Toward Full-text Searching Middleware over Hierarchical Documents

Kun Ma*, Bo Yang†, Ajith Abraham‡§
*Shandong Provincial Key Laboratory of Network Based Intelligent Computing
University of Jinan, Jinan, China
ise_mak@ujn.edu.cn
†Shandong Provincial Key Laboratory of Network Based Intelligent Computing
University of Jinan, Jinan, China
yangbo@ujn.edu.cn
‡Machine Intelligence Research Labs, Scientific Network for Innovation and Research Excellence, WA, USA
§IT4Innovations, VSB-Technical University of Ostrava, Czech Republic
ajith.abraham@ieee.org

***Abstract*—Currently, full-text searching can benefit from the emerging NoSQL databases and traditional indexing tools in the big data era. However, there are some drawbacks of current solutions. On one hand, the indexing documents lack of the hierarchy. On the other hand, big data have become the bottleneck of full-text searching. In the context of big data, we design a full-text searching middleware over hierarchical documents. We discuss the architecture of this middleware in detail. In addition, we propose a structure-independent hierarchical document model to present the hierarchical document. Moreover, the transformation engine is designed to translate the rich files into models. The core log event listener is responsible for capturing the changed documents and push them to the indexing storage at the same time. The experimental results show that our middleware is more advantageous than RDBMS with indexes and RDBMS with Lucene solutions.**

***Index Terms*—Full-text searching; middleware; hierarchical documents; NoSQL**

I. Introduction

Currently, NoSQL, also referred to as ”Not only SQL”, is finding significant and growing industry use in big data and real-time web applications, which emphasize that they do in fact allow relational query languages to be used [1]. Full-text search refers to techniques for searching a single formatted document or a collection in a full text database, which is distinguished from searches based on metadata or on parts of the original texts represented in databases [2]. However, the quantity of the hierarchical documents has become bottlenecks on the full-text searching in this context of big data [3]. For instance, how to full-text search millions of hierarchical publications with Digital Object Identifier (DOI) is challenging. To address this limitation, we design a full-text searching middleware over hierarchical documents in this paper.

The motivation of our paper is benefiting full-text searching from the integration of document stores and indexing tool. The contributions of this paper are several folds. First, we design a real-time full-text searching middleware for hierarchical documents. Second, we abstract a structure-independent document model to present the hierarchical document, and design a transformation engine from rich files to these models. Third, we integrate the indexing tool with document stores to support high-powered full-text searching. The log-based event listener intercepts the changes and synchronize them to the target indexing tool.

The rest of the paper is organized as follows. The related work is discussed in Section II. Section III abstracts the hierarchical document into structure-independent hierarchical document model. In Section IV, we design the architecture of full-text searching middleware. We introduce a transformation engine to translate the concrete hierarchical document to this model in detail. Moreover, we propose the integration solution of document stores and indexing tool to support full-text searching. Section V gives the experimental evaluation for our full-text searching middleware, and compares it with the existing approaches. Brief conclusions are outlined in the last section.

II. Related Research

Although the full-text searching of hierarchical documents is not a new problem, the traditional approaches become helpless once the quantity of the hierarchical documents is larger in the age of big data. Currently, there are several categories of strategies to full-text searching.

The first approach is direct full-text searching of rich text file [4]. In this solution, the files to be full-text searched are rich documents, such as PDF, Word, HTML. The search process is parsing this rich document to extract the text to make further index. The outstanding defect of this approach is the heterogeneity of hierarchical documents. As a improved version, the parser plugins are designed to expand the formats elastically, which detects and extracts metadata and structured text content from various documents using existing parser libraries.

The second approach is relational database management system (RDBMS) with indexes (abbreviated as RDBMS with indexes) [5]. Relational databases store data as rows and columns in tables. A table declaration specifies the columns, the type of data stored in each column, constraints over columns, as well as the optional indexes of columns. In this solution, the hierarchical document consists of processing those fields into terms that are added to the index. They are saved in relational tables. The disadvantage of this approach is the indexing speed limit.

The third alternative approach of full-text searching is specialized indexing tools integrated with RDBMS (abbreviated as RDBMS with Lucene), which is designed for rapid indexing of documents [6]. There are two classical methods. The first is trigger-based indexing. Generally, triggers solution listens the original RDBMS to store them in the indexing tools at the same time. As an improved form of this approach, some user-defined functions or hooks are developed for the RDBMS to capture the changes synchronously. However, a large amount of triggers will impact the database performance. The second is log-based indexing. The log event listener intercepts the changes of the original RDBMS to store them in the indexing tools. Generally, RDBMSs have native support for the binary logs. There are some deficiencies of this solution. First, relational schema can not reflect the hierarchy of documents. Second, RDBMS will have the I/O bottleneck issues encountered by handling huge amounts of data, especially for the query operation.

The fourth alternative approach of full-text searching is document store. A document-oriented database is a kind of NoSQL designed for storing, retrieving, and managing document-oriented information [7]. In contrast to well-known RDBMS, the document stores conform to the hierarchy of documents to be searched. Although document stores provide high performance read operations for frequently used queries, it has very limited functionality on the full-text searching of specific attributes.

III. Structure-independent Hierarchical Document Model

The hierarchical document is a document with different sections, just like an academic paper. In this paper, we only focus on this document to provide the full-text searching. Each hierarchical document is abstracted into a structure-independent hierarchical document model rather than the relational model. The structure of this model is shown in Figure 1 (a). The basic item of this structure-independent hierarchical document model is defined as the leaf node in Figure 1 (b), which is the content of the section of the hierarchical document. The non-leaf node in Figure 1 (b) is a section of the hierarchical document, which consists of several subsections and contents. The root is the accessible entrance of the hierarchical documents. This model reflects the hierarchical structure of the documents.

IV. Architecture

In this Section, we discuss the architecture of our proposed real-time full-text searching middleware over hierarchical documents.

A. Deployment Architecture

The typical architecture of full-text searching middleware over hierarchical documents is shown in Figure 2. The architecture is composed of four parts in order to reduce the complexity. From the left to the right, they are transformation engine from file to model, original document stores, log event listener and indexing storage respectively.

On the left is the transformation engine from files to structure-independent hierarchical document models, and original document stores. The inputs of this middleware are all kinds of files. The transformation engine is responsible to translate the rich files into a structure-independent hierarchical document models described in the last section. Since we design full-text searching middleware over hierarchical documents, we adopt MongoDB as the schema-free document stores. On the right is indexing storage. In our solution, we use Apache Lucene to store all the indexed data created based on the source NoSQL. The core of this architecture is a log event listener. First, this component captures the changes from the NoSQL database log. Second, it synchronizes the target indexing storage based on the increment. For the insert changes, we will create a new document index in the target indexing storage. For the update changes, we will obtain the corresponding document index and then update it at the same time. For the delete changes, we will delete the corresponding document index.

B. Transformation Engine from File to Model

The transformation engine is mainly in charge of converting the rich file to the structure-independent hierarchical document model. We design a content analysis toolkit to detect and extract the metadata and structured text content from various documents using existing parser libraries. There are literally thousands of different file formats in use, and most of those formats come in various different versions and dialects. We design different parser plugins to extract the unstructured hierarchical documents. For example, we use iText itextpdf [8] to parse PDF file, jsoup HTML Parser [9] to parse HTML file, and Apache POI [10] to extract the word files. we seek to build on the findings of the earlier work [11] [12] in this area to support transformation from file to model. The architecture of the transformation engine is shown in Figure 3.

The tentative plan for the transformation from a rich file to structure-independent hierarchical document model consists of the following steps: identification, parsing and extraction. First, we identify the file format based on the MINE header. Next, we parse the rich files using the configurable plugins. Finally, we change the extracted text into a hierarchical structure.

C. Log Event Listener

Our log-based event listener has been designed to intercept the changes from the source document stores. This flexibility is achieved by allowing different capturers to be developed and

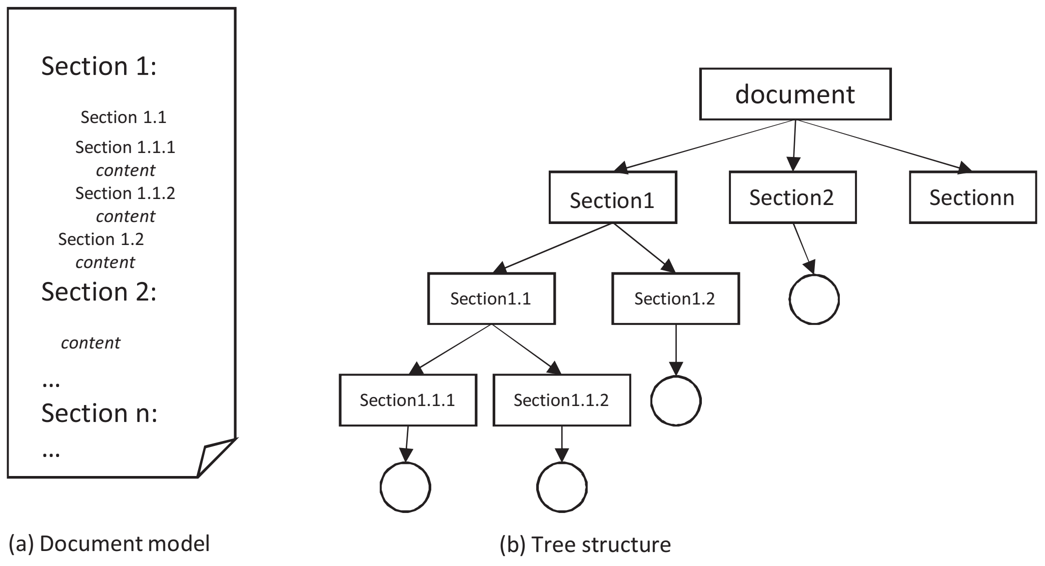
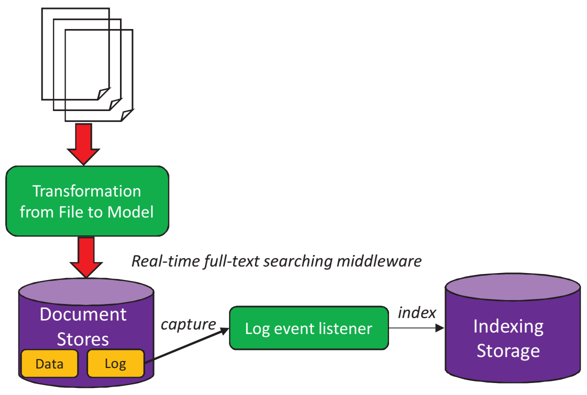


Fig. 1. Structure-independent hierarchical document model.



Transformation from File to Model
Real-time full-text searching middleware
Document Stores
Data
Log
capture
Log event listener
index
Indexing Storage

Fig. 2. Architecture of full-text searching middleware.

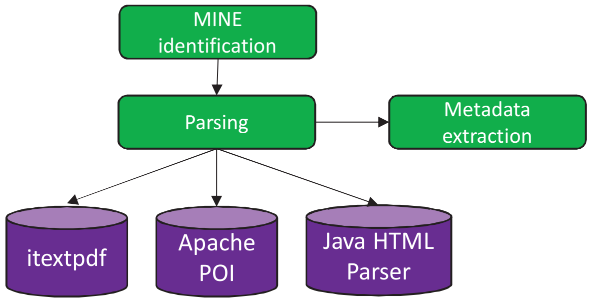


Fig. 3. Transformation from file to model.

plugged into the capture engine. For example, we use oplog to intercept the changed documents if we choose MongoDB as document stores. This capturer is typically an embedded thread and must follow a few semantic constraints.

In the context of change data capture, only the changed data reading from the operational log are concerned. Document store products often provide document-based replication (DBR) events to capture changed data from operational log. We can use this API to connect the listener to get the incremental data. By interpreting the contents of the database, operational log one can capture the changes made to the database in a non-intrusive manner.

Next, we will rectify the indexing storage according to the changed documents. Algorithm 1 presents this procedure. The function $getDocument$, $addDocument$, $updateDocument$, and $deleteDocument$ means obtaining, inserting, updating and deleting the index respectively.

D. Full-text Searching

After we have completed these above steps, all the indexing data have been stored in the indexed storage. If we need to do the full-text searching over hierarchical documents, what we need to do is to search the indexing storage.

V. Evaluation

We performed experiments on a two-core server (Core(R) CPU i5-2300 @2.80GHz $\times$ 2, 8G RAM, 120G SSD system disk, 2TB SATA-III disk, Gigabit Ethernet) running MongoDB, Apache Lucene and our full-text searching middleware over hierarchical documents. The systems were configured with a Windows Server 2012 x64. We have developed some test scripts to evaluate the performance of our proposed middleware. Besides, the performance of RDBMS with indexes, RDBMS with Lucene and document stores with Lucene are compared through experiment.

We have made an experiment to test the full-text searching time of RDBMS with indexes, RDBMS with Lucene, and document stores with Lucene solutions. We adopt MongoDB 2.4 as the source document stores, and Apache Lucene 4.4 as the indexing storage.

Algorithm 1 Log Event Listener *capture*

```
Input:
    Collection name $source$;
Output:
    indexing storage $indexing$;
1:  procedure capture($event$)
2:  switch ($source.event$) //log event
3:  case WriteDocumentsEvent:
4:      for each $document \in event.getRows()$ do
5:          for each $field \in document.getColumns()$ do
6:              function(doc) var ret=new Document();
                ret.addDocument(field); return ret;
7:          end for
8:      end for
9:  case UpdateDocumentsEvent:
10: for each $document \in event.getRows()$ do
11:     key=document.getKey();
12:     for each $field \in document.getColumns()$ do
13:         function(doc) var ret=Document.getDocument(key);
            ret.updateDocument(field); return ret;
14:     end for
15: end for
16: case DeleteDocumentsEvent:
17: for each $document \in event.getRows()$ do
18:     for each $field \in document.getColumns()$ do
19:         function(doc) var ret=Document.getDocument(key);
            ret.deleteDocument(field); return null;
20:     end for
21: end for
22: end switch
23: endprocedure
```

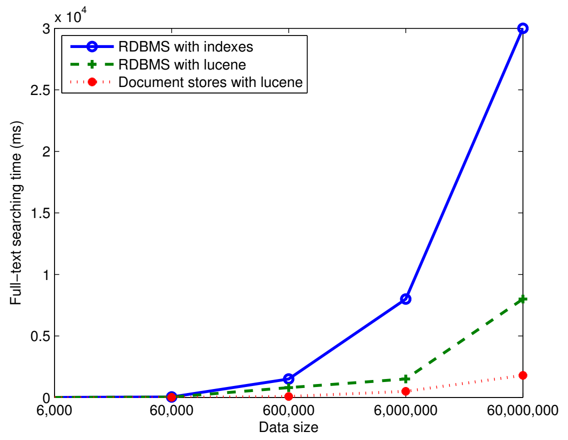


Fig. 4. Full-text searching time.

First, we initialize the document stores. We use our transformation engine to generate different hierarchical documents from the original PDF files, which are the academic papers. We initialize 6,000, 60,000, 600,000, 6,000,000 and 60,000,000 documents of the source NoSQL. Next, we test a simple full-text searching ”get all the papers that contain big data”. The full-text searching time is shown in Figure 4. We can see our document stores with Lucene solution is the most advantageous, since we split the long unstructured text into the fine-grained hierarchical documents. As the index will not significantly enhance the fuzzy full-text searching, the worst-case solution is RDBMS with indexes. The eclectic solution is RDBMS with Lucene. We can conclude that our full-text searching solution is suitable in the context of big data.

VI. Conclusions

As the emerging NoSQL databases, document-oriented databases provide high performance, high availability, and easy scalability of hierarchical documents. Compared with the current full-text searching methods, we design real-time full-text searching middleware over hierarchical documents. In order to design general full-text searching approach, we abstract the hierarchical document into structure-independent hierarchical document model. We also discuss the architecture and the important parts of this middleware. Experiment shows the superiority of our proposed middleware.

Acknowledgment

This work was supported by the Doctoral Fund of University of Jinan (XBS1237). Ajith Abraham was supported in the framework of the IT4 Innovations Centre of Excellence project, reg. no. CZ.1.05/1.1.00/02.0070 by operational programme Research and Development for Innovations funded by the Structural Funds of the European Union and state budget of the Czech Republic.

References

[1] R. Cattell, “Scalable sql and nosql data stores,” *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2010.

[2] J. Suchal and P. Nvrat, “Full text search engine as scalable k-nearest neighbor recommendation system,” in *Proceedings of Third IFIP TC 12 International Conference on Artificial Intelligence*, 2010, pp. 165–173.

[3] O. Trelles, P. Prins, M. Snir, and R. C. Jansen, “Big data, but are we ready?” *Nature Reviews Genetics*, vol. 12, no. 3, pp. 224–224, 2011.

[4] K. Anuradha, R. Sivakaminathan, and P. A. Kumar, “Open-source tools for enhancing full-text searching of opacs: Use of koha, greenstone and fedora,” *Program: electronic library and information systems*, vol. 45, no. 2, pp. 231–239, 2011.

[5] Z. H. Liu, T. Baby, S. Chakraborty, J. Ding, A. Novoselsky, and V. Arora, “Pay-as-you-go: an adaptive approach to provide full context-aware text search over document content,” in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010, pp. 1025–1036.

[6] C. Zhang and S. Zhan, “Research and implementation of full-text retrieval system using compass based on lucene,” in *Proceedings of the 2012 International Conference on Communication, Electronics and Automation Engineering*, 2013, pp. 349–356.

[7] E. Dede, M. Govindaraju, D. Gunter, R. S. Canon, and L. Ramakrishnan, “Performance evaluation of a mongodb and hadoop platform for scientific data analysis,” in *Proceedings of the 4th ACM workshop on Scientific cloud computing*, 2013, pp. 13–20.

[8] B. Lowagie, *iText in Action*, second edition ed. Manning Publications, 2010.

[9] J. Hedley, “jsoup: Java html parser,” 2013, available from http://jsoup.org/. [Online]. Available: http://jsoup.org/

[10] G. S. Ingersoll, T. S. Morton, and A. L. Farris, *Taming Text: How to Find, Organize, and Manipulate It*. Manning Publications, 2013.

[11] K. Ma, B. Yang, Z. Chen, and A. Abraham, “A relational approach to model transformation with qvt relations supporting model synchronization,” *Journal of Universal Computer Science*, vol. 17, no. 13, pp. 1863–1883, 2011.

[12] K. Ma, B. Yang, and A. Abraham, “A template-based model transformation approach for deriving multi-tenant saas applications,” *Acta Polytechnica Hungarica*, vol. 9, no. 2, pp. 25–41, 2012.