International Journal of Advances in Computer Science & Its Applications
Volume 6 : Issue 2 *[ISSN 2250-3765]*
Publication Date : 31 Auaust. 2016

Constructing Performance Comparison Environment of Search Engines

Takehiko Murakawa

***Abstract*—Comparison environments among different versions of free full-text search engines for verifying search performance and interoperability are reported. Those environments help the developers of retrieval services to decide whether the deployed search engine should be upgraded or not. In this research the target search engines are Apache Solr and Groonga. The constructed environment of Solr enabled one to make a crossover search using versions 1.4.1-5.4.0 together with a sequential search command, while for the variety of Groonga versions we adopted Docker to produce containers of the versions. The search comparison showed that relatively new versions made no difference with regard to search performance.**

***Keywords*—full-text search, search engine, developer support, version management.**

I. Introduction

Many people regularly use Internet retrieval services such as Google and Yahoo! to find desired information over the Internet. Apart from those worldwide or nationwide applications, many companies and organizations release the retrieval services for their own content to the public. Thanks to these independent full-text retrieval services, we enjoy instant, omission-free search, while the content holders and the developers are able to gain an understanding of their intellectual property and get on the trail of the service improvement, respectively, through the access logs. For the realization, the developers usually introduce a Web server and a full-text search engine which are ready-made while designing and implementing a Web application. Note that in this paper a "search engine" means a piece of software that performs a full-text search, but not an Internet service like Google.

It is important for the developer to select a suitable search engine in the development of the retrieval service. Among the previous works, a scalability evaluation was attempted [1]. Various open source search engines were reported in a workshop [2].

However performance should also be compared among versions of the same software. For example, Apache Solr [3] and Groonga [4] are search engines whose new versions have been frequently released. In the meanwhile, the developers of retrieval services suffer from the choice of version. Some have to select the appropriate version carefully while some have to judge whether or not the search engines used in their services should be upgraded according to the new version's release. They are concerned that a careless upgrade will cause the arrest or degrade the performance of their services. Therefore the operation check before the deployment or the upgrade has a significant role to play.

Wakayama University
Japan

Based on the circumstances describe above, we attempted comparative evaluation. In concrete terms, we constructed the comparison environment for major versions of Apache Solr and Groonga [5, 6], made indices of the same text files using the respectively installed versions, and confirmed the numbers of relevant documents by means of the common search terms.

II. About Solr and Groonga

Apache Solr (abbreviated as "Solr") and Groonga are high-performance search engines which are available freely. They have much in common; the software has been upgraded frequently; we can store and search documents by means of HTTP as well as using a traditional command line; the database is not unlocked while registering a document.

However the structures of those two search engines are entirely different. Solr holds Lucene to the heart of search functionality and includes practical features for document registration and search. Since this software is written in Java, we can extent the function by attaching one-of-a-kind Java class files. When configuring the properties, we usually edit some XML files and (re)start the server. The original version was created in 2004, and the latest version is 5.4.1. Newer versions of Solr have Kuromoji, a library for handling CJK (Chinese, Japanese, and Korean) strings properly, as a class file and described in a configuration file.

Groonga is a search engine made in Japan but supplies fulfilling English documentation. This program is written in C. While the basic functions are furnished in a C library, libraries for using Groonga in other programming languages, such as Python and Ruby, are available as well. It is not hard to serve a function in full-text search in cooperation with relational database management systems such as PostgreSQL and MySQL. Although a string is basically decomposed by means of N-gram, we can instead employ other natural language processing tools for decomposing documents to be registered and search terms. Taking advantage of the features described above, Groonga has been applied to the management of text data written in Japanese and other languages [7, 8]. The original version whose number is less than 1 was revealed in 2010, and the latest version is 5.1.1. In recent, a new version of Groonga is released on 29th every month.

This research attempts to compare the functionality of several versions, by constructing separate execution environments in parallel with the search engines. The comparison between Solr and Groonga is beyond our interest, although the difference among the Internet retrieval services [9] and the performance comparison between Lucene and Indri search engines [10] have been reported.

International Journal of Advances in Computer Science & Its Applications
Volume 6 : Issue 2 [ISSN 2250-3765]
Publication Date : 31 Auaust. 2016

III. Comparison among versions of Solr

A. *Constructing comparison environment*

The versions of Solr that we adopted for constructing the comparison environment, together with the release dates and the port numbers during execution, are shown in TABLE I. Although the default port number of the Solr server is 8983, we used other numbers depending on the versions so that we could run multiple versions in a server at the same instant and make a crossover search. We stored all the component files of Solr in a computer which runs Debian GNU/Linux.

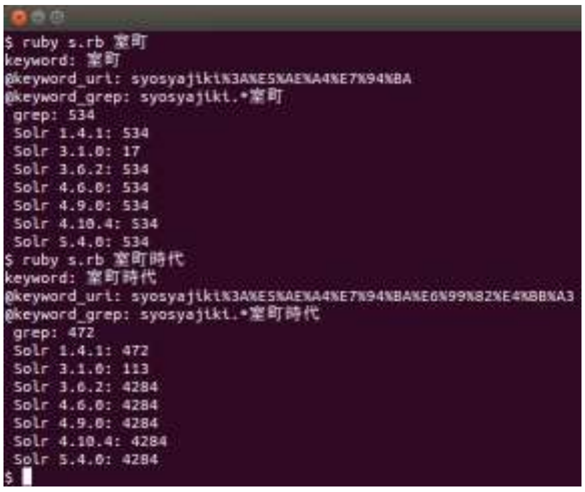
TABLE I. TARGET VERSIONS OF SOLR

Version	Release date (year-month-day)	Port number
1.4.1	2010-06-18	14183
3.1.0	2011-03-27	31083
3.6.2	2012-12-19	36283
4.6.0	2013-11-19	46083
4.9.0	2014.06-20	49083
4.10.4	2015-02-28	40084
5.4.0	2015-12-05	54083

Moreover we set on about 12,000 plain text files which describe bibliographic information about ancient documents, written in Chinese and (ancient and modern) Japanese, and converted them into XML files to register on each versions of Apache Solr.

For the registration and the retrieval, we made minimum amounts of configuration files' modification. We edited schema.xml to add "field" elements for the retrieval and remove unnecessary directives. In the configuration file of version 1.4.1, we additionally described "<tokenizer class="solr.CJKTokenizerFactory"/>" for the program to treat Chinese and Japanese characters appropriately. The "schemaless" mode, which newer versions of Solr provide, was not adopted.

We changed the port numbers of the Solr servers according as their versions so that every installed version could go live concurrently in the single Linux server. Moreover we made a Ruby script file. When we run the script with a search term in a command line, the program searches on the Solr servers in sequence, makes another search using the sequential search command grep, and reports the number of relevant documents of the search term. The screenshot of an execution example is shown in Figure 1.



```
$ ruby s.rb 室町
keyword: 室町
@keyword_url: syosyajiki%3A%E5%AE%A4%E7%94%BA
@keyword_grep: syosyajiki.*室町
 grep: 534
 Solr 1.4.1: 534
 Solr 3.1.0: 17
 Solr 3.6.2: 534
 Solr 4.6.0: 534
 Solr 4.9.0: 534
 Solr 4.10.4: 534
 Solr 5.4.0: 534
$ ruby s.rb 室町時代
keyword: 室町時代
@keyword_url: syosyajiki%3A%E5%AE%A4%E7%94%BA%E6%99%82%E4%BB%A3
@keyword_grep: syosyajiki.*室町時代
 grep: 472
 Solr 1.4.1: 472
 Solr 3.1.0: 113
 Solr 3.6.2: 4284
 Solr 4.6.0: 4284
 Solr 4.9.0: 4284
 Solr 4.10.4: 4284
 Solr 5.4.0: 4284
$ 
```

Figure 1. Example of command-line crossover searches.

B. *Results*

Typical search terms and the resulting number of relevant documents are shown in TABLE II. We describe the Roman spelling and the meaning of Chinese characters used in this table in Appendix. Before the search using grep, the developed Ruby script modified the search term including "AND" and "OR" to enable so-called AND and OR searches. We had no special handling for quoted search terms.

TABLE II. SEARCH RESULTS

Search term	Number of relevant documents			
	1.4.1	***3.1.0***	***3.6.2-5.4.0***	***grep***
室町	534	17	534	534
室町時代	472	113	4,284	472
室町時代写	68	68	4,284	68
"室町時代"	472	113	472	0
"室町時代写"	68	68	68	0
山寺	2,116	0	2,116	2,116
石山寺	1,282	560	2,116	1,282
"石山寺"	1,282	560	1,282	0
石山 AND 山寺	1,282	0	1,282	1,282
石山 OR 山寺	2,116	0	2,116	2,116

From the results of giving single search terms without quotation, we made sure that the numbers of the version 1.4.1 were in strict correspondence with those of grep, and that it indicated the numbers for two-letter search terms except for the version 3.1.0. Using the version 3.6.2 and newer, the number of relevant documents for a search term, denoted by S, having more than two characters was larger than those for the search term that consists of the first two characters of S. However if S is quoted, then the numbers are the same as what the version 1.4.1 reported.

When the search term S described above is tokenized by means of bi-gram (For example, if S is "石山寺", then the two tokens "石山" and "山寺" are derived.), and we made a search where the search term was the parted tokens with "AND" in between, we obtained the number just same as the

International Journal of Advances in Computer Science & Its Applications
Volume 6 : Issue 2 [ISSN 2250-3765]
Publication Date : 31 Auaust. 2016

quoted search term, apart from grep. When replacing "AND" with "OR" and searching, we found that the numbers by the version 1.4.1 and grep were the same as those by the versions 3.6.2-5.4.0.

These results follow that relatively new versions of Solr treat of a CJK search term as follows: (1) the search term is tokenized with bi-gram into two or more two-letter words; (2) if the original search term is quoted, then the relevant documents are the intersection of those of words; (3) if the original search term is not quoted, then the relevant documents are the union of those of words. In other words, the feature of morphological analysis using Kuromoji is not effective by default.

IV. Comparison among versions of Groonga

A. Constructing comparison environment

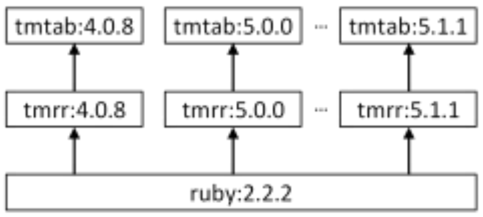
The versions of Groonga that we adopted for constructing the comparison environment together with the release dates are shown in Table III. Many versions were released on 29th in various months since the developer of Groonga would like to have special importance for the number 29, namely "niku," a Japanese word which means "meat," to produce a regular and aggressive improvement. The release of major upgrade to version 5.0.0 also includes "niku" by finding the number two in the second month of the year and combining the day. Version 5.0.2 was exceptionally released on a different day, including an urgent bug fix.

TABLE III. TARGET VERSIONS OF GROONGA

Version	Release date (year-month-day)
4.0.8	2014-11-29
5.0.0	2015-02-09
5.0.1	2015-03-29
5.0.2	2015-03-31
5.0.3	2015-04-29
5.0.4	2015-05-29
5.0.5	2015-06-29
5.0.8	2015-09-29
5.0.9	2015-10-29
5.1.1	2015-12-29

When introducing several versions of Groonga to a single computer, we need to take into consideration the versions of the required software, or the software that should be installed prior to Groonga's installation. To remove the burden of other software versions' management, we installed Rroonga, developed to use the functional capabilities of Groonga with a programming language Ruby. If we have a computer in which Ruby has been installed and the command line interface is available, and run the command "gem install groonga," then the latest version of Rroonga together with Groonga library will be installed. In addition, when executing "gem install groonga -v 5.0.0", the version 5.0.0 of Rroonga (and Groonga library) will be enabled. Although we are able to have two version or more installed in a computer, it seems that the later version is used when we run Ruby or a script file written in Ruby and load Rroonga. Conclusively we made a shell script file and execute it to introduce Rroonga of which the versions are listed on TABLE III, make an index for the prescribed text files, and forthwith remove Rroonga.

The generated index files can be used for static analysis such as file comparison, but it is inconvenient to get rid of Rroonga instantly since we cannot do the operability assessment. And then we attempted to make Docker's containers of Rroonga and the index, where Docker [11] is an open platform of virtualization. Figure 2 shows the Docker images used or developed for our purpose; "tmrr" is an image with Rroonga being installed, while "tmtab"



includes the document and index files. Each arrow denotes the dependency relationship.

Figure 2. Docker images developed for comparison environment of Groonga.

All the code of Dockerfile of tmrr:5.0.9, described for installing version 5.0.9 of Rroonga, is shown in Figure 3. The instruction is based on the ready-made image "ruby:2.2.2" and lets Docker install the library including Rroonga for future use. All the code of Dockerfile of tmtab:5.0.9, defined for indexing given documents using version 5.0.9 of Rroonga, is shown in Figure 4. This is derived from the image "tmrr:5.0.9"; after setting the environment variable for character code and importing document files and a script file "rr.rb" written in Ruby, the script runs to make an index for the document files and export all the database files to the host environment.

```
FROM ruby:2.2.2
MAINTAINER takehiko
RUN apt-get update && apt-get install locales locales-all
RUN gem install rroonga -v 5.0.9 --no-ri --no-rdoc
```

Figure 3. Dockefile of tmrr:5.0.9 described for installing version 5.0.9 of Rrronga.

```
FROM tmrr:5.0.9
MAINTAINER takehiko
ENV LANG=ja_JP.UTF-8
ADD ["text.tgz", "/root"]
ADD ["rr.rb", "/root"]
RUN cd /root && ruby rr.rb 5.0.9
```

International Journal of Advances in Computer Science & Its Applications
Volume 6 : Issue 2 [ISSN 2250-3765]
Publication Date : 31 Auaust. 2016

Figure 4. Dockefile of tmtab:5.0.9 described for indexing using version 5.0.9 of Rrronga.

It took about 70 seconds to complete a “tmrr” container and about 40 seconds for a “tmtab” container, using an Ubuntu 14.04 personal computer which equipped a CPU of Intel Core i7-4960X (3.60GHz). The difference of execution times among versions was not seen.

B. *Results*

The list of database files produced by Rroonga is shown in Table IV. Version 5.0.8 or older generated eight files while the latest two versions held an extra file. Common files were the same byte size, although the checksums obtained from the md5sum command were mostly different (the same check sums between some pairs of versions were found to a minor extent).

TABLE IV. DATABASE FILES PRODUCED BY GROONGA

Version	File name	Size (Bytes)
All	index.db	4,096
All	index.db.0000000	12,857,344
All	index.db.0000100	16,842,752
All	index.db.0000101	8,437,760
All	index.db.0000102	100,937,728
All	index.db.0000103	60,067,840
All	index.db.0000103.c	46,141,440
All	index.db.001	1,048,576
5.0.9 and 5.1.1	index.db.conf	8,437,760

In the next place, we activated “tmtab” containers and mounted another version’s database files. For several simple search terms, the number of the relevant documents were the same regardless of the pair of versions. The APIs (Application Programming interfaces) that are supplied for newer versions of Rroonga are operated on older version’s database file while Ruby threw an exception when we used the function available in newer versions in a container of older versions, no matter whether the mounted database was newer or older.

V. Conclusion

This paper reported comparison environments among various versions of Solr and Groonga so that we are able to verify search performance and interoperability. In the environment within Solr, we made sure that the number of the relevant documents undergo a huge change by the presence or absent of quoting the search term that has more than two characters, and that the numbers produced by version 3.1.0 differs from those by the others since that version holds an improper way of tokenization. In the environment within Groonga, we made sure that the database files generated by each version are compatible, that is, available to other versions. For both search engines, it was suggested that relatively new versions are compatible with one another and the developer can upgrade the search engine in the service with no incident.

Future works include the construction of comparison environment for a greater diversity or for other search engines such as Elasticsearch, and the provision of selection support system of search engines that will match the developers’ needs.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Numbers 26330365 and 25242037.

References

[1] R. Hayasaka, T. Hayashi, and R. Onai, “Development of a scalable search engine using open source softwares,” Computer Software, Japan Society for Software Science and Technology, Vol.26, No.4, pp.138-156, 2009 (in Japanese).

[2] A. Trotman, C. L. A. Clarke, I. Ounis, S. Culpepper, and M.-A. Cartright, “Open source information retrieval: a report on the SIGIR 2012 workshop,” ACM SIGIR Forum, Vol. 46, Issue 2, pp.95-101, 2012.

[3] Apache Solr. http://lucene.apache.org/solr/ (Accessed 28 January 2016).

[4] Groonga - An open-source fulltext search engine and column store. http://groonga.org/ (Accessed 28 January 2016).

[5] T. Murakawa and K. Fujii, “Performance comparison of search engines,” Proceedings of the 2015 IEICE General Conference, D-4-4, 2015 (in Japanese).

[6] T. Murakawa, “Performance comparison among versions of search engine Groonga,” Proceedings of the 2016 IEICE General Conference, D-4-12, 2016 (in Japanese).

[7] K. Sato, K. Takeuchi, and K. Kageura, “Terminology-driven Augmentation of Bilingual Terminologies,” Proceedings of the XIV Machine Translation Summit, pp.3-10, 2013.

[8] M. Yoshioka and T. Fujiwara, “Construction of a Japanese gazetteers for Japanese local toponym disambiguation,” Proceedings of the 7th Workshop on Geographic Information Retrieval (GIR ’13), pp.57-63, 2013.

[9] L. Vaughan, “New measurements for search engine evaluation proposed and tested,” Information Processing & Management, Vol.40, Issue 4, pp.677-691, 2004.

[10] H. Turtle, Y. Hegde, and S. Rowe, “Yet another comparison of Lucene and Indri performance,” SIGIR 2012 Workshop on Open Source Information Retrieval, pp.64-67, 2012.

[11] Docker - Build, ship, and run any app, anywhere. https://www.docker.com/ (Accessed 28 January 2016).

Appendix

We give an account of the Chinese characters used as the search terms in the comparative evaluation of Solr. “室町” (Muromachi) is a place where the Japan’s feudal government had been located in the 14-16th centuries, and the era of that government is called “室町時代” (Muromachi Jidai). Since the character “写” (utsushi) means “transcribed”, the search term “室町時代写” (Muromachi Jidai utsushi) will give a search query to find the bibliographic information whose document was transcribed in Muromachi Era. Although “山寺” (yamadera) indicates a mountain temple in Japanese, we use the word only as the last two characters of “石山寺” (Ishiyamadera) in this paper. “石山寺” (Ishiyamadera) is a well-known temple of Japan, located in “石山” (Ishiyama), near Lake Biwa.

About Author:

Takehiko Murakawa is an Associate Professor of Faculty of Systems Engineering, Wakayama University, Japan. He received the doctoral degree from Nara Institute of Science and Technology, Japan, in 1998. His research interests include information retrieval and recommendation, digital