A Cloud Computing Implementation of XML Indexing Method Using Hadoop*

Wen-Chiao Hsu[1], I-En Liao[2,**], and Hsiao-Chen Shih[3]

[1,2,3] Department of Computer Science and Engineering
National Chung-Hsing University,
250 Kuo Kuang Road, Taichung 402, Taiwan
phd9510@cs.nchu.edu.tw, ieliao@nchu.edu.tw, nic3p1217@gmail.com

Abstract. With the increasing of data at an incredible rate, the development of cloud computing technologies is of critical importance to the advances of researches. The Apache Hadoop has become a widely used open source cloud computing framework that provides a distributed file system for large scale data processing. In this paper, we present a cloud computing implementation of an XML indexing method called NCIM (Node Clustering Indexing Method), which was developed by our research team, for indexing and querying a large number of big XML documents using MapReduce. The experimental results show that NCIM is suitable for cloud computing environment. The throughput of 1200 queries per second for huge amount of queries using a 15-node cluster signifies the potential applications of NCIM to the fast query processing of enormous Internet documents.

Keywords: Hadoop, Cloud Computing, XML Indexing, XML query, Node Clustering Indexing Method.

1 Introduction

XML (eXtensible Markup Language) is widely used as the markup language for the web documents. The flexible nature of XML enables it to represent many kinds of data. However, the representation of XML is not efficient in terms of query processing. A number of indexing approaches for XML documents are proposed to accelerate query processing. Most of these works provide mechanisms to construct indexes and methods for query evaluation that deal with one or small amount of documents in a centralized fashion. In the real world, an XML database may contain a large number of XML documents which require the existing XML indexing methods to be scalable for high performance.

The concept of the "cloud computing" has been received considerable attention because it provides a solution to the increasing data demands and offers a shared,

* This research was partially supported by National Science Council, Taiwan, under contract no. NSC100-2221-E-005-070.

** Corresponding author.

J.-S. Pan, S.-M. Chen, N.T. Nguyen (Eds.): ACIIDS 2012, Part III, LNAI 7198, pp. 256–265, 2012.

distributed computing infrastructure [2]. With the increasing popularity of cloud computing, Apache Hadoop has become a widely used open source cloud computing framework that provides a distributed file system for large scale data processing. When a low-cost, powerful, and easily accessible parallel computational platform is available, it is important to better understand how it can solve a given problem [3].

Although there are many published papers on the subject of XML indexing and querying methods, most of them are confined to small data samples running in the centralized system. As cloud computing becomes popular, the issues of parallel XML parsing have been discussed recently. However, to the best of our knowledge, there is very little work that addresses the problem of indexing as well as querying XML documents on large distributed environments. Exploring whether the existing XML indexing methods can be scaled out is an important issue due to the enormous XML documents in the Web.

In our previous work [1], we presented an indexing method called NCIM (Node Clustering Indexing Method) which compresses XML documents effectively and supports complex queries efficiently. In this paper, we use Hadoop framework to present a mechanism for distributed construction and storage of indexes as well as distributed query processing for a large number of big XML documents on the basis of NCIM.

The contributions of our work are as follows. We modify the NCIM (Node Clustering Indexing Method) and design a system for indexing and querying a large number of XML documents by using the Hadoop cloud computing framework. We also consider two job processing modes, streaming query vs. batched query, for query evaluation in our experiments. The results show that the batched query processing will have much better throughput.

The rest of this paper is organized as follows. In the next section, we review related work. Section 3 describes preliminaries on Hadoop. Section 4 presents the proposed system that builds indexes for XML datasets and answers massive queries simultaneously. Experimental results are discussed in Section 5. Finally, Section 6 concludes the paper.

2 Related Work

Many index methods and query evaluation algorithms have been proposed in the literature. The most widely used approaches are structural summary and structural join. The structural summary indexing methods merge the same sub-structures in an XML document and form a smaller tree structure, which is used as the index of the XML document. Thus, instead of matching an input query against the XML document itself, the summarized index tree is used. The DataGuide [4] is a typical model. A strong DataGuide holds all the P-C (Parent-Child) edges in an XML file. Each node in a DataGuide has an extent for the corresponding nodes in the original XML document. Therefore, the P-C (Parent-Child) and A-D (Ancestor-Descendant) relationships can be evaluated using strong DataGuide directly. However, DataGuide is not feasible for twig queries, since the structure of the summarized index is not the same as the original XML document.

Structural Join [5] is one of the first proposed methods to process twig pattern matching. A twig query is decomposed into several binary P-C or A-D relationships. Each binary sub-query is separately evaluated and its intermediate result is produced. The final result is formed by merging these intermediate results in the second phase. This method generates a huge amount of intermediate results that may not be part of the final results. In addition, the phase of merge is expensive. Various follow-up techniques have been proposed to filter out useless partial solutions and avoid the expensive merging phase [6, 7, 8].

The NCIM [1] method labels each element node of an XML data tree with 3-tuple (level, $n^{\vdash}$, $n^{\dashv}$) for non-leaf node and a 2-tuple (level, $n^{\vdash}$) for leaf node, where "level" is the depth of the node n with the root as level 1, "$n^{\vdash}$" (start number) is the serial number of node n derived from a depth-first traversal of the data tree (the root node is assigned 1 also), and "$n^{\dashv}$" (end number) is the serial number after visiting all child nodes of n. The information is clustered with same (tag, level) pair and stores them in four hash-based tables, two for node indexes and two for level indexes. The advantage of using hash tables is to gain fast accesses on the needed data. NCIM can deal with single-path query as well as more complex query patterns. The experimental results show that NCIM can compress XML documents with high compression rate and low index construction time. There have been many indexing methods proposed in the literature for XML query processing. However, very few are known to scale out for a large number of big XML documents.

In recent years, parallel XML parsing and filtering have been discussed for processing streaming XML data in scientific applications. The results of parsing XML can be DOM-style or SAX-style. The parallel DOM-style parsing constructs a tree data structure in memory to represent the document [9, 10, 11]. The load-balancing scheme is widely applied that assigns work to each core as the XML document was being parsed. The parallel SAX-style parsing visits XML document in depth-first traversal. It is much more suitable when XML documents are streaming. In Pan et al. [12], they present algorithms on how to parallelize the parsing computations prior to issuing the SAX callbacks (representing the events). Although some of parallel XML parsing techniques have been proposed, indexing and querying XML documents on large distributed environments remains a challenging issue.

3 Preliminaries on Hadoop

The Apache Hadoop software library, inspired by Google Map-Reduce and Google File System, is a framework that allows for the distributed processing of large data sets across clusters of computers using a simple programming model [13]. Hadoop consists of two main services: high-performance parallel data processing using a technique called MapReduce and reliable data storage using the Hadoop Distributed File System (HDFS). Since Hadoop is well suited to process large data sets, the proposed system uses Hadoop as the cloud computing framework.

The MapReduce, illustrated in Fig. 1, has two computation phases, map and reduce [14]. In the map phase, an input is split into independent chunks which are distributed

to the map tasks. The mappers implement compute-intensive tasks in a completely parallel manner. The output of the map phase is of the form <*key, value*> pairs. The framework sorts the outputs of the mappers, which are then passed to the second phase, the reduce phase. The reducers then partition, process and sort the <*key, value*> pairs received from the Map phase according to the *key* value and make the final output.

The HDFS is a distributed file system designed to store and process large (terabytes) data sets. HDFS is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets [15]. HDFS has a master/slave architecture that consists of a single NameNode and a number of DataNodes. The NameNode, the master of the HDFS, maintains the critical data structures of the entire file system. The DataNodes, usually one per node in the cluster, manage storage attached to the nodes that they run on. Internally, a file is split into one or more blocks that are stored in a set of DataNodes with replication. The NameNode executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. The DataNodes are responsible for serving read and write requests from the file system's clients. The DataNodes also perform block creation, deletion, and replication upon instruction from the NameNode [15].

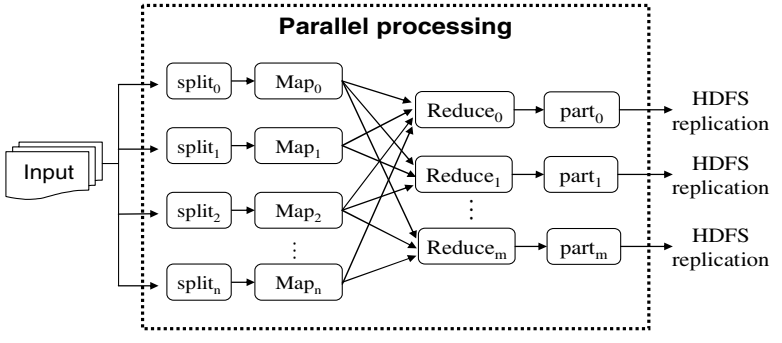


Fig. 1. The processing of MapReduce

Hadoop MapReduce framework runs on top of HDFS. Typically the compute nodes (the Map/Reduce framework) and the storage nodes (the HDFS) are running on the same set of nodes [14]. Thus data processing is co-located with data storage. A small Hadoop cluster will include a single master and multiple slaves. Fig. 2 [16] shows a hadoop system with multi-core cluster. The master for the MapReduce implementation is called the "JobTracker", which keeps track of the state of MapReduce jobs and the workers are called "TaskTrackers", which keep track of tasks within a job. The job tracker assigns jobs to available task tracker nodes in the cluster as close to the data as possible. If a task tracker fails or times out, that part of the job is rescheduled by job tracker.

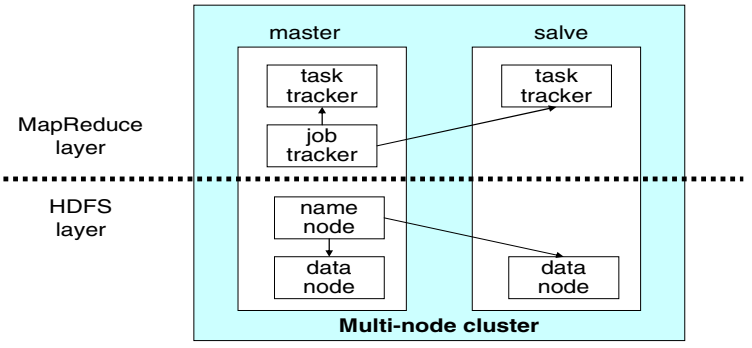


Fig. 2. The Hadoop system overview [16]

Since the release of Hodoop system, more and more researches use Hadoop as a framework to develop applications on large-scale data sets. For example, Zhang, et al. [17] describe a case study that uses the Hadoop framework to process sequences of microscope images of live cells. Dutta, et al. [2] address the problem of time series data storage on large distributed environments with a case-study of electroencephalogram using Hadoop.

4 The Proposed System

Consider a database that contains a large number of XML documents with different structures and sizes. It is a challenging task to retrieve required information from such huge amount of documents. In this paper, we develop a Hadoop-based XML query processing system for indexing and querying large number of XML documents. The proposed system consists of two subsystems. The first one is the preprocessor that parses XML documents and builds indexes. The second subsystem is the query processor that accepts queries from users and does query evaluation with the help of indexes.

4.1 Index Construction

The indexing method used in the proposed system is NCIM with some modification. We choose NCIM because the experimental results show that NCIM can compress XML documents effectively and support complex queries efficiently. The original NCIM method constructs and stores the indexes, which consist of four hashed-based tables, in main memory to support fast accesses. However, this implementation is not suitable for big data size of XML documents. Therefore, we write the indexes into files in the proposed system. The Non-leaf node index and the leaf node index are stored based on the hash keys. That is, data in each linked list to which a hash entry points is written into a file named using the corresponding hash key. Using this storage strategy, we need only to load required data while a query is processed. There is another modification in the proposed implementation of NCIM. Only text contents

which are less than 20 characters in the leaf node index are stored in the file for saving space. However, this restriction will be lifted in the future implementation for supporting wildcard characters in the queries.

In this phase, the input (see Fig. 1) is a sequence of files and each file represents an XML document. We refer to each file as a split, according to the Hadoop terminology, and feed splits to the map tasks. The splits are then processed in parallel. The SAX parser is used to parse the input XML document and the modified NCIM method is used to construct indexes. These two utilities reside in the Map function. Each Map function produces a list of <*key, value*> pairs, where *key* is a (*tag, level*) pair and the *value* is the corresponding *label* of a node. After that, the MapReduce framework collects all pairs with the same key from all lists and groups them together. The Reduce function is then applied in parallel to each group, which in turn produces a collection of values in the same *key,* and then the results are written into files. The output files may reside in different data nodes depending on HDFS. The flow of index construction is shown in Fig. 3.

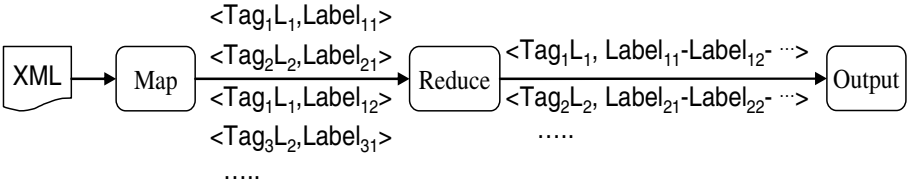


Fig. 3. The flow of index construction

4.2 Query Evaluation

After building XML indexes in the cloud servers, users may start sending queries to the cloud servers. The cloud servers may receive tens of thousands of queries per second and must be designed to respond in a very fast way. Fig.4 shows the overview of the proposed XML-Cloud service system. The cloud frontend receives user queries and submits them to the JobTracker. The JobTracker decides how many map tasks are required and distributes these tasks to the chosen TaskTracker nodes. Because the indexes are stored in HDFS, the JobTracker schedules tasks on the nodes where data is present or nearby. The TaskTracker loads indexes from HDFS and performs query evaluation. In this case, no reducer tasks are needed.

The query evaluation in our system is based on the algorithm of NCIM. The difference is that NCIM keeps indexes in main memory and the proposed system saves indexes in HDFS. Loading corresponding indexes are necessary before doing query evaluation in our system.

In the proposed system, we design two query processing modes. One is called streaming mode (Mode I). The second one is called batch mode (Mode II). In streaming mode, we treat user queries as a stream, and then each query is treated as a split and assigned to a map task. However, this may require a large number of I/O because each query will load corresponding part of indexes for evaluation. Two queries over the same documents may be assigned to different machines. The

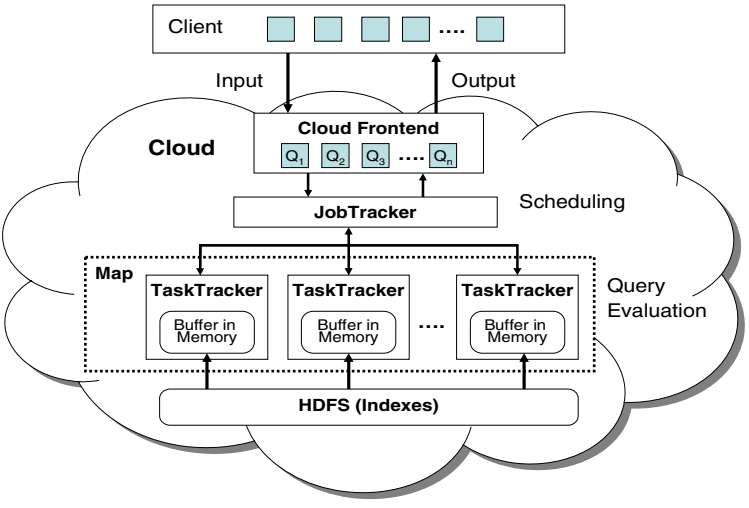


Fig. 4. The overview of XML-Cloud service system

common parts of indexes are loaded and released in different nodes. This will impose high cost for servers and result in poor performance.

In the batch mode, user queries are collected for a time period, e.g., one second, in the cloud frontend, and then classified into groups according to some similar characteristics. A query group is then treated as a split and assigned to a machine that loads common parts of indexes once and releases them after all queries in a group are finished. There may be a delay between query being entered into the system and the query being processed. However, the throughput of the system increases substantially when there are many user queries at the same time.

5 Experimental Results

In the experiments, we use a homogeneous cluster, which is composed of 15 slave nodes on Linux X86-64 with 4 CPUs and 8GB DRAM[1]. The Hadoop 0.20.1 is installed to run the experiments. Because it is a small cluster, the test datasets are not big comparing to real world datasets. The maximum size of datasets is about 2.5GB, which contains 50 XML files with different sizes. The maximum and minimum size of XML files are 75MB and 32MB, respectively. All essential functionalities are put inside the cloud. There is also a simple user interface at the client side for submitting XML documents and queries.

5.1 Performance of Index Construction

In the index construction phase, a set of XML files are parsed, and indexes are produced, which are then stored as HDFS files. In order to evaluate the performance

[1] Thanks to the National Center for High-Performance Computing, Taiwan, for providing the Hadoop computing cluster.

of the system under different data size, we form 5 datasets of size 0.5GB, 0.9GB, 1.4GB, 1.9GB, and 2.5GB with 10, 20, 30, 40, and 50 files, respectively. Fig. 5 shows the execution time for one file per task. The execution time includes loading XML files, executing MapReduce program, and saving index files to HDFS. It can be seen that the execution time is not increased linearly in terms of the number of files. The reason is because there are 15 slave nodes in the cloud, and the tasks are distributed unevenly when the number of input files is not the multiple of 15. We also observed that the Hadoop spent most of times in reduce tasks, where the HDFS creates multiple replicas of data blocks and distributes them on compute nodes. The *replication factor* is set to 3 for all tests.

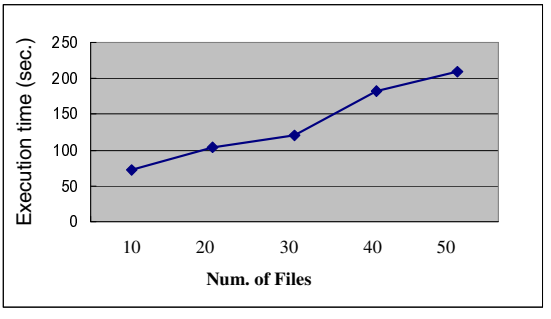


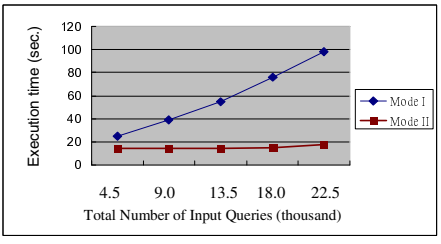
Fig. 5. Execution time of index construction

5.2 Performance of Query Evaluation

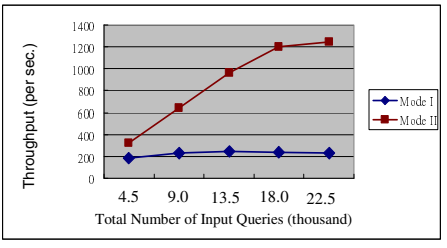
In the query evaluation phase, we feed a large number of queries to the system and examine the performance of query evaluation. The queries used in the experiments are generated by YFilter [18]. The query patterns may be either P-C or A-D relationships. We randomly choose required quantity of queries from the set of distinct queries. Duplicates are allowed in the experiments. We consider two types of query processing modes. Mode I is the streaming mode in which the incoming queries are entered as a stream. Each query is treated as a split and is assigned to a map task. Mode II is the batch mode. A set of queries over the same documents are classified as a group and also treated as a map task. We performed our experiments by using 30 files (1.4GB), a multiple of 15, to maximize the difference between two modes. Fig. 6 (a) shows the results of the execution time on the number of input queries form 4.5 thousand to 22.5 thousand in an increment of 4.5 thousand.

As we mentioned in Subsection 4.2, in Mode I, the system will load corresponding parts of indexes and release them once the end of a query evaluation is reached. The execution time increases steadily with the increase of the number of queries. In Mode II, the system holds the loaded indexes in memory until a group of queries is finished. The indexes can be reused and the cost of I/O is reduced. There is no obvious increase in execution time with the increase in the number of queries for Mode II. The reason is because the time spending on query evaluation is very low comparing to the time spending on I/O. Also because the queries may be duplicated, the loaded indexes in

different tests may be similar. Therefore, the differences in query processing time are not significant. Fig. 6 (b) illustrates the throughput of query evaluation. The throughput is defined as the average number of queries that can be processed in one second. It shows the throughput in Mode I is not improved when the number of queries increases. However, the throughput in Mode II increases rapidly due to the batch processing of the queries.



(a) Execution time of query evaluation



(b) Throughput of query evaluation

Fig. 6. Performance comparisons of query evaluation

6 Conclusions

In this paper, we proposed a system that builds indexes and processes enormous amount of queries for a large number of XML documents using Hadoop framework. The suitability of NCIM, which was developed by our research team, for large number of XML documents is demonstrated in this paper. The experimental results show that the proposed system can deal efficiently with large input XML files. The experimental results also show the throughput of the batch query processing mode is much higher than the streaming mode. In the batch processing mode, the throughput of 1200 queries per second for huge amount of queries using a 15-node cluster signifies the potential applications of NCIM to the fast query processing of enormous Internet documents.

References

1. Liao, I.-E., Hsu, W.-C., Chen, Y.-L.: An Efficient Indexing and Compressing Scheme for XML Query Processing. In: Zavoral, F., Yaghob, J., Pichappan, P., El-Qawasmeh, E. (eds.) NDT 2010. CCIS, vol. 87, pp. 70–84. Springer, Heidelberg (2010)
2. Dutta, H., Kamil, A., Pooleery, M., Sethumadhavan, S., Demme, J.: Distributed Storage of Large Scale Multidimensional Electroencephalogram Data using Hadoop and HBase. In: Grid and Cloud Database Management. Springer, Heidelberg (2011)
3. Thiébaut, D., Li, Y., Jaunzeikare, D., Cheng, A., Recto, E.R., Riggs, G., Zhao, X.T., Stolpestad, T., Nguyen, C.L.T.: Processing Wikipedia Dumps: A Case-Study comparing the XGrid and MapReduce Approaches. In: 1st International Conference on Cloud Computing and Services Science (2011)

4. Goldman, R., Widom, J.: DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In: 23rd International Conference on Very Large Data Bases, pp. 436–445 (1997)
5. Al-Khalifa, S., Jagadish, H.V., Koudas, N., Patel, J.M., Srivastava, D., Wu, Y.: Structural Joins: a Primitive for Efficient XML Query Pattern Matching. In: 18th IEEE International Conference on Data Engineering, pp. 141–152. IEEE Press, Washington, DC (2002)
6. Bruno, N., Koudas, N., Srivastava, D.: Holistic Twig Joins: Optimal XML Pattern Matching. In: 2002 ACM SIGMOD International Conference on Management of Data, pp. 310–321. ACM Press, New York (2002)
7. Chen, S., Li, H.G., Tatemura, J., Hsiung, W.P., Agrawal, D., Candan, K.S.: Twig^{2}Stack: Bottom-Up Processing of Generalized Tree-pattern Queries over XML Documents. In: 32nd International Conference on Very Large Data Bases, pp. 283–294 (2006)
8. Qin, L., Yu, X.J., Ding, B.: TwigList: Make Twig Pattern Matching Fast. In: 12th International Conference on Database Systems for Advanced Applications, pp. 850–862 (2007)
9. Pan, Y., Lu, W., Zhang, Y., Chiu, K.: A Static Load-Balancing Scheme for Parallel XML Parsing on Multicore CPUs. In: 7th IEEE International Symposium on Cluster Computing and the Grid, Brazil (2007)
10. Lu, W., Chiu, K., Pan, Y.: A Parallel Approach to XML Parsing. In: 7th International Conference on Grid Computing, pp. 28–29. IEEE Press, Washington, DC (2006)
11. Pan, Y., Zhang, Y., Chiu, K.: Simultaneous Transducers for Data-Parallel XML Parsing. In: 22nd IEEE International Parallel and Distributed Processing Symposium (2008)
12. Pan, Y., Zhang, Y., Chiu, K.: Parsing XML Using Parallel Traversal of Streaming Trees. In: Sadayappan, P., Parashar, M., Badrinath, R., Prasanna, V.K. (eds.) HiPC 2008. LNCS, vol. 5374, pp. 142–156. Springer, Heidelberg (2008)
13. Welcome to Apache™ Hadoop™!, `http://hadoop.apache.org/` (retrieved date: June 27, 2011)
14. Map/Reduce Tutorial, `http://hadoop.apache.org/common/docs/r0.20.2/mapred_tutorial.html` (retrieved date: June 27, 2011)
15. Welcome to Hadoop™ Distributed File System!, `http://hadoop.apache.org/hdfs/` (retrieved date: June 27, 2011)
16. Wikipedia, Apach Hadoop, `http://en.wikipedia.org/wiki/Apache_Hadoop` (retrieved date: June 29, 2011)
17. Zhang, C., De Sterck, H., Aboulnaga, A., Djambazian, H., Sladek, R.: Case Study of Scientific Data Processing on a Cloud Using Hadoop. In: Mewhort, D.J.K., Cann, N.M., Slater, G.W., Naughton, T.J. (eds.) HPCS 2009. LNCS, vol. 5976, pp. 400–415. Springer, Heidelberg (2010)
18. YFilter: Filtering and Transformation for High-Volume XML Message Brokering, `http://yfilter.cs.umass.edu/code_release.html` (retrieved date: June 29, 2011)