CS242: Information Retrieval & Web Search

Renjie Wu (862058526), Zhihui Shao (861257080), Tong Shen (862058291)

Part A: Collect your data and Index with Lucene

1. Collaboration

a) Coding

Renjie Wu: Overall design, implementation of class *"CrawlThread"* and class *"IndexThread"*.
Zhihui Shao: Implementation of class "*WriterThread*" and class "*RobotPolicy*".
Tong Shen: Implementation of class "*WikiCrawler*" and class "*Indexer*".

b) Report

Report is written by all of our three team members cooperatively.

2. Overview of the crawling system

2.1 Architecture

The crawling system generally consists of three components: crawler, writer, and database. These three components work in a pipeline: the crawler visits Wikipedia and crawls needed pages; the writer then writes those crawled pages into the database by batch; the database finally saves all necessary information associated with those pages and ensures the eventual consistency. Two intermediate layers are introduced to make those components work properly in a pipeline: a *LinkedBlockingQueue*, between the crawler and the writer, to play the queue's role in the producer-consumer model; independent database connections (SQL connection) built between the writer and the database, to perform writing transactions simultaneously. The basic architecture of our crawling system is demonstrated in Fig. 1.

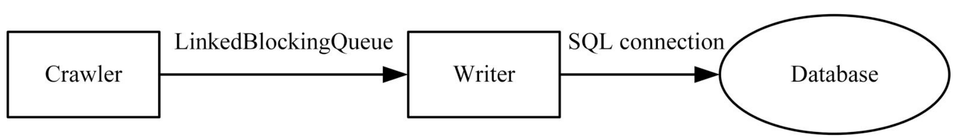


Figure 1 Basic architecture of the crawling system

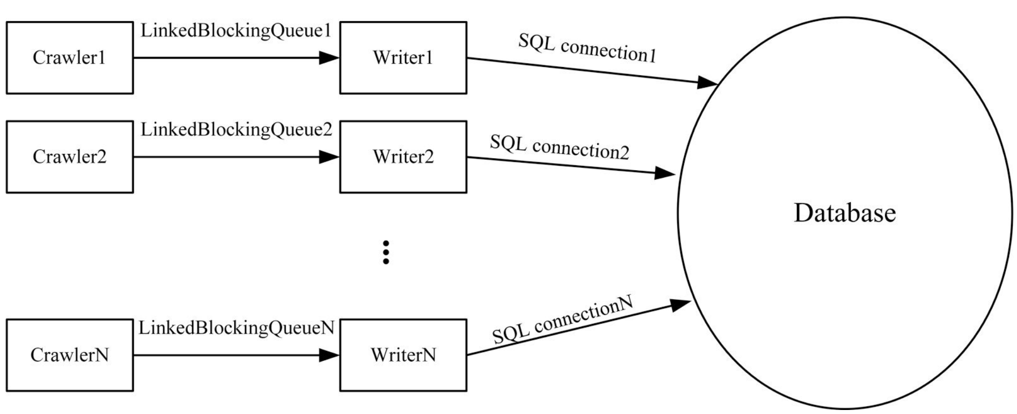


Figure 2 Full structure of the crawling system

To improve performance and efficiency, both the crawler and the writer would create multiple threads (*CrawlThread* for the crawler and *WriterThread* for the writer), to separate the workload and execute the crawling and the writing process parallelly. Each *CrawlThread* would work together with a *WriterThread*; they are connected by the LinkedBlockingQueue, where the *CrawlThread* pushes the crawled pages into the queue, and the *WriterThread* then removes the queue's head page and insert it into the database, through a dedicated database connection, associated with the *WriterThread*. The full structure of the crawling system is shown in Fig. 2.

2.2 Crawling Strategy

a) Strategy: *CrawlThread*'s routine

We adopted the general search algorithm in the *CrawlThread*. The default page depth limit is set to 10, since deeper pages may have already been crawled before. The following pseudo-code shows how *CrawlThread* works.

```
Start associated WriterThread;
Generate seed;
Add the random URL to Queue;
while Count<PagesToCrawl do
        if (Queue is empty) do
                Generate seed;
                Add the random URL to Queue;
        Get URL from Queue;
        Request the URL and download the web page;
        Add URL to visited set;
        Process content of the web page;
        Save processed web page into the LinkedBlockingQueue;
        if (depth of this web page exceeds limit) do
                continue;
        else do
                Get all links from the processed web page
                for URL in links do
                        if URL not in visited set and URL within Wikipedia do
                                Add URL to Queue;
end
```

b) Strategy: Seed generating and handling duplicate pages

A set of seeds or a set of initial URLs of Wikipedia pages are required to start the crawler. We set the entry url as *https://en.wikipedia.org/wiki/Special:Random*, since it would randomly redirect to a page, and that is ideal for seeding. The redirected page would act as the seed for each *CrawlThread*. In addition, to respect the crawler ethics, the crawler would first check website's robots.txt and determine whether the next url to crawl is allowed, and the *CrawlThread* would sleep for 5 seconds after finished processing a given page.

Duplicate pages handling is done through a set called *visitedURL*, saving all URLs of crawled pages and it is shared among all *CrawlThread*s, to check if a given URL has already been cralwed. *visitedURL* is implemented by *ConcurrentHashMap*, to ensure thread-safety. Page storage is done through SQLite, which provides support of eventual consistency and parallel writing, to improve the throughput of our writer.

c) Strategy: Wikipedia article processing

One of the important parts here is how to process the web page, and all the page process is coded in method *process()* in *CrawlThread* class. For almost all Wiki web page, there has one unique title; the main content, which is made of many paragraphs; and some words represent the category of this page. So, for every web page, we first split it into these three parts. And this is done by following three methods provided by jsoup.

Element elTitle = doc.getElementById("firstHeading");
Element elContent = doc.selectFirst("#mw-content-text .mw-parser-output");
Element elCategory = doc.getElementById("mw-normal-catlink");

i) Title

As the title is unique for all Wiki web pages, the title is used for the primary key when saving to SQLite database. Using title as primary key has two advantages than using URL: one is that primary key is shorter than URL; the other is that URL can be recovered from title when necessary.
The recovery process can be this:

URL = "https://en.wikipedia.org/wiki/" + title

For instance, the URL of web page with the title Computer is:

https://en.wikipedia.org/wiki/Computer

For title with more than one work, we need change the space to "_" in title. And then follows the recovery process. For instance, the URL of of web page with the title Computer science is:

https://en.wikipedia.org/wiki/Computer_science

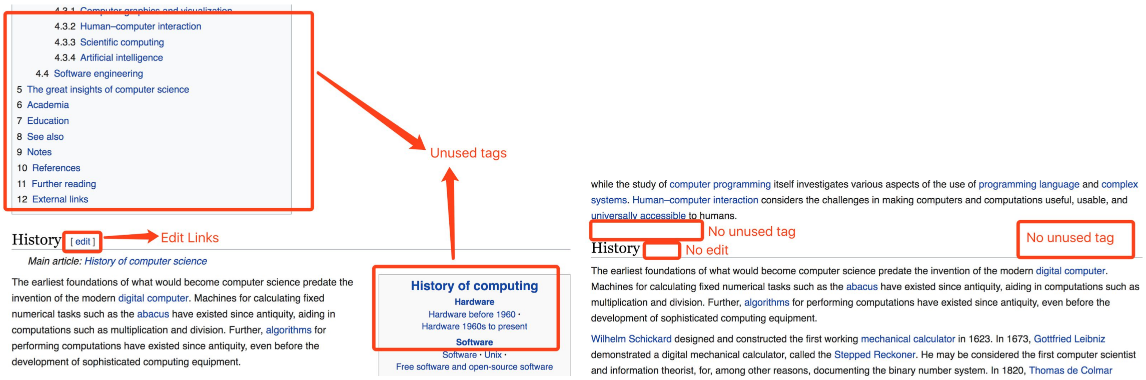
ii) Content

For the main content, we can simply convert the element to string using *text()* and *trim()* method.

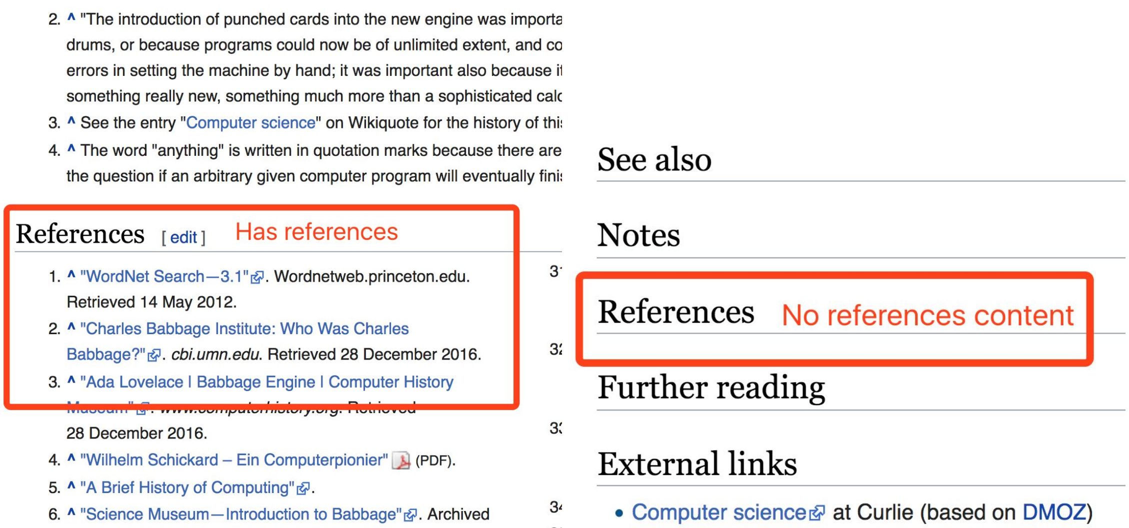
String title = elTitle.text().trim();

But it will generate references, edit links, spaces, \r\n, tables, and captions that is irrelevant the main content. In order to make the index part more efficiency, we need remove these irrelevant parts from the main content. As is shown in the Figure 3, after processing, we obtain the clean and plain content of the webpage.

To remove all reference, we use "*elContent.select("sup[class='reference']").remove()*"
To remove the `*edit*` links, we use "*elContent.select("span[class='mw-editsection']").remove()*"
To remove unused tags (such as table & div), we use "*Arrays.asList("table", "div").forEach(tag -> elContent.select(tag).remove())*". To remove empty headings with no paragraphs below it, we use "*Arrays.asList("h1", "h2", "h3", "h4", "h5", "h6").forEach(tag -> elContent.select(tag + "+" + tag).stream().map(Element::previousElementSibling).forEach(Element::remove));*"



(a) Raw content of edit and unused tag part (b) Content after process of edit and unused tag part



(c) Raw content of reference part (d) Content after process of reference part

Figure 3 Main content process example

iii) Category

For the words in category, we convert it to *List<String>*. And then saved into database.

d) Strategy: Storing articles into database

In our crawling system, one crawl thread will have one write thread to write the web page generated into the SQLite database. In order to improve the efficiency and throughput, the write thread use batch. Every 50 records from one batch, and they are written at one time. In order to concurrent the crawl thread and write thread, the page queue from crawl thread to write thread is blocked queue.

In our SQLite database, there are five attributes: *title*, *content*, *categories*, *lastModify*, *outLinks*. So, in the lucene index part, three of them are used for index process: title, content, and categories. Although the other two fields, *lastModify* and *outLinks*, is not used in index process, they can be used in query part. Here is a quick view of the result of our database for crawler:

<table>
<tr><th></th><th>title</th><th>content</th><th>categories</th><th>lastModify</th><th>outLinks</th></tr>
<tr><td>30</td><td>Sanskrit</td><td>Sanskrit (IAST: Saṃskṛtam; IPA: [sə̃…</td><td>Sanskrit|Indo-Aryan langu…</td><td>2018-02-06 21:07</td><td>IAST|Sacred language|Hinduism|Sikhism|Buddhism|I</td></tr>
<tr><td>31</td><td>Persian language</td><td>Persian (/ˈpɜːrʒən/ or /ˈpɜːrʃən/), …</td><td>Persian language|Language…</td><td>2018-02-09 11:57</td><td>Endonym|Western Iranian languages|Indo-Iranian I</td></tr>
<tr><td>32</td><td>Bengali literature</td><td>Bengali literature (Bengali: বাংলা সাহ…</td><td>Bengali-language literatu…</td><td>2018-02-06 16:20</td><td>Bengali language|Charyapada|Mangalkavya|Syed Sul</td></tr>
<tr><td>33</td><td>History of Bengali lite…</td><td>Ancient Age⏎Charyapada⏎The first wo…</td><td>Bengali-language literatu…</td><td>2018-01-12 14:10</td><td>Bengali language|Buddhism|Luipa|Bengali people|</td></tr>
<tr><td>34</td><td>Bangladeshi folk litera…</td><td>Bangladeshi Folk Literature (Bengali…</td><td>Bengali-language literatu…</td><td>2017-05-19 07:59</td><td>Bengali language|Bengali literature|Folklore|Ba</td></tr>
<tr><td>35</td><td>Bengali renaissance</td><td>The Bengali renaissance or simply Be…</td><td>Bengal Renaissance|Renais…</td><td>2018-01-23 19:32</td><td>Cultural movement|Social movement|Intellectual</td></tr>
<tr><td>36</td><td>Amar Sonar Bangla</td><td>Amar Sonar Bangla (Bengali: আমার স…</td><td>Asian anthems|Bangladeshi…</td><td>2018-01-30 13:15</td><td>Bengali language|National anthem|Bangladesh|Ode</td></tr>
<tr><td>37</td><td>Jana Gana Mana</td><td>"Jana Gana Mana" (Bengali: [ɟənə gəɳ…</td><td>Asian anthems|Bengali-lan…</td><td>2018-02-09 19:07</td><td>National anthem|India|Bengali language|Rabindra</td></tr>
<tr><td>38</td><td>Sri Lanka Matha</td><td>Sri Lanka Matha (Sinhalese: ශ්‍රී ල…</td><td>Asian anthems|1940 songs|…</td><td>2018-02-03 23:51</td><td>Sinhalese language|Tamil language|National anthe</td></tr>
<tr><td>39</td><td>Rabindranath Tagore</td><td>Rabindranath Tagore FRAS (/rəˈbɪndrə…</td><td>Rabindranath Tagore|1861 …</td><td>2018-01-28 02:38</td><td>Fellow of the Royal Asiatic Society|Sobriquet|Be</td></tr>
<tr><td>40</td><td>Sinhalese language</td><td>Sinhalese (/sɪnəˈliːz/), known nativ…</td><td>Southern Indo-Aryan langu…</td><td>2018-02-03 01:50</td><td>Sinhalese people|Sri Lanka|Indo-Aryan languages</td></tr>
<tr><td>41</td><td>Language Movement</td><td>The Language Movement (Bengali: ভাষা …</td><td>History of Bangladesh|His…</td><td>2018-02-09 03:35</td><td>Bengali language|East Bengal|Bangladesh|Officia</td></tr>
<tr><td>42</td><td>Dominion of Pakistan</td><td>Pakistan (Bengali: পাকিস্তান অধিরাজ্য p…</td><td>Former countries in South…</td><td>2018-01-28 17:37</td><td>Bengali language|Urdu language|Dominion|Pakistan</td></tr>
<tr><td>43</td><td>UNESCO</td><td>The United Nations Educational, Scie…</td><td>UNESCO|Organizations esta…</td><td>2018-02-06 17:33</td><td>French language|List of specialized agencies of</td></tr>
<tr><td>44</td><td>Language Movement Day</td><td>Language Movement Day or Language Re…</td><td>1952 protests|February ob…</td><td>2017-12-26 10:07</td><td>Bengali language|Bangladesh|Bengali Language Mov</td></tr>
<tr><td>45</td><td>International Mother La…</td><td>International Mother Language Day (I…</td><td>February observances|Inte…</td><td>2018-02-06 20:26</td><td>Linguistic diversity|Cultural diversity|Multilin</td></tr>
<tr><td>46</td><td>Bengali nationalism</td><td>Bengali nationalism (Bengali: বাংলা জ…</td><td>Political movements in Ba…</td><td>2018-01-29 06:51</td><td>Bengali language|Constitution of Bangladesh|Nati</td></tr>
<tr><td>47</td><td>Culture of Bengal</td><td>The culture of Bengal encompasses th…</td><td>Bengali culture|Banglades…</td><td>2018-02-06 00:53</td><td>Bengal|South Asia|Bangladesh|India|West Bengal|</td></tr>
<tr><td>48</td><td>1st millennium BC</td><td>The 1st millennium BC encompasses th…</td><td>1st millennium BC|Millenn…</td><td>2018-02-03 17:07</td><td>Before Christ|Iron Age|1000 BC|1 BC|Neo-Assyrian</td></tr>
<tr><td>49</td><td>Gupta Empire</td><td>The Gupta Empire was an ancient Indi…</td><td>Former monarchies of Asia…</td><td>2018-02-09 16:09</td><td>Outline of ancient India|Indian subcontinent|Gol</td></tr>
<tr><td>50</td><td>Vedic and Sanskrit lite…</td><td>Vedic and Sanskrit literature compri…</td><td>Indian literature by lang…</td><td>2017-12-14 06:07</td><td>Oral literature|Vedas|Indian epic poetry|Iron Ag</td></tr>
</table>

Figure 4 Result in SQLite database

3. Overview of the Lucene indexing strategy

When building Lucene index, we also run multi-threaded batch jobs to improve the performance of our system. In order to maximize the performance, all of the threads share one connection to database. When acquiring raw content, we use batch read to fetch records from our database. Intuitively, we chose the batch factor as 50 to make a balance between reducing I/O and not consuming too much main memory.

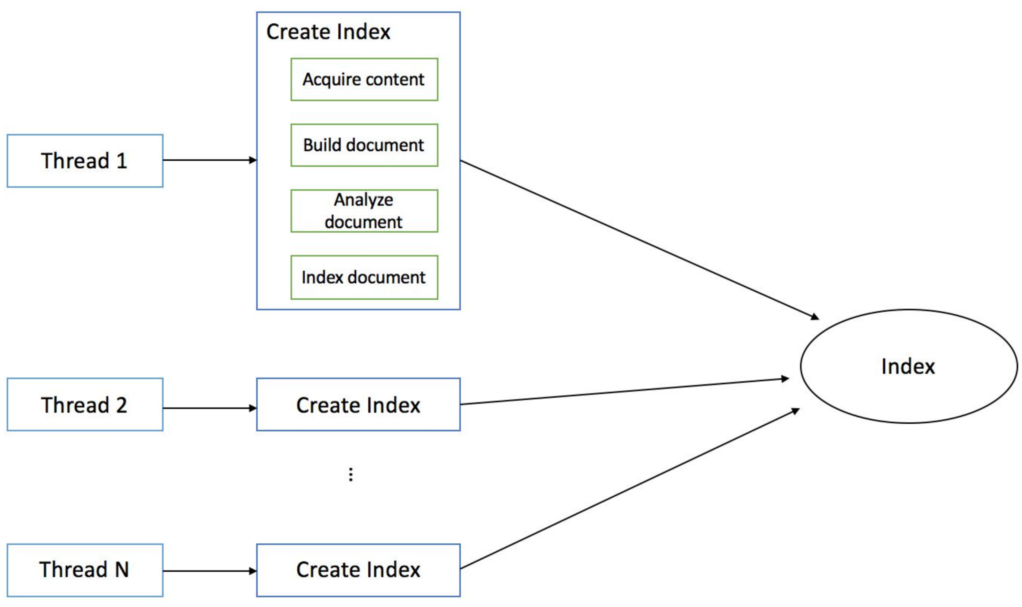


Figure 5 Architecture of Lucene Indexer

i) Fields in the Lucene indexing

We make one webpage as a document. To make it easier to maintain, we make each document has the three fields: title, content and categories. Since we want to search title, content and categories in the future, we build indexes for all of them, but in order to save some space, only title will be stored in the index and if user need to view the content or categories, the system will use the title as the key to fetch them from our database.

ii) Text analyzer choice

As for the analyzer part, since we already remove most of the irrelevant part and get the plain text while crawling the webpage as discussed before, the analyzer part is pretty simply and straightforward. We use a separate analyzer to analyze categories since when stored in the database we use the character "|" to separate different categories so that the categories field has one more special stop world "|". After that, we just use standardAnalyzer to analyze the documents.

iii) Progress reporting

We report the process when indexing every 1000 pages. As is shown in Figure 6, we finished indexing totally 744723 pages with in 3 hours and 20 minutes. Approximately, the running time is linear to the processed page number.

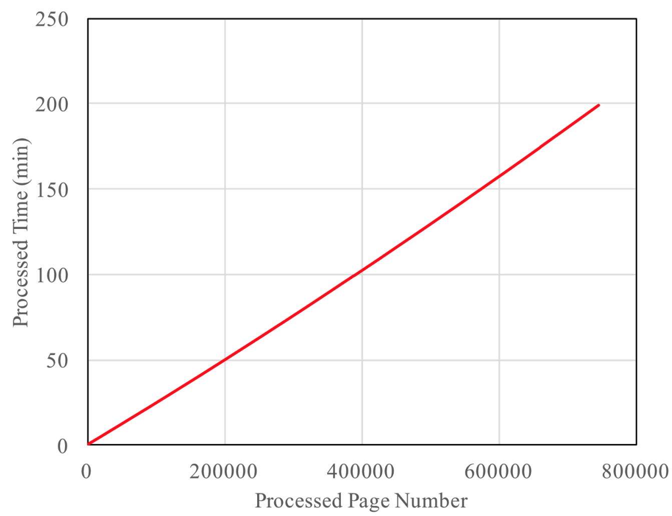


Figure 6 Run time of the Lucene index creation process

4. Obstacles and solutions

One big obstacle we encounter during coding is how to get only the main content of the web page. In Wiki page, there are many other parts other than the main content, such as the picture, table, edit hyperlink, reference link etc. If we do not remove these part from our main content text. There will be many irrelevant words and chars. So we spent a lot of time on this part. And it is handled using the method mentioned before.

5. Limitation and Achievement

There is one limitation in our crawling system: the consistency and corresponding efficiency. Although the *visitedURL* is used to avoid to search duplicate web page, the sequential consistency is still not ensured, because the *visitedURL* is not saved persistent with the database. Think about such case: Firstly, crawl the Wiki for some time, and stop; After some time, re-run the Crawler; when re-run the Crawler, the *visitedURL* is empty, and it is not consistent with the database.

To solve this, we ensure that this crawling system is eventually consistency by checking the page queue before write it database. This is done by the SQL statement: "INSERT **OR IGNORE** INTO pages". Therefore, every Wiki page in the database will be unique. But this solution is not very good. Because when we re-run the Crawler, it may crawler some pages that already saved in database before this run, and this reduce the efficiency of our crawling system.

In this project, we made the following achievements:

1. Both crawler and indexer are implemented with multi-threads and batch read & write database.
2. Respect the crawler ethic while crawling webpage, obeying robots.txt etc.
3. Wrap the code gracefully and make all parameters option.
4. Use database to store data, ensuring the eventual consistency and making reading and writing data more efficiently.
5. Crawler performs data cleaning, by removing unnecessary tags and saving paragraphs only.

6. Instruction

a) how to run the crawler: *./crawler.sh [options]*

Table 1 Options for crawler

Short Name	**Argument Name**	**Default Value**
t	*numOfThreads*	10
c	*numOfPages*	(this value will ensure 5GB data)750000
d	*crawlDepth*	10
i	*crawlInterval*	500
u	*entryUrl*	"https://en.wikipedia.org/wiki/Special:Random"
H	*crawlHostRegex*	"^en.wikipedia.org$"
P	*crawlPathRegex*	"^/wiki/[^:]*$"
	jdbcUrl	**Required parameter no default value**
l	*FILE NAME*	STDOUT

b) how to build the Lucene index: *./indexbuilder.sh [options]*

Table 2 Options for indexer

Short Name	**Argument Name**	**Default Value**
t	*numOfThreads*	10
l	*FILE NAME*	STDOUT
	jdbcUrl	**Required, no default value**
	indexOutputPath	**Required, no default value**