2015 IEEE International Conference on Smart City/SocialCom/SustainCom together with DataCom 2015 and SC2 2015

# MapReduce Implementation of XML Keyword Search Algorithm

Yong Zhang
[1]School of Computer and Information Technology
Liaoning Normal University, Dalian, China
[2]State Key Lab. for Novel Software Technology
Nanjing University, Nanjing, China
E-mail: zhyong@lnnu.edu.cn

Quanlin Li, Bo Liu
School of Computer and Information Technology
Liaoning Normal University
Dalian, China

***Abstract*—Keyword search for smallest lowest common ancestors (SLCAs) is an important approach to identify interesting data nodes in XML documents. With the rapid growth of XML data in Internet, how to effectively process massive XML data becomes an interesting topic. As an open-source cloud computing platform developed in recent years, Hadoop is a trend to process large-scale data, which makes possible massive storage and efficient search of XML data. In this paper, we first present two properties to improve the classical ILE algorithm. Then, a kind of parallel XML keyword search algorithm is proposed and realized on a MapReduce programming model. Two experiments on 4 datasets of different sizes in cluster are performed. The results show that our proposed algorithm is applicable to keyword search of massive XML data.**

***Keywords- XML; SLCA; keyword search; MapReduce; parallelization***

## I. INTRODUCTION

Extensible Markup Language (XML) is a simple, very flexible text format. XML is playing an increasingly important role in the exchange of a wide variety of data, and has become the de-facto standard for data storage and exchange on the Web. At present, with the rapid growth of data on the Web, it is a hot research topic to store and search massive XML data in an efficient and accurate way. XPath / XQuery technology and keyword-based search are two mainly approaches to obtain desired information from XML data.

Keyword-based search methods attempt to get the most compact fragment keyword tree through the operations on XML data [1]. The smallest lowest common ancestor (SLCA) is an important approach to keyword search in XML documents [2-4]. It does not require users to know the hierarchy of XML document in advance. Many literatures focus on the efficiency of XML keyword search, such as [5-7]. Xu *et al*. [1] propose three kinds of classical algorithms based on SLCA semantics, e.g., Indexed Lookup Eager (ILE), Scan Eager (SE) and stack-based algorithm (Stack), and verified that ILE has better performance through experiments. ILE algorithm starts from the smallest candidate node set $S_{min}$, through recursion between $S_{min}$ and other node sets to select node matches and binary search among nodes to determine each lowest common ancestor (LCA), then calculate SLCA set by removing ancestors in LCA results. Sun *et al*. [3] propose two multiway-SLCA algorithms, a basic multiway-SLCA (BMS) and an incremental multiway-SLCA (IMS), to enhance algorithm efficiency. IMS is an optimized variant of BMS that reduces the number of LCA computations. Li *et al*. [4] introduce the notion of Valuable Lowest Common Ancestor (VLCA) to accurately and effectively answer keyword queries over XML documents, propose the concept of Compact VLCA, and give an effective optimization strategy for speeding up the computation. Liu *et al*. [5] analyze XML data structure and keyword matching model and designed algorithm return generation result. Xu *et al*. [6] present a semantic Exclusive LCA (ELCA) based on the deficiency in SLCA and propose an IS (Index Stack) algorithm based on stack structure. Bao *et al*. [8] propose a novel XML keyword search strategy based on the related guide according to SLCA semantics and in combination with XReal and XSeek, and make the quality analysis of searching result. Top-*K* processing for XML keyword search is an important issue that has received very little attention. Chen *et al.* [9] present a series of join-based algorithms and return the first *K* results in XML keyword search in combination with top-*K* algorithm. Zhou *et al*. [10] propose a family of algorithms focused on efficient keyword query processing on XML data based on SLCA and ELCA semantics. The algorithms based on set intersection operation give a fast SCLA and ELCA computation for XML keyword query. Zhao *et al*. [11] describe a keyword search measure on probabilistic XML data based on extreme learning machine, and use this method to carry out keyword search on probabilistic XML data. Based on the previous researches and SLCA and ELCA semantics, Dimitriou *et al*. [12] propose the top-*k*-size stack-based algorithm on tree-structured data. The proposed algorithm implements ranking semantics for keyword queries based on the concept of LCA size.

However, the fast growth of XML data also brings big challenges for traditional data storage and processing manners. The algorithms above are realized in a single-computer environment and the scale of dataset used in experiments is not large. When these algorithms are used in large-scale XML documents, they may generate a huge tag-tree and exhausts system resources, which leads to

978-1-5090-1893-2/15 $31.00 © 2015 IEEE
DOI 10.1109/SmartCity.2015.154

uncompleted query. The parallelization is the most effective and feasible solution to settle the problem. The MapReduce programming model introduced by Google in [13] can process effectively large-scale data. At present, some scholars have researched the parallel XML keyword query in MapReduce framework. Zhou *et al*. [14] have realized XML data keyword search under MapReduce cluster. Li and Tao [15] also implement XML keyword search on MapReduce. In their framework, distributed XML parsing was conducted before searching for candidate node set. However, it does not optimize the original serial algorithm but performs the parallel on the basis of the original algorithms.

This paper presents two properties to optimize and improve the classical ILE algorithm, and implements the parallel algorithm of improved ILE on MapReduce framework. The simulation results show that the proposed algorithm is effective and efficient in processing keyword search in massive XML data.

The remaining of the paper is organized as follows. In Section II we simply introduce the related work, such as SLCA and MapReduce framework. We present two improved properties to ILE algorithm and give the parallel implement in Section III. We show the evaluation results through a series of experiments in Section IV. Finally, Section V concludes the paper and gives future work.

## II. PRELIMINARIES

In this section, we mainly introduce two issues as preliminaries for our work. They are SLCA semantics and MapReduce framework.

### *A. SLCA*

#### *1) SLCA semantics*

Keyword search is considered to be an effective information discovery method for both structured and semi-structured data. At present, most XML keyword search techniques are based on LCA, which can return the most compact segment tree. SLCA is an important concept for XML keyword search. It is used mainly for defining the significant result returned for the given keyword search, which is the core problem in keyword query research [2-4, 7, 8, 16]. The concrete definition of SLCA is to solve the subtree root nodes satisfying two conditions below:

(1) The subtree rooted by the nodes contains all keywords.

(2) No descendant node also contains all keywords.

XML keyword query work involves mainly in generation of candidate point set and reduction in SLCA calculation. As the return result of XML keyword query, SLCA reflects the inclusion relation between candidate point sets.

#### *2) SLCA solution*

ILE and SE are the classical algorithms for solving SLCA. Both algorithms produce part of the answers quickly so that users do not have to wait long to see the first few answers [2]. The basic idea of ILE algorithm is to design a kind of data format of $B^+$ tree structure, to facilitate the *lm* operation and *rm* operation and obtain all corresponding Dewey code sets of given keyword, and the nodes in set are sorted in order of size.

ILE algorithm is applicable to the keyword set containing low-frequency keyword; However, SE algorithm, as a variation of ILE algorithm, is applicable to all conditions with little frequency fluctuation of keyword. Both algorithms are intended for solving SLCA of $k$ keywords, on the basis of the same rule, needing $k$-1 intermediate variable sets in computation process, and a pair (two) node sets are required as the input for each SLCA calculation before generation of an intermediate result set.

The two algorithms are subjected to delicate design for reduction of time complexity for SLCA solution. The complexity of ILE is $O(|S_1|\sum_{i=2}^{k} d\log|S_i|+|S_1|^2)$. The complexity of SE is $O(k|S_1|+d\sum_{i=2}^{k}|S_i|)$ [2].

Although ILE algorithm has superior performance, it still has the following deficiencies: firstly, it saves XML data on B+ tree structure and therefore B+ tree structure must be modified to support the necessary Dewey code operation, which is complex to realize. Moreover, B+ index structure is not applicable to Dewey code data [4]; Secondly, SLCA computation process is "obstructed" in ILE algorithm, that is, Dewey set $S_i$ must be calculated in turn and the processing of the $i$-1$^{th}$ keyword must be completed before processing the $i^{th}$ keyword [2], which greatly reduces the algorithm speed.

### *B. MapReduce framework*

Hadoop is a well-known open-source software framework to process large-scale data in large clusters with thousands of nodes [17]. Hadoop mainly consists of the distributed file system (HDFS) and MapReduce programming model. HDFS is the storage layer for the Hadoop framework, which can realize the efficient storage and management of data on the cluster-based cloud. An HDFS cluster has a master/slave architecture and includes a single *NameNode* and series of *DataNodes*. The *NameNode* manages the file system namespace and regulates access to files by clients. The *DataNodes* are responsible for serving read and write requests from the file system's clients. MapReduce is a parallel and distributed programming model in support of large dataset processing. Figure 1 shows the overview of a MapReduce programming framework [13].

As a kind of parallel programming model, the MapReduce consists of the *Map* and the *Reduce* functions. The input for a MapReduce-based program is a list of ($k$1, $v$1) pairs and the *Map* function is applied to each key/value pair to produce one or more intermediate key/value pairs, i.e., ($k$2, $v$2) pairs, which will be consumed by the *Reduce* function later. This process is called *Map task*. Correspondingly, the nodes that run the *Map tasks* are called *Mappers*. The intermediate key/value pairs are then grouped by $k$2 values, i.e., ($k$2, *list*($v$2)). For each $k$2, the *Reduce* function reads a list of all values, *list(v2)*, then produces an aggregated result, i.e., ($k$3, $v$3). This process is called *Reduce task* and is the relevant nodes are called *Reducers*. Conceptually, the process can be described by the following two formulas:

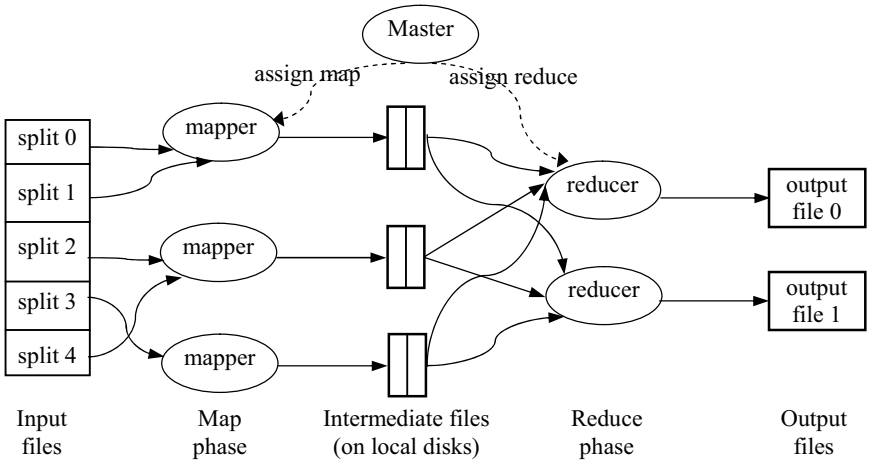



Figure 1. Overview of MapReduce programming framework

$$Map(k1,v1) \rightarrow list(k2, v2), \quad (1)$$

$$Reduce(k2, list(v2)) \rightarrow list(k3, v3). \quad (2)$$

## III. Improved Strategies to ILE Algorithm and Its Parallel Implement

### A. Optimization and improvement of ILE algorithm

Aiming at the above mentioned issues emerged in ILE algorithm, an improved ILE algorithm is proposed for data optimization from the source in this paper. The optimization and improvement of ILE algorithm are described from such two properties as follows.

The time complexity of ILE algorithm is $O(|S_1|\sum_{i=2}^{k} d\log|S_i| + |S_1|^2)$, where $S_1$ is the minimum size of keyword lists $S_i$ ($i$=1 to $k$). So the algorithm efficiency is hugely affected by $S_i$, and the computing cost can be reduced before SLCA computing by means of decrease in $S_i$ ($i$ = 1 to $k$). We give the first property to optimize ILE algorithm.

**Property 1**: $slca(S_1, S_2, \ldots, S_k)=slca(removeAncestor(S_1), removeAncestor(S_2), \ldots, removeAncestor(S_k))$.

In Property 1, *removeAncestor* is the function for removing the ancestor node in Dewey code. According to SLCA semantics, to solve the SLCA of keyword set $w_i$ is to solve the longest common substring between the corresponding Dewey code set of keyword set. It is obvious that to remove the ancestor node in $S_i$ will not affect the correctness of SLCA solution but accelerate the computation process.

For example, supposing the corresponding Dewey code set of keyword $w_0$ is $S_0$={0.0.0.1} and the corresponding Dewey code set of keyword $w_1$ is $S_1$={0.0, 0.0.0, 0.1.1.0.1, 0.1.1.1.0, 0.1.1.2.1.0, 0.1.1.3.0, 0.1.2.0.0, 0.2.0.0.0}. When we calculate SLCA between $S_0$ and $S_1$, an intermediate result {0.0} will be generated before *removeAncestor* processing, and a *slca* calculation will be reduced obviously and then it may get the result SLCA={0.0.0} directly after execution of *removeAncestor* operation.

Property 1 provides a basis for cutting $S_i$ size and can reduce some intermediate calculation steps by removing the ancestor node in $S_i$.

Moreover, the algorithm parallelization will be realized in the following part of this paper. In the process of algorithm parallelization, $S_1$ can be spitted and correct SLCA solving is the foundation and guarantee for realization of algorithm parallelization. We give the second property to improve the parallelization of ILE algorithm.

**Property 2**: $slca(S_1, \ldots, S_i, \ldots, S_k)=slca(slca(S_1, S_2), \ldots, slca(S_i, S_{i+1}), \ldots, slca(S_{k-1}, S_k))$, where $1 \le i < k$.

If $v_{12} \in slca(S_1, S_2)$, a match={$v_1$, $v_2$} with $v_{12}$ as the anchor point must exist (where satisfies $v_1 \in S_1$ and $v_2 \in S_2$), so $v_{12}=lca(v_1, v_2)$ [10]; Similarly, $v_{34}=lca(v_3, v_4)$.

Consider A=$slca\{S_1, S_2\}$ and B=$slca\{S_3, S_4\}$. if $v_{AB} \in slca(A, B)$, then a match={$v_{12}$, $v_{34}$} with $v_{AB}$ as the anchor point must exist (where satisfies $v_{12} \in A$ and $v_{34} \in B$), so $v_{AB}=lca(v_{12}, v_{34})$.

$v_{AB}=lca(v_{12}, v_{34})$, $v_{12}=lca(v_1, v_2)$ and $v_{34}=lca(v_3, v_4)$, so $v_{AB}=lca(v_1, v_2, v_3, v_4)$ and $slca(S_1, S_2, S_3, S_4) = slca(A, B)$. In conclusion, $slca(S_1,\ldots,S_i,\ldots,S_k)=slca(slca(S_1, S_2), \ldots, slca(S_i, S_{i+1}), \ldots, slca(S_{k-1}, S_k))$.

Based on such two properties and in combination with Hadoop's parallel mechanism, the ancestor will be removed at first during reading metadata in Map process. And then, $S_i$ will be grouped and each group will be delivered to a Reduce for parallel processing, for accelerating SLCA solution. At last, the results are collected and subjected to removing duplicates and removing ancestor.

### B. Realization of Hadoop platform-based algorithm

This section will present the realization of the proposed algorithm under Hadoop and MapReduce framework. The algorithm realization may be divided into four major parts as shown in Figure 2, which presents the flow chart of SLCA keyword query of keyword Title, Ben and John in School.xml document tree [2] as shown in Figure 3.

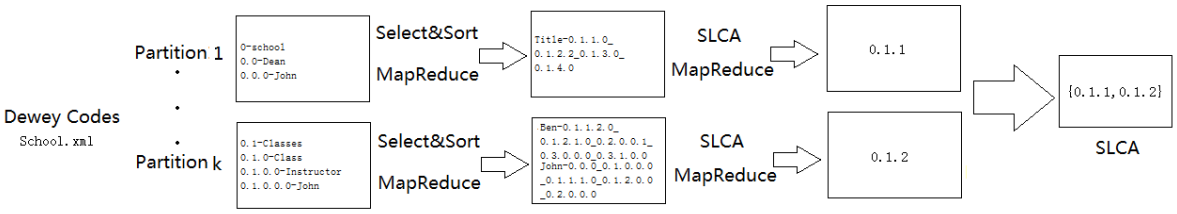



Figure 2. Flow chart of Hadoop-based SLCA keyword query.

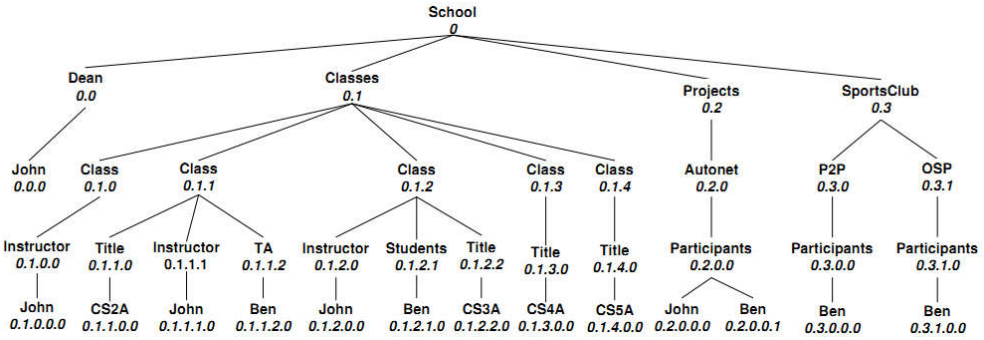



Figure 3. School.xml document tree (each node is identified by Dewey code).

### C. *Description of algorithm*

According to above two properties, this subsection gives a detail description of our proposed algorithm based on MapReduce framework.

Algorithm 1 first selects keyword to be looked up and its corresponding Dewey code, and saves all Dewey codes of the same keyword to the same set.

**Algorithm1. SelectMapReduce**

```
1:SelectMapper(key=offset, value=(dewey, keyword)){
2:  if keyword= w_i then
3:    output (keyword, dewey);
4:  end if
5:}
6:SelectReduce(key=(keyword, dewey), values){
7:  for all the same keyword do
8:    put dewey into S_i
9:  end for
10:  output (keyword, S_i);
11:}
```

**Algorithm2. SortMapReduce**

```
1:SortMapper(key=keyword, value=S_i){
2:  length =the size of dewey in S_i;
3:  output(length, value);
4:}
5:SortReducer(key=length, values){
6:  according to length, sort S_i from small to large
7:  output((keyword, S_i), Text());
8:}
```

**Algorithm3. SLCAMapReduce**

```
1:SLCAMapper(key=offset, value=(keyword, S_i)){
2:   removeAncestor(S_i);
3:   for i=2 to m do
4:     put S_i into Ss;
5:   end for
6:   split S_1 into several Sets;
7:   for each Set in S do
8:     result=ILE Algorithm(Set, Ss)[2];
9:     if result is not null then
10:      output(size of result, result);
11:    end if
12:  end for
13:}
14:SLCAReduce(key=(size of result, result), values){
15:  output(key, values);
16:}
```

In Algorithm 1, the $m$ keywords to be looked up are $\{w_1, w_2, \ldots, w_i, \ldots, w_m\}$. We select the keyword same as keyword to be looked up and its corresponding Dewey code in data file in *SelectMapper* function and merge its Dewey code set to save in $S_i$ in *SelectReduce* function. Then, the results are outputted to HDFS and will be used in the next sorting function module.

Algorithm 2 arranges the Dewey codes of each keyword in the ascending order. In Algorithm 2, $S_i$ will be sorted in the ascending order according to the number of Dewey code. By means of the characteristic of automatic sorting of Reduce, *SortReducer* function sorts keywords according to the number of Dewey code of the same keyword. The

number of SLCA must be smaller than $|S_{min}|$ at last, so the sequence from small to large, must be beneficial to computation process simplification.

Then, Algorithm 3 realizes the parallel of improved ILE algorithm. In Line 2, ancestor code is removed from Dewey code set according to Property 1. Apart from $S_1$, $S_i$ is included in the set *Ss*. Dewey codes are classified intelligently, based on which SLCA is computed in parallel, results of which are summarized.

## IV. EXPERIMENTAL ANALYSIS

The paper designs two groups of experiments by combining the several factors (the number and frequency of the key words, size of XML data sets, number of parallel cluster nodes) of XML keywords query and validates the high efficiency of the XML keywords query based on Hadoop by comparing the experiment results.

### *A. Experimental environment*

In our experiments, 16 nodes are involved in the cluster. The configuration information and operation environments of all nodes are the same. The operating systems are ubuntu10.12, Hadoop 0.20.2 and Java 1.6.0_21. One of them is used as the master node, and the rest 15 are used as slave nodes.

### *B. Experimental data*

#### *1) Data Set*

The following experiments use 4 XML datasets which are downloaded from the Internet data website (available at https://www.monetdb.org/Downloads) as follows: FR_meta.xml (**data1**), od_bsz-tit_130516_20.xml (**data2**), DE_meta.xml (**data3**), and od-up_bsz-tit_150316_01.xml (**data4**). Their basis information is summarized in Table 1.

TABLE I. BASIC STATISTICS OF THE TESTED DATASETS

| dataset | The size of dataset | The number of nodes | The number of Keywords |
|---|---|---|---|
| **data1** | 125.2MB | 3737432 | 92633 |
| **data2** | 226.3MB | 6999749 | 1050132 |
| **data3** | 247.1MB | 7389164 | 176431 |
| **data4** | 1.1GB | 35852619 | 3951735 |

#### *2) Queried keywords information*

Based on the need of the experiment, 6 subsets were selected randomly from each dataset in accordance with the number and frequency to search for the keyword groups. 24 keyword groups were available in total, which were numbered from Q1-Q24 and separated by commas, and the number and frequency of the keywords were separated by "-" as shown in Table II [8]. For example, 4-100 in query condition means 4 keywords and the frequency of each keyword is 100 (100±10%), i.e. the frequencies of all the keywords that appear 90-100 times are all 100.

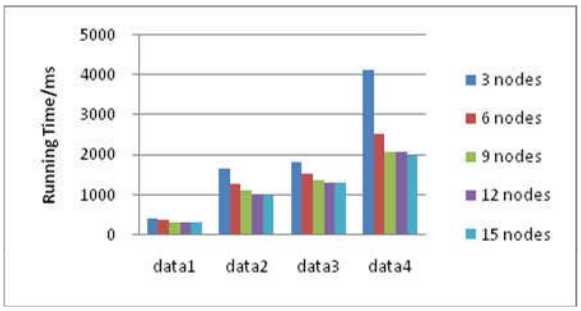
TABLE II. INFORMATION OF KEYWORD GROUPS

| dataset | ID | keywords | query condition |
|---|---|---|---|
| **data1** | Q1 | 90.217,1988_01_02,MQ,Villeurbanne | 4-10 |
| | Q2 | Organization,90.164,16,person_first_name | 4-100 |
| | Q3 | Station_city,street_name,population,NO2 | 4-1000 |
| | Q4 | Data_set,UNKNOWN,Max8,1.000 | 4-5000 |
| | Q5 | 90.164,16 | 2-100 |
| | Q6 | Organization,90.164,16,person_first_name,1999_06_01,organization_city,organization_name,unknown | 8-100 |
| **data2** | Q7 | C,M,DVD ROM,Growth | 4-10 |
| | Q8 | Boon,Duncker,Ratgeber,780 | 4-100 |
| | Q9 | Jpn,spa,lat,trl | 4-1000 |
| | Q10 | Aut,rvk,prf,pup | 4-5000 |
| | Q11 | Ratgeber,780 | 2-100 |
| | Q12 | Boon,Duncker,Ratgeber,780,UB,Jena,aui,grc | 8-100 |
| **data3** | Q13 | K,90.209,1993_03_031,Magdeburg | 4-10 |
| | Q14 | As,Ni,1983_07_01,station_description | 4-100 |
| | Q15 | 1988_01_01,NO,Background,SPM | 4-1000 |
| | Q16 | 361,362,363,364 | 4-5000 |
| | Q17 | 1983_07_01,station_description | 2-100 |
| | Q18 | As,Ni,1983_07_01,station_description,81.500,7018,Conductimetry,light_scattering | 8-100 |
| **data4** | Q19 | Set,sankt,UQ_1225,UB_2780 | 4-10 |
| | Q20 | Tests,Tempel,Testmaterial,UY | 4-100 |
| | Q21 | 1957,Stuttgart,DE_21_32a,DE_25_75 | 4-1000 |
| | Q22 | Politologie,adp,ad18,De_1033 | 4-5000 |
| | Q23 | Tests,Tempel | 2-100 |
| | Q24 | Tests,Tempel,Testmaterial,UY,UF_1500,UF_4000,UVK_Lucius,AF_02000 | 8-100 |

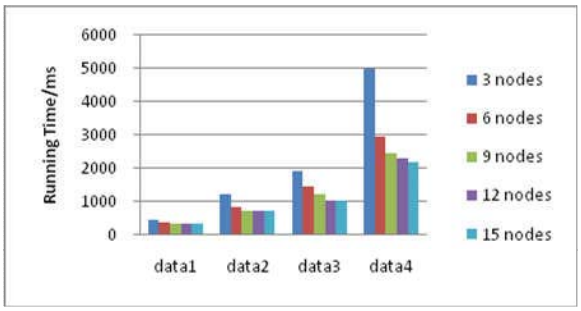
### C. *Experimental results and analysis*

**Experiment 1:** change the query condition and compare the running time.

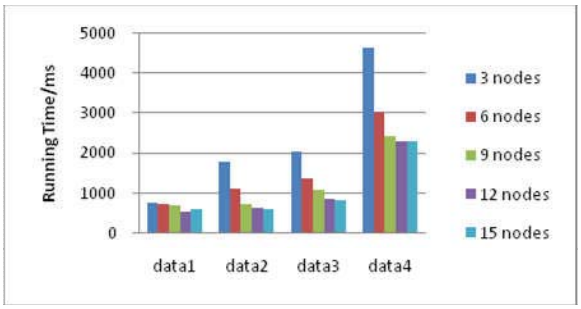
In this experiment, we change the size of nodes in the cluster, from 3 to 15, to test the running time of the proposed algorithm. The experimental results are shown in Figure 4.



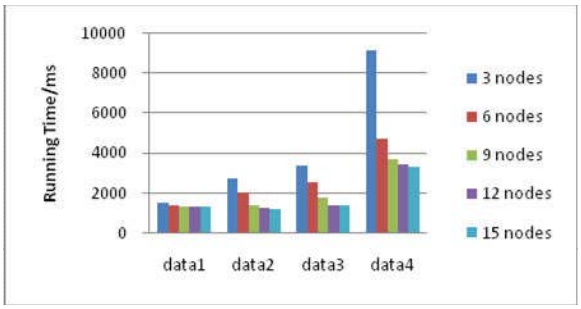

(a) Running time at query condition of 4-10



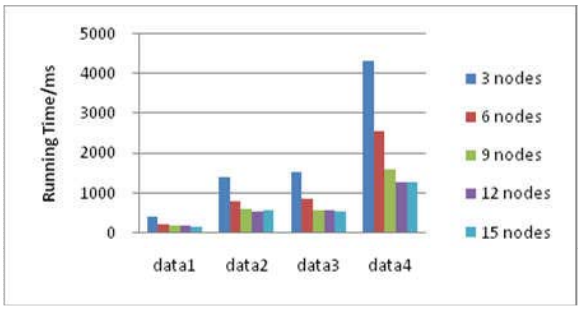

(b) Running time at query condition of 4-100



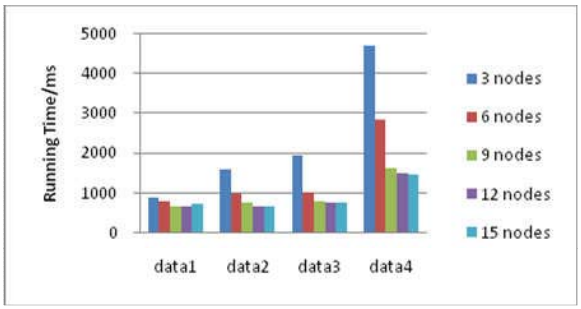

(c) Running time at query condition of 4-1000




(d) Running time at query condition of 4-5000




(e) Running time at query condition of 2-100



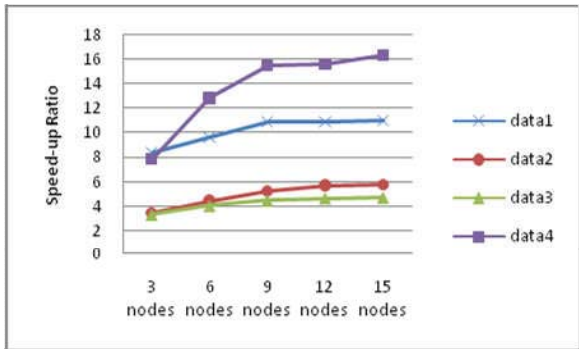

(f) Running time at query condition of 8-100

Figure 4. Running time at different query coditions.

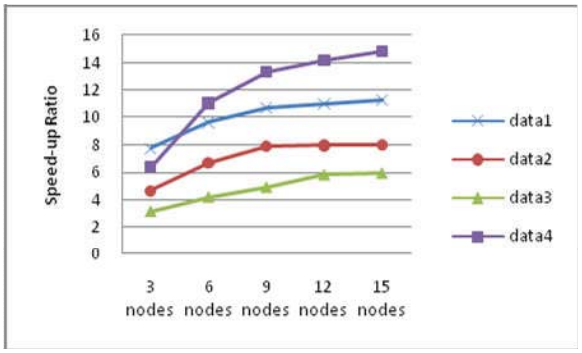
It can be seen from Figure 4 that, running effect of small-sized dataset is not ideal, such as data1-data3. The running time changes not obviously and even more slowly while the size of nodes gradually increases. However, for the data of large-sized data sets it shows obvious advantages, such as data4. The running time obviously decreases when the size of nodes changes from 3 to 6 as shown in Figure 4 (e and f). The running time changes slowly when the size of nodes reaches 9. The query condition has small effect on the running time. For the same dataset, the running time in the query condition of 4-5000 is longer than that in other query conditions. It indicates that the size and the frequency of keyword will affect the running time of the proposed algorithm. The experiment results have verified that the proposed algorithm is more suitable for larger sized data sets, which makes it possible to query massive xml keywords.

**Experiment 2:** change the number of cluster nodes and compare the speed-up ratio.

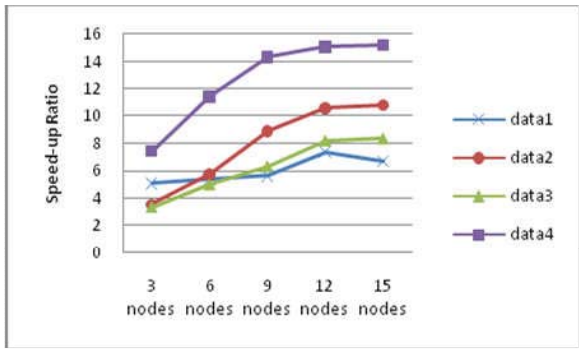
We have introduced the speed-up ratio to analyze the experiment results and measured performance and efficiency of parallel system or program parallelization. The specific definition of speed-up ratio is: Speed-up ratio = Runtime on single machine/Runtime on clusters.



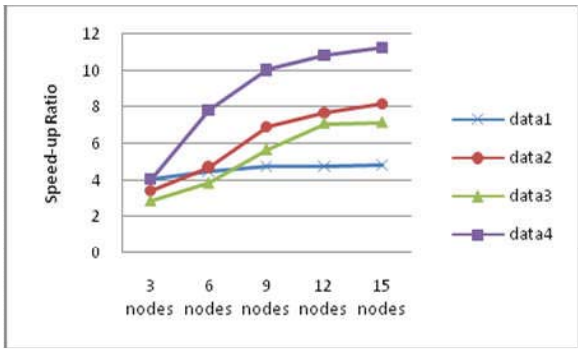

(a) Speed-up ratio at query condition of 4-10



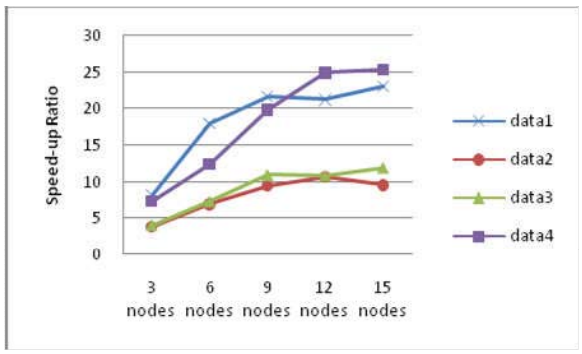

(b) Speed-up ratio at query condition of 4-100



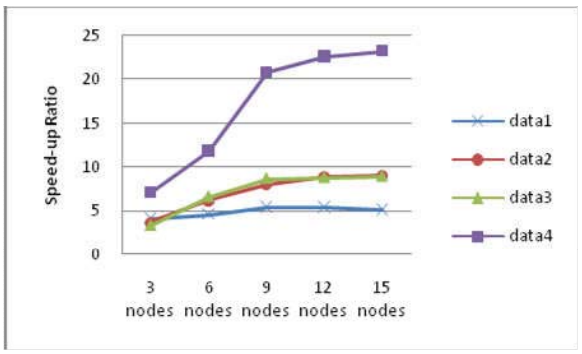

(c) Speed-up ratio at query condition of 4-1000




(d) Speed-up ratio at query condition of 4-5000




(e) Speed-up ratio at query condition of 2-100




(f) Speed-up ratio at query condition of 8-100

Figure 5. Speed-up ratio curve at different qurey conditions.

From the speed-up ratio curve in Figure 5, we can clearly see that, the speed-up ratio shows a tendency of increase as the size of nodes increases, while running time of the algorithm decreases significantly.

However, this is not the case for small sized data sets. We can also see that after 9 nodes, the increase of the speed-up ratio starts to slow down. This is because there is also an information exchange between the nodes of the same data set, which will occupy some system consumption. It is not true that the algorithm speed can increase unlimitedly as the increase of the number of nodes.

## V. CONCLUSION AND FUTURE WORK

In this paper, MapReduce-based XML keyword search parallelization algorithm is further researched in depth. XML documents are split into several parts to be processed parallel. In our algorithm, the classical ILE algorithm is improved through our given two properties. The paper employs Hadoop as the platform and implements improved ILE algorithm in MapReduce framework. The experiment results show that our proposed algorithm can process SLCA-based keyword search of large-scale XML data. Parallelization of XML Dewey code has not yet been implemented in place. In the future work, we will study it and give a more efficient

algorithm for keyword search and applications of the same in cloud computing environment.

## ACKNOWLEDGMENTS

This work is partly supported by National Natural Science Foundation of China (No. 61373127), the China Postdoctoral Science Foundation (No. 20110491530), and the University Scientific Research Project of Liaoning Education Department of China (No. 2011186).

## REFERENCES

[1] Y. Lin and G. J. Xu, "Distributed keyword search algorithm in XML databases using MapReduce," Lecture Notes in Electrical Engineering, 2012, 107: 1307-1316.

[2] Y. Xu and Y. Papakonstantinou, "Efficient keyword search for smallest LCAs in XML databases," Proceedings of SIGMOD, 2005: 537-538.

[3] C. Sun, C. Y. Chan, and A. K. Goenka, "Multiway SLCA-based keyword search in XML data," Proceedings of the 16th international conference on World Wide Web, 2007: 1043-1052.

[4] G. L. Li, J. H. Feng, J. Y. Wang, and L. Z. Zhou, "Effective keyword search for valuable LCAs over XML documents," Proceedings of the sixteenth ACM conference on Conference on Information and Knowledge Management, ACM Press, 2007: 31-40.

[5] Z. Liu and Y. Chen, "Identifying meaning return information for XML keyword search," Proceedings of SIGMOD, 2007: 329-340.

[6] Y. Xu and Y. Papakonstantinou, "Efficient LCA based keyword search in XML data," Proceedings of EDBT, 2008: 535-546.

[7] J. Li, C. Liu, R. Zhou R, and Y. Xu, "Quasi-SLCA based keyword query processing over probabilistic XML data," IEEE Transactions on Knowledge and Data Engineering, 2014, 26(4): 957-969.

[8] Z. Bao, J. Lu, T. W. Ling, and B. Chen, "Towards an effective XML keyword search," IEEE Transactions on Knowledge and Data Engineering, 2010, 22(8): 1077-1092.

[9] L. J. Chen and Y. Papakonstantinou, "Supporting top-k keyword search in xml databases," Proceedings of IEEE 26th International Conference on Data Engineering (ICDE), 2010: 689-700.

[10] J. Zhou, Z. Bao, W. Wang, T. W. Ling, Z. Chen, X. Lin, and J. Guo, "Fast SLCA and ELCA computation for XML keyword queries based on set intersection," Porceedings of IEEE 28th International Conference on Data Engineering (ICDE), 2012: 905-916.

[11] Y. Zhao, Y. Yuan, and G. Wang G, "Keyword search over probabilistic XML documents based on node classification," Mathematical Problems in Engineering, 2015: 135-144.

[12] A. Dimitriou, D. Theodoratos, and T. Sellis, "Top-k-size keyword search on tree structured data," Information Systems, 2015, 47: 178-193.

[13] J. Dean and S. Ghemawat, "Mapreduce: Simplied data processing on large clusters,"in the 6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, USA, 2004, pp. 137-150.

[14] M. Zhou, H. Hu, and M. Zhou, "Searching XML data by SLCA on a MapReduce cluster," Proceeding of 4th International conference on Universal Communication Symposium (IUCS), IEEE, 2010: 84-89.

[15] Z. Li and S. Tao, "A XML Keyword Search Algorithm Based on MapReduce," International Journal of Digital Content Technology & its Applications, 2012, 6(17): 307-316.

[16] C. Aksoy, A. Dimitriou, D. Theodoratos, and X. Wu, "XReason: A semantic approach that reasons with patterns to answer XML keyword queries," Proceedings of Database Systems for Advanced Applications, Springer Berlin Heidelberg, 2013: 299-314.

[17] Hadoop. Apache Software Foundation. http://hadoop.apache.org