A Document-Based Data Warehousing Approach for Large Scale Data Mining

Hualei Chai, Gang Wu, and Yuan Zhao

School of Software,
Shanghai Jiao Tong University, Shanghai, China
{joestone_chai,zhaoyuan}@sjtu.edu.cn
wugang@cs.sjtu.edu.cn

Abstract. Data mining techniques are widely applied and data warehousing is relatively important in this process. Both scalability and efficiency have always been the key issues in data warehousing. Due to the explosive growth of data, data warehousing today is facing tough challenges in these issues and traditional method encounters its bottleneck. In this paper, we present a document-based data warehousing approach. In our approach, the ETL process is carried out through MapReduce framework and the data warehouse is constructed on a distributed, document-oriented database. A case study is given to demonstrate details of the entire process. Comparing with RDBMS based data warehousing, our approach illustrates better scalability, flexibility and efficiency.

Keywords: Data Warehousing, Document-based, Big Data, MapReduce.

1 Introduction

Data mining has always been a hotspot issue in computer science. Having the rapid development in IT industry in the past two decades, data mining techniques are now widely applied in almost every aspect of our economic and social life and have enjoyed an explosive growth. From scientific research to Business Intelligence systems, data mining is an irreplaceable part of the Knowledge Discovery process.

Data warehouse is the basis of data mining, and is playing an important role in modern IT industry and has evolved into an unique and popular business application class. Early builders of data warehouses already consider those systems to be key components of their data mining and even decision-support system architecture [1].

Since the emergence of cloud computing, the demand of mining massive data has become relatively urgent. In order to obtain valuable information hidden in the ocean of data, data mining techniques are being applied to almost every aspect of modern society. The ability of handling big data is so crucial that no one could neglect. However, big data poses more challenges, among which, both scalability and efficiency have always been the most important issues and have

Q. Zu, B. Hu, and A. Elçi (Eds.): ICPCA-SWS 2012, LNCS 7719, pp. 69–81, 2013.

attracted tremendous interests. In recent years, large-scale data mining has been extensively investigated [2], and many approaches have been proposed [3][4][5][6].

Traditional data warehousing is more like constructing a larger database, and follows certain steps including requirements analysis, data modeling, normalization or denormalization and so on [7]. However, as a RDBMS based data warehouse, it is facing tougher challenges today. First of all, the problem comes from scalability. The collected data is getting bigger and bigger, and even beyond the tolerance of a traditional DBMS. Second, dealing with heterogeneous data sources is complex. The schema of data sources could change more frequently than before as the real business keep changing. The corresponding modification job is a great burden, as redesign and reconstruction of the data warehouse may take place from time to time. This could be tremendous time and money consuming. Third, efficiency has become the bottleneck of RDBMS based data warehouse. Data mining usually calls for millions of aggregation operations and mathematical computations. Those operations are very likely to be inefficient and complex because all data is organized as linked tables in RDBMS, thus each query leads to several operations like projection and OUTER-JOIN. As a result, the operating efficiency is sacrificed.

This paper focuses on presenting a better solution of data warehousing in the big data era. We explore a document-based data warehousing approach. This approach includes three phases, namely documentization phase, aggregation phase and data loading phase. In the documentization phase, we extract all the data from respective heterogeneous data sources, and write them to basic text files. In the aggregation phase, a MapReduce process is applied to accomplish ETL of data from multiple sources, and transform all the results into JSON objects [8]. The main difference between our approach and the traditional ones is that after the aggregation phase we keep all the data as recursive key/value pairs, getting rid of their original schema which includes the table format and foreign keys. The data loading phase is responsible for putting all the output JSON objects into a document-oriented data warehouse. In order to demonstrate details of the entire process, a case study is given in Section 4 and the result of our experiments shows that our data warehouse is more scalable, efficient and flexible.

The rest of this paper is organized as follows. Section 2 presents the background and related work. The Third section gives our solution and the fourth section is a real case study in order to show details of the whole process. Evaluations and discussions are also given in Section 4. Section 5 presents our future plans and conclusion of this paper.

2 Background

2.1 Difference between Database and Data Warehouse

Data warehousing is an essential element of data mining or decision-support system and has always been a focus in the database industry. Data warehouse is a "subject-oriented, integrated, time-varying, non-volatile collection of data

that is used primarily in organizational decision making" [9]. The main difference between a data warehouse and a database is that the former one is targeting at collecting useful data from heterogeneous sources, and seeking information and patterns hidden behind the data. Whereas the later one is used to record data generated during a transaction process. Therefore the ability of handling a huge amount of data is more critical for a data warehouse and database emphasis more on the atomic operation, which refers to accuracy and consistency in a single operation.

Figure 1 is used to demonstrate different requirements for a data warehouse and an ordinary database application. Data warehouse is a basic support for data mining, whereas transaction process, for example, OLTP, is a typical application of a database. In general, data mining is usually read-intensive and calls for complex mathematical operations, but transaction processes are write-intensive with simple operations. Therefore operating efficiency, including throughput and response time, is more important for data warehouses and consistence and accuracy is prior to an ordinary database.

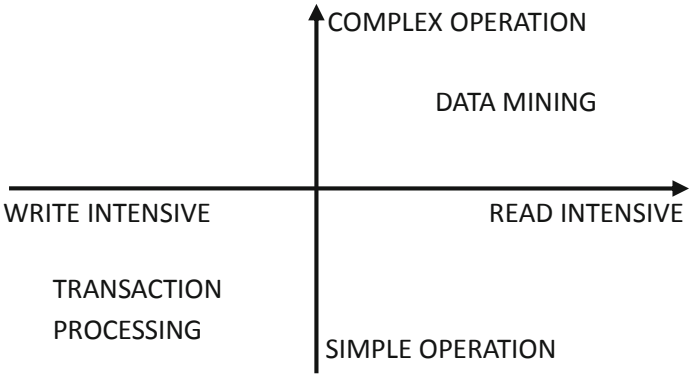


Fig. 1. Different Database Application Characteristics

2.2 Challenges in Data Warehouse

Considering both the different characteristics between data warehouses and databases, and the demand of big data, we conclude the following three features for a data warehouse as the most critical requirements.

Scalability

The data warehouse should have full support for dynamic scaling. When dealing with big data, any database server would have the risk of running out of storage and there is not a "large-enough-ever" storage. In the coming cloud era, dynamic scaling is definitely the best solution.

Efficiency

This word here refers to two aspects, the efficiency of construction and maintenance of the data warehouse, and operating efficiency of the data warehouse

responds to each query. Data mining usually studies large data set from multiple sources. How convenient is the data immigration? How fast does the data warehouse respond for millions of queries from data mining engine? Efficiency is always of great concern.

Heterogeneity

In a real data mining application, data sources are usually heterogeneous. Data sources may be RDBMS with different schema, or even different types of data set, like XML files, logs or other NoSQL databases. Furthermore, changes in data sources may take place at any time, including introducing new data sources, or adjustment of schema due to commercial reasons. If the data warehouse is not flexible enough to deal with heterogeneous sources, it will lead to reconstruction of the data warehouse, which is really both money and time consuming. The only solution is to put heterogeneity into consideration in designing your data warehouse.

2.3 Related Work

Traditional data warehouse is constructed based on RDBMS, following certain steps includes defining architecture of storage, integrating servers, designing warehouse schema and views, implementing data extraction, cleansing, transformation, loading and so on [10].

Even though Entity Relationship diagrams and other RDBMS techniques are popularly used for database design in OLAP, the database designs recommended by ER diagrams are inappropriate for decision support systems [10]. On one hand, heterogeneous data sources are difficult to be integrated in one schema, which is flexible enough to face possible adjustments of sources in the long run. On the other hand, the RDBMS based data warehouse can not transcend limitations of dynamic scaling due to the fixed schema.

The problem is so important that there have been lots of engineers working on it and they have proposed lots of solutions. Yang Lai and Shi Zhongzhi proposed an indexing method, aiming at more efficient data accessing for large scale data mining [3]. Jane Zhao, proposed an optimization which adds a OLAP service layer in between the data warehouse and data mining application [4]. Vuda Sreenlvase Rao focus on distributed storage and has made some improvements on the hardware [5]. Jin Han, et. al. believed that applying a shared-memory in order to store temporary data, just like the idea of MemCache, would greatly improve accessing efficiency [6].

These works have made some progress, yet they are still facing challenges. On one hand, scalability remains a problem. When collected data gets larger than the origin infrastructure thus corresponding hardware should be extended, but that is not easy for these added in shared-memory [6] or OLAP query layer [4]. On the other hand, heterogeneity is making the problem harder. Both the schema design and indexing system [3] lead to great modification job, because neither has a support of incremental modification. And real business world changes rapidly, any fixed schemas or indexes have problems adjust to any structural updates.

3 Approach

This section gives a document-based data warehousing approach to tackle challenges in big data era. The entire process consists of three phases, namely documentization phase, aggregation phase and data loading phase, and is demonstrated in Fig.2.

3.1 Glossary

MapReduce is a programming model and associated implementation for processing and generating large data sets [11]. It is proposed by Google and used for handling large data analyze tasks. MapReduce consists of two functions: Map and Reduce. The Map function takes input key/value pairs and produces intermediate data, also key/value pairs. The Reduce function collects the intermediate data and produces the final output. This framework is a high efficient distributed programming model and has been validated in recent years.

DFS (Distributed file system) is a file system that allows access to files from multiple hosts via a shared computer network. This makes it possible for multiple users on multiple machines to share files and storage resources. There are various implementations, such as Google File System [12], BigTable [13] and Hadoop's HDFS [14].

JSON (JavaScript Object Notation) is a lightweight data-interchange format [8]. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a completely language-independent text format, and is built on a collection of name/value pairs and an ordered list of values.

3.2 Documentization Phase

The first challenge to tackle is heterogeneous data sources, varying from traditional RDBMS, to XML files, even log files and so on. Documentization refers to extract data from original sources and transform them into independent documents. In this phase, we export all the data from each original sources into a set of typical text files, and an extra document indicating corresponding structure or format of each text file. This is a very important step which guarantees the flexibility of our approach. After this phase, the overall system shall deals with documents only, as shown in Fig.2.

3.3 Aggregation Phase

The core idea of our data warehousing approach is carried out in this phase. In the big data era, thanks to distributed file systems, storage capacity is no

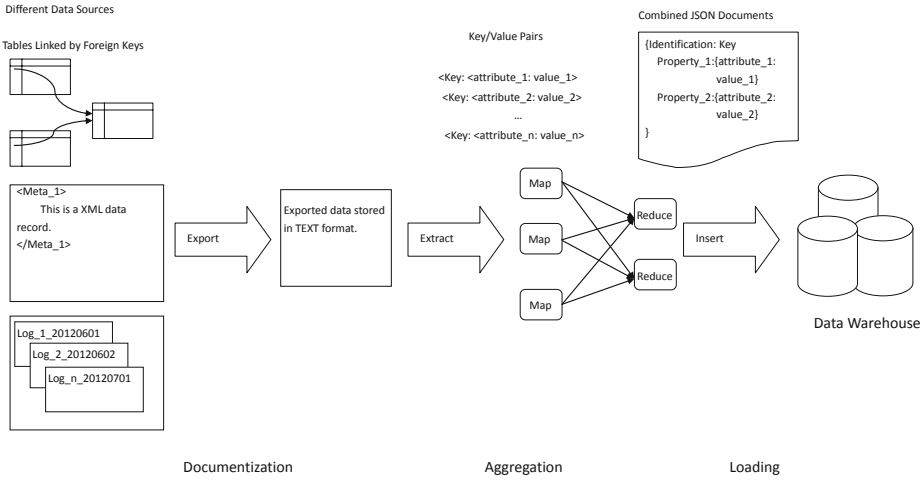


Fig. 2. Document-Based Data Warehousing Process

longer a tough constraint. And the aggregation phase is aimed at promoting query efficiency of the overall system.

Before getting started, all data exported from original databases are uploaded on DFS. And afterwards, a MapRedeuce process is applied. The Map function is applied to read through the data set by line, and transform each line into recursive key/value pairs, having each attribute name as keys and its correspond data as values, thus the intermediate output is produced. The Reduce function is used to collect all the lines by separate identification key, as shown in Fig.2. Lines from different table, which were linked by foreign keys in its previous sources, are now gathered into one JSON object, and written to a document file on DFS. Since data originally from different tables are now gathered via each primary key, it is also convenient to kick out inconsistent data, wrong type data and many other types of dirty data. The algorithm is given in Algorithm 1.

3.4 Data Loading Phase

For load balance and query efficiency considerations, we gather all the JSON objects produced in last phase into multiple documents. In this phase, we read all these files and insert the JSON objects into a document-oriented database. Since all data are stored as JSON objects, which has a nature support for Object-Oriented programming languages, such as JAVA and C++. All need to do in future data mining process is to implement analyze algorithms in any Object-Oriented programming languages without considering any SQL queries. This is both agile and high efficient.

Algorithm 1: MapReduce Data Preparation

```
   Input: FileNames, OutputDirectory
   Output: OutputFile
 1 Procedure Mapper(key=Line Number, value=Line String)
 2 begin
 3     foreach attribute_item in each Line do
 4         if attribute_item is not primary key then
 5             attribute_item ← {"attibute_name:", "attribute_item"}
 6         end
 7         transform Line String into JSON Object
 8         output(key=collect_key, value=Line String in JSON format)
 9     end
10 end
11 Procedure Reducer(key=collect_key, value=Line String in JSON format)
12 begin
13     collect all Line String with same collect_key
14     make new JSON Object: {"collect_key: property"}
15     foreach Line String do
16         add Line String to {"collect_key": {Line String}}
17     end
18     output(key=NULL, value= {"collect_key": {Line String},...,{Line String}})
19 end
```

4 Case Study

In this section, we use a real case in order to show the details of our approach and validate it.

4.1 Data Set

The dataset we use is released in KDD Cup 2012, provided by Tencent, one of the largest micro-blog websites in China [15]. The entire data set is over 10 Gega Byte, consists of 13 entity-relationship tables. The dataset is a subset of basic customer information of Tencent Weibo users, including personal information, such as tweeting activity, comments and each person's Follower and Followee list, and advertisement information. The basic logic view of the dataset is demonstrated in the Fig.3.

In order to estimate a big data input, we replicate this data set 100 times to build a large data collection, which is over 1 TB. We use this data set for two reasons. First, social networks have become tremendously popular in recent years. Popular social network websites like Facebook, Twitter, and Tencent Weibo are adding thousands of enthusiastic new users each day to their existing billions of actively engaged users. Currently, there are more than 200 million registered users on Tencent Weibo, generating over 40 million messages each

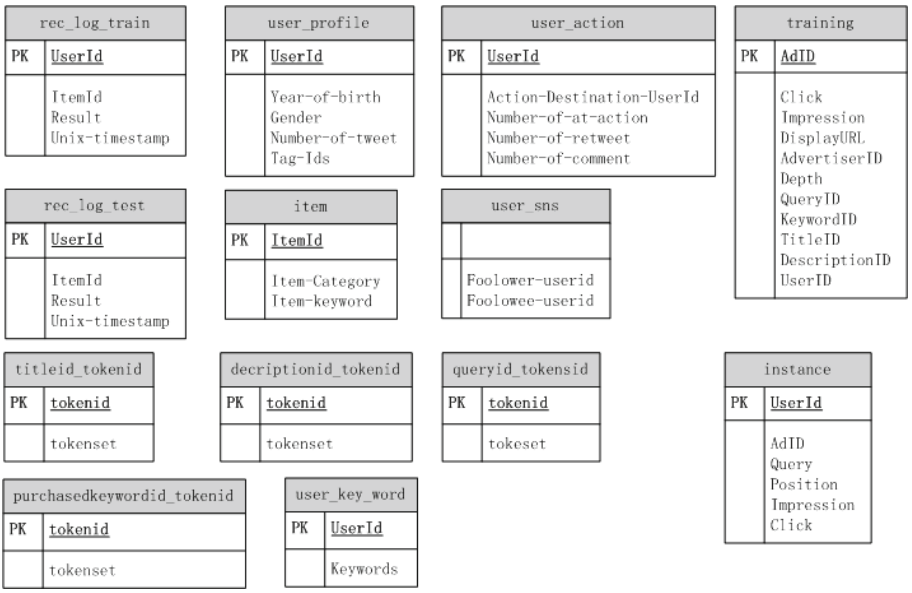


Fig. 3. OR mode of original dataset

day [15]. Valuable information interfering with almost every aspect of social life, including economic, social and political issues, state and regional security issues, scientific and technological researches and so on. As a result, there is an urgent demand of studying this kind of data. Second, the dataset also shares the features of Web 2.0 service, which is a new trend in IT industry and needs to be further investigated. One of the most unique characters of Web 2.0 is that the database stores customer-provide data, therefore the database must be heterogeneous-tolerant and robust enough in order to face customers' different requirements. Under this scenario, our approach is given full play.

4.2 System Environment

The experiment is carried out on a distributed Cluster consists of seven machines. Environmental Parameters of each node is shown in Table 1.

Hadoop is an opensource implementation of MapReduce by Apache Foundation [14]. The Apache Hadoop project consists of many subprojects, among which HDFS(Hadoop DFS) and MapReduce are the most commonly used.

MongoDB is a C++ implemented, opensource document oriented database, developed by 10gen [16]. It is a BSON(Binary JSON) based high performance database, features its schema-free structure, well support for dynamic scaling, powerful Javascript-like query language and high accessing speed.

Table 1. System Environment Parameter

Environment	Parameter
CPU	Intel(R)Xeon(R)CPU E5405@2.00GHZ (2 cores)
Memory	8 GB RAM
Hard Disk	1 TB
Operating System	Ubuntu 12.04 64-bit
JAVA Runtime Environment	JDK 1.7
Hadoop Version	Apache Hadoop 0.20.2
Database	MongoDB 2.0.6 (Linux 64-bit)

4.3 Implementation

The structure of the overall system is shown in Fig.4. There are three parts, data source, MapReduce Cluster and MongoDB Cluster.

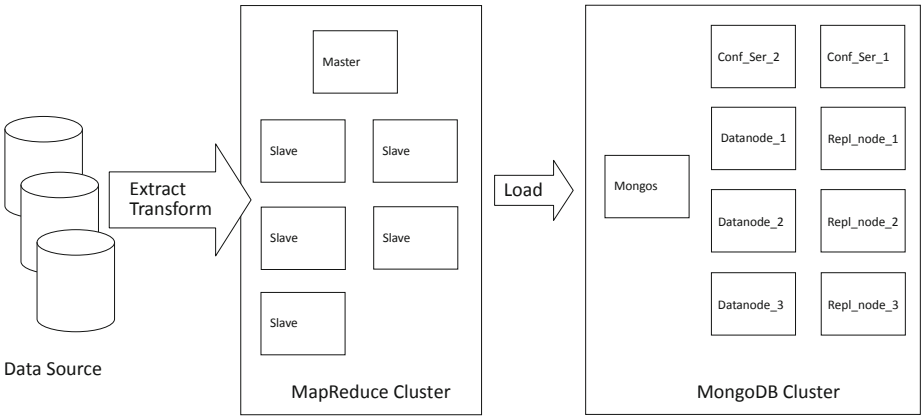


Fig. 4. System Structure of Implementation

In our case, the data source consists of 100 subsets, each of which is a document collection exported from Tencent Weibo's operating database. As for the MapReduce cluster, we built a 6-node Hadoop cluster. One acts as the master and the other five as slaves. Slave node can be dynamically dropped out or added in, in order to validate the system's scalability. The entire experiment is carried out on this cluster separately with 3 nodes and 6 nodes. The last part of Fig.4 is MongoDB Cluster, based on which we constructed our data warehouse. The logical view of the data warehouse's structure is demonstrated in this figure.

- *Mongos* is the accessing node of the cluster.
- *Conf_ser_i* is the ith Configuration Server, maintaining name space, data servers' information and index.

- *Datanode_i* is the ith Data Server.
- *Repl_node_i* is the replica of the ith Data Server.

We choose MongoDB for two reasons. First, it has a very agile support of dynamic scaling. If the data warehouse run out of storage, it is very easy to add in a new node into the distributed storage environment. Second, MongoDB has a special key/value storage mechanism which has a nature support of MapReduce framework. Since the data extraction and loading job is carried out in MapReduce framework, the data loading phase doesn't need to wait till the aggregation phase is totally finished. Therefore, much time and memory is saved.

4.4 Experiment Result and Discussion

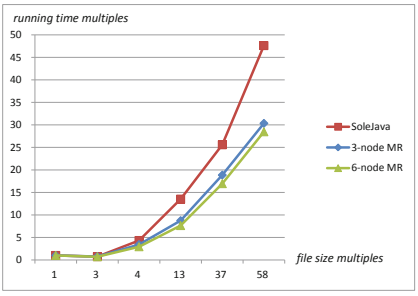
Scalability

Definition 1. *Scale Sensitivity (ScaS) measures a system's sensitivity facing changes of workload. ScaS equals to the average value of the multiples of processing time changes divided by the corresponding growth of input data size in multiples. A system with a lower ScaS is less sensitive to workload change, therefore this system features better scalability.*

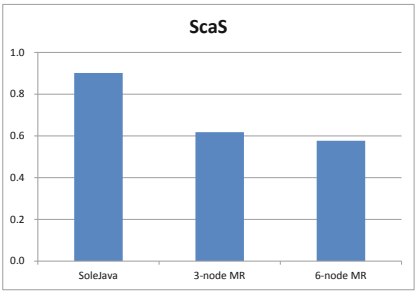
$$ScaS = Ave(\frac{T/t}{F/f}) \tag{1}$$

where F *is the size of input file and* f *is the size of standard input file, and we use the average size of input subsets as* f *in this case.* T *and* t *are the corresponding running time of input* F *and* f *respectively.*

In our experiment, we implement the entire process as mentioned in Fig.2, using our MapReduce approach, marked as MR in Fig.5. As for comparison, we also implement the same function as a none-MapReduce Java program, which is named SoleJava here. SoleJava runs on single node, while MR is implemented respectively on a 3-node and 6-node cluster. We use ScaS to compare the scale sensitivity of each system.



(a) Sensitivity Curve



(b) System ScaS

Fig. 5. System Scalability

In Fig.5(a), the horizontal axis marks multiples of input data size of *f*, and the vertical axis records the corresponding running time multiples. As input data set gets larger, the running time multiples of both methods increase. However, the Curve of MR grows always slower than SoleJava. And Fig.5(b) demonstrates the ScaS of SoleJava, 3-node MR and 6-node MR. Facing increasing workload, running time of SoleJava has almost a linear growth of almost the same speed, while ScaS of MR is about two thirds of that of SoleJava. Therefore, facing growing workload, data warehouse built through our approach is less sensitive. That is to say, our approach is more scalable.

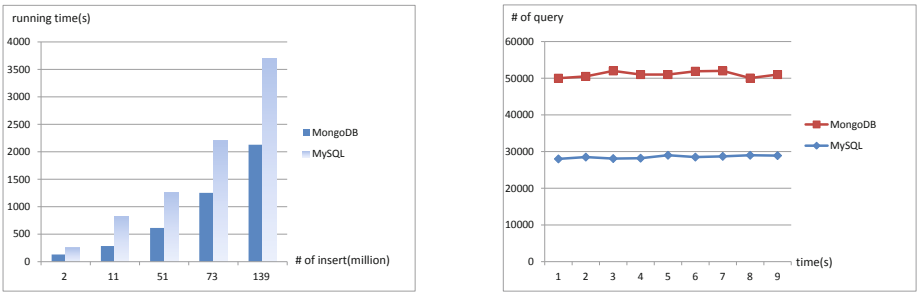
Efficiency

We evaluate overall system's efficiency from two perspectives, the efficiency of constructing and operating the data warehouse. As for constructing the data warehouse, Table 2 shows a comparison between traditional approach and our document-based approach. It is straightforward that our approach requires fewer work to do and the process is very agile.

Table 2. Comparison of Data Warehousing Approaches

Phases	Rational DW	Document-based DW
Preparation	Requirement Analysis	Requirement Analysis
	Data Modeling	Data Modeling
Design	Design Table Format	
	Choose Foreign Keys	
Construction	Data Extraction	Documentization
	Data Cleansing	
	Transformation	Aggregation
	Loading	Loading
Operating	SQL query	Object Operation
	OUTTER-JOIN, Projection	
Modification	Change Schema	(Not Affected)
	Possible Reconstruction	

As for operating efficiency, we use an experiment to estimate accessing the data warehouse and validate its performance. Figure 6 shows the efficiency comparison between the MongoDB based data warehouse and a MySQL Cluster based data warehouse, both are build on the same physical machines. A write experiment and a query experiment are carried on both data warehouses. The pillar graph (a) shows the comparison of data insertion performance and the graph (b) shows the QPS (query per second) curve of both system. As demonstrated in Fig.6, the average processing capacity of MongoDB based data warehouse is 50,000 requests per second while that of MySQL based data warehouse is less than 30,000. Therefore our data warehouse is more efficient.



(a) System Respond Time (b) QPS Curve

Fig. 6. Efficiency Comparison

Heterogeneity

Heterogeneity, as mentioned in Section 2, refers to the overall system's flexibility facing both heterogeneous structure and changes of data source. In our experiment, the format of data source is given to our program as an input configuration document, which is read automatically. The unique key/value storage format and schema-free designment the flexibility of our data warehouse. Our data warehouse has no dependence on the schema of data source. When input file is changed, very few lines of code are changed. Also we find this approach to be convenient enough facing similar kinds of data mining requirements.

5 Conclusion

This paper presents a better solution for data warehousing in the big data era. Comparing with traditional RDBMS-based data warehousing, our approach shows better performance in scalability, efficiency and heterogeneity. The approach consists of three phases, namely documentization, aggregation and data loading. Our data warehouse is constructed on a distributed environment and the MapReduce framework is applied for efficiency consideration. Even though it is agreed to all that there is not, and will never be, a "one-fits-for-all" solution, our approach definitely boosts its unique characteristic.

In future, we will introduce more data mining applications based on this data warehouse structure under the similar scenario. More data mining algorithms will be implemented on this platform. We will also work on more friendly documentization tools for different data sources.

References

1. Gupta, V.R.: An Introduction to Data Warehousing. System Services Corporation (1997)
2. Tan, A.X., et al.: A Comparison of Approaches for Large-Scale Data Mining. Technical Report UTDCS-24-10 (2010)
3. Yang, L., Shi, Z.: An Efficient Data Mining Framework on Hadoop using Java Persistentce API. In: 10th IEEE International Conference on Computer and Information Technology (2010)
4. Zhao, J.: Designing Distributed Data Warehouses and OLAP Systems. In: ISTA 2005, pp. 254–263 (2005)
5. Sreenivasa Rao, V., Vidyavathi, S.: Distributed Data Mining And Mining Multi-agent Data. International Journal on Computer Science and Engineering (IJCSE) 02(04), 1237–1244 (2010)
6. Han, J., et al.: A Novel Solution of Distributed Memory NoSQL database for Cloud Computing. In: 2011 10th IEEE/ACIS International Conference on Computer and Information Science (2011), 978-0-7695-4401-4/11$26.00
7. Sen, A., Sinha, A.P.: A comparison of data warehousing methodologies. Communications of The ACM 48(3) (2005)
8. JSON, `http://www.json.org/`
9. Inmon, W.H.: Building the Data Warehouse. John Wiley (1992)
10. Chaudhuri, S., Dayal, U.: An Overview of Data Warehousing and OLAP Technology. ACM Sigmod Record (1997)
11. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: OSDI (2004)
12. Ghemawat, S., et al.: The Google File System. In: SOSP 2003. ACM (2003)
13. Chang, F., et al.: BigTable: A Distributed Storage System for Structured Data. In: OSDI (2006)
14. Apache Hadoop, `http://hadoop.apache.org/`
15. KDD Cup 2012, `http://www.kddcup2012.org/`
16. MongoDB, `http://www.mongodb.org/`