# Methodology of Selecting the Hadoop Ecosystem Configuration in Order to Improve the Performance of a Plagiarism Detection System

Andrzej Sobecki(✉) and Marcin Kepa

Gdansk University of Technology,
ul. G. Narutowicza 11/12, 80-233 Gdansk, Poland
{andrzej.sobecki,marcin.kepa}@pg.gda.pl
http://task.gda.pl/

**Abstract.** The plagiarism detection problem involves finding patterns in unstructured text documents. Similarity of documents in this approach means that the documents contain some identical phrases with defined minimal length. The typical methods used to find similar documents in digital libraries are not suitable for this task (plagiarism detection) because found documents may contain similar content and we have not any warranty that they contain any of identical phrases. The article describes an example method of searching for similar documents contains identical phrases in big documents repositories, and presents a problem of selecting storage and computing platform suitable for presented method using in plagiarism detection systems. In the article we present comparison of the mentioned above method implementations using two computing platforms: KASKADA and Hadoop with different configurations in order to test and compare their performance and scalability. The method using the default tools available on the Hadoop platform i.e. HDFS and Apache Spark offers worse performance than the method implemented on the KASKADA platform using the NFS (Network File System) and the processing model Master/Slave. The advantage of the Hadoop platform increases with the use of additional data structures (hash-map) and tools offered on this platform, i.e. HBase (NoSQL). The tools integrated with the Hadoop platform provide a possibility of creating efficient and a scalable method for finding similar documents in big repositories. The KASKADA platform offers efficient tools for analysing data in real-time processes i.e. when there is no need to compare the input data to a large collection of information (patterns) and to use the advanced data structures. The Contribution of this article is the comparison of the two computing and storage platforms in order to achieve better performance of the method used in the plagiarism detection system to find similar documents containing identical phrases.

J. Szymański and Y. Velegrakis (Eds.): IKC 2017, LNCS 10546, pp. 56–69, 2018.
https://doi.org/10.1007/978-3-319-74497-1_6

## 1 Introduction

Along with the prevalent computerization, the number of processes in which digital documents replaced their traditional paper counterparts has largely increased. The effectiveness of using the digital documents by employees in an organization depends most on the available tools, such as the digital repositories of documents, methods used to search for the documents that comply with the defined requirements, and programs that provide the possibility of analysing the information stored in the documents. One of the obvious trends in the development of tools, used for handling the digital documents, is the increase of interest in the efficient methods of searching for similar documents. Methods of searching for similar documents may be used in different areas, according to the type of document content, e.g.

- health records with a description of treatment methods in a medical repository [1];
- court rulings in judgment document repository [2,3];
- the results of scientific research, articles and books in scientific institutes repositories [4,5];
- theses in the universities repositories [6];
- different unstructured, text documents in the plagiarism detection systems [7,8].

Areas of application for the methods of searching for similar documents are defined by the function used to calculate the degree of similarity between the two documents. We could divide these areas into two classes in which one:

- user expect documents with similar content based on e.g., keywords or a bibliography;
- user expect documents contain patterns defined by him in the request like e.g., set of text phrases;

To the first class, we could assign solutions to use in digital libraries in order to find books based on the meta-data or keywords specified by the user. In this class user generally, does not specify his requirements concern to the phrases uses in sought books. The second of a mentioned-above class includes solutions that offer for user possibility to define requirements concerning the patterns that he expects to find in sought documents. An example of such solution belong to the second class is the plagiarism detection system, where the method searches for text fragments (phrases) from published documents, which were used in a new, unpublished document, omitting the information that relates them to the sources of the borrowed fragments. One of the obstacles in common use of the methods of searching for similar documents in the plagiarism detection systems is the number of documents stored in the digital repositories.

One of the proposed approach used to increase the performance of these methods is to perform calculations in parallel-way in the cloud [9] using the special platforms for storing documents and management of the parallel execution of tasks in the cloud computing environment. The aim of the article is to present a comparison of the storage and computing platform suitable for the problem of finding patterns in unstructured, text documents that each one is smaller than 1 MB. In the article, we have compared two platforms: KASKADA [10] (actual used) and Hadoop [11]. The analysis of these platforms was conducted using the existed plagiarism detection system SowiDocs[1]. *SowiDocs* is an anti-plagiarism system that uses the *KASKADA* platform in order to have the possibility to perform parallel computations on the data streams in real time, using the services working in the *Master/Slave* model. *KASKADA* serves the tools for creating user defined functions *UDF* in C++ language and publishes the created *UDFs* as services. This system has been developed at the Technical University of Gdansk and is used since 2010 in the University to detect plagiarism in student works. The article presents the comparison of the different method implementations for retrieving similar documents containing defined patterns, using the *KASKADA* and *Hadoop* platforms.

The Contribution of this article is the comparison results which presents advantages and disadvantages of two computing and storage platforms and their capabilities to achieve better performance of algorithms used in the plagiarism detection system to finding similar documents containing identical phrases.

## 2 State of Art

Algorithms for solving one of the following problems: Longest Common Substring (LCSg) [12] or Longest Common Subsequence (LCSe) [13–16] are most commonly used to detect plagiarism in the text documents. A drawback of these algorithms is their high computational complexity, which prevents the development of an efficient method for scanning the document repository in order to operate in a time acceptable to the user. One of the methods used to increase the performance of the plagiarism detection process is the pre-selection of similar documents that contain patterns defined in the analysed document. The purpose of the pre-selection is to reduce the set of input documents for algorithms that solve the problems of *LCSg* and *LCSe*. Methods used for pre-selection do not compare the original document content instead they use some kind of representation that is called a document profile. An example of methods used for pre-selection of documents based on the document profiles, are:

- inverted index [17–20],
- n-grams [21–23],
- semantic similarity [24–26],
- natural language processing [27,28],
- hashing document content [29–34].

Regardless of the methods used to pre-select the similar documents, their performance often depends on their implementation, the number of documents

[1] https://sowi.pg.gda.pl/.

stored in the repository and the computing platform used. The number of documents in the repositories is growing steadily, so the presented methods must be implemented in a computing platform that provides high performance and scalabilities like *KASKADA* or *Hadoop*.

The advantage of the *KASKADA* platform is mediating in the access to resources provided by cloud computing and automatic management of the running tasks. The second of the mentioned platforms is *Hadoop*, which is now one of the most popular solutions to create scalable and high-performance computing platforms. The reason for its popularity is the portability of solutions created using *Hadoop* platform tools, ease of installation and maintenance, as well as a set of additional components of the *Hadoop* ecosystem like *HBase* – a distributed, scalable, big data store (Big Table implementation [35]), *YARN* (Yet Another Resource Negotiator) and *Apache Spark* [36] – a fast and general engine for large-scale data processing.

## 3 Details of Our System

The *SowiDocs* system comprises of several processing stages, designed to analyse similarities between documents with increasing accuracy. The tasks are running in the cloud computing environment, according to the process described in the selected web service scenario. The scenario describes the process of the documents similarity computation in the *SowiDocs* system, using the services divided into the following stages:

- Converting – converting a received document to the text format without formatting and images,
- Mapping – computing the document profile based on the content generated by the preceding service (*Converting*) and use the following algorithms: Rabin fingerprinting [37], BSW [38] (Basic Sliding Window) and TTTD [39] (Two Thresholds Two Divisors),
- Searching – finding the similar documents by counting identical numerical values, occurring in the compared documents profiles,
- Filtering – filtering the found documents by calculating the similarity between two documents based on similarity of their contents.

An accuracy of the plagiarism detection process mostly depends on the algorithms used in the filter stage where documents are analyzed in order to find all longest common substrings. While the performance of this process mostly depends on the algorithms used in the searching stage where we try to find only this documents that contain some patterns and omit the documents that not comply with this requirement.

To achieve that in the *SowiDocs* we use document profiles that represent original text documents by fingerprint values. Each fingerprint is calculated based on the content of the sliding window. The first fingerprint value for a document is calculated based on Eq. (1) [34] and for every other position of the sliding window a single fingerprint value is calculated based on Eq. (2).

$$F_1 = (t_1 \cdot p^{n_t - 1} + t_2 \cdot p^{n_t - 2} + \cdots + t_{n_t}) mod 2^x \tag{1}$$

$$F_{i+1} = (F_i \cdot p + t_{n_t - i} - t_i \cdot p^{n_t}) mod 2^x \quad (2)$$

where:

- $F_i$ – single *i-th* fingerprint value,
- $p$ – a prime number,
- $t_j$ – *j-th* char from the document,
- $n_t$ – a length of the sliding window,
- $x$ – exponent, e.g. 30 or 31.

## 4 Problem Statement

In order to achieve satisfying performance, and accuracy of the plagiarism detection process, we should find in the third stage of this process only these documents that contain patterns defined by the user. Moreover, the method used in the third stage can not omit any of the document contains that patterns. Because of that, the method using for searching for documents in the third stage was chosen to test the profitability of migrating the system from existing platform KASKADA to the Apache Hadoop Platform. The problem of a selection the platform to perform the computation arises from the number of files and its size. In one repository we have about 500.000 documents with each is smaller than 1 MB. In the one analysis process, we could use few different repositories depending on the content of the analysed document and user requirements.

Originally, the third stage of the analysis process was developed as a Master/Slave algorithm on the *KASKADA* platform. The analysed documents were saved in the repository as files in a *NFS* [40]. The task of analysing a single document was divided by the master between the slave nodes. Each slave had to compare the document with a subsection of the repository. This was done by iterating over the document profiles from the subsection and comparing them to the test document. First, every profile form the subsection was loaded into the memory as a key-value hash-map (with the fingerprints as the keys), after that, every fingerprint in the test document profile was searched for in the hash-map. A simplified scheme of processing the documents using the *KASKADA* platform can be seen in Fig. 1.

As the repository grew bigger, the time needed to process a single document grew linearly relative to its size. What's more important, limitations of the *NFS*, combined with the complexity of iterating over every file in the repository to create its hash-map, made the entire process long, even for small files.

## 5 Proposed Solutions

Three possible approaches were designed and tested. All three approaches were in fact MapReduce algorithms written in the *Apache Spark* environment:

- a naive approach, using a *HDFS* file repository – further called the *HDFS* approach,
- a complex approach, using a hash-map and a *HDFS* file repository – further called the *HDFS* map approach,
- a *Hbase* based approach – further called the *Hbase* approach.

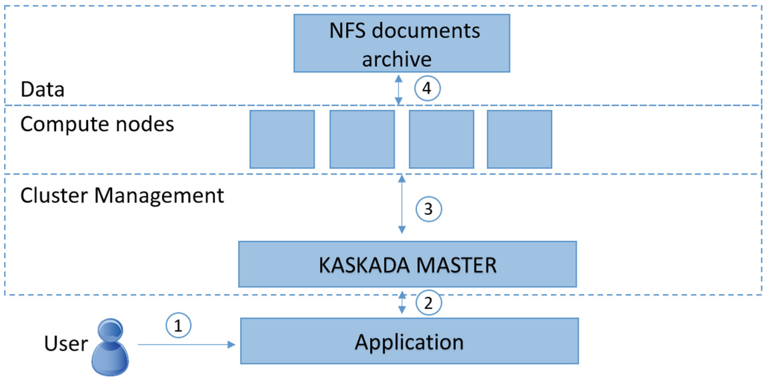

**Fig. 1.** *KASKADA* schema

## 5.1 HDFS Approaches

Both the first and the second approach work on a *HDFS* file-based repository of documents. It is important to note, that unlike in the *KASKADA* approach, the file system here is distributed among the compute nodes using *HDFS* (with data blocks replicated between nodes). A simplified scheme of documents processing in such *Hadoop* ecosystem can be seen in Fig. 2.

The first approach works by comparing every line of the profiles repository (saved as files in *HDFS*) to every line in the tested document – this means that for a document containing $n$ fingerprints and a repository containing $m$ fingerprints the solution does $n \cdot m$ comparison operations.

The *HDFS* map approach is an improved version of the first approach. Instead of doing $n \cdot m$ comparison operations, the tested document is converted into a key-value hash-map (with the fingerprint as the key). Because of that the search operation is simplified to $m$ searches in the hash-map - provided there is enough system memory the computational complexity is reduced from $O(n \cdot m)$ to $O(m)$. This means that the analysis time is largely independent from the size of the tested document.

## 5.2 HBase Approach

The third and final approach uses, a *Hbase* implementation of a reversed index. The database was created on top of the existing data nodes of the Hadoop cluster. On each data node a *Hbase* Region server was installed. A *Hbase* master server was also installed on one of the master nodes. A simplified scheme of the Hadoop platform in the *HBase* setup can be seen in Fig. 3.

The schema of the database was designed as a reversed index:

– every row is indexed by a value of a fingerprint,
– every row contains a number of columns (part of a single column family) - each named after a single document containing this fingerprint,

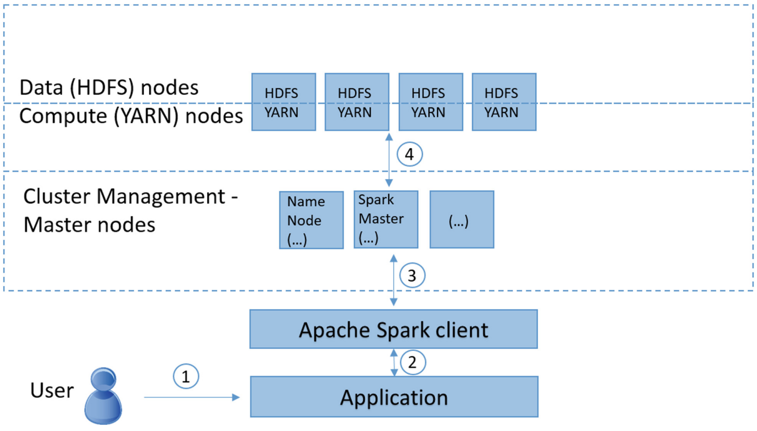

**Fig. 2.** Hadoop HDFS schema

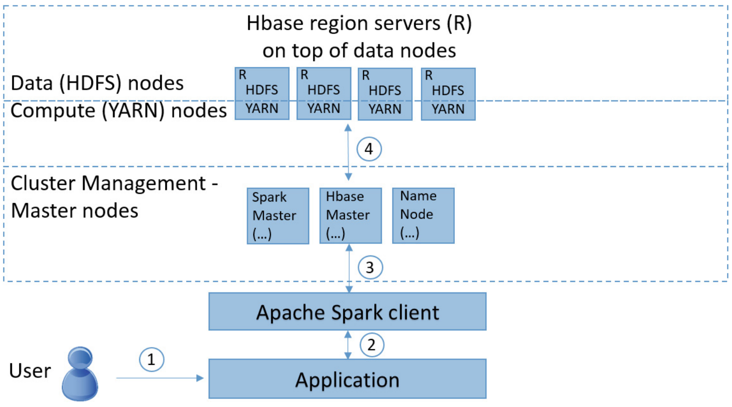

**Fig. 3.** *Hadoop* and *HBase* schema

– each column contains a single integer value, representing the number of occurrences of this fingerprint in the related document.

This in summary means, that the computational complexity is further reduced to be $O(n \cdot f(m))$. – where $f(m)$ is the number of operations required to retrieve a row from *HBase*. As $m \gg n$ and $f(m) < m$ this solution should be the fastest one of the three.

## 6 Experiments

In order to sufficiently evaluate the proposed solutions we planned a series of experiments, which was designed to test such features, as time performance and scalability in relation to the size of a repository and the size of the analysed document. Later, we run those tests in our testing environment on a supercomputer *TRYTON*[2]. Mentioned tests were done on a prepared dataset, created from real-world diplomas that had been writing by students at the Gdansk University of Technology. The minimal similarity between selected documents was about 2% and it resulted from the similarity of title pages, statements and university affiliations.

### 6.1 Planned Tests

As mentioned before, we planned a series of tests in order to evaluate our approaches and their viability to process a constantly growing database of text documents. Details of these tests were as follows:

- the time needed to search our repository depending on its size – in total of 17 000 real and unique (excluding cases of plagiarism) documents of varying size.
- the time needed to insert a new document into the repository – as it is almost equally important to the search time (since the standard use-case consists of both similar documents search and repository inserts). As before, this time was measured for different sizes of the repository, in order to evaluate the scalability of this operation.
- the scalability of each solution – depending on the size of the tested document.
- the scalability of the *KASKADA* and the *Hadoop* ecosystem – depending on the number of concurrent requests and the number of available processors.

Each test was conducted on a random batch of $n$ documents from the dataset, and all the results were averaged over this batch. Results of those tests can be seen in Subsect. 6.4.

### 6.2 Dataset

Our dataset consisted of around 17 000 text documents (stripped of formatting and images), varying in size from few kilobytes to one megabyte. Each document was a real and unique diploma – the repository consists of works from students of the Gdansk University of Technology. For every document there are two files in the repository – one containing its text and another, containing it's profile. Every profile contains a few hundreds to several hundred thousands of fingerprints.

### 6.3 Test Environment

All Hadoop tests were run in a cluster environment, comprising of eight nodes in an *openstack* environment – three master/manager nodes and five *HDFS* data nodes with *HBase* Region Servers. Each data node was equipped with 8 CPU cores and 32 GB of memory. Tests were run with *Apache Spark*, each *YARN* task was limited to 6 cores per data node, giving a total of 30 processors.

[2] Specification at TOP500 website: http://www.top500.org/system/178552.

All *KASKADA* tests were run in a cluster environment comprising of 8 nodes – 5 management nodes and 3 compute nodes. Each compute node was equipped with 12 CPU cores and 32 GB of memory.

## 6.4 Results

The first test we run was the experiment measuring the search operation time per 10 000 fingers, depending on the number of documents in the repository. The results of this test can be seen in Fig. 4. The naive iterative *HDFS* approach is a few orders of magnitude slower than the other three. The reason of that probably depends on the number of files in the repository. In this approach, we did not compact small files into larger buckets. Furthermore, the *KASKADA* map and the *HDFS* map approaches achieved similar results – the *HDFS* map approach is only two times faster. The *HBase* based approach is the fastest – achieving performance better by an order of magnitude from the original *KASKADA* approach. Also the *HBase* approach scales much better (close to logarithmically) than the *HDFS*, *HDFS* map and the *KASKADA* map approaches, which all scale close to linearly. Results for the every approach achieved the same level of accuracy.

Next, we run the import operation time test – in order to check whether the operation of importing new files into *HBase* doesn't impact the performance gain negatively. The results of this test can be seen in Fig. 5. The time of import a new document is almost independent of the repository size. Furthermore, this time is almost negligible compared to the search operation times.

The third test we run was aimed at measuring the scalability of all three solutions in relation to the processed document size. The results of this test can be seen in Fig. 6. The *HBase* and *HDFS* solutions scale close to linearly. The *KASKADA* map solution time grows very slowly depending on the size of the analysed document, the *HDFS* map solution time was almost constant.

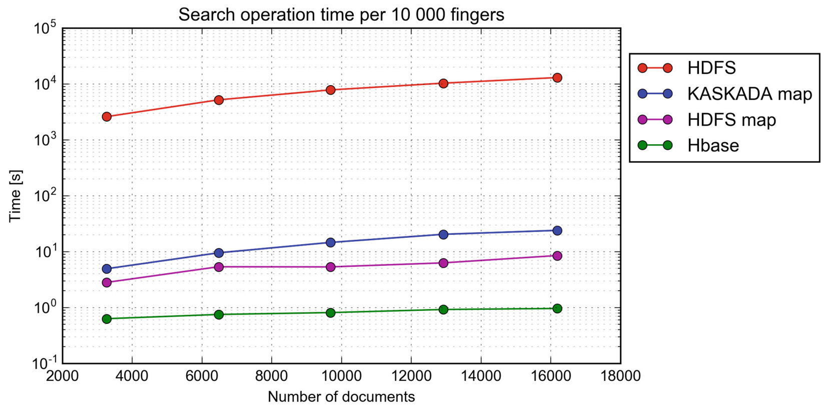

**Fig. 4.** Search operation time per 10 000 fingers

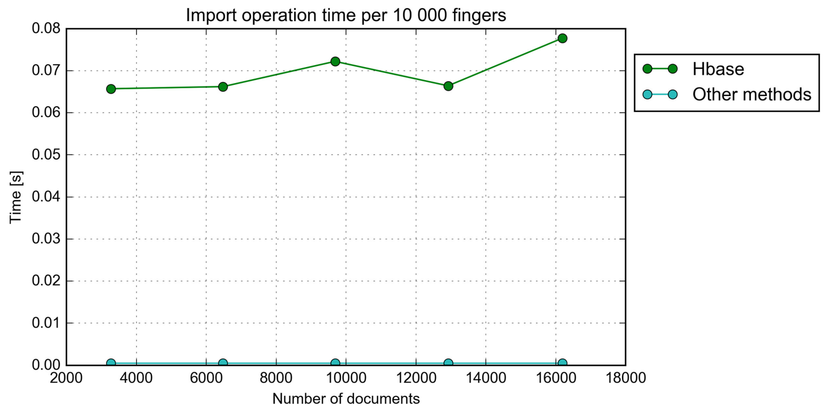

**Fig. 5.** Import operation time per 10 000 fingers

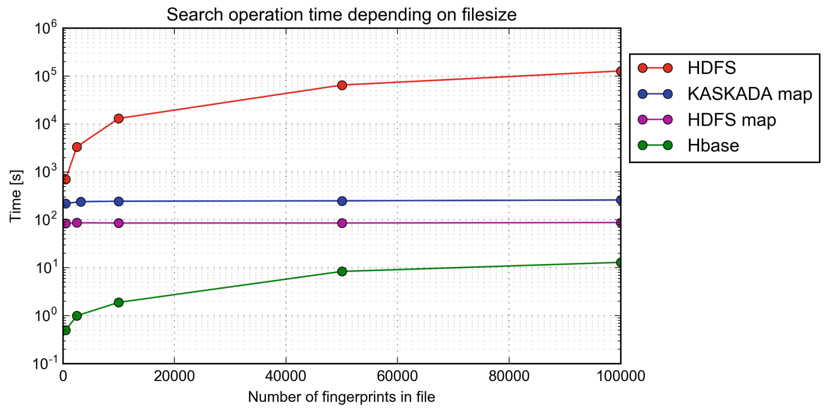

**Fig. 6.** Search operation time depending on file size

The fourth experiment we conducted was aimed to test the scalability of both platforms and the three fastest solutions, depending on the number of processors used in computations. The result of this test can be seen in Fig. 7. All three solutions scale fairly well in a close to linear manner.

The final experiment was run in order to test how both of the platforms handle many concurrent requests. All three platforms were limited to the same number of processors and memory. The result of this test can be seen in Fig. 8. The time of processing grows linearly with tasks load on the *KASKADA* platform. The *KASKADA* platform rejects new tasks, when there are no available resources – the number of rejected requests is represented as bars on the chart. The *Hadoop* ecosystem handles heavy load much better, no tasks were rejected and the processing times grow at a smaller pace (most likely, thanks to the advanced caching and load distribution algorithms of the platform). The sudden jump in processing time between 4 and 5 *HBase* approaches is caused by memory limitations of the system – the fifth task had to wait for free memory in order to start).

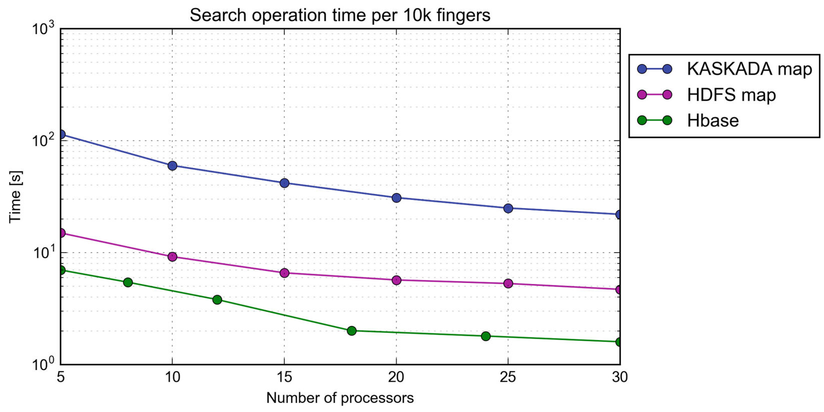

**Fig. 7.** Search time depending on the number of processors used

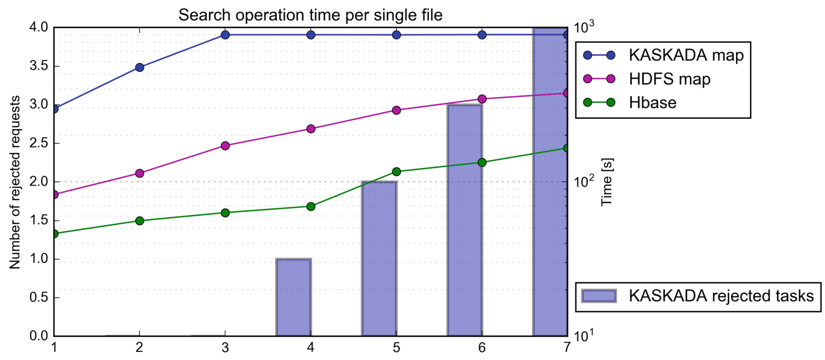

**Fig. 8.** Cumulative search time for $n$ parallel tasks

## 7 Conclusion

Conducted experiments confirm the benefits of migrating the selected algorithm to the *Hadoop* platform from the *KASKADA* platform. The naive, iterative method, using the distributed file system *HDFS* and the integrated computing cluster *YARN* did not prove to be as effective as the original algorithm on the *KASKADA* platform. However the improved version, using a in-memory map, was measured to be several times faster than the original *KASKADA* algorithm and just as scalable. The difference is most visible when using the *HBase* database to store and analyse the documents profiles. Searching for similar documents using *HBase* is around 25× faster than the search method running on the *KASKADA* platform. This is caused by the lack of a dedicated, distributed file system for the *KASKADA* platform (which is using a standard disk matrix), as well as by the better scaling of the *HBase* algorithm. Results of experiments depends on the size of files stored in the repositories. For the plagiarism detection system where the size of files is smaller than 1 MB the most suitable computing platform configuration is HBase using reversed index as described above.

In summary, our tests proved, that the *HBase* solution is much faster and scales much better, in relation to the profiles repository size. Apart from that, the results quality was exactly the same as for the original *KASKADA* algorithm. The *KASKADA* platform allows for fast and efficient analysis of multimedia data streams, exploiting the compute power of a supercomputer or a compute cluster. The *Hadoop* platform is much better suited to support an anti-plagiarism system, since it provides efficient tools for big-data processing.

## References

1. Fragidis, L.L., Chatzoglou, P.D., Aggelidis, V.P.: Integrated nationwide electronic health records system: semi-distributed architecture approach. Technol. Health Care **24**(6), 827–842 (2016)
2. Aletras, N., Tsarapatsanis, D., Preotiuc-Pietro, D., Lampos, V.: Predicting judicial decisions of the European court of human rights: a natural language processing perspective. PeerJ Comput. Sci. **2**, e93 (2016)
3. Hall, M.A., Wright, R.F.: Systematic content analysis of judicial opinions. Calif. Law Rev. **96**(1), 63–122 (2008)
4. Jurik, B.A., Blekinge, A.A., Ferneke-Nielsen, R.B., Moldrup-Dalum, P.: Bridging the gap between real world repositories and scalable preservation environments. Int. J. Digit. Libr. **16**(3–4), 267–282 (2015)
5. Beel, J., Gipp, B., Langer, S., Breitinger, C.: Research-paper recommender systems: a literature survey. Int. J. Digit. Libr. **17**(4), 305–338 (2016)
6. Tuarob, S., Bhatia, S., Mitra, P., Giles, C.L.: AlgorithmSeer: a system for extracting and searching for algorithms in scholarly big data. IEEE Trans. Big Data **2**(1), 3–17 (2016)
7. Kong, L., Zhao, Z., Lu, Z., Qi, H., Zhao, F.: A method of plagiarism source retrieval and text alignment based on relevance ranking model. Int. J. Database Theory Appl. **9**(12), 35–44 (2016)
8. Velasquez, J.D., Covacevich, Y., Molina, F., Marrese-Taylor, E., Rodriguez, C., Bravo-Marquez, F.: Docode 3.0 (document copy detector): a system for plagiarism detection by applying an information fusion process from multiple documental data sources. Inf. Fusion **27**, 64–75 (2016)
9. Buyya, R., Yeo, C.S., Venugopal, S.: Market-oriented cloud computing: vision, hype, and reality for delivering it services as computing utilities. In: 10th IEEE International Conference on High Performance Computing and Communications, 2008, HPCC 2008, pp. 5–13. IEEE (2008)
10. Krawczyk, H., Proficz, J.: KASKADA - multimedia processing platform architecture. In: Proceedings of the 2010 International Conference on Signal Processing and Multimedia Applications (SIGMAP), pp. 26–31, July 2010
11. White, T.: Hadoop: The Definitive Guide. O'Reilly Media Inc., Sebastopol (2012)
12. Kasai, T., Lee, G., Arimura, H., Arikawa, S., Park, K.: Linear-time longest-common-prefix computation in suffix arrays and its applications. In: Amir, A. (ed.) CPM 2001. LNCS, vol. 2089, pp. 181–192. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-48194-X_17
13. Hunt, J.W., MacIlroy, M.: An algorithm for differential file comparison. Citeseer (1976)

14. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Phys. Dokl. **10**(8), 707–710 (1966)
15. Winkler, W.E.: The state of record linkage and current research problems. In: Statistical Research Division, US Census Bureau. Citeseer (1999)
16. Baeza-Yates, R., Navarro, G.: A faster algorithm for approximate string matching. In: Hirschberg, D., Myers, G. (eds.) CPM 1996. LNCS, vol. 1075, pp. 1–23. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-61258-0_1
17. Cutting, D., Pedersen, J.: Optimization for dynamic inverted index maintenance. In: Proceedings of the 13th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 405–411. ACM (1989)
18. Anh, V.N., Moffat, A.: Inverted index compression using word-aligned binary codes. Inf. Retr. **8**(1), 151–166 (2005)
19. Yan, H., Ding, S., Suel, T.: Inverted index compression and query processing with optimized document ordering. In: Proceedings of the 18th International Conference on World Wide Web, pp. 401–410. ACM (2009)
20. Gabrilovich, E., Markovitch, S.: Computing semantic relatedness using wikipedia-based explicit semantic analysis. In: IJCAI, vol. 7, pp. 1606–1611 (2007)
21. Mcnamee, P., Mayfield, J.: Character n-gram tokenization for European language text retrieval. Inf. Retr. **7**(1–2), 73–97 (2004)
22. Mayfield, J., McNamee, P.: Single n-gram stemming. In: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 415–416. ACM (2003)
23. Ogawa, Y., Matsuda, T.: An efficient document retrieval method using n-gram indexing. Syst. Comput. Jpn. **33**(2), 54–63 (2002)
24. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. J. Am. Soc. Inf. Sci. **41**(6), 391 (1990)
25. Hofmann, T.: Probabilistic latent semantic indexing. In: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 50–57. ACM (1999)
26. Kanerva, P., Kristofersson, J., Holst, A.: Random indexing of text samples for latent semantic analysis. In: Proceedings of the 22nd Annual Conference of the Cognitive Science Society, vol. 1036. Citeseer (2000)
27. Lewis, D.D., Jones, K.S.: Natural language processing for information retrieval. Commun. ACM **39**(1), 92–101 (1996)
28. Strzalkowski, T.: Natural language information retrieval. Inf. Process. Manag. **31**(3), 397–417 (1995)
29. Schleimer, S., Wilkerson, D.S., Aiken, A.: Winnowing: local algorithms for document fingerprinting. In: Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, pp. 76–85. ACM (2003)
30. Heintze, N., et al.: Scalable document fingerprinting. In: 1996 USENIX Workshop on Electronic Commerce, vol. 3, no. 1 (1996)
31. Forman, G., Eshghi, K., Chiocchetti, S.: Finding similar files in large document repositories. In: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, pp. 394–400. ACM (2005)
32. Willett, P.: Document retrieval experiments using indexing vocabularies of varying size. II. Hashing, truncation, digram and trigram encoding of index terms. J. Doc. **35**(4), 296–305 (1979)

33. Dhillon, I.S., Fan, J., Guan, Y.: Efficient clustering of very large document collections. In: Grossman, R.L., Kamath, C., Kegelmeyer, P., Kumar, V., Namburu, R.R. (eds.) Data Mining for Scientific and Engineering Applications. MC, vol. 2, pp. 357–381. Springer, Boston (2001). https://doi.org/10.1007/978-1-4615-1733-7_20
34. Manber, U., et al.: Finding similar files in a large file system. In: USENIX Winter, vol. 94, pp. 1–10 (1994)
35. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: a distributed storage system for structured data. ACM Trans. Comput. Syst. **26**(2), 4:1–4:26 (2008). http://doi.acm.org/10.1145/1365815.1365816
36. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud 2010, p. 10. USENIX Association, Berkeley (2010). http://dl.acm.org/citation.cfm?id=1863103.1863113
37. Rabin, M.O., et al.: Fingerprinting by random polynomials. Center for Research in Computing Technology, Aiken Computation Laboratory, University (1981)
38. Muthitacharoen, A., Chen, B., Mazieres, D.: A low-bandwidth network file system. In: ACM SIGOPS Operating Systems Review, vol. 35, no. 5, pp. 174–187. ACM (2001)
39. Eshghi, K., Tang, H.K.: A framework for analyzing and improving content-based chunking algorithms. Hewlett-Packard Labs Technical Report TR, vol. 30, p. 2005 (2005)
40. Shvachko, K., Kuang, H., Radia, S., Chansler, R.: The hadoop distributed file system. In: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pp. 1–10, May 2010. https://doi.org/10.1109/MSST.2010.5496972. ISSN 2160-195X