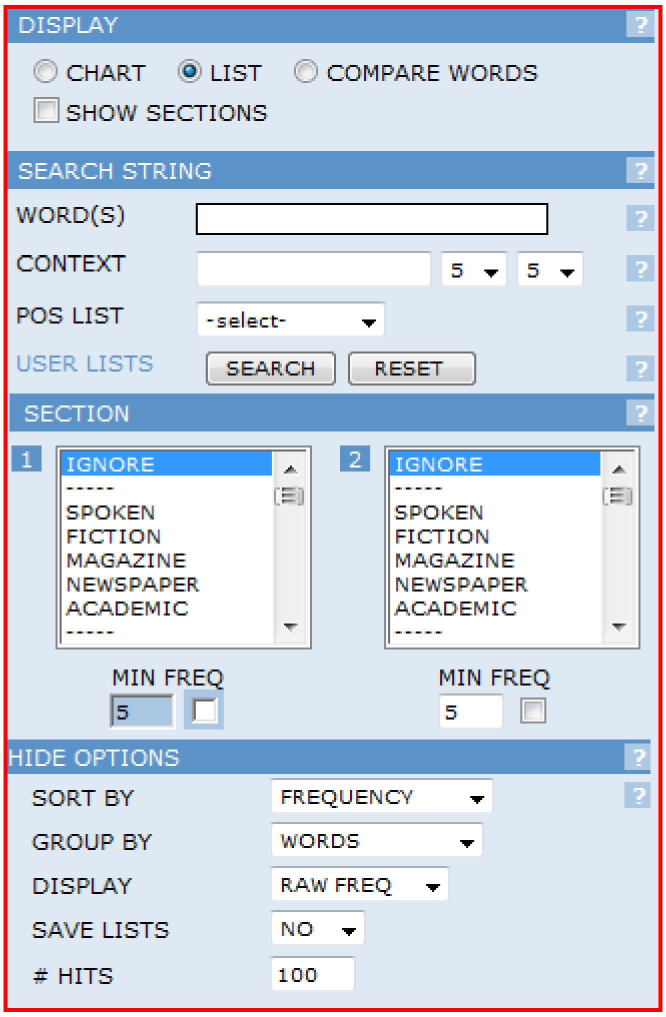

DISPLAY

CHART [✓ LIST] COMPARE WORDS

☐ SHOW SECTIONS

SEARCH STRING

WORD(S)

CONTEXT 5 5

POS LIST -select-

USER LISTS

SEARCH RESET

SECTION

1 IGNORE ----- SPOKEN FICTION MAGAZINE NEWSPAPER ACADEMIC -----

2 IGNORE ----- SPOKEN FICTION MAGAZINE NEWSPAPER ACADEMIC -----

MIN FREQ 5 ☐

MIN FREQ 5 ☐

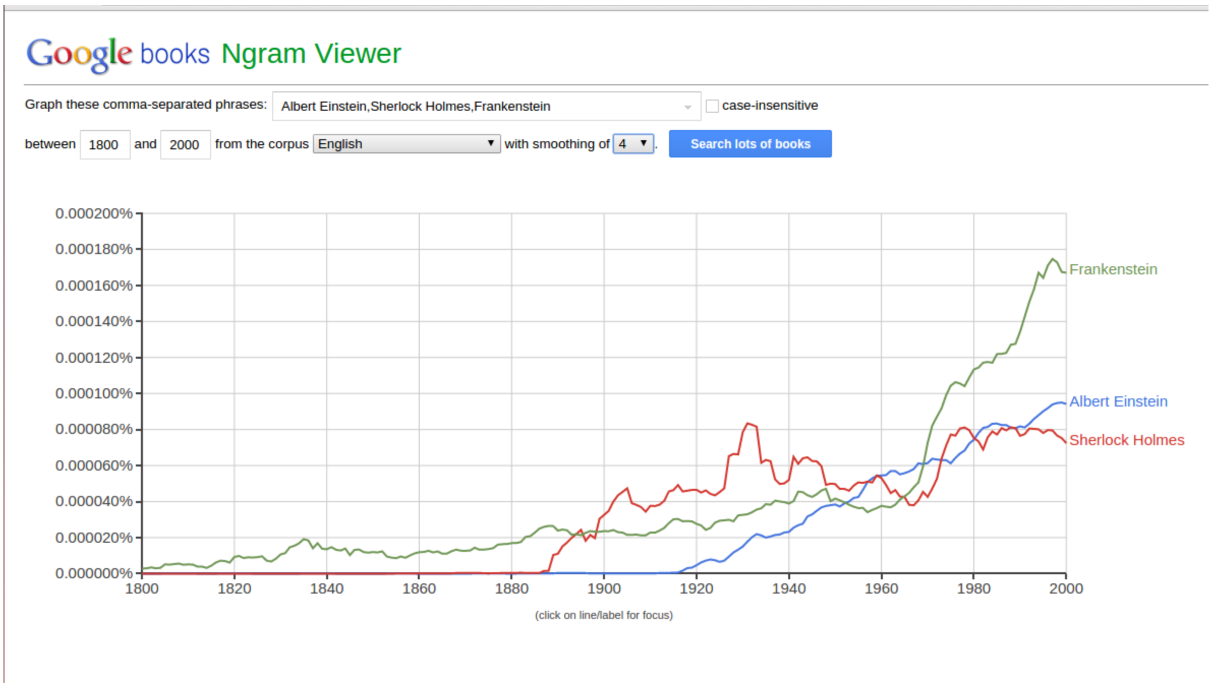
HIDE OPTIONS

SORT BY	FREQUENCY
GROUP BY	WORDS
DISPLAY	RAW FREQ
SAVE LISTS	NO
# HITS	100

Figure 3: User inputs of COCA interface [15]

For effective data visualization, we can look at the interface of Google Ngram Viewer [17] also. Even though it doesn't give access to most details, the functionalities it has are very effective. Figure 4 is a screenshot of its web interface.

Here users can compare the usage of different words over different time periods. The graph used here is different from graphs used in previously mentioned.



Graph these comma-separated phrases: Albert Einstein,Sherlock Holmes,Frankenstein

between 1800 and 2000 from the corpus English with smoothing of 4

Figure 4: Google Ngram Viewer [17]

2.7 Extracting Linguistic Features of the Sinhala Language

A main usage of a language corpus is extracting linguistic features of a language. Existing corpora have been widely used in this for extracting features of various languages.

[38] describes how a corpus has been used to identify the colligations of “TO” and “FOR” in their particular function as prepositions in sentences in the corpus to discover the similarity and differences of the colligations between “TO” and “FOR” in their particular function as prepositions in sentences, and to examine whether the students had applied these two particular words correctly in their essays with reference to the function as prepositions. This has identified the following similarities and differences of “TO” and “FOR” using the corpus.

Using corpus this study has also identified many other features like patterns of using ‘to’ and ‘for’, incorrect uses of ‘to’ and ‘for’ in language, etc.

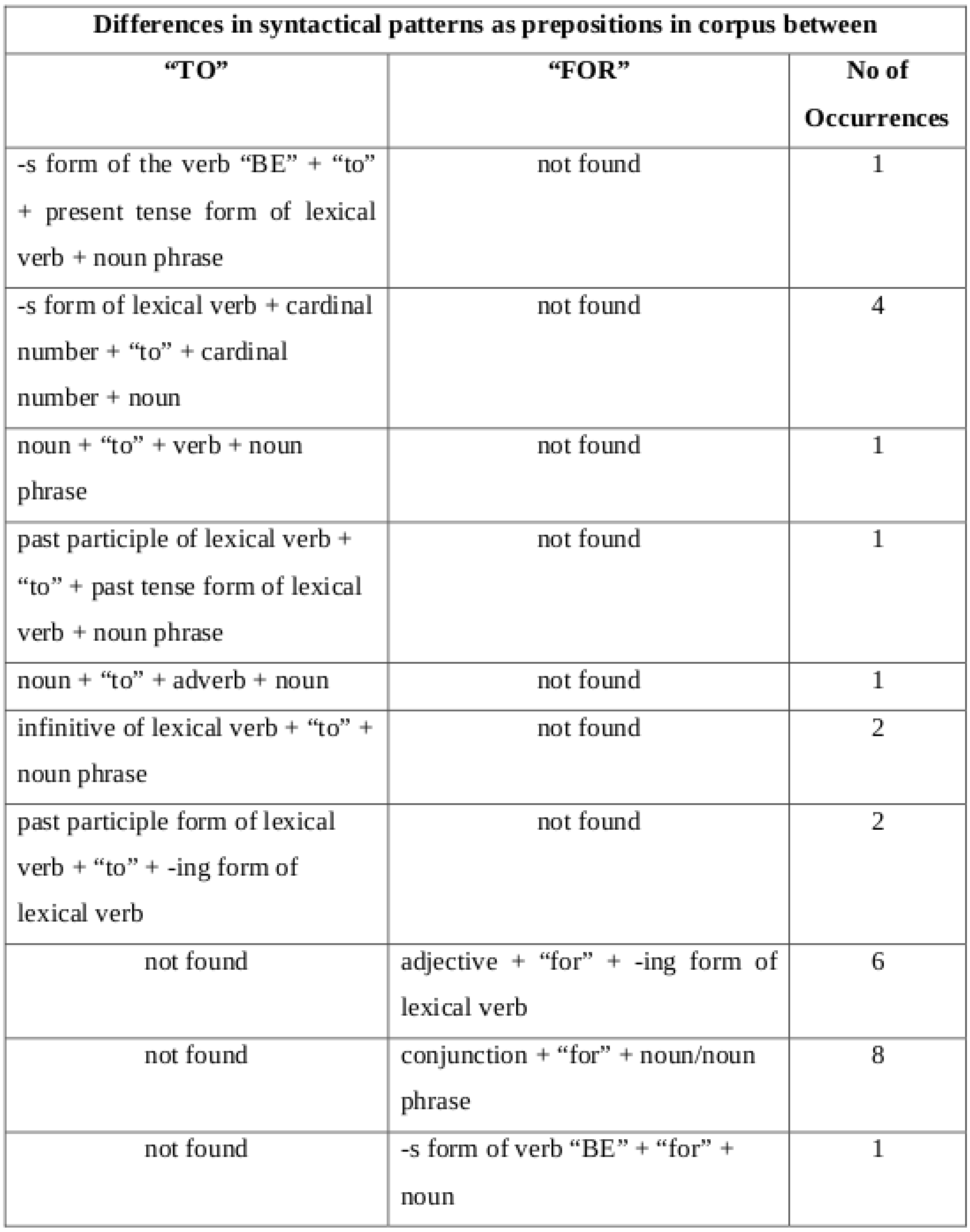
[18] also describes how linguistics features can be extracted using a corpus and some popular use cases.

Since there is no proper corpus for Sinhala language, there is not much work done in this area before this project.

Table 7: Identified Similarities of “TO” and “FOR” [38]

No	**The same syntactical patterns in corpus which “TO” and “FOR” share**
1	noun + “to”/”for” + noun/noun phrase
2	adjective + “to”/”for” + noun/noun phrase
3	past participle form of lexical verb + “to”/”for” + noun/noun phrase
4	preposition + “to”/”for” + noun/noun phrase
5	-ing form of lexical verb + “to”/”for” + noun/noun phrase
6	adverb + “to”/”for” + noun/noun phrase
7	base form of lexical verb + “to”/”for” + noun/noun phrase

Table 8: Identified Differences of “TO” and “FOR” [38]



Differences in syntactical patterns as prepositions in corpus between		
“TO”	**“FOR”**	**No of Occurrences**
-s form of the verb “BE” + “to” + present tense form of lexical verb + noun phrase	not found	1
-s form of lexical verb + cardinal number + “to” + cardinal number + noun	not found	4
noun + “to” + verb + noun phrase	not found	1
past participle of lexical verb + “to” + past tense form of lexical verb + noun phrase	not found	1
noun + “to” + adverb + noun	not found	1
infinitive of lexical verb + “to” + noun phrase	not found	2
past participle form of lexical verb + “to” + -ing form of lexical verb	not found	2
not found	adjective + “for” + -ing form of lexical verb	6
not found	conjunction + “for” + noun/noun phrase	8
not found	-s form of verb “BE” + “for” + noun	1

3.0 Design

3.1 Introduction

Chapter 3 presents the architecture of SinMin. Initially, an overview of the overall architecture is discussed, followed by a detailed description of each component within SinMin. The objective of this chapter is to enlighten the reader with design considerations and functionalities of SinMin.

3.2 Overall System Architecture

SinMin consists of 4 main components, web crawlers, data storage, REST API and web interface.

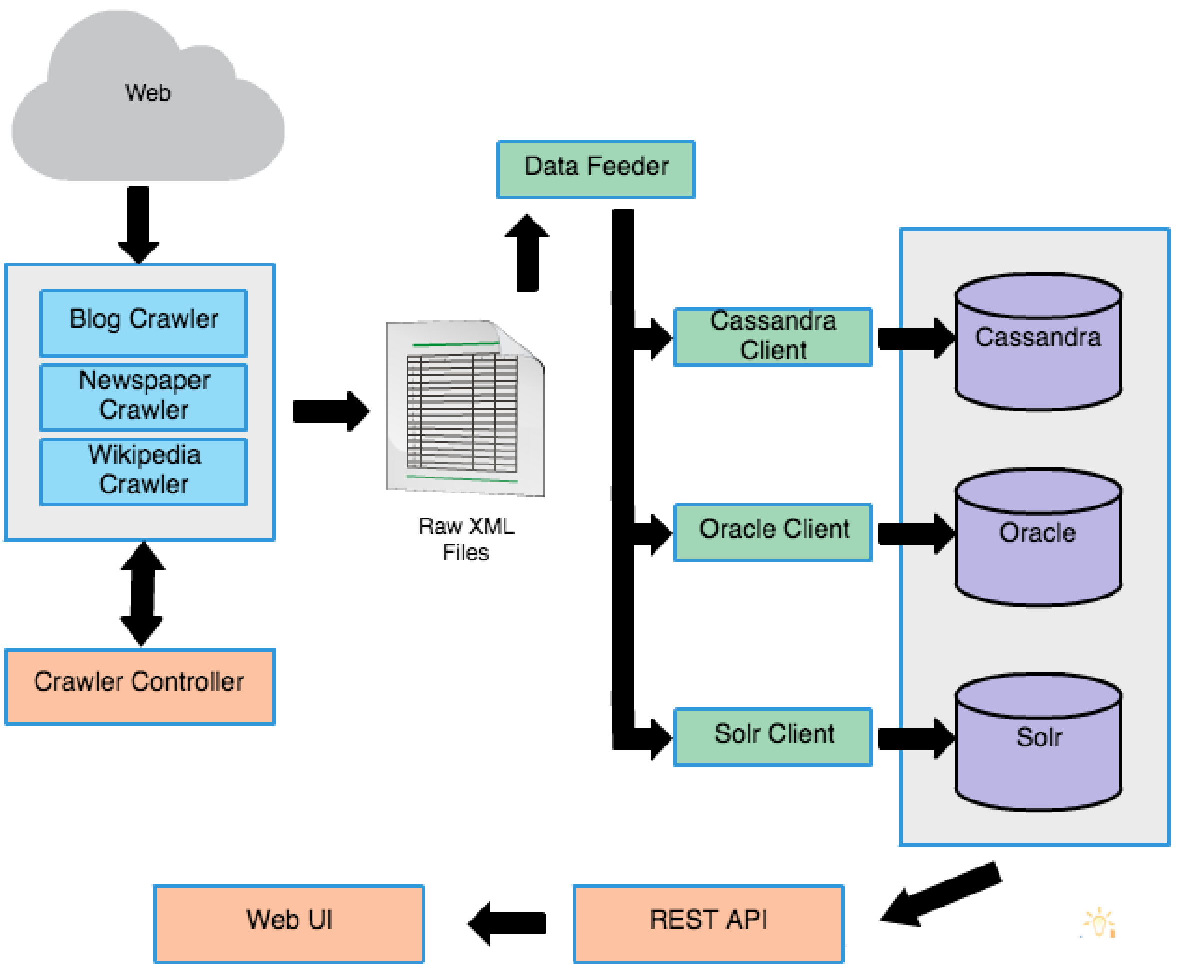


Figure 5: Overall architecture of SinMin

Crawlers are designed so that, when a period of time given to the crawler, they will go into the websites which contain the Sinhala language sources and will collect all Sinhala language resources (articles, subtitles, blogs, etc.) and the corresponding metadata of each source. Then those collected resources are saved as XML files in the server with those metadata. Then these unstructured data is saved into the Cassandra database in a structured manner.

The API accesses the database and makes information in the corpus available for the outside users. User Interface of SinMin allows users to view a visualized and summarized view of statistical data available in the corpus.

3.3 Crawler Design

Crawlers are responsible of finding web pages that contain Sinhala content, fetching, parsing and storing them in a manageable format. Design of a particular crawler depends on the language resource that is expected to be crawled. The following list contains a list of online Sinhala language sources which were identified up to now.

- Sinhala Online Newspapers
  - Lankadeepa - http://lankadeepa.lk/
  - Divaina - http://www.divaina.com/
  - Dinamina - http://www.dinamina.lk/2014/06/26/
  - Lakbima - http://www.lakbima.lk/
  - Mawbima - http://www.mawbima.lk/
  - Rawaya - http://ravaya.lk/
  - Silumina - http://www.silumina.lk/
- Sinhala News Sites
  - Ada Derana - http://sinhala.adaderana.lk/
- Sinhala Religious and Educational Magazines
  - Aloka Udapadi - http://www.lakehouse.lk/alokoudapadi/
  - Budusarana - http://www.lakehouse.lk/budusarana/
  - Namaskara - http://namaskara.lk/
  - Sarasawiya - http://sarasaviya.lk/
  - Vidusara - http://www.vidusara.com/
  - Wijeya - http://www.wijeya.lk/
- Sri Lanka Gazette in Sinhala - http://documents.gov.lk/gazette/

- Online Mahawansaya - http://mahamegha.lk/mahawansa/
- Sinhala Movie Subtitles - http://www.baiscopelk.com/category/සිංහල-උපසිරැස/
- Sinhala Wikipedia - http://si.wikipedia.org/
- Sinhala Blogs

When collecting data from these sites, the first thing to be done is creating a list of web pages to visit and collect data. There are three main ways that can be followed to do this.

1. Going to a root URL (may be the home page) of a given website, list all links available in that page, then continuously going to each listed page and doing the same thing while no more new pages could be found.
2. Identify a pattern in the page URL and generate URL’s available for each day and visit them.
3. Identify a page where all articles are listed and then get the list of URL’s from that page.

All 3 above mentioned methods have their advantages as well as disadvantages.

When using the first method, the same program can be used for all newspapers and magazines. So it will minimize the workload of implementation. But if this method is used, it will be difficult to keep track on what time periods are already crawled and what are not. Also the pages that do not have an incoming link from other pages, will not be listed. By using this, it is difficult to do a controlled crawling, which includes crawling from a particular date to another.

When considering sources like ‘Silumina’ newspaper, its article URLs look like “http://www.silumina.lk/2015/01/18/_art.asp?fn=aa1501187”. All URL share a common format which is

http://www.silumina.lk/ + date + _art.asp?fn= + unique article id

unique article id = article type+ last 2 digits of year + month + date + article number

For newspapers and magazines those have that kind of unique format, the second method can be used. But some newspapers have URLs which includes article name in it, e.g. : http://tharunie.lk/component/k2/item/976-අලුත්-අවුරුද්දට-අලුත්-කිරි-බතක්-හදමුද.html . The method we discussed is impossible to use for these kind of sources.

Most of the websites has a page for each day which has the list of articles published in that day, e.g.: http://www.vidusara.com/2014/12/10/viduindex.htm . This URL can be generated using resource (newspaper, magazine, etc.,) name and corresponding date. By referring to the HTML content of the page, the crawler gets the list of URLs of the articles published on that day. We used this 3rd method for URL generating since it can be used for all resources we identified and since we can track the time periods that are already crawled and those that are not crawled yet.

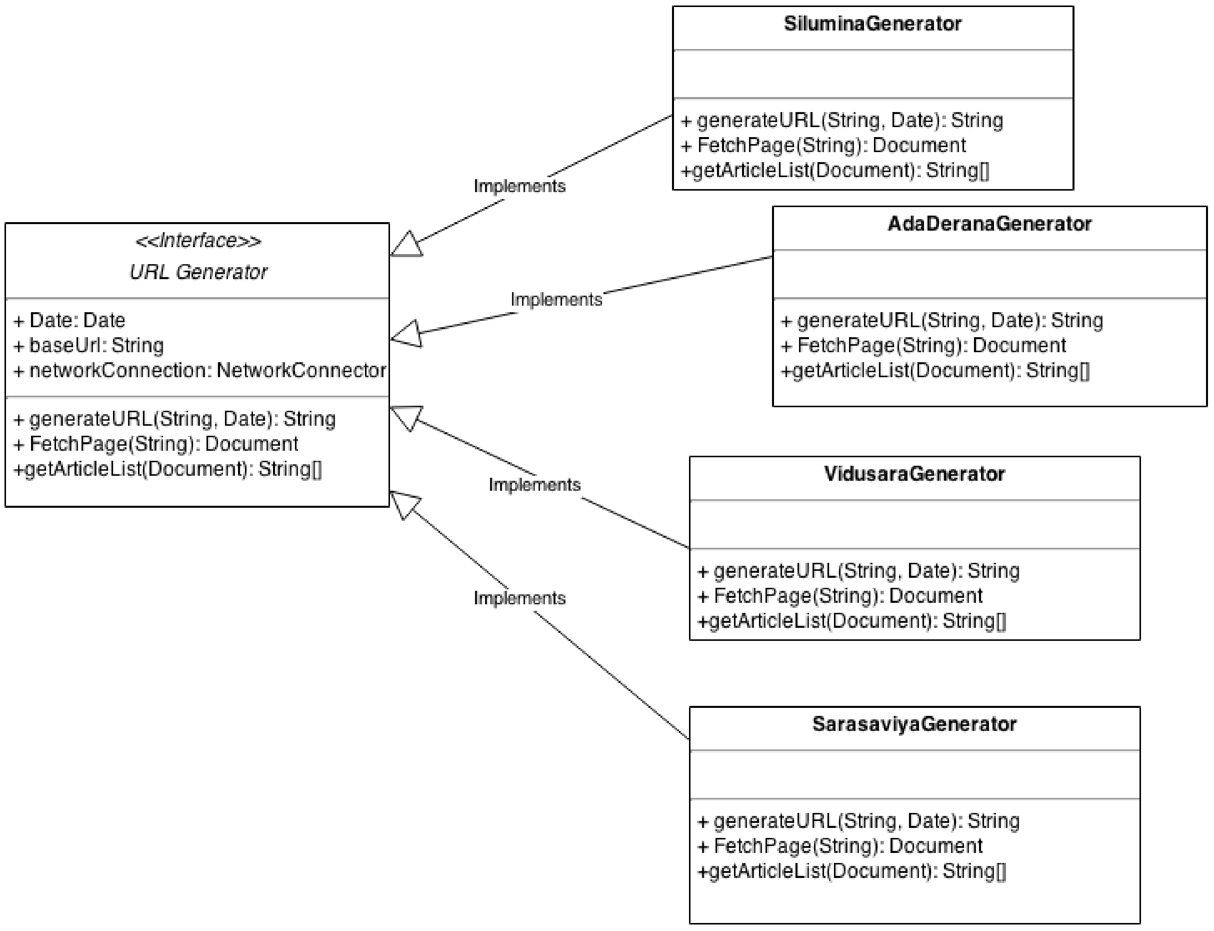
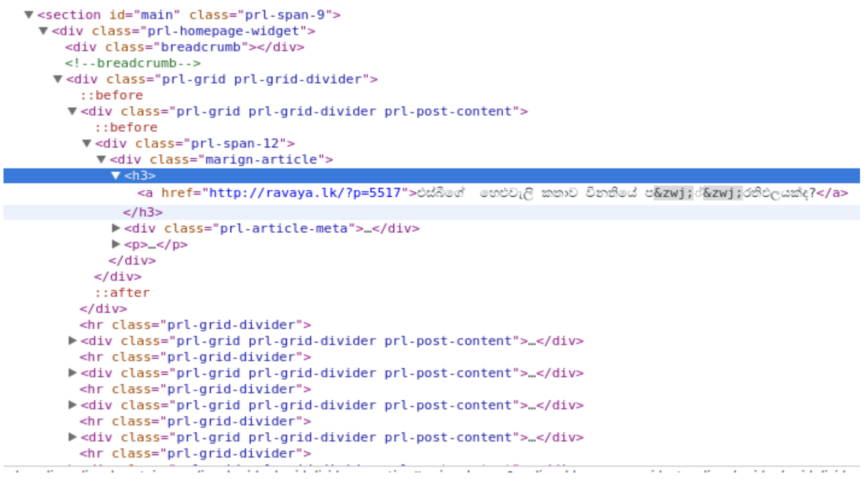

SiluminaGenerator
+ generateURL(String, Date): String
+ FetchPage(String): Document
+getArticleList(Document): String[]
Implements
<<Interface>>
URL Generator
+ Date: Date
+ baseUrl: String
+ networkConnection: NetworkConnector
+ generateURL(String, Date): String
+ FetchPage(String): Document
+getArticleList(Document): String[]
AdaDeranaGenerator
+ generateURL(String, Date): String
+ FetchPage(String): Document
+getArticleList(Document): String[]
Implements
Implements
VidusaraGenerator
+ generateURL(String, Date): String
+ FetchPage(String): Document
+getArticleList(Document): String[]
Implements
SarasaviyaGenerator
+ generateURL(String, Date): String
+ FetchPage(String): Document
+getArticleList(Document): String[]

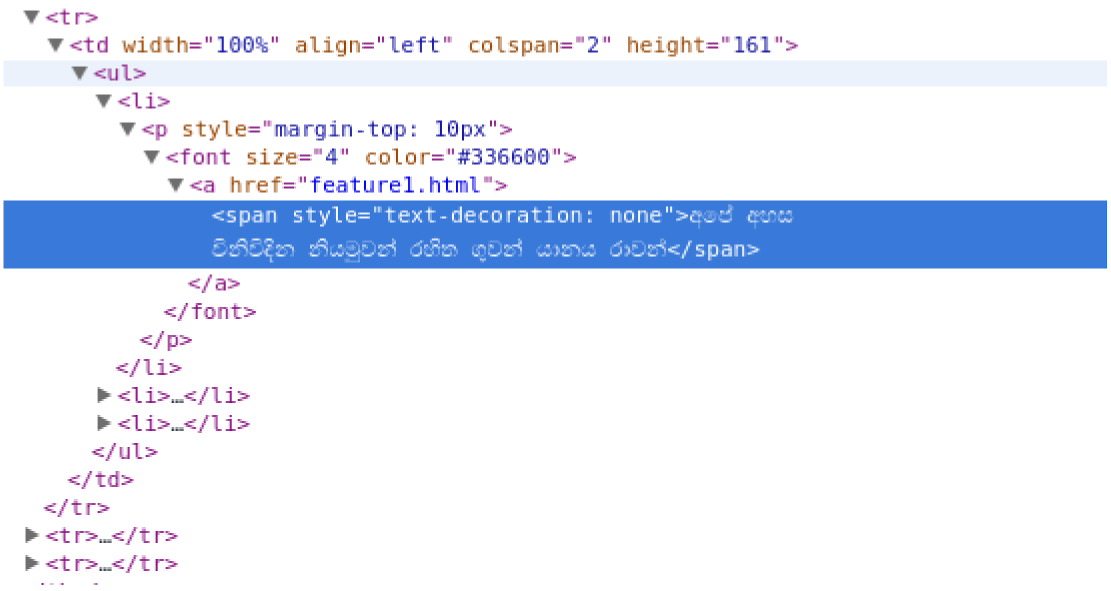
Figure 6: URLGenerators Architecture

All URL generators implement the methods *generateURL* to generate the URL of the page which contains the list of articles using base URL and date, *fetchPage* to connect to internet and fetch the page that contains URL list using given URL, and *getArticleList* to extract URL list of articles using a given html document. The way of extracting the URL list varies from one resource to another, because the styling of each page that contains the URL list also varies from one source to another.



```
▼<section id="main" class="prl-span-9">
  ▼<div class="prl-homepage-widget">
     <div class="breadcrumb"></div>
     <!--breadcrumb-->
    ▼<div class="prl-grid prl-grid-divider">
       ::before
      ▼<div class="prl-grid prl-grid-divider prl-post-content">
         ::before
        ▼<div class="prl-span-12">
          ▼<div class="marign-article">
            ▼<h3>
               <a href="http://ravaya.lk/?p=5517">එස්කිමෝ  හෙප්වැලි කතාව විනාශයේ ප&zwj;්&zwj;රතිඵලයක්ද?</a>
             </h3>
            ▶<div class="prl-article-meta">…</div>
            ▶<p>…</p>
           </div>
         </div>
         ::after
       </div>
       <hr class="prl-grid-divider">
      ▶<div class="prl-grid prl-grid-divider prl-post-content">…</div>
       <hr class="prl-grid-divider">
      ▶<div class="prl-grid prl-grid-divider prl-post-content">…</div>
       <hr class="prl-grid-divider">
      ▶<div class="prl-grid prl-grid-divider prl-post-content">…</div>
       <hr class="prl-grid-divider">
      ▶<div class="prl-grid prl-grid-divider prl-post-content">…</div>
       <hr class="prl-grid-divider">
```

Figure 7: URL list page of Rawaya Newspaper



```
▼<tr>
  ▼<td width="100%" align="left" colspan="2" height="161">
    ▼<ul>
      ▼<li>
        ▼<p style="margin-top: 10px">
          ▼<font size="4" color="#336600">
            ▼<a href="feature1.html">
               <span style="text-decoration: none">අපේ අහස
               විනිවිදින නියමුවන් රහිත ගුවන් යානය රාවන්</span>
             </a>
           </font>
         </p>
       </li>
      ▶<li>…</li>
      ▶<li>…</li>
     </ul>
   </td>
 </tr>
▶<tr>…</tr>
▶<tr>…</tr>
```

Figure 8: URL list page of Widusara Newspaper

Generators for Mahawansaya, Subtitles and Wikipedia follow the same architecture, but with some small modifications. Mahawansaya sources, Wikipedia articles and subtitles we used for corpus cannot be listed under a date. So the above mentioned method for listing pages didn't work for them. However, they had pages that list all the links to articles/files. So when crawling those resources, the crawler goes through the whole list of items available in the above mentioned page. When it crawls a new resource, it will save its details in a mySQL database. So, before crawling, it will check the database if it has already crawled or not.

After creating the list of articles to crawl, the next task that needs to be done is collecting content and other metadata from each of these pages. This was done by the Parser class.

All parsers other than subtitleParser and gazetteParser read the required content and metadata from HTML pages. Since subtitles are available as zip files and gazettes are available as pdf files, those parsers have additional functionalities to download and unzip as well.

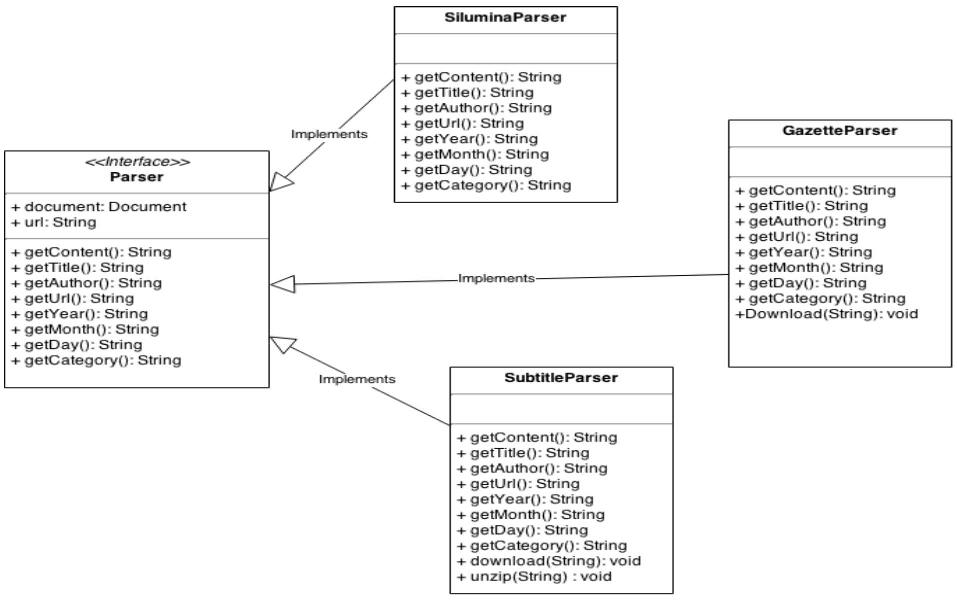


Figure 9: Parser Architecture

After extracting all the URLs in the list page as described above, crawler goes through each URL and passes the page to the *XMLFileWriter* using the *addDocument* method. All data is kept with *XMLFileWriter* until they are written to the file. When a crawler finishes extracting articles of a particular data, it notifies the *XMLFileWriter* so that it can write the content to the XML file and send the finished date with a notification. With this notification *XMLFileWriter* writes the content to a file named with the date of the article in a folder named with the ID of that particular crawler. At the same time, the finished date is written to the database.

Figure 10 shows the process of web crawling as a sequence diagram.

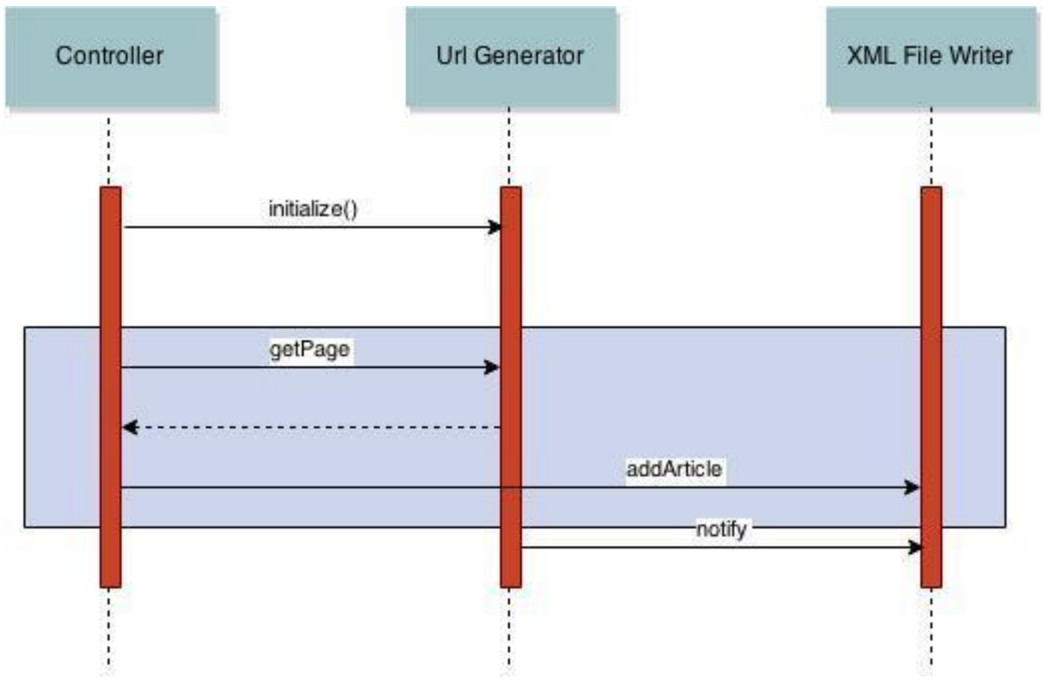


Figure 10: Sequence diagram of web crawling

Figure 11 shows the architecture of a crawler of SinMin.

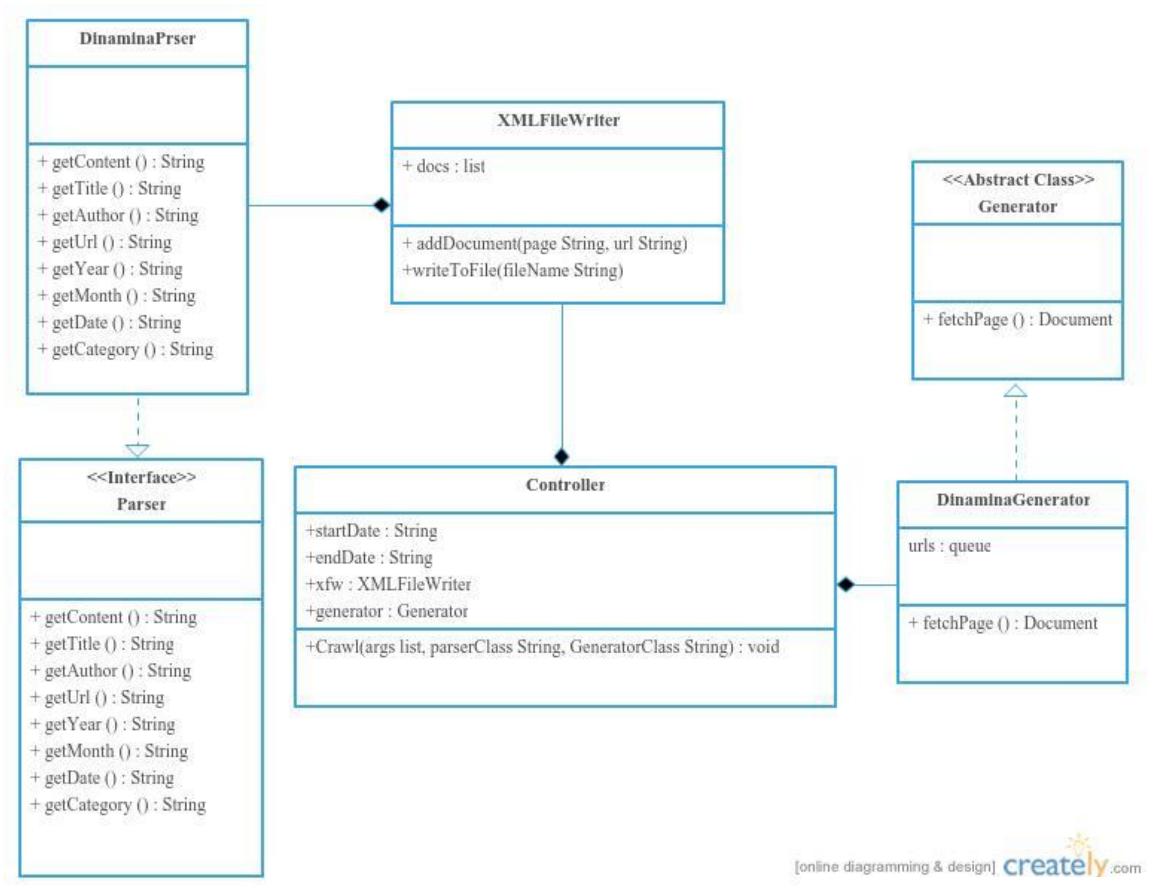
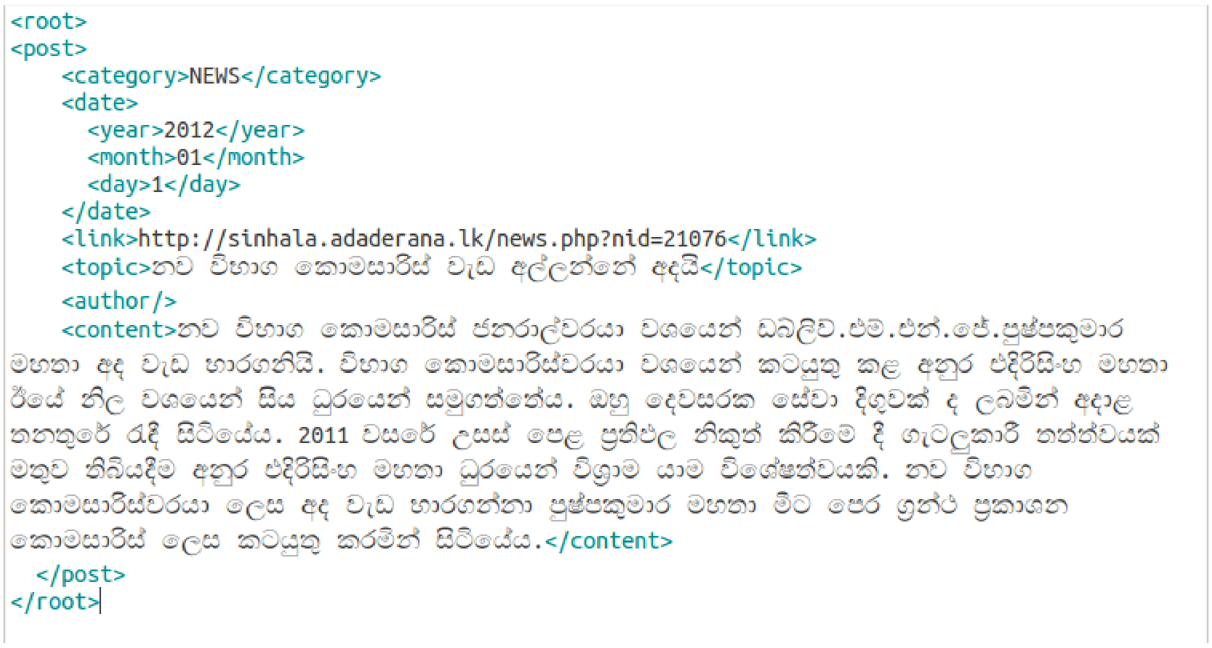


Figure 11: Overall architecture of a crawler

Figure 12 shows a sample xml file with one article stored in it.



```
<root>
<post>
    <category>NEWS</category>
    <date>
      <year>2012</year>
      <month>01</month>
      <day>1</day>
    </date>
    <link>http://sinhala.adaderana.lk/news.php?nid=21076</link>
    <topic>නව විභාග කොමසාරිස් වැඩ අල්ලන්නේ අදයි</topic>
    <author/>
    <content>නව විභාග කොමසාරිස් ජනරාල්වරයා වශයෙන් ඩබ්ලිව්.එම්.එන්.ජේ.පුෂ්පකුමාර
මහතා අද වැඩ භාරගනියි. විභාග කොමසාරිස්වරයා වශයෙන් කටයුතු කළ අනුර එදිරිසිංහ මහතා
ඊයේ නිල වශයෙන් සිය ධුරයෙන් සමුගත්තේය. ඔහු දෙවසරක සේවා දිගුවක් ද ලබමින් අදාළ
තනතුරේ රැඳී සිටියේය. 2011 වසරේ උසස් පෙළ ප්‍රතිඵල නිකුත් කිරීමේ දී ගැටලුකාරී තත්ත්වයක්
මතුව තිබියදීම අනුර එදිරිසිංහ මහතා ධුරයෙන් විශ්‍රාම යාම විශේෂත්වයකි. නව විභාග
කොමසාරිස්වරයා ලෙස අද වැඩ භාරගන්නා පුෂ්පකුමාර මහතා මීට පෙර ග්‍රන්ථ ප්‍රකාශන
කොමසාරිස් ලෙස කටයුතු කරමින් සිටියේය.</content>
  </post>
</root>
```

Figure 12: Sample XML file

3.4 Data Storage Design

In a corpus, the data storage system is a vital part because the performance of data insertion and retrieval mainly depends on it. Most of the existing corpora have used relational databases or indexed file systems as the data storage system. But no study has been done about how other existing database systems like column store databases, graph databases and document databases perform when they are used in a corpus. So we carried out a performance analysis using several database systems to identify what is the most suitable data storage system for implementing SinMin.

3.4.1 Performance analysis for selecting data storage mechanism

In this study, we compared the performance of various database systems and technologies as a data storage component of a corpus and tried to find an optimal solution which provides the best performance. To achieve that, we have carried out a comprehensive comparison of a set of widely used database systems and technologies in their role of being a data storage component of a corpus in order to find an optimal solution with optimal performance.

We used H2 database as a relational database system because it's licensing permitted us to publish benchmarks and according to H2 benchmark tests, H2 has performed better than other relational databases such as MySQL, PostgreSQL, Derby and HSQLDB that allows us to publish benchmarks. We did not do such performance testing on relational databases such as Oracle, MS SQL Server and DB2 because their licenses are not compatible with benchmark publishing.

We used Apache Solr which was powered by Apache Lucene as an indexed file system. We selected Solr because Lucene is a widely used text search engine and it is licensed under Apache License which allows us to do benchmark testing on that.

When storing details about words, bigrams, trigrams and sentences, one of the biggest challenges is to store the relationships between each of these entities. One way to represent relationships between entities is to use a graph. Therefore we also considered graph databases in our study as a candidate for the optimal storage system. Currently there are several implementations of graph databases such as Neo4j, OrientDB, Titan and DEX. In our study we used Neo4j as our graph database system because, it has been identified by Jouili and Vansteenberghe [21] that Neo4j has performed better than the other graph databases.

Column databases improve the performance of information retrieval at the cost of higher insertion time and lack of consistency. Since one of our main goals is fast information retrieval, we considered column databases also as a candidate. We used Cassandra since it has been proven to give higher throughput than other widely used column databases.

The data set we used in this study contains data which were crawled from online sources that are written in the Sinhala language. The final dataset consisted of 5 million word tokens, 400,000 sentences and 20,000 posts.

All tests were run in a 2.4 GHz core i7 machine with 12GB of physical memory and a standard hard disk (non-solid state). Operating system was Ubuntu 14.04 with java version 1.8.0_05.
For every database system that was mentioned above, words were added in 50 iterations in which each iteration added 100,000 words with relevant sentences and posts.

At the end of each iteration that was mentioned above, queries to retrieve information for the following scenarios were passed to each database. The Same query was executed 6 times in each iteration and the median of recorded values was selected .

1. Get frequency of a given word in corpus using same word for every database system

2. Get list of frequencies of a given word in different time periods and different categories
3. Get the most frequently used 10 words in a given time period or a given category
4. Get the latest 10 posts that include a given word
5. Get the latest 10 posts that include a given word in a given time period or a given category
6. Get the 10 words which are most frequent as the last word of a sentence
7. Get the frequency of a given bigram in given time period and a given category
8. Get the frequency of a given trigram in given time period and a given category
9. Get most frequent bigrams
10. Get most frequent trigrams
11. Get the most frequent word after a given word
12. Get the most frequent word after a given bigram

3.4.1.1 Setting up Cassandra

Cassandra uses a query based data modeling which includes maintaining different column families to address querying needs; so when designing our database, we also maintain different column families for different querying needs and consistency among them was maintained in the application that we used to insert data.

The following are the column families we used with indexes used in each of them.

1. word_frequency (id bigint, content varchar, frequency int, PRIMARY KEY(content))
2. word_time_frequency (id bigint, content varchar, year int, frequency int, PRIMARY KEY(year, content))
3. word_time_inv_frequency (id bigint, content varchar, year int, frequency int, PRIMARY KEY((year), frequency, content))
4. word_usage (id bigint, content varchar, sentence varchar, date timestamp, PRIMARY KEY(content,date,id))
5. word_yearly_usage (id bigint, content varchar, sentence varchar, position int, postname text, year int, day int, month int, date timestamp, url varchar, author varchar, topic varchar, category int, PRIMARY KEY((content, year, category),date,id))
6. word_pos_frequency (id bigint, content varchar, position int, frequency int, PRIMARY KEY((position), frequency, content))

7. word_pos_id (id bigint, content varchar, position int, frequency int, PRIMARY KEY(position, content))
8. bigram_with_word_frequency (id bigint, word1 varchar, word2 varchar, frequency int, PRIMARY KEY(word1, frequency, word2))
9. bigram_with_word_id (id bigint, word1 varchar, word2 varchar, frequency int, PRIMARY KEY(word1, word2))
10. bigram_time_frequency (id bigint, bigram varchar, year int, frequency int, PRIMARY KEY(year, bigram))
11. trigram_time_frequency (id bigint, trigram varchar, year int, frequency int, PRIMARY KEY(year, trigram))
12. bigram_frequency (id bigint, content varchar, frequency int, category int, PRIMARY KEY(category,frequency, content))
13. bigram_id (id bigint, content varchar, frequency int, PRIMARY KEY(content))
14. trigram_frequency (id bigint, content varchar, frequency int, category int, PRIMARY KEY(category,frequency, content))
15. trigram_id (id bigint, content varchar, frequency int, PRIMARY KEY(content))
16. trigram_with_word_frequency (id bigint, word1 varchar, word2 varchar, word3 varchar, frequency int, PRIMARY KEY((word1,word2), frequency, word3))
17. trigram_with_word_id (id bigint, word1 varchar, word2 varchar, word3 varchar, frequency int, PRIMARY KEY(word1, word2,word3))

Even though we are only evaluating performance for 12 types of queries, we had to create a database with 17 column families, because we had to use extra 5 column families when updating frequencies when inserting data into database. Data insertion, querying and performance evaluation was done using java. Corresponding source files are available at https://github.com/madurangasiriwardena/performance-test .

3.4.1.2 Setting up Solr

Solr is an open source full-text search engine which runs on top of Apache Lucene which is a information retrieval system written in Java. Solr version 4.9.0 was used to do the performance testing.

Cache sizes of LRUCache, FastLRUCache and LFUCache were set to 0 and auto WarmCount of each cache was set to 0.

3.4.1.2.1 Defining the schema of the Solr core

The following field types were defined to store data in Solr.

- text_general - This is implemented using solr TextField data type. The standard tokenizer is being used.
- text_shingle_2 - This is implemented using solr TextField data type and ShingleFilterFactory at with minShingleSize="2" and maxShingleSize="2"
- text_shingle_3 - This is implemented using solr TextField data type and ShingleFilterFactory with minShingleSize="2" and maxShingleSize="2"

To evaluate the performance, fields shown in table 1 were added to documents.

Table 9: Schema of Solr database

Field name	**Field type**
id	string
date	date
topic	text_general
author	text_general
link	text_general
content	text_general
content_shingled_2	text_shingle_2
content_shingled_3	text_shingle_3

For each document, a unique id of 7 characters was generated. All fields except links were indexed and all the fields were sorted.

3.4.1.3 Setting up H2

H2 is a relational database. H2 can be used in embedded and server modes. Here we have used the server mode. We used H2 version 1.3.176 for this performance analysis. Evaluation was done using JDBC driver version 1.4.182. Cache was set to 0 Mb.

We used the relational schema shown in figure 13 for this performance analysis.

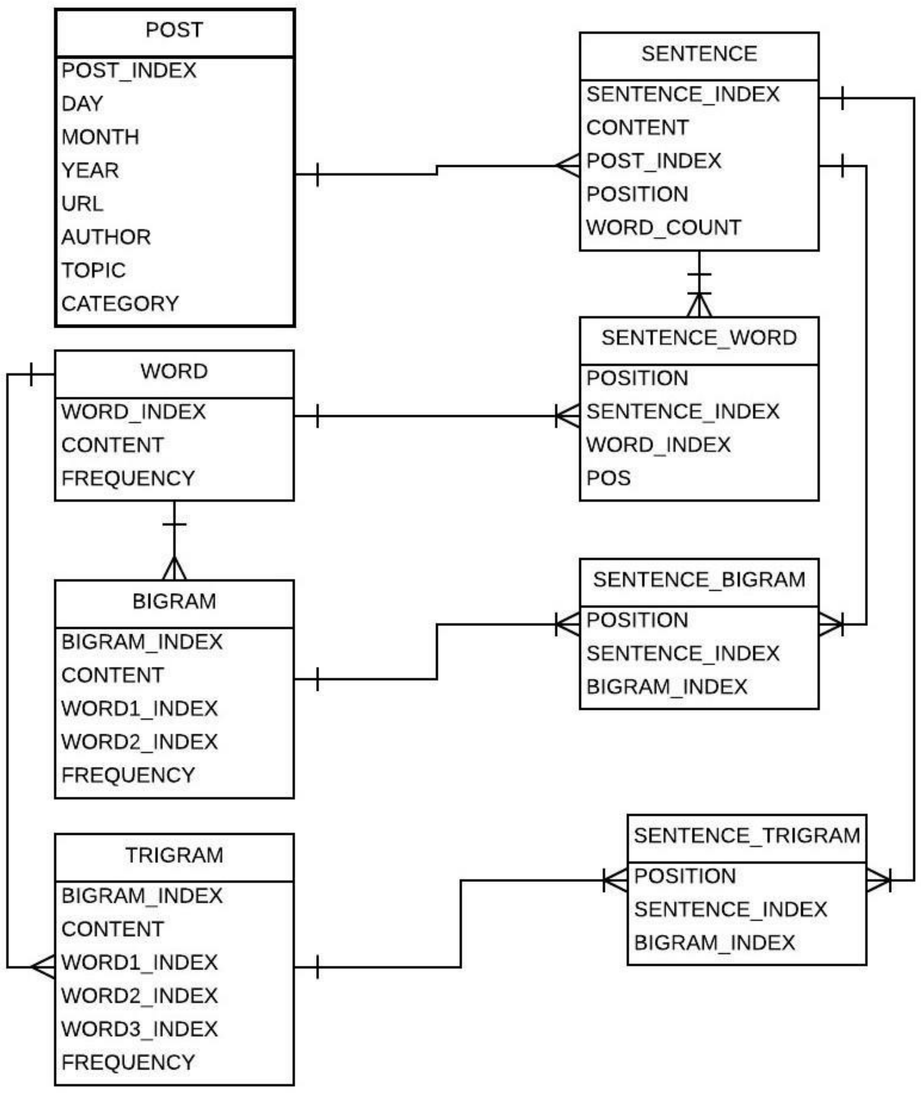


Figure 13: Schema used for H2 database

We used the indexes given below in the database.

- Index on CONTENT column of the WORD table
- Index on FREQUENCY column of the WORD table
- Index on YEAR column of the POST table
- Index on CATEGORY column of the POST table
- Index on CONTENT column of the BIGRAM table
- Index on WORD1_INDEX column of the BIGRAM table
- Index on WORD2_INDEX column of the BIGRAM table
- Index on CONTENT column of the TRIGRAM table
- Index on WORD1_INDEX column of the TRIGRAM table
- Index on WORD2_INDEX column of the TRIGRAM table

- Index on WORD3_INDEX column of the TRIGRAM table
- Index on POSITION column of the SENTENCE_WORD table

When inserting data to the database we dropped all the indexes except the one on CONTENT column of the WORD table. The indexes were recreated when retrieving information.

3.4.1.4 Setting up Neo4J

Neo4j is a graph database system that can effectively store data in a graph structure and retrieve data from the graph structure using cypher queries. We used Neo4j community distribution 2.1.2 licensed under GPLv3. Evaluation was done using java embedded Neo4j database mode which can pass database queries through a java API with a heap size of 4048 MB. We measured performance using both warm cache mode and cold cache mode. In warm cache mode, all caches were cleared and a set of warm up queries were run before running each actual query. In cold cache mode, queries were run with an empty cache. We did batch insertion through csv import function with pre generated csv files.

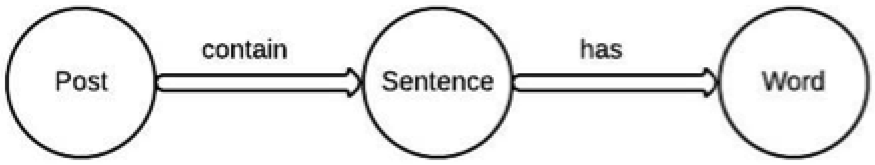


Figure 14: Graph structure used in Neo4j

For nodes following properties were assigned

● Post - post_id, topic, category, year, month, day, author and url
● Sentence - sentence_id, word_count
● Word - word_id, content, frequency

Relationships were created among nodes with properties included

● Contain - position
● Has - position

To feed data into the database we used database dumps as CSV files of a relational database using its CSV import function.

● Word.csv includes word_id, content and frequency
● Post.csv includes post_id, topic, category, year, month, day, author and url
● Sentence.csv includes sentence_id, post_id, position and word_count
● Word_Sentence.csv includes word_id, sentence-_id and position

Evaluations were done for both data insertion times and data retrieval times. Data retrieval time was calculated by measuring the execution time for each cypher query. For each cypher query, two time measurements were recorded for warm cache mode and cold cache mode.

3.4.1.5 Results and observations

Fig. 15 shows the comparison between the data insertion time for each data storage mechanism. Fig. 16 and Fig.17 illustrate the comparison between times taken for each information retrieval scenario in each data storage mechanism. When plotting graphs, we have ignored relatively higher values to increase the clarity of graphs.

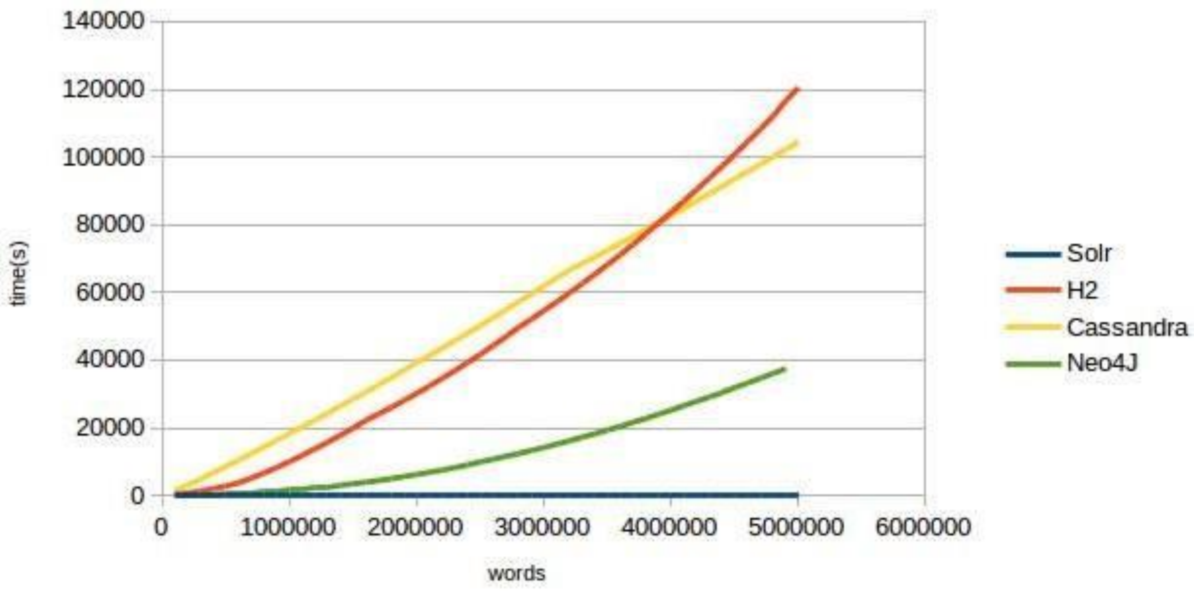


Figure 15: Data insertion time in each data storage system

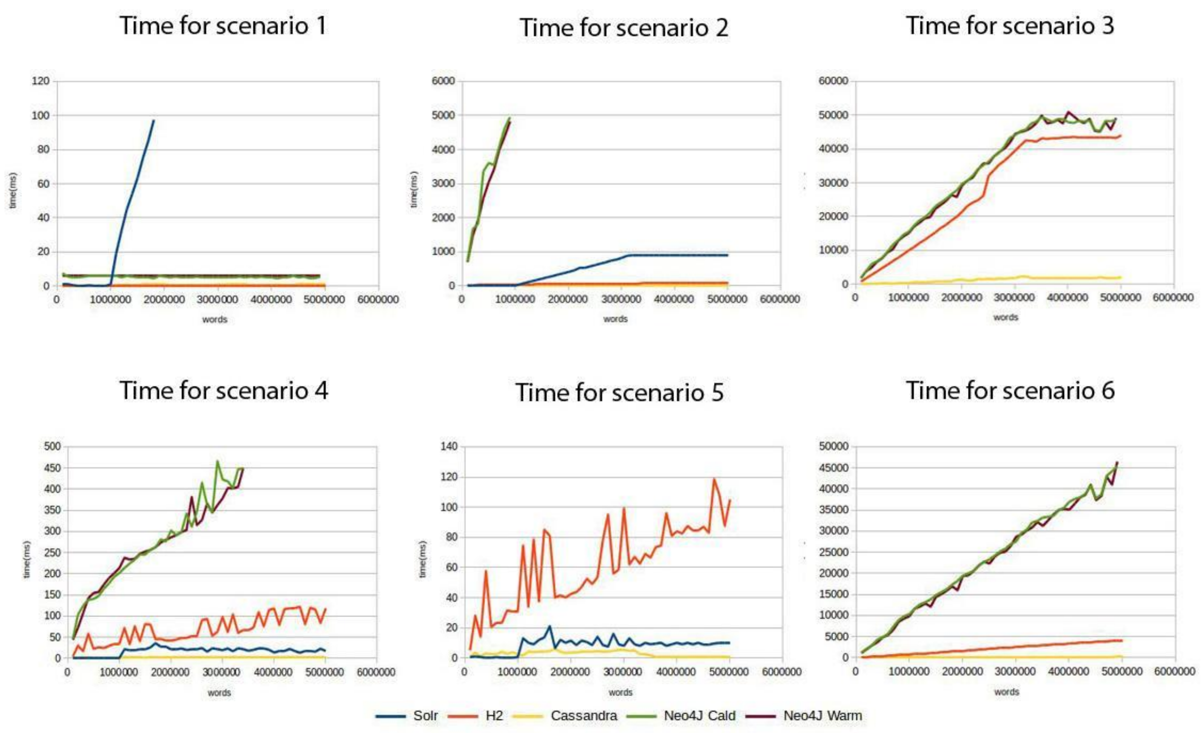


Figure 16: Data retrieval time for each scenario in each data storage system - part 1

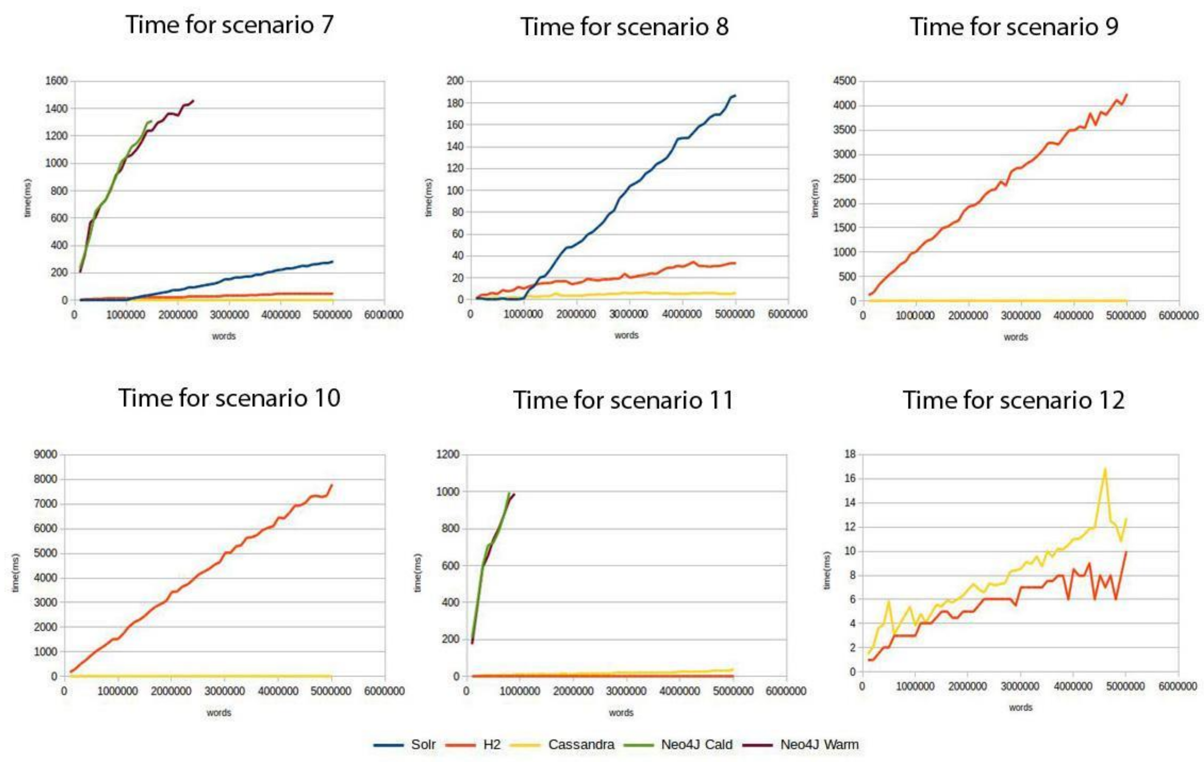


Figure 17: Data retrieval time for each scenario in each data storage system - part 2

Based on our observations, from the data insertion point of view, Solr performed better than the other three databases with a significant amount time gap (Solr: H2:Neo4j: Cassandra= 1:2127:495:3256). That means Solr is approximately 2127 times faster than H2, 495 times faster than Neo4j and 3256 times faster than Cassandra with respect to data insertion time.

From the data retrieval point of view, Cassandra performed better than other databases. In a couple of scenarios H2 outperformed Cassandra. But in those scenarios also Cassandra showed a considerably good speed. Neo4j didn't perform well in any scenario so it is not suitable to use in a structured dataset like a language corpus. Solr also marked a decent performance in some scenarios but there were issues in implementing it in other scenarios because its underlying indexing mechanism didn't provide the necessary support .

The only issue with Cassandra is its problems in supporting new queries. If we have another information need other than the above mentioned scenarios, we have to create new column families in Cassandra that support a given information need and insert data from the beginning which is a very expensive process.

Considering the above facts Cassandra was chosen as the data storage system for SinMin corpus because

1. Data insertion time of Cassandra is linear which can be very effective in the long term data insertion process
2. Performed well in 10 scenarios out of all 12 scenarios.

3.4.2 Data storage architecture of SinMin

According to the study described in section 3.4.1, Cassandra is the suitable candidate for the storage system of SinMin. So our main focus is towards the Cassandra data model design. There it uses a separate column family for each information need while keeping redundant data in different column families. Because Cassandra can't provide support immediately for new requirements, we keep an Oracle instance running as a backup database that can be used when new requirements occur. Another Solr instance was also designed in order to support wildcard search using Permuterm indexing.

3.4.2.1 Cassandra data model

The Following table shows the information needs of the corpus and column families defined to fulfill those needs with corresponding indexing.

Table 9: Cassandra data model with information needs

Information need	**Corresponding column family with indexing**
Get frequency of a given word in a given time period and given category	corpus.word_time_category_frequency (id bigint, word varchar, year int, category varchar, frequency int, PRIMARY KEY(word,year, category))
Get frequency of a given word in a given time period	corpus.word_time_category_frequency (id bigint, word varchar, year int, frequency int, PRIMARY KEY(word,year))
Get frequency of a given word in a given category	corpus.word_time_category_frequency (id bigint, word varchar, category varchar, frequency int, PRIMARY KEY(word, category))
Get frequency of a given word	corpus.word_time_category_frequency (id bigint, word varchar, frequency int, PRIMARY KEY(word))
Get frequency of a given bigram in given time period and given category	corpus.bigram_time_category_frequency (id bigint, word1 varchar, word2 varchar, year int, category int, frequency int, PRIMARY KEY(word1,word2,year, category))
Get frequency of a given bigram in given time period	corpus.bigram_time_category_frequency (id bigint, word1 varchar, word2 varchar, year int, frequency int, PRIMARY KEY(word1,word2,year))
Get frequency of a given bigram in given category	corpus.bigram_time_category_frequency (id bigint, word1 varchar, word2 varchar, category varchar, frequency int, PRIMARY KEY(word1,word2, category))
Get frequency of a given bigram	corpus.bigram_time_category_frequency (id bigint, word1 varchar, word2 varchar, frequency int, PRIMARY KEY(word1,word2))
Get frequency of a given trigram in given time period and in a given category	corpus.trigram_time_category_frequency (id bigint, word1 varchar, word2 varchar, word3 varchar, year int, category int, frequency int, PRIMARY KEY(word1,word2,word3,year, category))
Get frequency of a given trigram in given time period	corpus.trigram_time_category_frequency (id bigint, word1 varchar, word2 varchar, word3 varchar, year int,

	frequency int, PRIMARY KEY(word1,word2,word3,year))
Get frequency of a given trigram in a given category	corpus.trigram_time_category_frequency (id bigint, word1 varchar, word2 varchar, word3 varchar, category varchar, frequency int, PRIMARY KEY(word1,word2,word3, category))
Get frequency of a given trigram	corpus.trigram_time_category_frequency (id bigint, word1 varchar, word2 varchar, word3 varchar, frequency int, PRIMARY KEY(word1,word2,word3))
Get most frequently used words in a given time period and in a given category	corpus.word_time_category_ordered_frequency (id bigint, word varchar, year int, category int, frequency int, PRIMARY KEY((year, category),frequency,word))
Get most frequently used words in a given time period	corpus.word_time_category_ordered_frequency (id bigint, word varchar, year int,frequency int, PRIMARY KEY(year,frequency,word))
Get most frequently used words in a given category, Get most frequently used words	corpus.word_time_category_ordered_frequency (id bigint, word varchar,category varchar, frequency int, PRIMARY KEY(category,frequency,word))
Get most frequently used bigrams in a given time period and in a given category	corpus.bigram_time_ordered_frequency (id bigint, word1 varchar, word2 varchar, year int, category varchar, frequency int, PRIMARY KEY((year,category),frequency,word1,word2))
Get most frequently used bigrams in a given time period	corpus.bigram_time_ordered_frequency (id bigint, word1 varchar, word2 varchar, year int, frequency int, PRIMARY KEY(year,frequency,word1,word2))
Get most frequently used bigrams in a given category, Get most frequently used bigrams	corpus.bigram_time_ordered_frequency (id bigint, word1 varchar, word2 varchar, category varchar, frequency int, PRIMARY KEY(category) ,frequency,word1,word2))
Get most frequently used trigrams in a given time period and in a given category	corpus.trigram_time_category_ordered_frequency (id bigint, word1 varchar, word2 varchar, word3 varchar, year int, category varchar, frequency int, PRIMARY KEY((year, category),frequency,word1,word2,word3))
Get most frequently used trigrams in a given time period	corpus.trigram_time_category_ordered_frequency (id bigint, word1 varchar, word2 varchar, word3 varchar, year int,frequency int, PRIMARY KEY(year,frequency,word1,word2,word3))
Get most frequently used trigrams in a given category	corpus.trigram_time_category_ordered_frequency (id bigint, word1 varchar, word2 varchar, word3 varchar, category varchar, frequency int, PRIMARY KEY(

	category,frequency,word1,word2,word3))
Get latest key word in contexts for a given word in a given time period and in a given category	corpus.word_year_category_usage (id bigint, word varchar, year int, category varchar, sentence varchar, postname text, url varchar, date timestamp, PRIMARY KEY((word,year,category),date,id))
Get latest key word in contexts for a given word in a given time period	corpus.word_year_category_usage (id bigint, word varchar, year int, sentence varchar, postname text, url varchar, date timestamp, PRIMARY KEY((word,year),date,id))
Get latest key word in contexts for a given word in a given category	corpus.word_year_category_usage (id bigint, word varchar, category varchar, sentence varchar, postname text, url varchar, date timestamp, PRIMARY KEY((word,year),date,id))
Get latest key word in contexts for a given word	corpus.word_year_category_usage (id bigint, word varchar,sentence varchar, postname text, url varchar, date timestamp, PRIMARY KEY(word,date,id))
Get latest key word in contexts for a given bigram in a given time period and in a given category	corpus.bigram_year_category_usage (id bigint, word1 varchar, word2 varchar, year int, category varchar, sentence varchar, postname text, url varchar, date timestamp, PRIMARY KEY((word1,word2,year,category),date,id))
Get latest key word in contexts for a given bigram in a given time period	corpus.bigram_year_category_usage (id bigint, word1 varchar, word2 varchar, year int, sentence varchar, postname text, url varchar, date timestamp, PRIMARY KEY((word1,word2,category),date,id))
Get latest key word in contexts for a given bigram in a given category	corpus.bigram_year_category_usage (id bigint, word1 varchar, word2 varchar, category varchar, sentence varchar, postname text, url varchar, date timestamp, PRIMARY KEY((word1,word2,category),date,id))
Get latest key word in contexts for a given bigram	corpus.bigram_year_category_usage (id bigint, word1 varchar, word2 varchar, sentence varchar, postname text, url varchar, date timestamp, PRIMARY KEY((word1,word2),date,id))
Get latest key word in contexts for a given trigram in a given time period and in a given category	corpus.trigram_year_category_usage (id bigint, word1 varchar, word2 varchar, word3 varchar, year int, category varchar, sentence varchar, postname text, url varchar, date timestamp, PRIMARY KEY((word1,word2,word3,year,category),date,id))
Get latest key word in contexts for a given trigram in a given time period	corpus.trigram_year_category_usage (id bigint, word1 varchar, word2 varchar, word3 varchar, year int, sentence varchar, postname text, url varchar, date timestamp,

	PRIMARY KEY((word1,word2,word3,year),date,id))
Get latest key word in contexts for a given trigram in a given category	corpus.trigram_year_category_usage (id bigint, word1 varchar, word2 varchar, word3 varchar,category varchar, sentence varchar, postname text, url varchar, date timestamp, PRIMARY KEY((word1,word2,word3,category),date,id))
Get latest key word in contexts for a given trigram	corpus.trigram_year_category_usage (id bigint, word1 varchar, word2 varchar, word3 varchar, sentence varchar, postname text, url varchar, date timestamp, PRIMARY KEY((word1,word2,word3),date,id))
Get most frequent words at a given position of a sentence	corpus.word_pos_frequency (id bigint, content varchar, position int, frequency int, PRIMARY KEY(position, frequency, content)) corpus.word_pos_id (id bigint, content varchar, position int, frequency int, PRIMARY KEY(position, content))
Get most frequent words at a given position of a sentence in a given time period	corpus.word_pos_frequency (id bigint, content varchar, position int, year int, frequency int, PRIMARY KEY((position,year), frequency, content)) corpus.word_pos_id (id bigint, content varchar, position int, year int,frequency int, PRIMARY KEY(position,year,content))
Get most frequent words at a given position of a sentence in a given category	corpus.word_pos_frequency (id bigint, content varchar, position int, category varchar,frequency int, PRIMARY KEY((position,category), frequency, content)) corpus.word_pos_id (id bigint, content varchar, position int, category varchar,frequency int, PRIMARY KEY(position, category,content))
Get most frequent words at a given position of a sentence in a given time period and in a given category	corpus.word_pos_frequency (id bigint, content varchar, position int, year int, category varchar,frequency int, PRIMARY KEY((position,year,category), frequency, content)) corpus.word_pos_id (id bigint, content varchar, position int, year int, category varchar,frequency int, PRIMARY KEY(position,year,category,content))
Get the number of words in the corpus in a given category and year	CREATE TABLE corpus.word_sizes (year varchar, category varchar, size int, PRIMARY KEY(year,category));

3.4.2.2 Oracle data model

Unlike Cassandra, the Oracle data model is designed so that it can support more general queries. This allows more flexibility in querying and retrieving data. Figure 18 shows a database diagram for the Oracle data model.

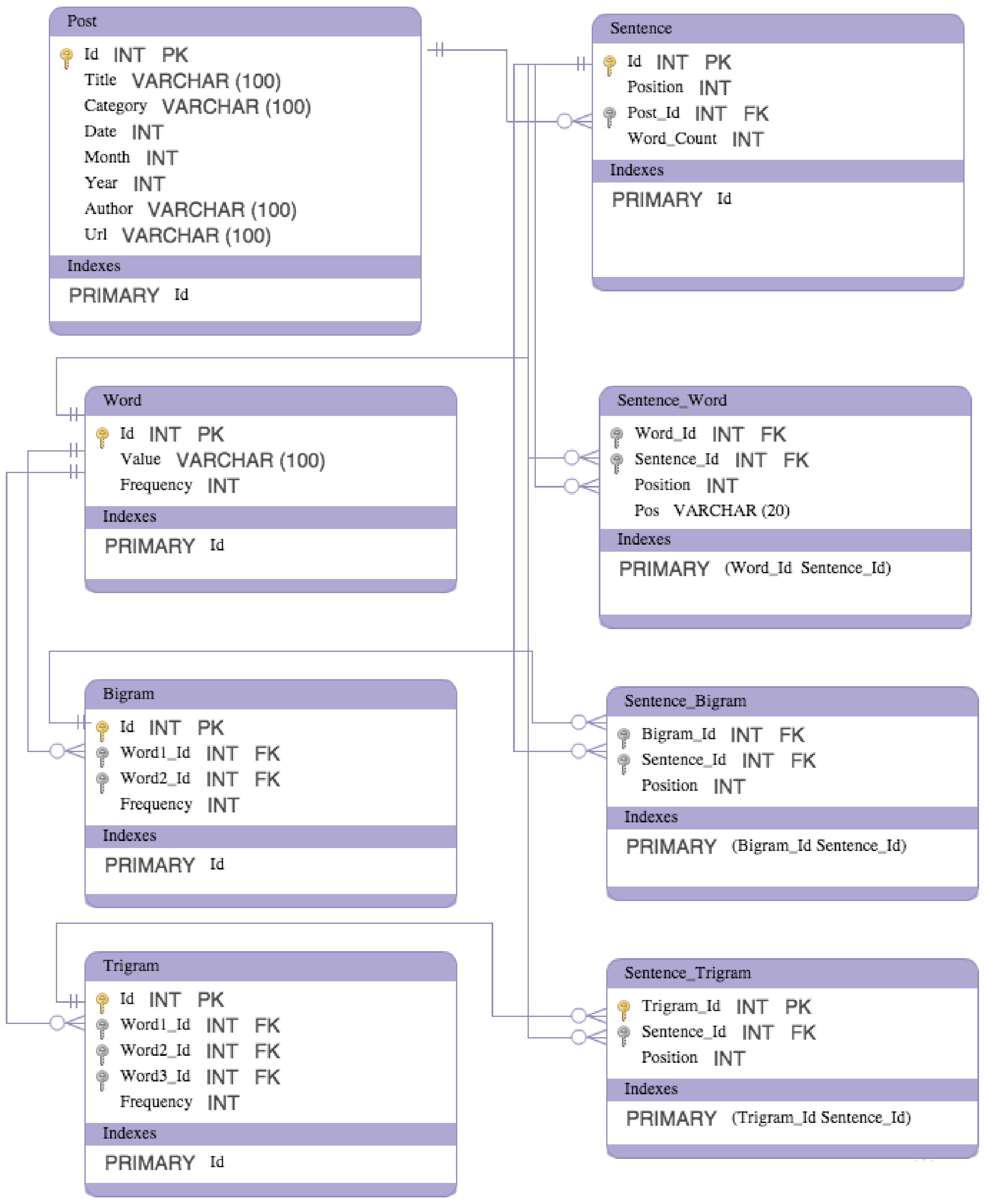


Figure 18: Schema used for Oracle database

3.4.2.3 Solr data model

Apache Solr is used to implement the wildcard search feature of the corpus. Apache Solr version 4.10.2 was used. Below is the schema of the Solr database. Table 10 shows the schema of the Solr database.

Table 10: Schema of Solr database

Field name	**Field type**
id	string
content	text_rvswc
content_encoded	text_rvswc
frequency	text_general

As there are about 1.3 million distinct words a unique id of 7 characters is generated for each word.

Solr provides a feature to do wildcard searching without any permuterm indices. However the performance can degrade if multiple search characters appear. This search can be efficiently done using permuterm indexing.

Therefore Solr.ReversedWildcardFilterFactory is used. It generates permuterm indices for each word at the insertion of data. The definition of ‘text_rvswc’ is below.

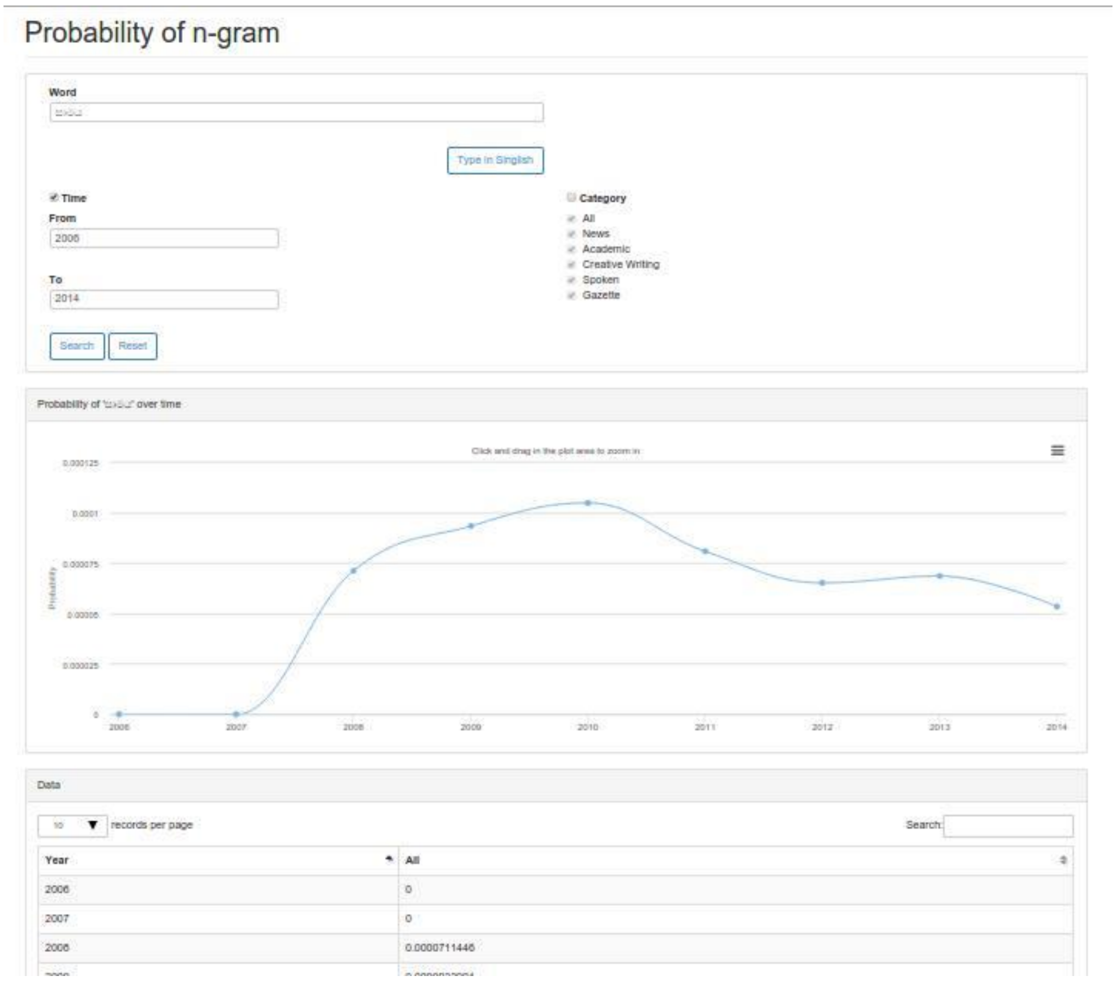
```
<fieldType name="text_rvswc" class="solr.TextField"
positionIncrementGap="100">
     <analyzer type="index">
         <tokenizer class="solr.WhitespaceTokenizerFactory"/>
         <filter class="solr.ReversedWildcardFilterFactory"
withOriginal="true"
            maxPosAsterisk="10" maxPosQuestion="10" minTrailing="10"
maxFractionAsterisk="0"/>
     </analyzer>
     <analyzer type="query">
         <tokenizer class="solr.WhitespaceTokenizerFactory"/>
     </analyzer>
 </fieldType>
```

At the indexing phase, the ReversedWildcardFilterFactory generates the possible patterns of the word. It will support at most 10 asterisk marks (*) and 10 question marks (?). The minimum number of trailing characters supported in a query after the last wildcard character is also 10.

3.5 API and User Interface Design

3.5.1 User interface design

We have designed the web interface of SinMin for the users who prefer a summarized and visualized view of the statistical data of the corpus. The Visual design of the interface has been made so that a user without prior experience of the interface will be able to fulfill his information requirements with a little effort.



Probability of n-gram

Word: [illegible]

Type in Singlish

✓ Time

From: 2006

To: 2014

Category: ✓ All ✓ News ✓ Academic ✓ Creative Writing ✓ Spoken ✓ Gazette

Search Reset

Probability of '[illegible]' over time

Data

10 records per page Search:

Year	All
2006	0
2007	0
2008	0.0000711446

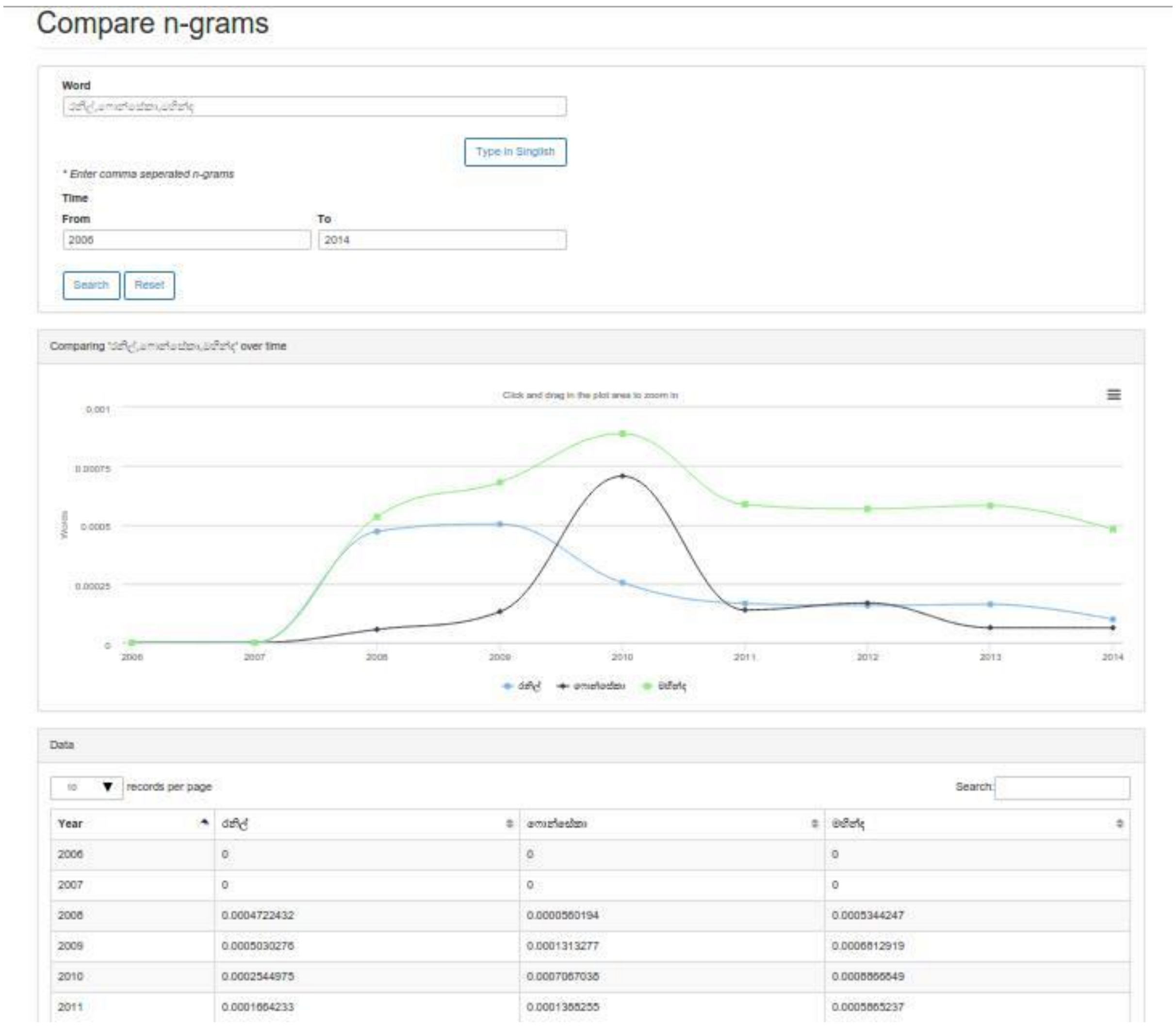
Figure 19: User interface for retrieving of the probability of a n-gram

Here we have included the designs of some sample user interfaces of the SinMin.

User interface in the Figure 19 can be used to get the fraction of occurrences of a particular word according to the category and the time period. Results are represented as a graph and a table. Figure shows the graph and table generated when a user selects to search from the time period only.

Figure 20 shows the user interface that can be used to compare the usage patterns of two n-grams. User can specify the desired time period for the search.

Figure 21 shows the user interface that can be used to find the most frequent ten words that comes after a given n-gram. Fraction of usage of that particular n-gram (given n-gram with the words retrieved) is graphed according to the time.


Compare n-grams

Word

[illegible]

* Enter comma seperated n-grams

Time

From: 2006

To: 2014

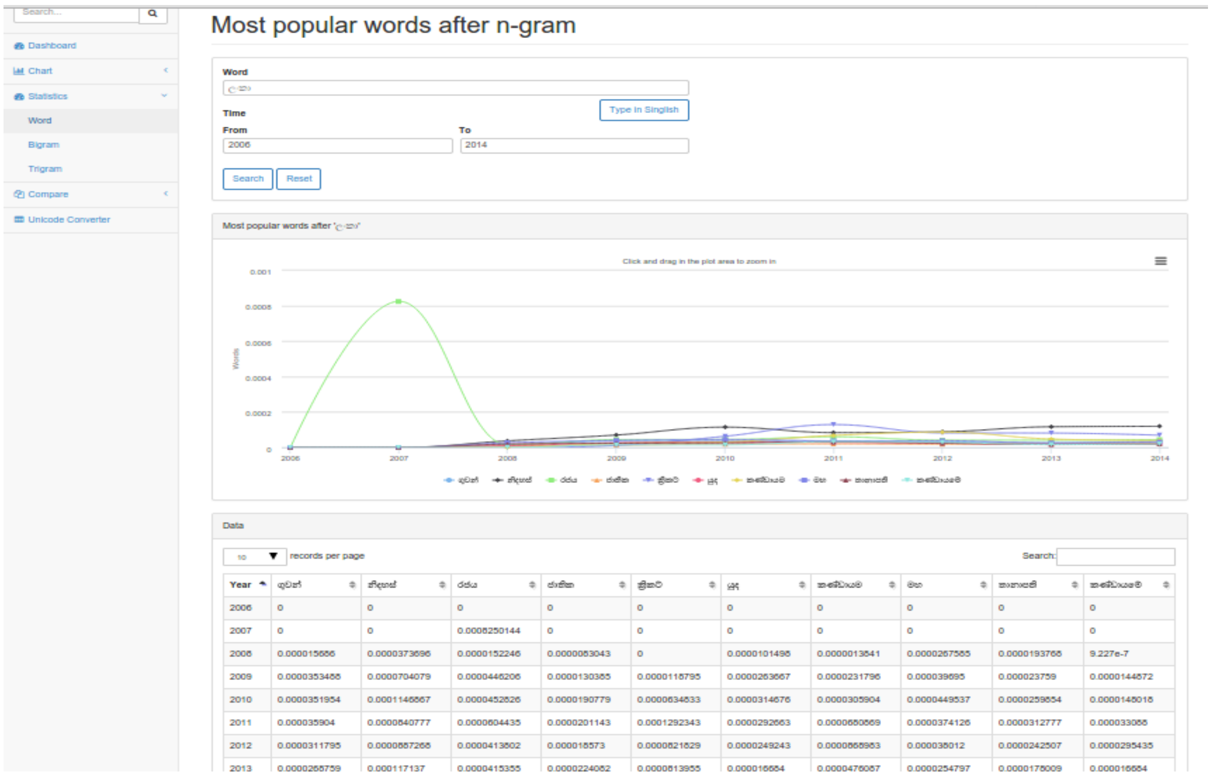
Comparing '[illegible]' over time

Data

10 records per page

Year	[illegible]	[illegible]	[illegible]
2006	0	0	0
2007	0	0	0
2008	0.0004722432	0.0000560194	0.0005344247
2009	0.0005030276	0.0001313277	0.0006812919
2010	0.0002544975	0.0007067036	0.0008866849
2011	0.0001664233	0.0001366255	0.0005865237

Figure 20: User interface for comparing n-grams



Most popular words after n-gram

Word: ලංකා

From: 2006 To: 2014

Most popular words after 'ලංකා'

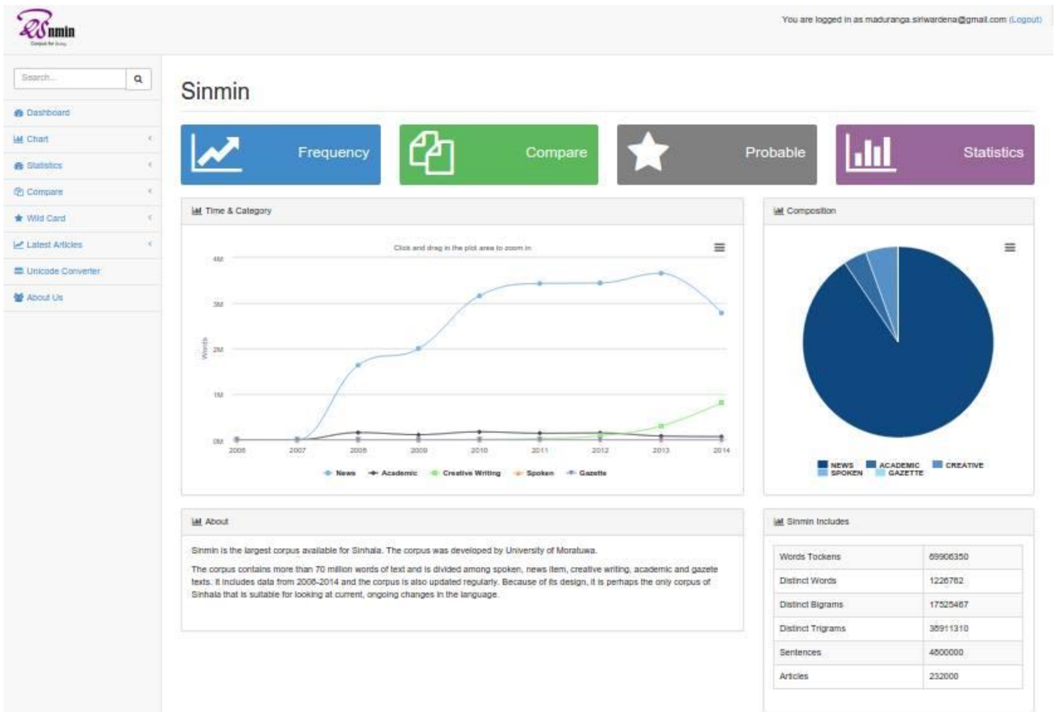
Data

10 records per page Search:

Year	ගුවන්	නිදහස්	රජය	ජාතික	ක්‍රිකට්	පුද	කණ්ඩායම	මහ	සංචාරක	කණ්ඩායමේ
2006	0	0	0	0	0	0	0	0	0	0
2007	0	0	0.0008250144	0	0	0	0	0	0	0
2008	0.000015686	0.0000373696	0.0000152246	0.0000083043	0	0.0000101498	0.0000013841	0.0000267565	0.0000193768	9.227e-7
2009	0.0000353488	0.0000704079	0.0000446206	0.0000130385	0.0000118795	0.0000263667	0.0000231796	0.000039695	0.000023759	0.0000144872
2010	0.0000351954	0.0001146867	0.0000452626	0.0000190779	0.0000634633	0.0000314676	0.0000305904	0.0000449537	0.0000259654	0.0000148018
2011	0.000035904	0.0000840777	0.0000604435	0.0000201143	0.0001292343	0.0000292663	0.0000680869	0.0000374126	0.0000312777	0.000033088
2012	0.0000311795	0.0000887268	0.0000413602	0.000018573	0.0000821629	0.0000249243	0.0000868983	0.000038012	0.0000242507	0.0000295435
2013	0.0000268759	0.000117137	0.0000415355	0.0000224082	0.0000813955	0.000016684	0.0000476087	0.0000254797	0.0000178009	0.000016684

Figure 21: User interface for retrieving most popular words comes after a n-grams

A dashboard in the web interface of SinMin is shown in the figure 22. It shows the composition of words according the categories in the corpus and the number of words inserted in each year.



You are logged in as maduranga.siriwardena@gmail.com (Logout)

Sinmin

About

Sinmin is the largest corpus available for Sinhala. The corpus was developed by University of Moratuwa.

The corpus contains more than 70 million words of text and is divided among spoken, news item, creative writing, academic and gazete texts. It includes data from 2006-2014 and the corpus is also updated regularly. Because of its design, it is perhaps the only corpus of Sinhala that is suitable for looking at current, ongoing changes in the language.

Sinmin Includes

Words Tockens	69906350
Distinct Words	1226762
Distinct Bigrams	17525467
Distinct Trigrams	38911310
Sentences	4800000
Articles	232000

Figure 22: Dashboard

3.5.2 API design

Showing the content and analytics of the corpus using a web portal is not enough when it comes to integrating the corpus with other consumer applications. To enhance the adaptability of SinMin we designed a comprehensive REST API for SinMin. Due to various information needs and to enhance the data retrieval speed, API depends on multiple database like Cassandra, Oracle and Apache Solr which contains same dataset.

Cassandra is the primary storage system that is used by API. Most of the common and performance intensive data requirements are fetched through Cassandra database. Because Cassandra column families are designed according to the defined set of data requirements, it can perform only in that subset of requirements. If a new requirement comes into the system, it takes some time to integrate it into the Cassandra because we have to create one or more separate column families and feed data to them from the beginning. To overcome this issue, an Oracle instance is used as a backing up storage system. Unlike Cassandra, Oracle instance has a generic table structure that can give support to almost all data requirements. Because of this generic structure, Oracle needs to do a large number of table joining in order to give results for data requirements. This increases the latency of Oracle than Cassandra. Apache Solr is used as a prefix search engine because in some cases we need to find words with its first few prefixes and wild cards. Solr's permuterm indexing can be used very effectively for these kind of requirements.

Figure 23 shows the architecture of the API. Requests from outside comes to API. API passes it to Request Dispatcher to separate requests according to the suitable database system. Request Dispatcher provides a simplified and uniform interface to the API by hiding underlying complexities of different databases. Each database is connected to Request Dispatcher through a dedicated adapter that is capable of translating data requirements into actual database queries and passing them to database.

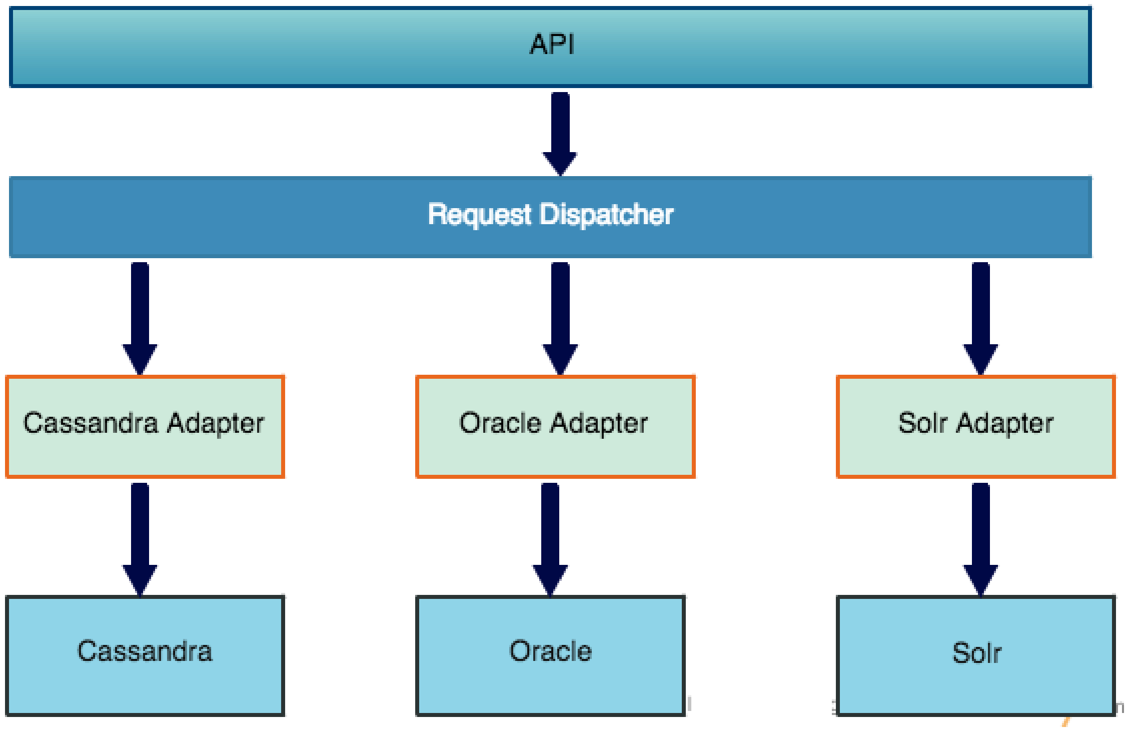


Figure 23: Architecture of the API

3.5.2.1 API functions

Name	**Description**
wordFrequency	Returns frequency of a given word over given time and category
bigramFrequency	Returns frequency of a given bigram over given time and category
trigramFrequency	Returns frequency of a given trigram over given time and category
frequentWords	Returns most frequent set of words over given time and category
frequentBigrams	Returns most frequent set of bigrams over given time and category
frequentTrigrams	Returns most frequent set of trigrams over given time

	and category
latestArticlesForWord	Returns latest articles that includes a given word over given time and category
latestArticlesForBigram	Returns latest articles that includes a given bigram over given time and category
latestArticlesForTrigram	Returns latest articles that includes a given trigram over given time and category
frequentWordsAroundWord	Returns most frequent words that appear around a given word and range over given time and category
frequentWordsInPosition	Returns most frequent words that appear in a given position of a sentences over given time and category
frequentWordsInPositionReverse	Returns most frequent words that appear in a given position (from reverse) of a sentences over given time and category
frequentWordsAfterWordTimeRange	Returns frequency of a given word over a given time range and category
frequentWordsAfterBigramTimeRange	Returns frequency of a given bigram over a given time range and category
wordCount	Returns word count of over a given time and category
bigramCount	Returns bigram count of over a given time and category
trigramCount	Returns trigram count of over a given time and category

Table 11: API functions

4.0 Implementation

Chapter 4 presents the Implementation details of SinMin. This describes the details of the approaches and tools we used in the implementation as well as the models we used, various management tools that were used to manage code, builds and code quality.

4.1 Crawler Implementation (Technology and process)

4.1.1 News items crawler

Web crawlers were implemented using java. We have used maven as the build tool. HTML parsing is done using *jsoup 1.7.3* library and date handling process is done using *joda time 2.3* library. Handling the XML is done using *Apache Axiom 1.2.14*. Writing the raw XML data into the files in a human readable format is done using *StAX Utilities* library version *20070216*.

Procedure of crawling a particular online Sinhala source is described here. From the user interface of the crawler controller we specify to crawl a source from a particular date to another. Crawler controller finds the path of that jar from the database using the ID of the crawler. Then it runs the jar with the specified time period as parameters, after which the crawler class asks for a web page from the URL generator class. So the URL generator class tries to get a URL from its list. Since this is the first time of crawling, the list is empty. So it tries to find URLs to add to its listand it seeks   a page that lists a set of articles as described in the design section. URL generator knows the URL format of this particular page and it generates the URL for the first page of the specified time period. As the URL generator now has a URL of the page that list a set of articles, it extracts all the URLs of articles in that page and adds them to the list of URLs it keeps. Now the URL generator returns a web page to the crawler class. Crawler class now adds this page to its XML file writer.

Data extraction from the web page is initiated by the XML file writer. The XML file writer has a HTML parser written specifically to extract the required details from a web page of this particular online source. The XML file writer asks the parser to extract the article content and the required meta data from the web page. All the data is added to an article element (OMElements objects are used in Axiom to handle XML elements) and the article is added to the document element.

The crawler continuously asks for web pages and the URL generator returns pages from the specified time period. When the URL generator finishes sending back the web pages for particular date, it notifies the XML file writer about the status. So the XML file writer writes the articles of that particular date to a single file with filename as the date.

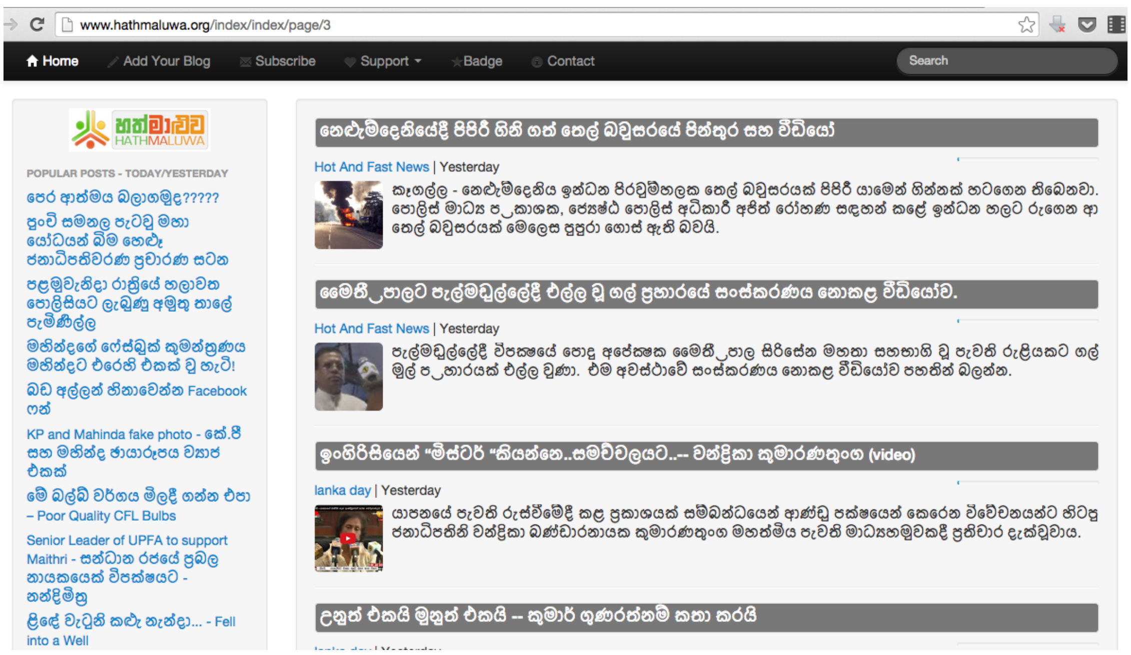
This procedure goes until there are no more web pages from the required time period. The same procedure is used to crawl all the online articles except for blogs which is described next.

4.1.2 Blog crawler

Crawling Blogs from the Sinhala Language is a relatively tricky task because

1. There is no pre-defined set of Sinhala Blogs available in the internet
2. New Sinhala Blogs are added to the web daily and it's hard to find them manually
3. Different blog sites have different page layouts so the traditional approach to get data from the site using HTML parsing may not work

To address problems of 1 and 2, instead of manually collecting blog URLs, we used the support of a Sinhala blog aggregator: hathmaluwa.org which is continuously updating with latest updates of different blog sites. We implemented a crawler for hathmaluwa.org to fetch URLs of blogs it is showing. Because it is continuously updating we keep track of the last position we did crawling in previous run.



POPULAR POSTS - TODAY/YESTERDAY

පෙර ආත්මය බලාගමුද?????

පුංචි සමනල පැටවු මහා යෝධයන් බිම හෙළූ ජනාධිපතිවරණ ප්‍රචාරණ සටන

පළමුවැනිදා රාත්‍රියේ හලාවත පොලිසියට ලැබුණු අමුතු තෑග්ගේ පැමිණිල්ල

මහින්දගේ ෆේස්බුක් කුමන්ත්‍රණය මහින්දට එරෙහි එකක් වූ හැටි!

බඩ අල්ලන් හිනාවෙන්න Facebook ෆන්

KP and Mahinda fake photo - කේ.පී සහ මහින්ද ඡායාරූපය ව්‍යාජ එකක්

මේ බල්බ් වර්ගය මිලදී ගන්න එපා – Poor Quality CFL Bulbs

Senior Leader of UPFA to support Maithri - සන්ධාන රජයේ ප්‍රබල නායකයෙක් විපක්ෂයට - නන්දිමිත්‍ර

ළිඳේ වැටුනි කළු නැන්දා... - Fell into a Well

නෙළුම්දෙනියේදී පිපිරී ගිනි ගත් තෙල් බවුසරයේ පින්තූර සහ වීඩියෝ

Hot And Fast News | Yesterday

කෑගල්ල - නෙළුම්දෙනිය ඉන්ධන පිරවුම්හලක තෙල් බවුසරයක් පිපිරී යාමෙන් ගින්නක් හටගෙන තිබෙනවා. පොලිස් මාධ්‍ය ප්‍රකාශක, ජ්‍යෙෂ්ඨ පොලිස් අධිකාරී අජිත් රෝහණ සඳහන් කළේ ඉන්ධන හලට රුගෙන ආ තෙල් බවුසරයක් මෙලෙස පුපුරා ගොස් ඇති බවයි.

මෛත්‍රීපාලට පැල්මඩුල්ලේදී එල්ල වූ ගල් ප්‍රහාරයේ සංස්කරණය නොකළ වීඩියෝව.

Hot And Fast News | Yesterday

පැල්මඩුල්ලේදී විපක්ෂයේ පොදු අපේක්ෂක මෛත්‍රීපාල සිරිසේන මහතා සහභාගි වූ පැවති රැළියකට ගල් මුල් ප්‍රහාරයක් එල්ල වුණා. එම අවස්ථාවේ සංස්කරණය නොකළ වීඩියෝව පහතින් බලන්න.

ඉංගිරිසියෙන් "මිස්ටර් "කියන්නෙ..සමච්චලයට..-- චන්ද්‍රිකා කුමාරණතුංග (video)

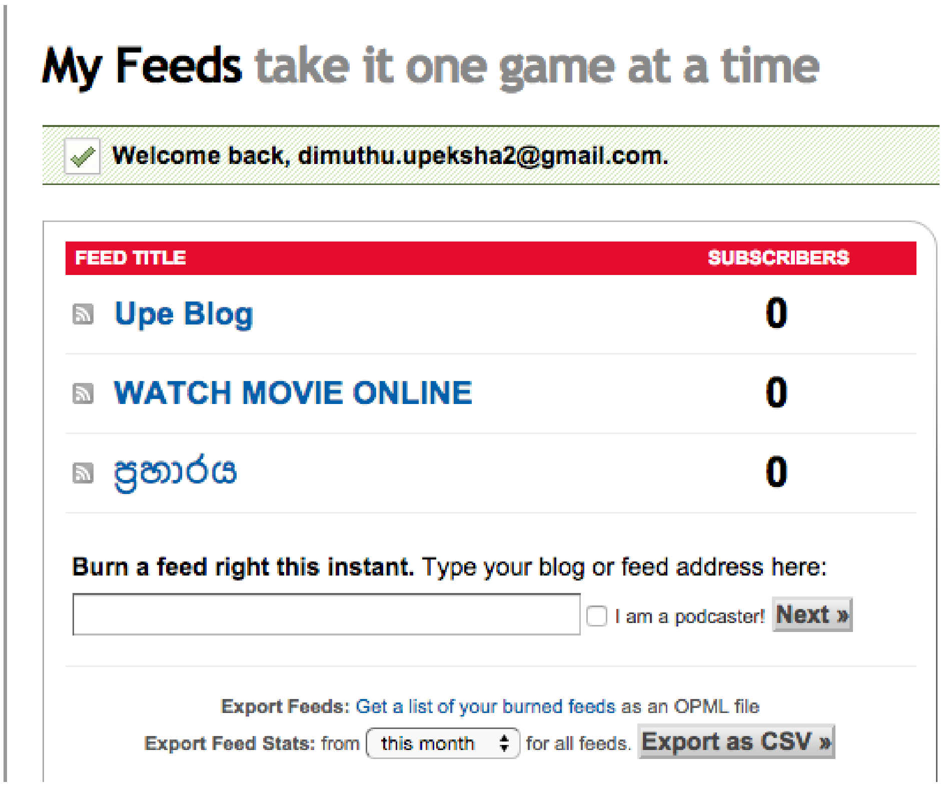
lanka day | Yesterday

යාපනයේ පැවති රැස්වීමේදී කළ ප්‍රකාශයක් සම්බන්ධයෙන් ආණ්ඩු පක්ෂයෙන් කෙරෙන විවේචනයන්ට හිටපු ජනාධිපතිනි චන්ද්‍රිකා බණ්ඩාරනායක කුමාරණතුංග මහත්මිය පැවති මාධ්‍යහමුවකදී ප්‍රතිචාර දැක්වූවාය.

උනුත් එකයි මුනුත් එකයි -- කුමාර් ගුණරත්නම් කතා කරයි

Figure 24: A sample page of Hathmluwa blog aggregator

After collecting URLs of blogs they are then passed through Google Feedburner to get the URL of RSS Feed. Using RSS feed, important metadata such as Blog ID can be retrieved. Using the ULS of RSS feeds, RSS feeds are fetched. This RSS feed is a XML file and it is parsed using java DOM parser. The main requirement of RSS feed is to extract blog id of the particular blog. Because Feedburner doesn't provide an API support to do this using a program, we had to consider another approach. We used Selenium web driver, a browser automation tool that can automate the function that we performed on a web page. We used a script to automate functions such as going to Feedburner site, logging in using Google credentials and storing session cookies, inserting blog URL in particular search box, searching for the feed URL and extracting feed URL.


My Feeds take it one game at a time

✓ Welcome back, dimuthu.upeksha2@gmail.com.

FEED TITLE	SUBSCRIBERS
Upe Blog	0
WATCH MOVIE ONLINE	0
ප්‍රහාරය	0

Burn a feed right this instant. Type your blog or feed address here:

Export Feeds: Get a list of your burned feeds as an OPML file

Export Feed Stats: from this month for all feeds. Export as CSV »

Figure 25: RSS feed tracking by Google FeedBurner

After fetching blog id of the blog site, the next goal is to extract the content of the blog site in a machine readable way. To do this task we used the Google blogger API. It requires the blog ID of the particular blog site and returns a set of blog post in JSON format.

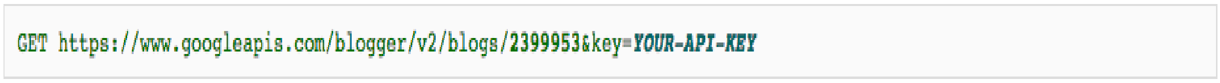
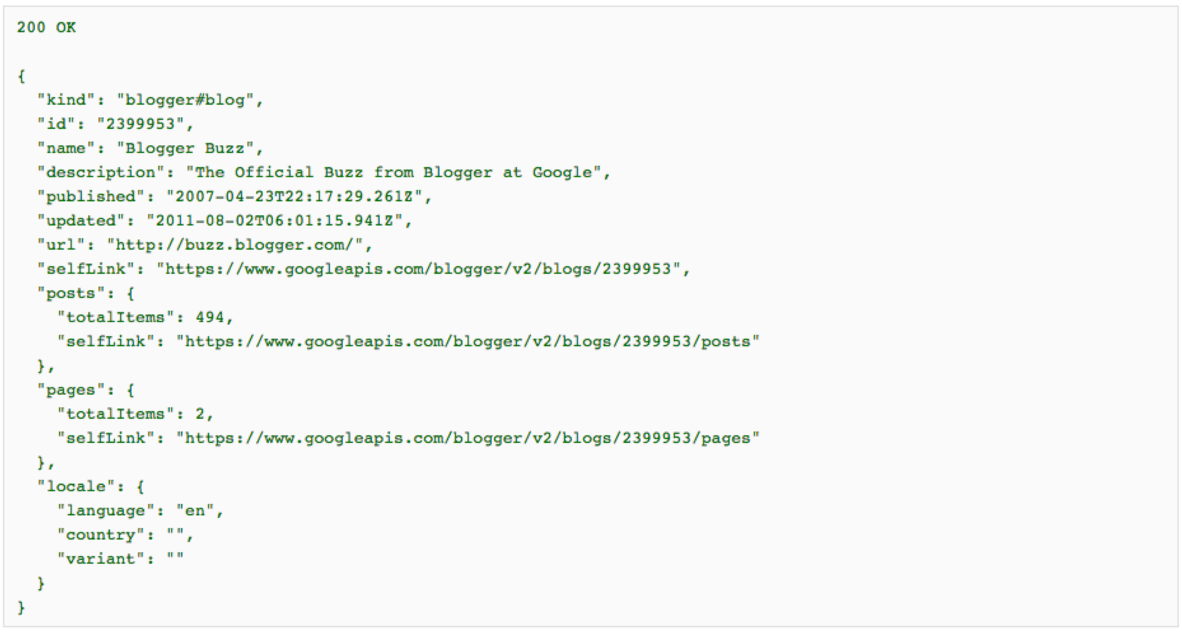

```
GET https://www.googleapis.com/blogger/v2/blogs/2399953&key=YOUR-API-KEY
```

Figure 26: Request format to get a blog by blogID



```
200 OK

{
  "kind": "blogger#blog",
  "id": "2399953",
  "name": "Blogger Buzz",
  "description": "The Official Buzz from Blogger at Google",
  "published": "2007-04-23T22:17:29.261Z",
  "updated": "2011-08-02T06:01:15.941Z",
  "url": "http://buzz.blogger.com/",
  "selfLink": "https://www.googleapis.com/blogger/v2/blogs/2399953",
  "posts": {
    "totalItems": 494,
    "selfLink": "https://www.googleapis.com/blogger/v2/blogs/2399953/posts"
  },
  "pages": {
    "totalItems": 2,
    "selfLink": "https://www.googleapis.com/blogger/v2/blogs/2399953/pages"
  },
  "locale": {
    "language": "en",
    "country": "",
    "variant": ""
  }
}
```

Figure 27: Sample response of a blog entity

This content is parsed using a JSON parser and necessary content and metadata is extracted. But still some extracted parts had some unnecessary sections like html tags and long full stops. So those contents were filtered using filters implemented in SinMin Corpus Tools project raw data was stored in XML files for further processing.

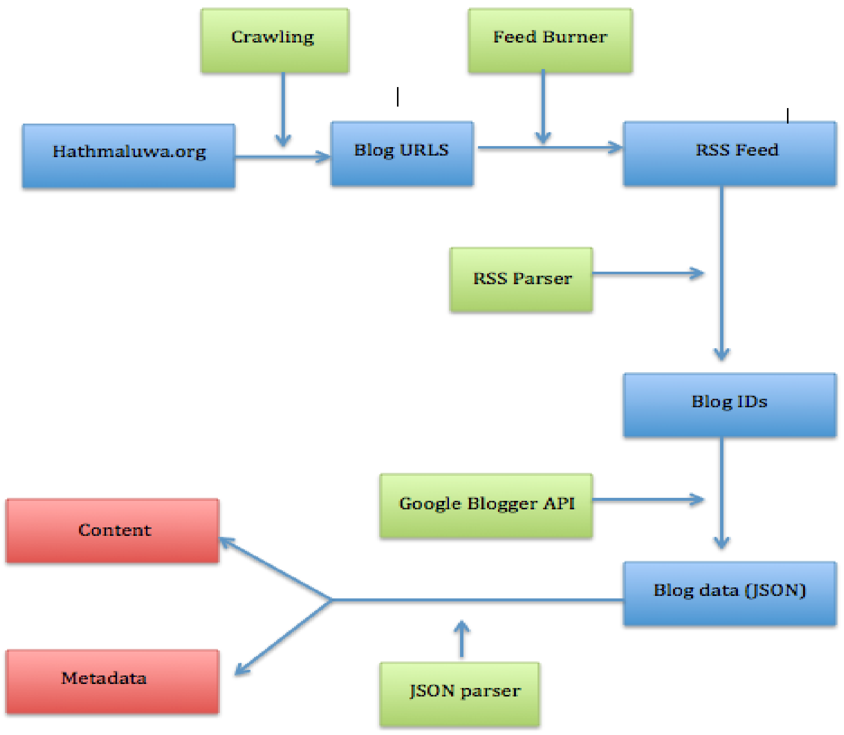


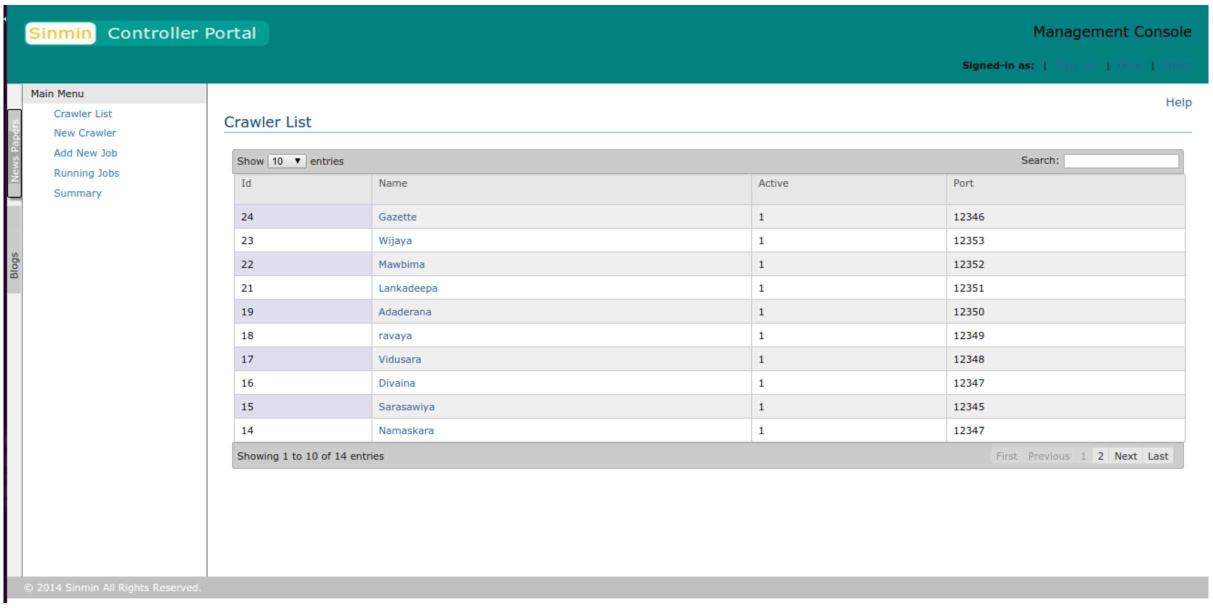
Figure 28: Architecture of blog crawler

4.2 Crawl Controller

Crawler controller is the component that is responsible for managing the web crawlers of SinMin. It consists of two components, front end component and the back end component.

The front end component allows the users to monitor the status of the crawlers and crawl the required source within the required time period. Figure 29 and figure 30 show the user interfaces that can be used to list the available web crawlers and see the crawled time periods of a particular source.

The back end component is actually managing the web crawlers. When a user specifies a time period to crawl, this component receives those details. Then it searches for the path of jar file of the crawler and runs the jar file with the specified parameters. It opens a port to receive the completed dates. When the crawler finishes crawling a particular date, it sends back the finished date. Then the crawler controller writes those details to the database, for the users to keep track of the crawled time periods.



Crawler List

Id	Name	Active	Port
24	Gazette	1	12346
23	Wijaya	1	12353
22	Mawbima	1	12352
21	Lankadeepa	1	12351
19	Adaderana	1	12350
18	ravaya	1	12349
17	Vidusara	1	12348
16	Divaina	1	12347
15	Sarasawiya	1	12345
14	Namaskara	1	12347

Showing 1 to 10 of 14 entries

Figure 29: Crawler controller showing crawler list on web interface

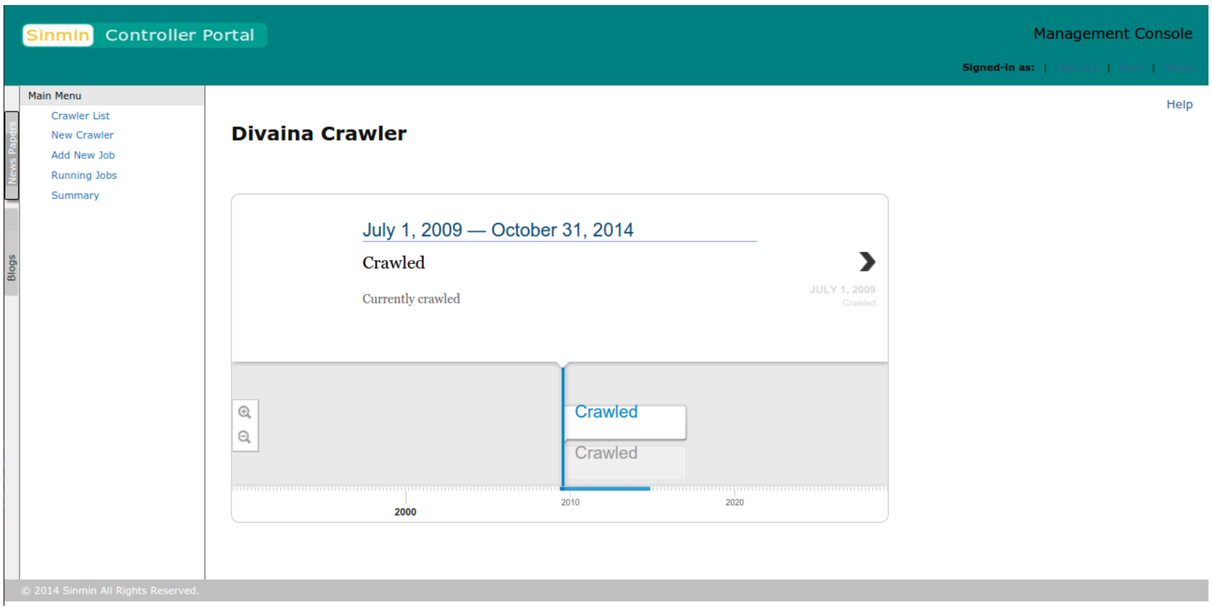


Figure 30: Crawled time period of a crawler

Flow diagram of the crawler controller is shown in figure 31.

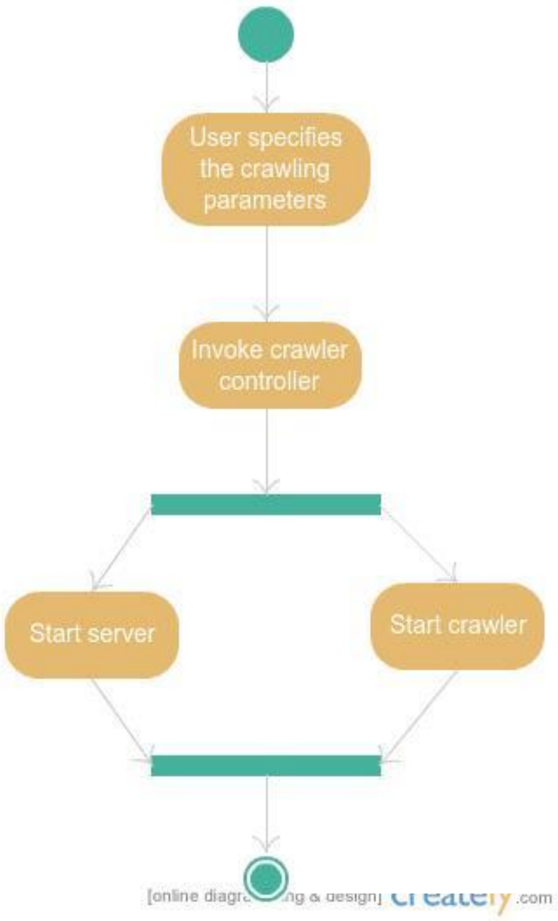

User specifies the crawling parameters
Invoke crawler controller
Start server
Start crawler
[online diagramming & design] creately.com

Figure 31: Crawling process

4.3 Database Installation

4.3.1 Cassandra database installation

We used Apache Cassandra 2.1.2 for implementing the database of SinMin. It is available to download at http://cassandra.apache.org/download/ .

Due to the limited server instances we could get, we hosted our Cassandra database in a single node. The main database of SinMin which follows the schema mentioned in 3.4.2.1 was created using cqlsh terminal. For this we used cqlsh version 5.0.1.

4.3.2 Oracle database installation

We used Oracle database 11g Release 2 Standard Edition. It can be downloaded from http://www.oracle.com/technetwork/database/enterprise-edition/downloads/index-092322.html.

However Oracle does not give binaries that support Ubuntu server versions. So we had to use Oracle Linux versions and carry out some configuration changes in Ubuntu server including kernel parameter changes and adding symlinks.

We did the following kernel parameter changes in /etc/sysctl.conf file.

```
#
# Oracle 11g
#
kernel.sem = 250 32000 100 128
kernel.shmall = 2097152
kernel.shmmni = 4096
# Replace kernel.shmmax with the half of your memory in bytes
# if lower than 4Go minus 1
# 1073741824 is 1 GigaBytes
kernel.shmmax=1073741824

# Try sysctl -a | grep ip_local_port_range to get real values
net.ipv4.ip_local_port_range = 9000  65500

net.core.rmem_default = 262144
net.core.rmem_max = 4194304
net.core.wmem_default = 262144
net.core.wmem_max = 1048576

# Max value allowed, should be set to avoid IO errors
fs.aio-max-nr = 1048576
# 512 * PROCESSES / what really means processes ?
fs.file-max = 6815744

# To allow dba to allocate hugetlbfs pages
# 1001 is your oinstall group, id. grep oinstall /etc/group will
give this value
vm.hugetlb_shm_group = 1001
```

The following packages were newly installed to Ubuntu server

```
sudo apt-get install alien
sudo apt-get install autoconf
sudo apt-get install automake
sudo apt-get install autotools-dev
sudo apt-get install binutils
sudo apt-get install bzip2
sudo apt-get install doxygen
sudo apt-get install elfutils
sudo apt-get install expat
sudo apt-get install gawk
sudo apt-get install gcc
sudo apt-get install gcc-multilib
sudo apt-get install g++-multilib
sudo apt-get install ksh
sudo apt-get install less
sudo apt-get install lesstif2
sudo apt-get install lesstif2-dev
sudo apt-get install lib32z1
sudo apt-get install libaio1
sudo apt-get install libaio-dev
sudo apt-get install libc6-dev
sudo apt-get install libc6-dev-i386
sudo apt-get install libc6-i386
sudo apt-get install libelf-dev
sudo apt-get install libltdl-dev
sudo apt-get install libmotif4
sudo apt-get install libodbcinstq4-1 libodbcinstq4-1:i386
sudo apt-get install libpth-dev
sudo apt-get install libpthread-stubs0
sudo apt-get install libpthread-stubs0-dev
sudo apt-get install libstdc++5
sudo apt-get install lsb-cxx
sudo apt-get install make
sudo apt-get install openssh-server
sudo apt-get install pdksh
sudo apt-get install rlwrap
sudo apt-get install rpm
sudo apt-get install sysstat
sudo apt-get install unixodbc
sudo apt-get install unixodbc-dev
sudo apt-get install unzip
sudo apt-get install x11-utils
sudo apt-get install zlibc
```

The following symlinks were created in-order to mock paths required from Linux Oracle installer.

```
sudo ln -sf /bin/bash /bin/sh
sudo ln -s /usr/bin/awk /bin/awk
sudo ln -s /usr/bin/rpm /bin/rpm
sudo ln -s /usr/bin/basename /bin/basename
```

To run the Oracle Universal Installer (OUI) X-Windows is required because installer is a GUI tool. So we logged in to the server using SSH with X-Forwarding

```
ssh -X oracle@server_ip_address
./runInstaller -ignoreSysPrereqs
```

After installation, a new Oracle instance was created using Oracle DBCA (Database Configuration Assistant) tool.

`Instance SID = corpus.`

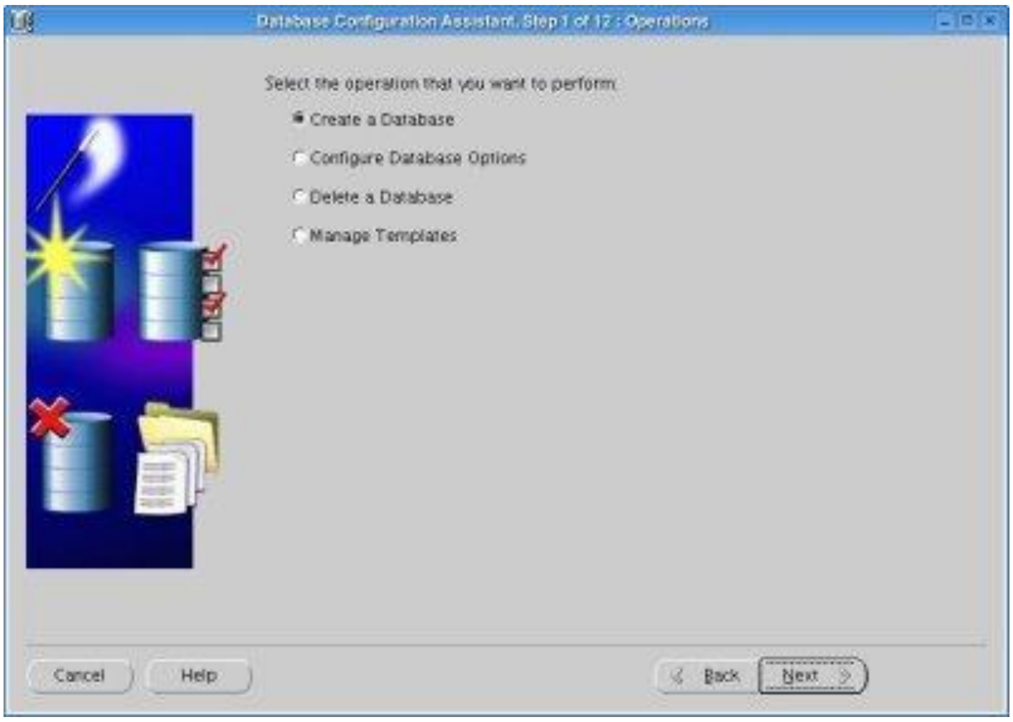
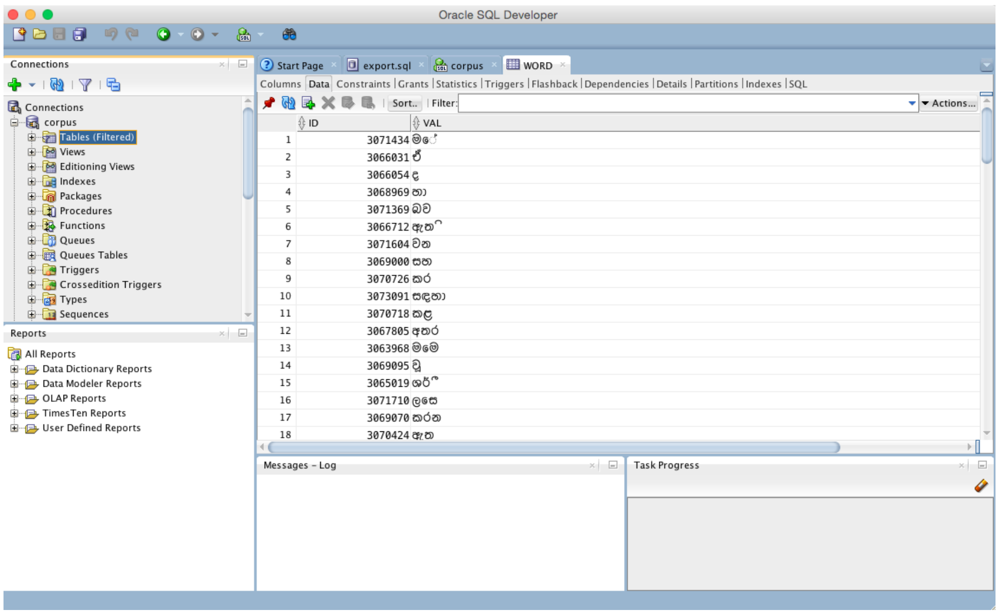

Database Configuration Assistant, Step 1 of 12 : Operations
Select the operation that you want to perform:
● Create a Database
○ Configure Database Options
○ Delete a Database
○ Manage Templates
Cancel
Help
Back
Next

Figure 32: Oracle Database Cconfiguration Assistant tool

To start newly created instance

```
ORACLE_SID=corpus
export ORACLE_SID
sqlplus / as sysdba
startup
```

To connect to the database and create structure, Oracle SQL Developer was used.



	ID	VAL
1	3071434	මේ
2	3066031	ඒ
3	3066054	ද
4	3068969	හා
5	3071369	බව
6	3066712	ඇත්
7	3071604	වන
8	3069000	සහ
9	3070726	කර
10	3073091	සඳහා
11	3070718	කළ
12	3067805	අතර
13	3063968	මෙම
14	3069095	වූ
15	3065019	ශ්‍රී
16	3071710	ලෙස
17	3069070	කරන
18	3070424	ඇත

Figure 33: Oracle SQL Developer tool

4.4 Data Feeding Mechanism

The data saved to XML files by crawlers were fed into the Cassandra database. This was done using an application developed using Java. This application is developed in such a way that it will check the location of the server where XML files with language content is received and if it finds a new file there, the file will be processed. The following is the flow chart for that process.

4.4.1 Identifying newly crawled files

The new file identifier tool uses an in-memory tree structure to track current files and newly added files. Once a particular location is given as the root folder, it recursively goes into the folders and creates the tree which consists of file and folder nodes. Figure 35 shows a sample tree structure.

Figure 37 shows the logic that was used to create the file tree using recursive travelling. This algorithm is run once in 60 seconds to identify newly added files. Figure 36 illustrates the class diagram that was used to create this file tree. To represent file nodes we implemented DFile class and to represent folders we implemented Folder class. A Folder object is capable of keeping its children DFile and Folder objects.

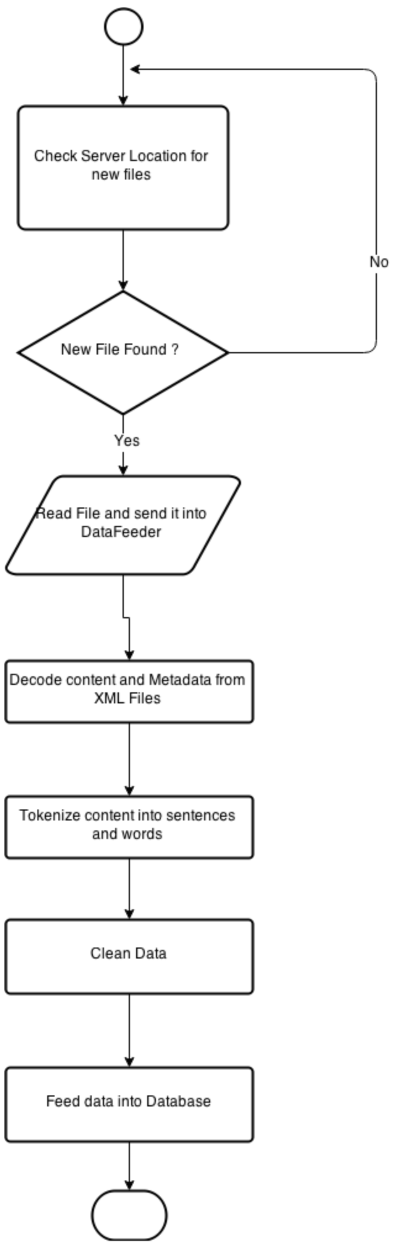

Check Server Location for new files
New File Found ?
No
Yes
Read File and send it into DataFeeder
Decode content and Metadata from XML Files
Tokenize content into sentences and words
Clean Data
Feed data into Database

Figure 34: Flow chart for data feeder

Then newly founded XML files will be sent into the Cassandra or Oracle data feeder as necessary. We used Apache Axiom library for decoding XML files and for extracting relevant information from them. As we have mentioned earlier, <*root*> element of each XML file will contain one or more <*post*> elements inside it. Each <post> element will have content and metadata of a post marked with corresponding tags.

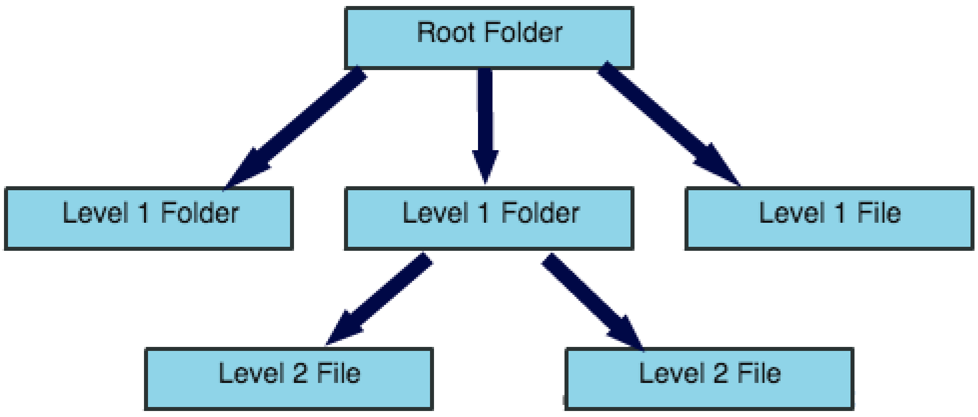


Figure 35: File tree structure to handle data feeding

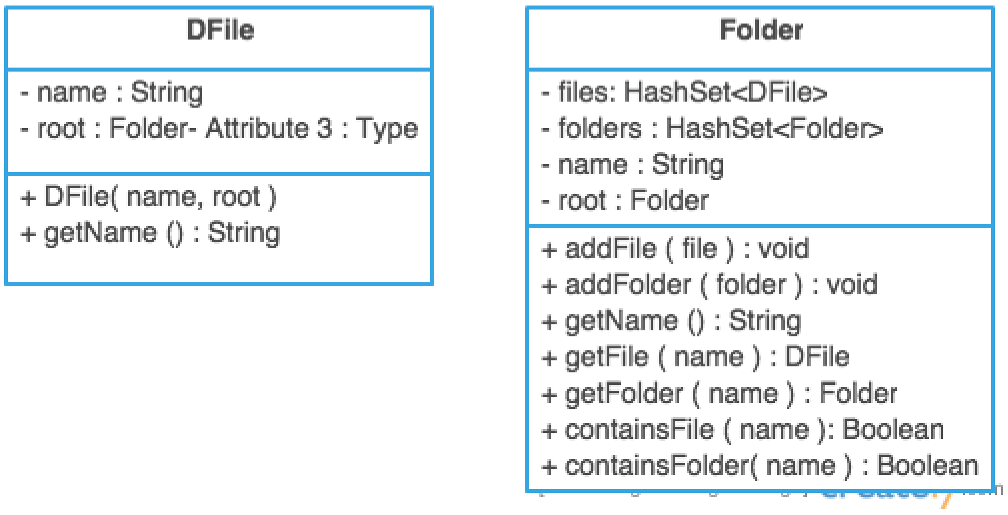

DFile
- name : String
- root : Folder- Attribute 3 : Type
+ DFile(name, root)
+ getName () : String
Folder
- files: HashSet<DFile>
- folders : HashSet<Folder>
- name : String
- root : Folder
+ addFile (file) : void
+ addFolder (folder) : void
+ getName () : String
+ getFile (name) : DFile
+ getFolder (name) : Folder
+ containsFile (name): Boolean
+ containsFolder(name) : Boolean

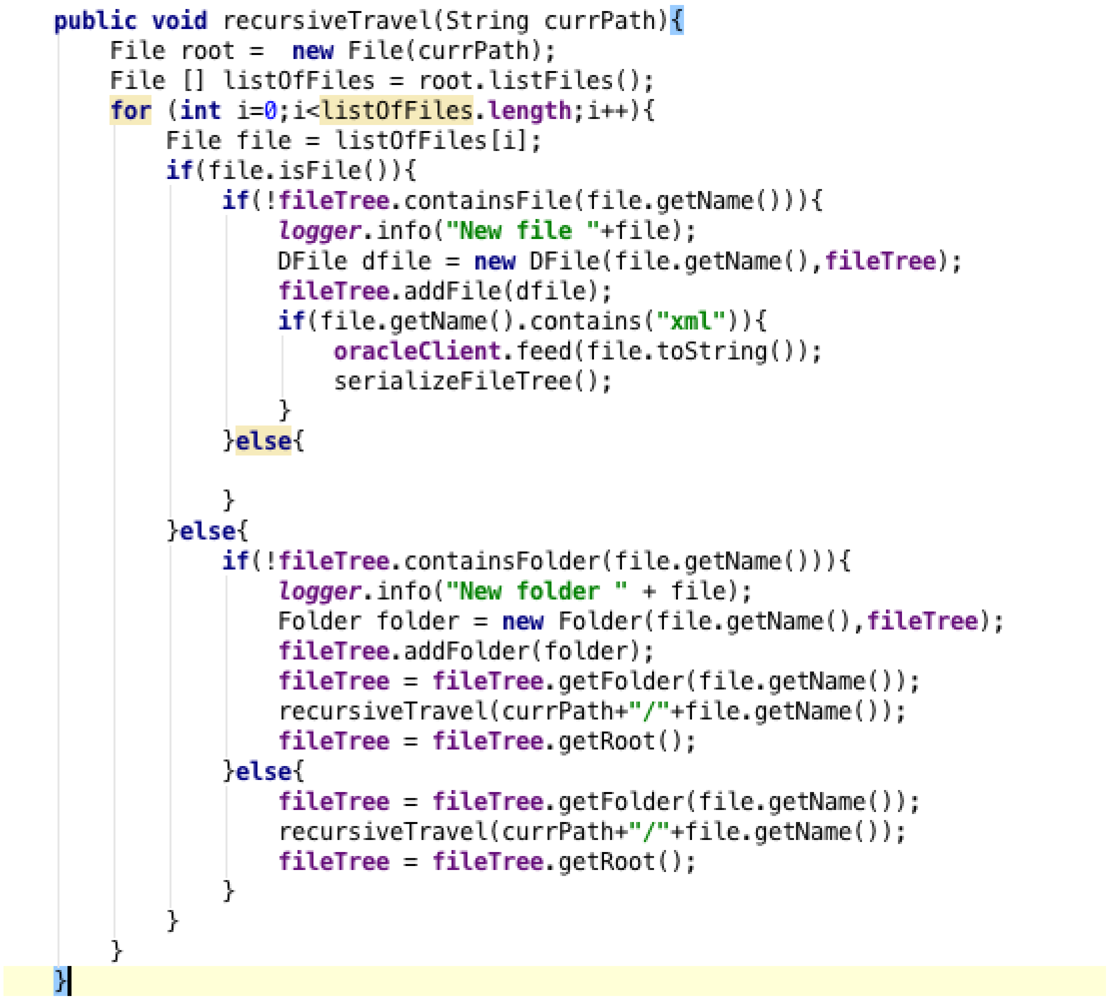
Figure 36: Class diagram of Dfile and folder classes

4.4.2 Data insertion into Cassandra

Data inserting and information extraction from database was done using java application which uses datastax java driver for Cassandra. It was added to the maven projects using the following code.

```
<dependency>
<groupId>com.datastax.cassandra</groupId>
<artifactId>cassandra-driver-core</artifactId>
<version>2.0.1</version>
</dependency>
```

We used prepared statements for inserting data into database via java driver to avoid security issues. The following code shows how data insertion to Cassandra done using prepared statements.



```
public void recursiveTravel(String currPath){
    File root =  new File(currPath);
    File [] listOfFiles = root.listFiles();
    for (int i=0;i<listOfFiles.length;i++){
        File file = listOfFiles[i];
        if(file.isFile()){
            if(!fileTree.containsFile(file.getName())){
                logger.info("New file "+file);
                DFile dfile = new DFile(file.getName(),fileTree);
                fileTree.addFile(dfile);
                if(file.getName().contains("xml")){
                    oracleClient.feed(file.toString());
                    serializeFileTree();
                }
            }else{

            }
        }else{
            if(!fileTree.containsFolder(file.getName())){
                logger.info("New folder " + file);
                Folder folder = new Folder(file.getName(),fileTree);
                fileTree.addFolder(folder);
                fileTree = fileTree.getFolder(file.getName());
                recursiveTravel(currPath+"/"+file.getName());
                fileTree = fileTree.getRoot();
            }else{
                fileTree = fileTree.getFolder(file.getName());
                recursiveTravel(currPath+"/"+file.getName());
                fileTree = fileTree.getRoot();
            }
        }
    }
}
```

Figure 37: Algorithm to create file tree

```
PreparedStatement statement = session.prepare("select * from
corpus.word_time_category_frequency WHERE word=? AND year=? AND
category=?");
ResultSet results = session.execute(statement.bind(words[j],
yearInt, category));
```

4.5 Bulk Data Insertion Optimization

4.5.1 Oracle

Inserting data to Oracle relational, normalized structure is a complex task when it comes to batch data insertion because one new entry for a particular table may depend on another entry from another table. Our first approach was to directly pass SQL queries through Oracle Database Connector for each new database entry. Due to significant overhead of communication within JDBC connector and the database, this method was very slow and became slower when more data is inserted to database.

Figure 38 shows the process we followed to insert data as a batch.

1. Batch inserter reads all posts from given XML file
2. Metadata related to a particular post (Author, Date, etc) are saved in Post table and primary keys Are retrieved.
3. Select posts one by one and extract sentence list.
4. Add sentence metadata to Sentence table and retrieve primary key for each inserted sentence.
5. Extract word list from each sentence using our tokenizer.
6. Add words to Word table. If word already exists in word table, update entry with frequency. Else create a new entry.
7. Add an entry to Sentence_Word table to connect newly added sentence and word entries.
8. Do step 5,6 and 7 repeatedly to insert bigram and trigrams. Only difference is, bigrams and trigram tables use Word ids rather than their exact values. To do this, once a new entry is inserted to Word table, we store word value and word id in a hash map. This improves performance since inserting bigrams and trigrams does not need to look into database for their existence. They are all stored locally in in-memory data structure.

Oracle Database 11g comes with the support of Procedural Language/Structured Query Language (PL/SQL). This provides much flexibility in inserting data in batches when data contains conditional dependencies. In the previous method we checked those conditional dependencies within Java code. This required needed passing much f data between Java code and the database. Using PL/SQL scripts we were able to handover those tasks to the database side. Because PL/SQL scripts run inside the database, this significantly increased the performance of data insertion. In this method, rather than passing SQL queries from Java to Oracle, we send an array of data to Oracle. PL/SQL script within Oracle processes those data and creates SQL queries according to conditional dependencies. Figure 39 shows a part of the PL/SQL script we implemented to batch data insertion.

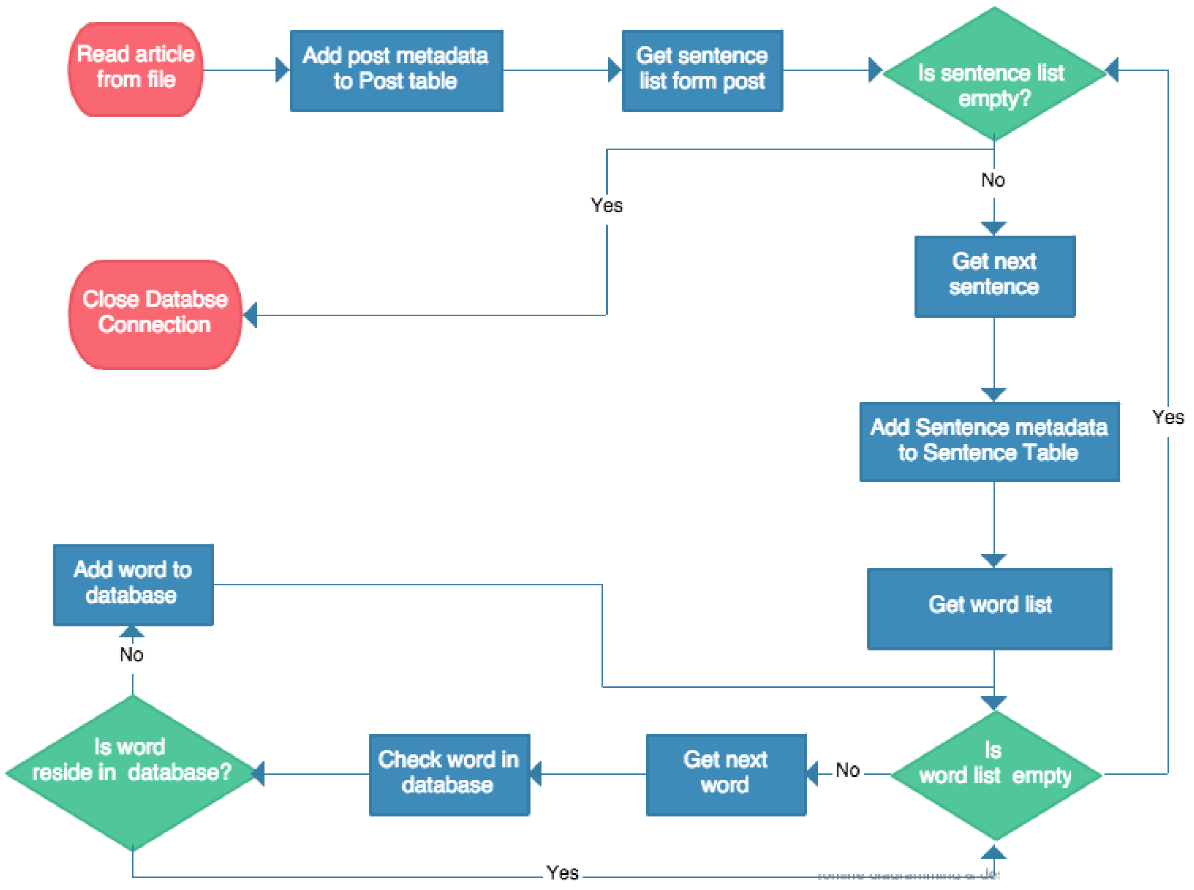


Figure 38: Oracle data feeding flow

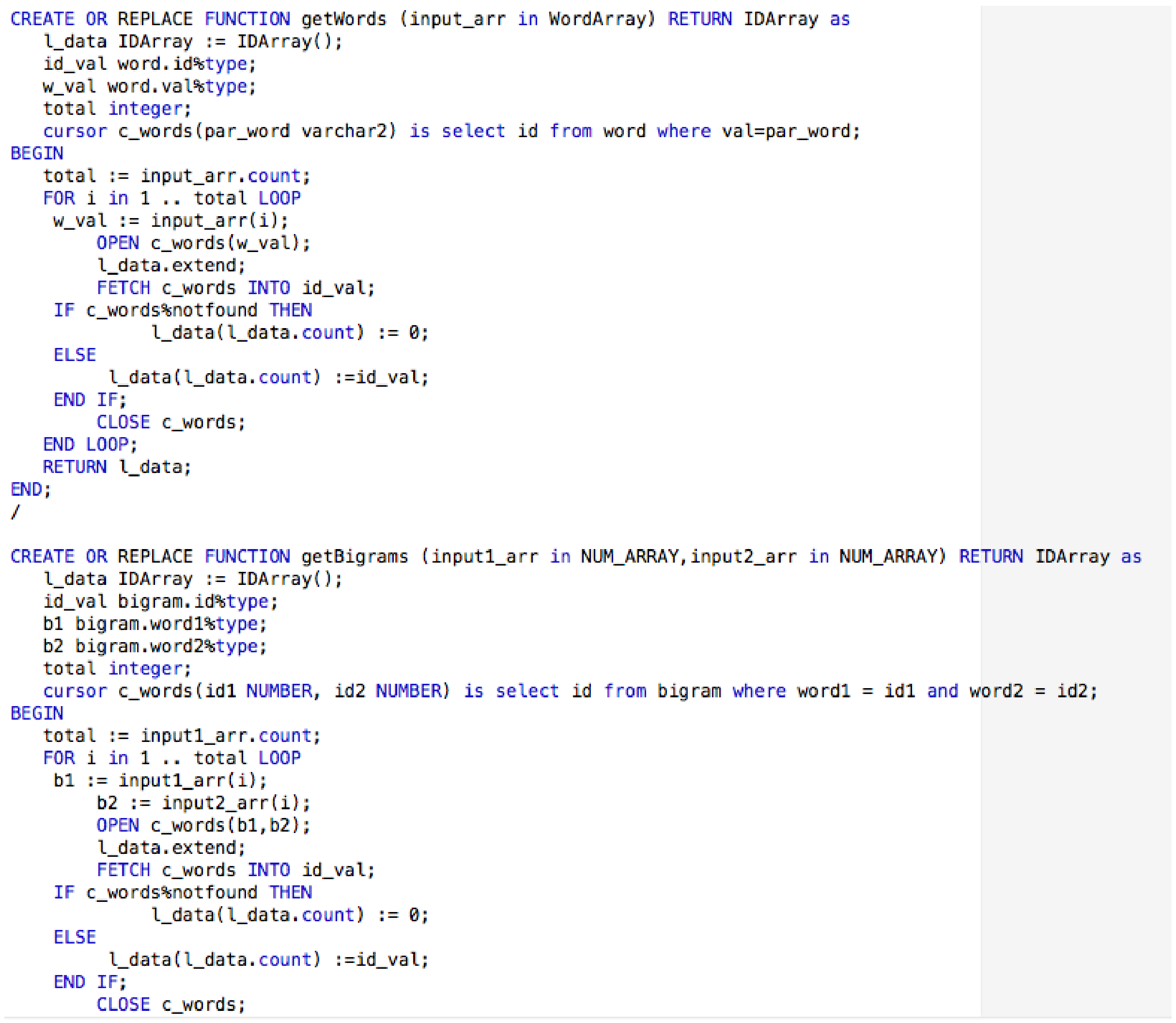
4.6 Data cleaning

The data of raw XML files are cleaned in two steps.

1) Cleaning the erroneous characters of the texts (this is done at tokenizing operation)
2) Sinhala consecutive vowel sign problem fixing

4.6.1 Sinhala Tokenizer

Split a given Sinhala text into words or Sentences. The first objective was to identify delimiters for both cases.



```
CREATE OR REPLACE FUNCTION getWords (input_arr in WordArray) RETURN IDArray as
   l_data IDArray := IDArray();
   id_val word.id%type;
   w_val word.val%type;
   total integer;
   cursor c_words(par_word varchar2) is select id from word where val=par_word;
BEGIN
   total := input_arr.count;
   FOR i in 1 .. total LOOP
    w_val := input_arr(i);
        OPEN c_words(w_val);
        l_data.extend;
        FETCH c_words INTO id_val;
    IF c_words%notfound THEN
             l_data(l_data.count) := 0;
    ELSE
         l_data(l_data.count) :=id_val;
    END IF;
        CLOSE c_words;
   END LOOP;
   RETURN l_data;
END;
/

CREATE OR REPLACE FUNCTION getBigrams (input1_arr in NUM_ARRAY,input2_arr in NUM_ARRAY) RETURN IDArray as
   l_data IDArray := IDArray();
   id_val bigram.id%type;
   b1 bigram.word1%type;
   b2 bigram.word2%type;
   total integer;
   cursor c_words(id1 NUMBER, id2 NUMBER) is select id from bigram where word1 = id1 and word2 = id2;
BEGIN
   total := input1_arr.count;
   FOR i in 1 .. total LOOP
    b1 := input1_arr(i);
        b2 := input2_arr(i);
        OPEN c_words(b1,b2);
        l_data.extend;
        FETCH c_words INTO id_val;
    IF c_words%notfound THEN
             l_data(l_data.count) := 0;
    ELSE
         l_data(l_data.count) :=id_val;
    END IF;
        CLOSE c_words;
```

Figure 39: Oracle PL/SQL script to insert data

4.6.1.1 **Initial implementation**

After data was crawled by web crawlers, all of them were spited into words using a simple tokenizer which consists of the below delimiters.

- u+002E - period mark
- u+0021 - exclamation mark
- u+0020 - space
- u+002C - comma
- u+003F - question mark

Then the result word set was examined. From the list some words were extracted which have at least one letter which is not a Sinhala character or a number. Then the words were examined manually. The following problems were identified.

- Some words contained invalid Unicode characters
- Some words contained symbols
- Some words contained unwanted non-Sinhala characters (removing them doesn’t change the meaning of the word)
- Non-symbolic characters which were terminating words

4.6.1.1.1 **Invalid Unicode characters**

Crawled Sinhala texts contained following invalid Unicode characters.

- u+fffd - Replacement character
- u+f020 - Character in a private user area
- u+f073 - Character in a private user area
- u+f06c - Character in a private user area
- u+f190 - Character in a private user area

The words which contained the above letters were separating two words. Therefore these letters were listed as word separators.

4.6.1.1.**2 Unwanted non-Sinhala characters in Sinhala words**

These were the letters which were in Sinhala letters but were not terminating words. There were 101 such letters. These were not giving any meaning to the text. Depending on the source the text was extracted; the amount of occurrences of these words varies. It was decided to remove all these letters from Sinhala texts before adding the data to corpus database.

E.g.: u+200C, Á, À, ®, ¡, ª, º, ¤, ¼, ¾, Ó, ø, ½,etc.

Table 12: Rejecting Unicode characters

u+200C	u+00FF	u+00AB	u+00D1	u+00E4
u+0160	u+00B7	u+00B7	u+00CD	u+00D4
u+00AD	u+00ED	u+00A8	u+00D1	u+00F5
u+0088	u+03A9	u+2026	u+00E7	u+00C8
u+F086	u+00B0	u+22C6	u+00C6	u+00DD
u+200B	u+00D7	u+203A	u+00F4	u+00DF

u+FEFF	u+00B5	u+00A5	u+017D	u+00F5
u+00C1	u+F02E	u+22C6	u+20AC	u+00F9
u+00C0	u+007E	u+02DD	u+00A7	u+00E5
u+00AE	u+0192	u+F075	u+00C6	u+00D8
u+00A1	u+008F	u+F02C	u+00F7	u+0152
u+00AA	u+00EB	u+25CA	u+00E9	u+00D4
u+00BA	u+00CE	u+0141	u+00AF	u+00DC
u+00A4	u+2030	u+00EA	u+00E9	u+00D6
u+00BC	u+00BB	u+00D5	u+00E6	u+00DB
u+00BE	u+00AB	u+00C4	u+00EE	u+00CF
u+00D3	u+00E0	u+00E1	u+00EF	u+00F1
u+00F8	u+00FD	u+007F	u+00D2	u+0081
u+00BD	u+0153	u+00C9	u+00AF	u+00A2
u+02C6	u+00B9			

4.6.1.1.3 **Symbols**

In the extracted text, there were many symbols. All of them were separating words, but a few of them were separating sentences.

The following characters were identified as sentence separators:
“?”, “!”, “;”, “:”, “•” (bullet mark)

Identified word separators:
".", ",", "\n", " ", "¸", ",", "\"", "/", "-", "|", "\\", "—", "ˌ", "””", "‘", """", "““", "’", "′", "′", "!", "@", "#", "$", "%", "^", "&", "\*", "+", "\\-", "£", "\\?", "~", "\\(", "\\)", "\\[", "\\]", "{", "}", ":", ";", "\u2013"

4.6.1.1.3 **Short forms**

Short forms consists of full stops. But those full stop marks aren’t separating sentences nor words.
E.g.: ප. ව. (pm), රු. (Rupees)

Therefore these should be identified before tokenizing is done. In order to prevent unnecessary splitting the following steps are taken.

- The full stops of the short forms are replaced with “u+0D80” letter. This is an unassigned Unicode letter which lies in the Sinhala Unicode block.
- After splitting is done, the “u+0D80” will be replaced with a full stop mark.

Some of the identified short forms are below.
"ඒ.", "බී.", "සී.", "ඩී.", "ඊ.", "එෆ්.", "ජී.", "එච්.", "අයි.", "ජේ.", "කේ.", "එල්.", "එම්.", "එන්.", "ඕ.", "පී.", "කිව්.", "ආර්.", "එස්.", "ටී.", "යූ.", "ඩබ්.", "ඩබ්ලිව්.", "එක්ස්.", "වයි.", "ඉසෙඩ්.", "පෙ.", "ව.", "ප.", "රු.", "0.", "1.", "2.", "3.", "4.", "5.", "6.", "7.", "8.", "9."

4.6.1.2 **Current implementation**

Sentence tokenizing:

1) Remove all ignoring chars
2) Replace full stop characters of short forms with “u+0D80”
3) Split by delimiters {".", "?", "!", ":", ";", "\u2022"}
4) Replace “u+0D80” characters with full stop character

Word tokenizing:

1) Remove all ignoring characters
2) Replace full stop characters of short forms with “u+0D80”
3) Split words using following delimiters

   Numbers: {"0", "1", "2", "3", "4", "5", "6", "7", "8", "9"}

   Invalid characters: {"Ê", "\u00a0", "\u2003", "\ufffd", "\uf020", "\uf073", "\uf06c", "\uf190", "\u202a", "\u202c", "\u200f" }

   Punctuation marks: {".", ",", "\n", " ", "‚", ",", "\"", "/", "-", "|", "\\", "—", "¦", "”", "‘", "'", "“", "’", "´", "´", "!", "@", "#", "$", "%", "^", "&", "\*", "+", "\\-", "£", "\\?", "~", "\\(", "\\)", "\\[", "\\]", "{", "}", ":", ";", "\u2013" }
4) Replace “u+0D80” characters with full stop character
5) Ignore words which doesn’t contain at least one character of Unicode Sinhala block

4.6.2 Sinhala Vowel letter fixer

Depending on the rendering algorithm the visual representation of a Sinhala Unicode text changes. The main reason is the Sinhala vowel signs. It’s possible to have the same visual representation for different Unicode representations on different rendering algorithms.

E.g.:

Algorithm1 (ල + ෝ) ->ලෝ

Algorithm2 (ල + ො + ්) ->ලෝ

In such cases, although they give the same visual representation, they won't be equal. Most of the time the lengths are not equal. In these cases there is more than one vowel sign character for a Sinhala letter character (In Sinhala Unicode the vowel signs comes after the Sinhala letter). These issues can be found often in Sinhala texts.

It's possible to represent any Sinhala text using only one vowel sign character for a Sinhala letter character.

E.g.: "ල + ො + ්" can be represented by "ල + ෝ"

The SinhalaVowelLetterFixer does a many-to-one mapping on Unicode Sinhala words to eliminate the above issue.

4.6.2.1 **Map of vowel signs:**

Table 13: Vowel sign mapping

ෙ + ්	ේ
් + ෙ	ේ
ෙ + ා	ො
ා + ෙ	ො
ේ + ා	ෝ
ො + ්	ෝ
ෙ + ෙ	ෛ
◌a + ◌a	◌aa

ෙ◌ + ◌ෟ	ෙ◌ෟ
◌ෟ + ෙ◌	ෙ◌ෟ
◌් + ◌්	◌්
◌් + ◌්	◌්

4.6.2.2 **Duplicating same symbol**

Sometimes there is duplicating of vowels. Some rendering algorithms do not visualize the duplicate ones. However these vowel signs need to be removed.

Table 14: Eliminating duplicate vowel signs

ෙ◌් + ◌්	ෙ◌්
ෙ◌් + ෙ◌	ෙ◌්
ෙ◌ා + ◌ා	ෙ◌ා
ෙ◌ා + ෙ◌	ෙ◌ා
ෙ◌ා් + ◌ා	ෙ◌ා්
ෙ◌ා් + ◌්	ෙ◌ා්
ෙ◌ා් + ෙ◌	ෙ◌ා්
ෙ◌ා් + ෙ◌්	ෙ◌ා්
ෙ◌ා් + ෙ◌ා	ෙ◌ා්
ෙ◌ෟ + ◌ෟ	ෙ◌ෟ
ෙ◌ෟ + ෙ◌	ෙ◌ෟ

4.7 Wildcard Searching Implementation

As described in section 3.4.2.3 every word is stored in two ways. The word itself and an encoded form of it. The fields in schema of Solr are ‘content’ and ‘content_encoded’ respectively.

4.7.1 Sinhala vowel sign problem at wildcard search

In Sinhala Unicode the Vowel Sign (Pillama) is a separate character that comes after the Sinhala letter.
Eg: මේ = ම + ේ (Sinhala letter + Sinhala vowel Sign)
When doing wildcard search in Sinhala, the search results may not be correct from a language point of view..
Eg: Search results of “ම?” would include the word “මේ” (because මේ = ම + ේ)

4.7.2 Encoding Sinhala letters

The problem here is that the Sinhala Vowel signs separate Unicode characters. To solve this problem, the Sinhala letter and Sinhala vowel sign should be represented as one entity. In implementation, one entity will contain at most one vowel sign. Each entity will be created by 4 a digit integer as follow.
The following characters are indexed from 0 to 61 respectively.
"අ", "ආ", "ඇ", "ඈ", "ඉ", "ඊ", "උ", "ඌ", "ඍ", "ඎ", "ඏ", "ඐ", "එ", "ඒ", "ඓ", "ඔ", "ඕ", "ඖ", "ක", "ඛ", "ග", "ඝ", "ඞ", "ඟ", "ච", "ඡ", "ජ", "ඣ", "ඤ", "ඥ", "ඦ", "ට", "ඨ", "ඩ", "ඪ", "ණ", "ඬ", "ත", "ථ", "ද", "ධ", "න", "ඳ", "ප", "ඵ", "බ", "භ", "ම", "ඹ", "ය", "ර", "ල", "ව", "ශ", "ෂ", "ස", "හ", "ළ", "ෆ", "ං, "ඃ, "u+200d"

Note: The above are listed as the Sinhala letters. Although the zero-width-joiner character (u+200d) isn’t in the Sinhala Unicode block, it is listed here because it is used by some rendering algorithms to represent Yansaya, Reepaya and Rakaraya vowel signs. The vowel signs "ං and "ඃ are also listed here to prevent breaking the notation “one entity will contain at most one vowel sign”.
The characters below are indexed from 0 to 18 respectively.
"්", "ා", "ැ", "ෑ", "ි", "ී", "ු", "ූ", "ෘ", "ෙ", "ේ", "ෛ", "ො", "ෝ", "ෞ", "ෟ", "ෲ", "ෳ", "෴"

The 4 digit integer will be generated as follow. Let’s consider the Sinhala word “මේ“

$$\underbrace{\text{මේ}}_{\text{“4710”}} = \underbrace{\text{ම}}_{\text{“47”}} + \underbrace{\text{ේ}}_{\text{“10”}}$$

To separate each entity, a special character is used which isn’t a numerical character. At the end of the representation of each word a special character will be present.
E.g.: Let’s consider Sinhala word “මෙය”. (The * is used to represent that special character)

$$\underbrace{\text{මෙය}}_{} = \text{මෙ} + \text{ය}$$

$$\underbrace{\text{මෙය}}_{4709*4900*} = \text{ම} \quad \underbrace{+ \quad \text{ෙ}}_{4709} \quad \underbrace{+ \quad \text{ය}}_{4900}$$

4.7.2 Security of Solr

Solr instance is running on port 8983. By default, Solr accepts queries from any IP addresses. To ensure security, access to the port 8983 is limited. Currently the wildcard search feature is exposed through the REST API. It queries Solr through the Solr controller (corpus.sinhala.wildcard.search). Both of them are lying on the same machine that the Solr is running. Therefore the access to the port 8983 is limited only to the local host. That means the queries from other hosts to the port 8983 are dropped.
This is done by setting up an IP table on a Linux server.

```
sudo iptables -A INPUT -p tcp -s localhost --dport 8983 -j ACCEPT
sudo iptables -A INPUT -p tcp --dport 8983 -j DROP
```

The first command tells iptables, "accept all connections from localhost on port 8983", and the second tells iptables, "drop all traffic to port 8983"—since iptables obeys rules in order from top to bottom, localhost traffic is let through, while other traffic is denied.

4.8 API and User Interface

4.8.1 User interface

User interface is created using PHP. Most of the interactive features of the user interface are developed using JavaScript. We have used several JavaScript libraries,

- jQuery
- Highcharts for graph generation
- jQuery Datatables to create tables

API calls are made using Ajax. We have used bootstrap front-end framework in the user interface implementation.

The process used to generate the graph of fraction of usage of a given word over time is described here. First the user entered time range is read. Then two API calls are made to retrieve the word count and the word frequencies of each year during the given time period.

First a request is sent to the API function wordFrequency with the message body below..

```
{
        "time":["2007","2008","2009","2010","2011","2012","2013","2014"],
        "value":"ලංකා"
}
```

It will send a response with a message body as shown below.

```
{
        [
                {"date":2007,"category":"all","frequency":0},
                {"date":2008,"category":"all","frequency":1373},
                {"date":2009,"category":"all","frequency":2228},
                {"date":2010,"category":"all","frequency":3524},
                {"date":2011,"category":"all","frequency":4891},
                {"date":2012,"category":"all","frequency":4873},
                {"date":2013,"category":"all","frequency":4191},
                {"date":2014,"category":"all","frequency":3114}
        ]
}
```

Then another request is sent to API function wordCount with the message body below.

```
{
        "time":["2007","2008","2009","2010","2011","2012","2013","2014"]
}
```

It will send back a response with a message body as shown below.

```
{
        [
                {"year":2007,"category":"all","count":0},
                {"year":2008,"category":"all","count":1630137},
                {"year":2009,"category":"all","count":1999548},
                {"year":2010,"category":"all","count":2974013},
                {"year":2011,"category":"all","count":3194107},
                {"year":2012,"category":"all","count":3261664},
                {"year":2013,"category":"all","count":3653421},
                {"year":2014,"category":"all","count":3497363}
        ]
}
```

The word frequency in each year is now divided by the word count in that year (unless it is zero) to find the fraction of usage of that word in the particular year over all the words used in that year.

Highcharts requires the data to be in the format below to generate the graph over time.

    [[time_1, amount_1],[time_2, amount_2],[time_3, amount_3],....,[time_n, amount_n]]

Here the *time_k* should be the x axis value which is the year (this should be in UNIX time format) and *amount_k* should be the fraction of usage. Highchart needs the time values to be sorted in the ascending order. So the values are formatted in the required format and passed to Highchart library to generate the graph.

Figure 40 shows the JavaScript code segment used to generate the graph.

Then the data is represented in a table. The table has two columns and the first column is the year and the next is the value. So the data has to be in the format below.

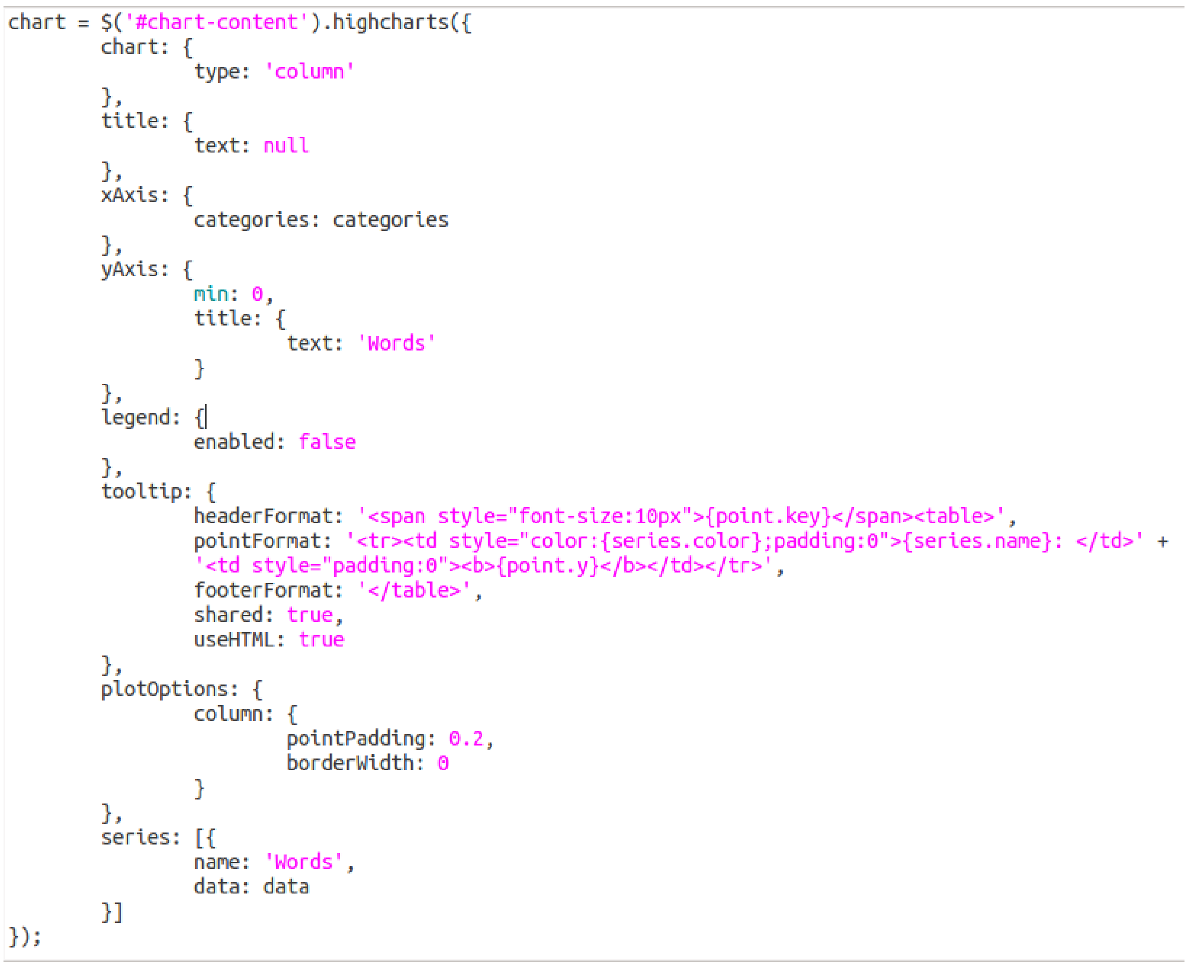
    [[time_1, amount_1],[time_2, amount_2],[time_3, amount_3],....,[time_n, amount_n]]

Here the *time_k* is the year and the *amount_k* is the fraction of usage. In this case this format is also equal to the format required for highcharts except that highchart need the time to be in the UNIX time format.

An equal procedure is used in creating other user interfaces.

4.8.2 API

To implement the RESTful web service of SinMin, we used Jersey https://jersey.java.net/ framework as a reference implementation based on JAX-RS specification. Figure 41 shows the folder structure of REST API.

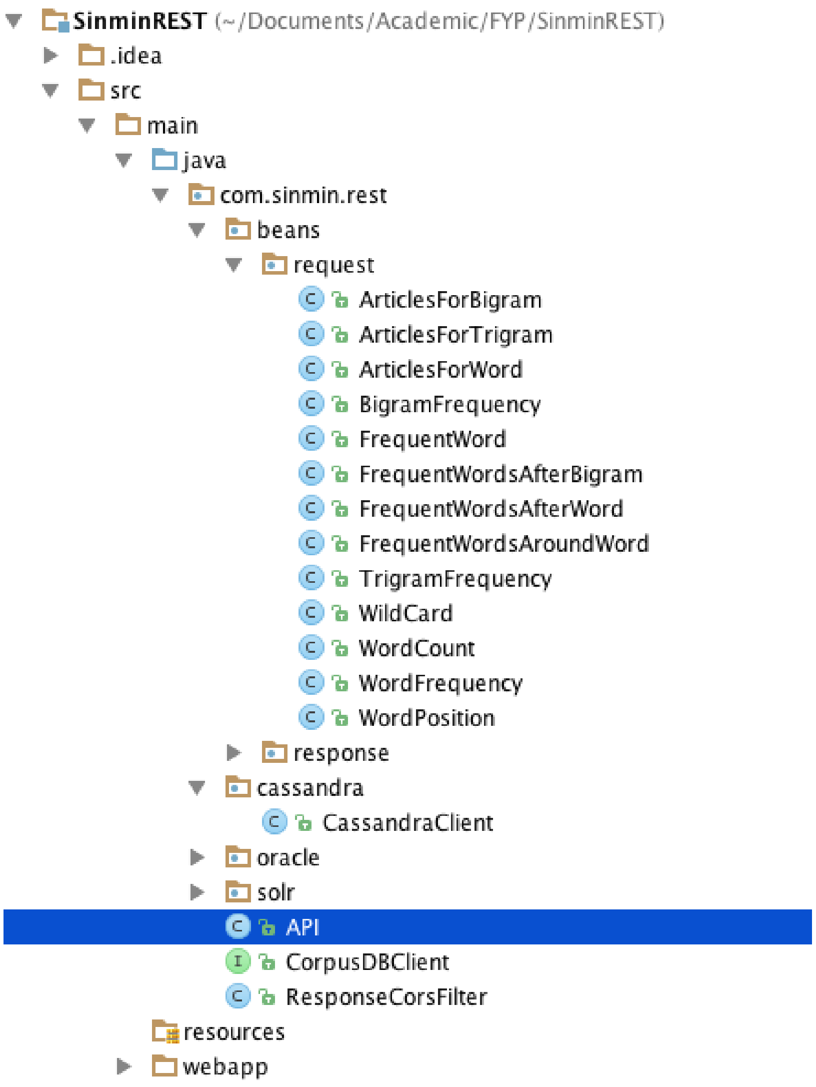


```
chart = $('#chart-content').highcharts({
        chart: {
                type: 'column'
        },
        title: {
                text: null
        },
        xAxis: {
                categories: categories
        },
        yAxis: {
                min: 0,
                title: {
                        text: 'Words'
                }
        },
        legend: {
                enabled: false
        },
        tooltip: {
                headerFormat: '<span style="font-size:10px">{point.key}</span><table>',
                pointFormat: '<tr><td style="color:{series.color};padding:0">{series.name}: </td>' +
                '<td style="padding:0"><b>{point.y}</b></td></tr>',
                footerFormat: '</table>',
                shared: true,
                useHTML: true
        },
        plotOptions: {
                column: {
                        pointPadding: 0.2,
                        borderWidth: 0
                }
        },
        series: [{
                name: 'Words',
                data: data
        }]
});
```

Figure 40: Javascript to generate the graph

Though communication is done using JSON objects, to increase the usability and clearness of the code, we used bean binding provided by Jersey framework. In that case for each JSON request and response, we implemented separate Java beans (objects). These objects are encoded and decoded by Jersey framework so this makes developer's life easy by minimizing the work done at JSON parsing and generating.

Authentication of the API is done using Basic Auth. User store for API is stored in a LDAP installed in the server. Because Basic Auth needs a HTTPS transport in order to provide reliable data transfer, API is wrapped from WSO2 API Manager. API Manager works as a proxy for the API by exposing custom HTTPS URLs and disabling HTTP transport for original API. At the same time API Manager was connected with WSO2 Business Analytics Monitor (BAM) in order to get real time analytics of API usage. Figure 42 shows API usage patterns recorded by WSO2 API Manager.



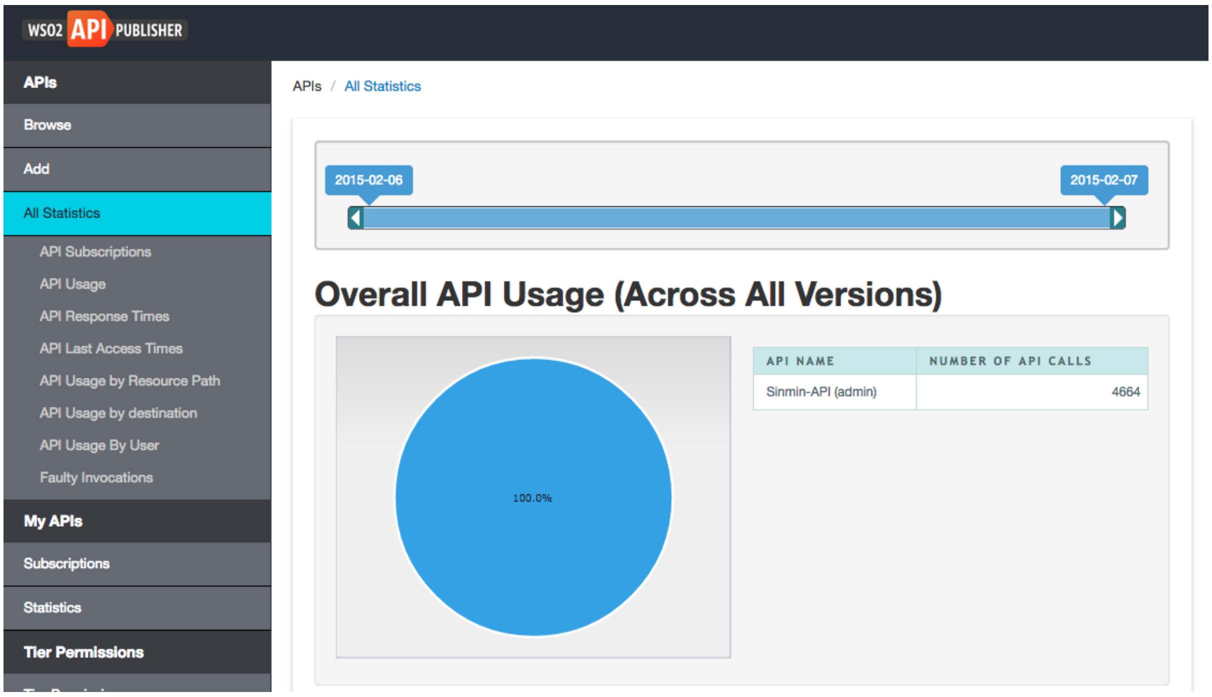
```
SinminREST (~/Documents/Academic/FYP/SinminREST)
  .idea
  src
    main
      java
        com.sinmin.rest
          beans
            request
              ArticlesForBigram
              ArticlesForTrigram
              ArticlesForWord
              BigramFrequency
              FrequentWord
              FrequentWordsAfterBigram
              FrequentWordsAfterWord
              FrequentWordsAroundWord
              TrigramFrequency
              WildCard
              WordCount
              WordFrequency
              WordPosition
            response
          cassandra
            CassandraClient
          oracle
          solr
          API
          CorpusDBClient
          ResponseCorsFilter
      resources
      webapp
```

Figure 41: Class structure of REST API

4.9 Version Controlling

Throughout the project we used Git as our main version controlling system (VCS) for the source code. Other than that we used Google drive for managing documents related to the project.

We used Github (https://github.com) to host the project source code which is the most popular and recognized code hosting service available for Git version control system. Free version of Github projects are public. There are a few other options for Git hosting such as bitbucket to make private projects. Since our project was implemented as open-source system we could use free Github projects without any issue. All source codes related to SinMin can be found from https://github.com/SinMin/core. Github page for project can be found from https://SinMin.github.io



Overall API Usage (Across All Versions)

API NAME	NUMBER OF API CALLS
Sinmin-API (admin)	4664

Figure 42: WSO2 API Manager to manage REST API

We used Google drive to manage documents related to the project. As we needed to edit documents collaboratively, we used Google docs. There are a few other alternatives to manage documents such as OneDrive, but as all of us had Google accounts Google drive was a better option.

5.0 Testing and Results

5.1 Unit testing for crawlers

Unit testing for crawlers was done using junit 4 test framework. By using junit framework we tested getTitle, getAuthor, getContent, getUrl, getYear, getMonth, getDate and getCategory methods of Parser classes, and getDocumentFromURL and generateUrlList methods of generator classes.

In parser classes, we used sample pages of each resource web site and passed them as HTML document to the constructor of the parser. Then we parsed content and metadata from that document and compared them with the correct values that should be returned from those methods.

We used the same way for testing methods of the generator class as well. For getDocumentFromURL we had to check if the content is correctly downloaded from the internet. But it cannot be tested by downloading from the corresponding site. So we used a JettyServer instance to create a dummy website. The following code was used to create and destroy the JettyServer.

```
import org.eclipse.jetty.server.Handler;
import org.eclipse.jetty.server.Server;
import org.eclipse.jetty.server.handler.DefaultHandler;
import org.eclipse.jetty.server.handler.HandlerList;
import org.eclipse.jetty.server.handler.ResourceHandler;

public class JettyServer {
   private Server server;

   public void start() throws Exception{

       ResourceHandler resource_handler = new ResourceHandler();
       resource_handler.setDirectoriesListed(true);
       resource_handler.setResourceBase("/Users/dimuthuupeksha/Docum
ents/Academic/FYP/crawlers/Blog/BlogCrawler/src/test/resources/web")
;
```

```
        // Add the ResourceHandler to the server.
        HandlerList handlers = new HandlerList();
        handlers.setHandlers(new Handler[] { resource_handler, new
DefaultHandler() });
        server = new Server(9880);
        server.setHandler(handlers);
        server.start();
    }

    public void stop() throws Exception{
            server.stop();

    }
}
```

Instead of giving a URL of the real site, we gave URL of site hosted in JettyServer and checked whether content is properly downloaded.

5.2 API Performance Testing

REST API is the most important interface for SinMin to provide services to outside applications. So it is essential to test the stability of the API with different load conditions. The Stability of an API mainly depends on the request processing speed of backend databases. Some methods use Cassandra database while some methods use Oracle database to retrieve data. So in this test we measured the stability of each method of API with different load conditions

5.2.1 Tuning operating system

Since the server receives a large set of requests at a test, some OS settings of the server were changed to get maximum performance. These settings are the recommended settings that were used in ESB performance testing runs of http://esbperformance.org. These settings are mainly focused on utilizing the full port range, reducing the TCP fin timeout for better socket reuse, and setting a good limit on the number of file handles permitted to the process.

Edit /etc/sysctl.conf and append

```
net.ipv4.ip_local_port_range = 1024 65535
net.ipv4.tcp_fin_timeout = 30
fs.file-max = 2097152
net.ipv4.tcp_tw_recycle = 1
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_syncookies = 0
net.core.somaxconn = 1024
```

Edit /etc/security/limits.conf and append

```
* soft nofile 32768
* hard nofile 65535
```

Edit /etc/pam.d/common-session and append

```
session required pam_limits.so
```

5.2.2 Testing method

For load testing we used uterm console with jb-* commands that ships as a load testing tool with UltraESB of AdroitLogic. To send requests we used jb-single command with pre created payload files and header files. Different numbers of requests (different amount of threads with multiple iterations) were sent for each method of API and data like time taken for tests, number of completed/ failed requests, number of write errors, amount of transferred bytes, number of requests per second, time per request and transfer rates were collected.
Here is a sample report generated by jb-show command.

```
Server Software:       WSO2-PassThrough-HTTP
Server Hostname:       127.0.0.1
Server Port: 8243
```

```
Document Path:
https://127.0.0.1:8243/rest/1.0.0/trigramCount
Document Length:       119 bytes

Concurrency Level:     6
Time taken for tests:  308.551268 seconds
Complete requests:     45
Failed requests:       25
Write errors:          0
Kept alive:            0
Total transferred:     11192 bytes
Requests per second:   0.15 [#/sec] (mean)
Time per request:      41,140.169 [ms] (mean)
Time per request:       6,856.695 [ms] (mean, across all concurrent
requests)
Transfer rate:         0.02 [Kbytes/sec] received
                       0.01 kb/s sent
                       0.04 kb/s total
```

5.2.3 Results

Table 15 shows the time in seconds taken for each method against a different amount of request loads. Finally the average time per different amount of request loads were calculated. Figure 43 shows the graph of average time against different request loads.

Table 15: Time taken for each method against different ammount of request loads

	Time taken for tests (s)				
Method	30 Requests	40 Requests	50 Requests	60 Requests	70 Requests
wordFrequency	1.599386	2.344126	0.579143	2.241463	9.204325
bigramFrequency	68.416795	29.493321	15.499555	22.509835	67.38347
trigramFrequency	151.251321	112.030685	20.49532	43.604188	85.496288
frequentWords	190.800163	272.424459	332.931827	398.498049	444.787395
latestArticlesForWord	0.859966	0.69925	0.871029	2.452922	1.237825
latestArticlesForBigram	0.935384	0.605487	1.012713	1.326227	1.093929
latestArticlesForTrigram	0.972167	0.758981	0.798879	1.066257	1.976048
frequentWordsAroundWord	126.358204	85.416918	103.301834	174.091198	201.247924
frequentWordsInPosition	1.015491	1.152789	1.393903	1.362462	2.885438
frequentWordsAfterWordTimeRange	146.344746	189.518254	228.003734	309.681383	359.662293
frequentWordsAfterBigramTimeRange	30.095017	49.804954	45.801494	116.848206	219.868231
wordCount	0.669373	0.672034	0.749927	0.937789	0.879869
bigramCount	138.589387	167.755784	214.136583	232.108656	331.581301
trigramCount	136.726991	159.333983	225.1805	336.628791	398.551268
wildCardSearch	0.5057	0.471972	0.670411	10.584435	11.032443
Average Time	66.34267273	71.49886647	79.4284568	110.2627907	142.4592031

Table 16 shows the number of requests per second that each method was able to process

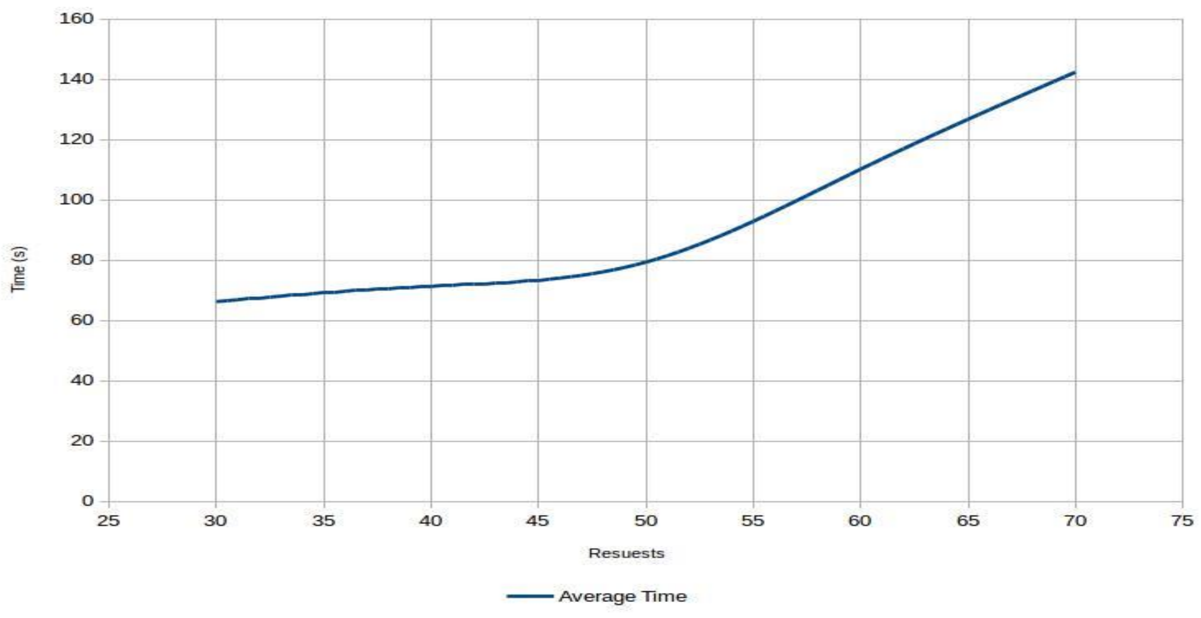


Figure 43: Average time against different request loads

Table 16: Number of requests per second under a different request load

	Requests per second				
Method	30 Requests	40 Requests	50 Requests	60 Requests	70 Requests
wordFrequency	18.13195814	17.06392916	86.33446316	26.76823128	6.518674645
bigramFrequency	0.409256236	1.356239265	3.225899066	2.487801443	1.009149573
trigramFrequency	0.152064788	0.303488281	2.439581329	1.192546	0.736874097
frequentWords	0.15723257	0.139488209	0.114137481	0.065244987	0.020234386
latestArticlesForWord	34.88510011	57.2041473	51.66303303	24.46062288	48.47211843
latestArticlesForBigram	32.07238952	64.41096175	46.4099898	43.73308642	54.84816656
latestArticlesForTrigram	30.85889564	51.38468552	62.58770102	56.27161182	30.36363489
frequentWordsAroundWord	0.22159226	0.468291305	0.484018512	0.310182253	0.268325749
frequentWordsInPosition	29.54235931	32.96353452	35.87050175	43.30396004	20.79407009
frequentWordsAfterWordTimeRange	0.204995402	0.195231853	0.201751082	0.122706763	0.088972352
frequentWordsAfterBigramTimeRange	0.731018029	0.783054633	1.091667446	0.462138032	0.20466804
wordCount	43.32412571	55.0567382	65.33969306	63.98027701	64.78237101
bigramCount	0.216466792	0.232480807	0.219486084	0.163716428	0.138729174
trigramCount	0.219415346	0.244768876	0.204280566	0.112883987	0.112908937
wildCardSearch	55.36879573	76.27571127	70.10624826	5.479744549	5.257221814
Average Requests Per Second	16.43304437	23.8721834	28.41949678	17.92765026	15.57440798

5.2.4 Discussion

Looking at the test results, the API provides the maximum throughput when load is 50 simultaneous requests. It means that the API can handle an average of 50 requests at a time.

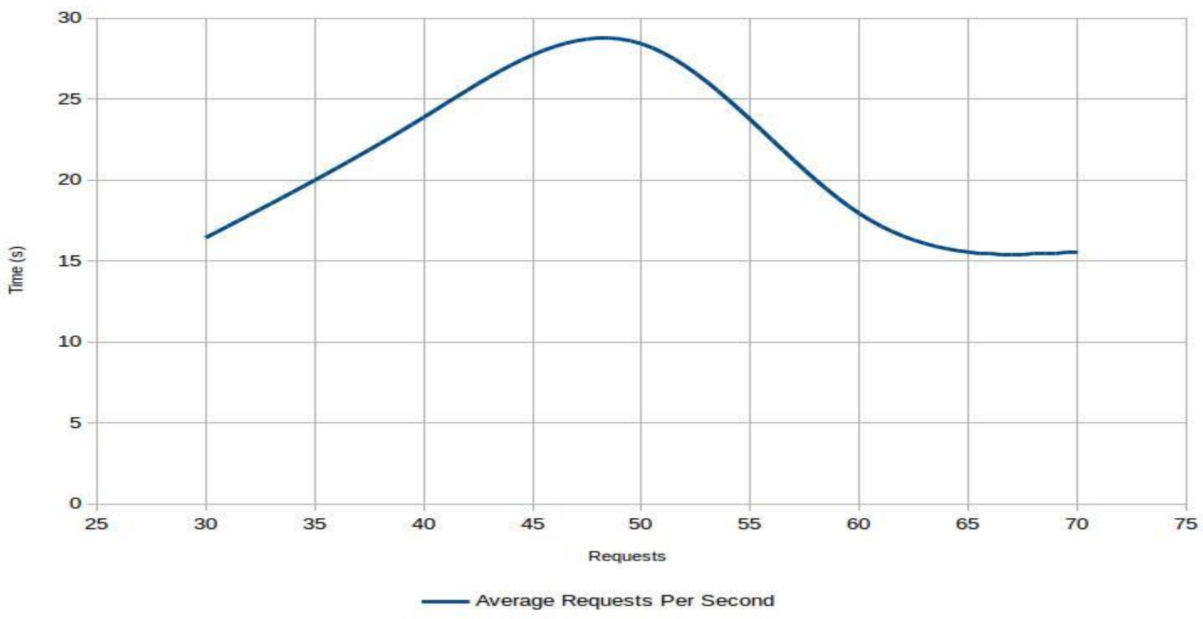
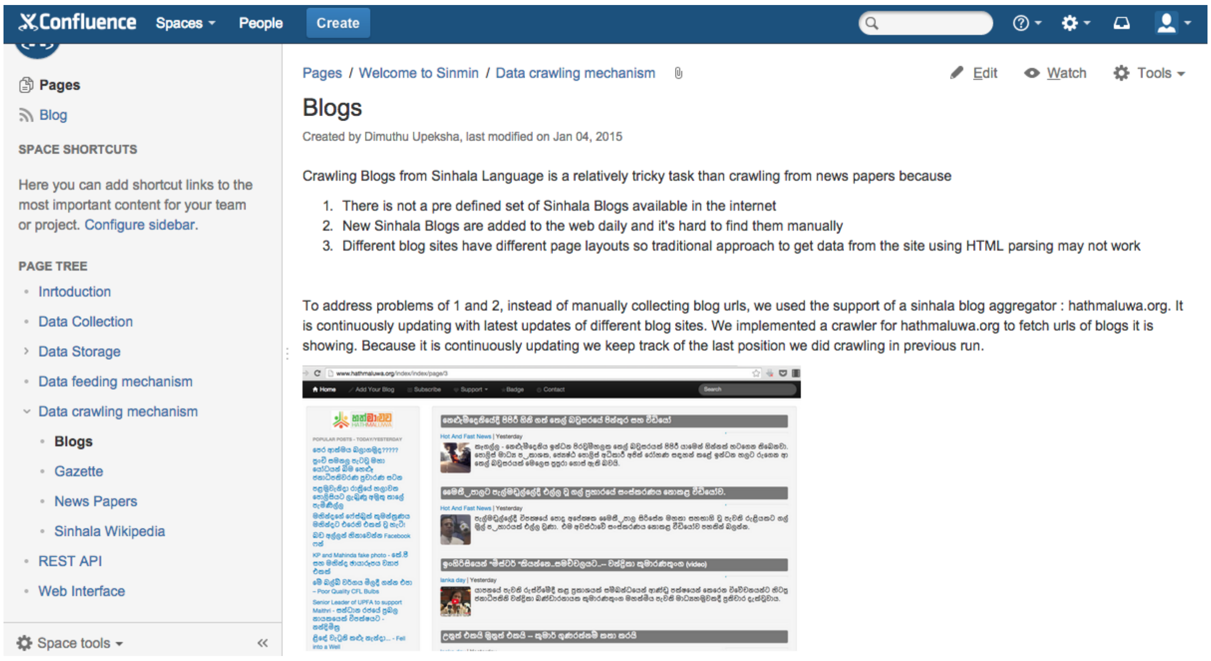


Figure 44: Average requests per second against different load conditions

6.0 Documentation

Documentation is an important part of any software product. Documentation describes how the software works and how to use it.

For documenting SinMin, we used Atlassian Confluence. Atlassian Confluence provides a platform where teams can share, find, and collaborate on information they need to get work done. We have used it for sharing information about how each components of SinMin operates and details about their architecture.



Here you can add shortcut links to the most important content for your team or project. Configure sidebar.

Pages / Welcome to Sinmin / Data crawling mechanism

Blogs

Created by Dimuthu Upeksha, last modified on Jan 04, 2015

Crawling Blogs from Sinhala Language is a relatively tricky task than crawling from news papers because

1. There is not a pre defined set of Sinhala Blogs available in the internet
2. New Sinhala Blogs are added to the web daily and it's hard to find them manually
3. Different blog sites have different page layouts so traditional approach to get data from the site using HTML parsing may not work

To address problems of 1 and 2, instead of manually collecting blog urls, we used the support of a sinhala blog aggregator : hathmaluwa.org. It is continuously updating with latest updates of different blog sites. We implemented a crawler for hathmaluwa.org to fetch urls of blogs it is showing. Because it is continuously updating we keep track of the last position we did crawling in previous run.

Figure 45 Confluence User Interface

7.0 Discussion

In this project we designed and developed a generalized language corpus for Sinhala language. Though there are some corpus implementations for Sinhala language, SinMin becomes significant because:

1. Continuous updating with new language sources that are added to the internet
2. Wide coverage of the language context
3. Powerful database backend for real time analytics
4. Pluggability of new crawlers at runtime
5. Fully open source
6. Rich front-end design and API for third party users

SinMin Corpus contains more than 70 million word tokens with more than 1.2 million distinct word tokens. Altogether SinMin aggregates 232, 000 articles from different language contexts and 4.8 million distinct sentences. Further, it keeps track of 17 million distinct bigrams and 39 million distinct trigrams. SinMin is the only Sinhala corpus that can provide bigram and trigram support. Looking at word tokens, SinMin contains 7 times bigger content than other available corpuses. Distinct word tokens content is also 10 times bigger than others. So, content wise SinMin is the largest corpus that is currently implemented for the Sinhala language.

Currently SinMin Corpus runs with more than 20 web crawlers which can crawl data from all available online sources like online newspapers, online news sites, blogs, Wikipedia, gazettes, subtitles and etc. In case of a new crawler, it can be added easily as a new module through Crawler Controller web application. Crawler Controller is responsible to control crawlers and schedule their running periods. This provides administrators an easy interface to control all crawlers through a single console. All crawled data is parsed and saved as annotated XML files by crawlers. Data Feeder application is then responsible for storing that raw data in databases.

SinMin is powered by two main database systems as the backend. Cassandra runs as the primary database. We selected Cassandra as our main storage system after following a very comprehensive research in performance measurements of different database systems and technologies. Cassandra is a NOSQL database that uses column stores as the storage mechanism. Due to its linear time performance in data insertion and low data retrieval latency, Cassandra works well with our requirement. Oracle provides support as a backup database. Unlike Cassandra, the design for Oracle database was done for more general data requirements. In addition to that, Apache Solr instance is also running as a prefix search engine. Solr uses permuterm indexing on distinct word tokens in order to support wildcard searches.

SinMin has a very rich user interface that can illustrate real time analytics of language patterns using interesting graphs and tables. We have implemented a comprehensive REST API that can be used by third party applications. It is very well documented in SinMin's official documentation with good examples.

8.0 Conclusion

SinMin is a dynamic and continuously updating corpus for the Sinhala language. SinMin Corpus contains about 70 million words belonging to time period of 2004-2014. They were collected from online resources which can be mainly divided into 5 categories: news, academic, creative writing, spoken and gazette.

SinMin uses Cassandra as storage, since Cassandra has performed better when retrieving information from a corpus compared to other competitive technologies. SinMin provides a REST API for outside users to retrieve information from it. It also has a web interface where people can access all the features in a summarized and interactive manner. This interface also uses the REST API when retrieving information from the database.

When comparing to other corpora, one limitation of SinMin is that words are not annotated with their Part of Speech (POS) tags and lemmas. The reason for this is currently there is no implemented POS tagger or morphological analyzer for Sinhala language.

Since Cassandra uses a query based data modeling, it uses a separate column family for each information need. Therefore if a new information need occurs, new column families may need to be created for them and data has to be inserted again.

The major use of SinMin is identifying statistical and linguistic features of the Sinhala Language. Other than that it can be also used in developing dictionaries, translators, POS taggers, optical character recognition tools and spell checkers for the Sinhala language.

9.0 Future Works

Currently there are many researches going on the field of language localization for Sinhala language. Most of those related researches will benefit from having a large corpus for the Sinhala language. Two of the most valuable research areas in language localization for the Sinhala language are developing a POS tagger for Sinhala and developing a morphological analyzer for the Sinhala language. Currently there are ongoing projects to address those two research problems, but they suffer from not having a large corpus for the Sinhala language. So after the creation of SinMin, those projects can use it and create effective POS tagger and Morphological analyzer for the Sinhala language. We have already discussed with a team working on developing a morphological analyzer for the Sinhala language and they intend to use SinMin in implementing their tools.

One of the biggest limitations in SinMin is not having words annotated with their POS taggers and lemmas. Once the POS taggers and morphological analyzers have been implemented, they can be used in annotating words in SinMin with their POS taggers and morphological analyzer, so more language related features can be extracted from the corpus.

A corpus is required to carry out natural language processing based studies on a particular language. With the creation of SinMin, we believe NLP based studies for Sinhala language will have a rapid growth in the next few years.

Research can be carried out for developing language related tools which uses SinMin such as spell checkers, grammar checkers and dictionaries.

The current OCR tool for the Sinhala language has accuracy of about 90% which makes it very less useful. Research can be carried out to identify how the accuracy of OCR can be improved using a corpus. We have identified areas of OCR such as full stop prediction of an image which can be enhanced using a corpus. However these are very wide research areas that have to be carried out as separate studies.

Other than the above mentioned studies, that are related to the Computer Science area, many Sinhala language related researches can be carried out using SinMin to identify what the recently evolved language patterns of the language are , how language has been changed from time to time, varieties of language used by people of different groups, etc.

References

1. Aksan, Y. and Aksan, M (2009) *Building a National Corpus of Turkish: Design and Implementation*, Working Papers in Corpus-based Linguistics and Language Education no. 3, 299-31. Tokyo: TUFS

2. Aston, G (1999) Corpus use and learning to translate, *Textus* 12, 289-314

3. Aston, G. and Burnard, L. (1997) *The BNC Handbook:Exploring the British National Corpus with SARA.*

4. baiscopelk.com (2008) බයිස්කෝප්සිංහලෙන් | Free Sinhala Subtitles, Available from: http://www.baiscopelk.com/ [Accessed: 02nd November 2014].

5. Bennet, G. R. (2010) *Using Corpora in the Language Learning Classroom*, Michigan ELT.

6. Biber, D. and Conrad, S. (2010) Corpus Linguistics and Grammar Teaching, Pearson Longman English Language Teaching Newsletter.

7. Biber, D., Conrad, S., & Reppen, R. (1998). Corpus linguistics: Investigating language structure and use.New York: Cambridge University Press.

8. Brigham Young University (1980) British National Corpus (BYU-BNC) Available from: http://corpus.byu.edu/bnc/ [Accessed: 02nd November 2014]

9. Brigham Young University (1990) Corpus of Contemporary American English (COCA) Available from: http://corpus.byu.edu/coca/ [Accessed: 02nd November 2014].

10. Burnard. L. (2007)Reference Guide for the British National Corpus (XML Edition), Available from: http://www.natcorp.ox.ac.uk/docs/URG/ [Accessed: 02nd November 2014].

11. Corpus Technologies, Tatar Corpus, Available at : http://web-corpora.net/TatarCorpus/search/?interface_language=en [Accessed: 02nd November 2014].

12. Davies, M. Google Books corpora (BYU), Available from: http://googlebooks.byu.edu/# [Accessed: 02nd November 2014].

13. Davies, M. (2002) *CORPORA: 1.9 billion - 45 million words each: free online access* Available from: http://corpus.byu.edu [Accessed: 02nd November 2014].

14. Davies, M. (2005) The advantage of using relational databases for large corpora, *International Journal of Corpus Linguistics* 10:3.p.307-335

15. Davies, M. (2009) The 385+ million word Corpus of Contemporary American English (1990–2008+) Design, architecture, and linguistic insights, *International Journal of Corpus Linguistics* 14:2.p.159-191

16. Denuro, J. and Uszkoreit, J. (2011) Inducing Sentence Structure from Parallel Corpora for Reordering, *In proceeding of EMNLP*.

17. Google (2013) Google Ngram Viewer,Available from: https://books.google.com/ngrams [Accessed: 02nd November 2014].

18. Granger, S. (2003) *The corpus approach: a common way forward for Contrastive Linguistics and Translation Studies?* Corpus-based Approaches to Contrastive Linguistics and Translation Studies, eds. S. Granger, J. Lerot, S. Petch-Tyson. Amsterdam/Atlanta: Rodopi.

19. Handford, M. and McCarthy, M. J. (2004) "Invisible to us" - A preliminary corpus based study of spoken business english, *Discourse In the Profession: Perspectives from Corpus Linguistics* 167-201

20. Jayaweera, A.J.P.M.P. and Dias, N.G.J. (2014) Hidden Markov Model Based Part of Speech Tagger for Sinhala Language, *International Journal on Natural Language Computing (IJNLC)* Vol. 3, No.3.

21. Jouili, S. and Vansteenberghe, V. (2013) *An empirical comparison of graph databases*, International Conference on Social Computing (SocialCom), 2013, pp. 708–715.

22. Koehn Philipp (2005). EuroParl: A Parallel Corpus for Statistical Machine Translation. *Machine Translation Summit 2005*. Phuket, Thailand.

23. Liu, Y. L., Wang, S. and Su, K. Y. Corpus based automatic rule selection in designing a grammar checker.

24. Maekawa, K. (2008) Balanced Corpus of Contemporary Written Japanese, *The 6th Workshop on Asian Language Resources*.

25. Muaidi, H. and Al-Tarawneh, R. (2012) Towards Arabic Spell-Checker Based on N-Grams Scores, International Journal of Computer Applications 53:3.

26. Naber, D. (2003) A Rule-Based Style and Grammar Checker, Diploma Thesis, Computer Science - Applied, University of Bielefeld.

27. Nazar, R., & Renau, I. (2012). Google Books N-gram Corpus used as a grammar checker. *Proceedings of EACL 2012: Second Workshop on Computational Linguistics and Writing (CL&W 2012): Linguistic and Cognitive Aspects of Document Creation and Document Engineering*. April 23, 2012, Avignon, France.

28. Parra Escartín, C. (2012). Design and compilation of a specialized Spanish-German parallel corpus. In Calzolari, N. C. C., Choukri, K., Declerck, T., Uğur Doğan, M., Maegaard, B., Mariani, J., Odijk, J., and Piperidis, S., editors, Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12), Istanbul, Turkey. European Language Resources Association (ELRA).

29. Pirinen, T. A. and Linden, K. (2010) Finite-State Spell-Checking with Weighted Language and Error Models—Building and Evaluating Spell-Checkers with Wikipedia as Corpus, *In proceedings of the 7th SaLTMil workshop on creation and use of basic lexical resources for less resourced languages*, Valletta, Malta.

30. Rachakonda R. T. and Sharma, D. M. (2011) , Creating an Annotated Tamil Corpus as a Discourse Resource, *Proceedings of the Fifth Law Workshop (LAW V),* pages 119–123,Portland, Oregon.

31. Real Academia Española (2014) Real Academia Española, Available from: http://www.rae.es/ [Accessed: 02nd November 2014].

32. Someya, Y. (2014) A Corpus-based Study of Lexical and Grammatical Features of Written Business English, Available from: http://www.someya-net.com/09-MA/ [Accessed: 02nd November 2014].

33. The TEI Consortium (2014) *TEI P5:Guidelines for Electronic Text Encoding and Interchange*,

34. Tian, L. et al. (2014) UM-Corpus: A Large English-Chinese Parallel Corpus for Statistical Machine Translation, *In proceeding of LREC*, European Language Resource Association

35. Weerasinghe,R.,Herath, D. and Welgama, V. (2009) *Corpus-based Sinhala Lexicon,* 7th Workshop of Asian Languages and Resources(ALR7).

36. Whitelaw, C. et al. (2009) Using the Web for Language Independent Spellchecking and Autocorrection, In EMNLP, pp. 890-899.

37. Xiaoping, J. and van Rij-Heyligers, J. (2008) Parallel Corpus in Translation Studies: An Intercultural Approach, *The international symposium on Using Corpora in Contrastive and Translation Studies Hangzhou China*.

38. Yusuf, Y. Q. (2009). *A corpus-based linguistics analysis on written corpus: colligation of "TO" and "FOR."* Journal of Language and Linguistic Studies, 5(2), 104-122

View publication stats