Comput Syst Sci & Eng (2018) 2: 125–135
© 2018 CRL Publishing Ltd

International Journal of
Computer Systems
Science & Engineering

MapReduce Implementation of an Improved XML Keyword Search Algorithm

Yong Zhang[1,2], **Jing Cai**[1] **and Quanlin Li**[1]

[1] *School of Computer and Information Technology, Liaoning Normal University, Dalian, China*
[2] *State Key Lab. for Novel Software Technology, Nanjing University, Nanjing, China*

Extensible Markup Language (XML) is commonly employed to represent and transmit information over the Internet. Therefore, how to effectively search for keywords of massive XML data becomes a new issue. In this paper, we first present four properties to improve the classical ILE algorithm. Then, a kind of parallel XML keyword search algorithm, based on intelligent grouping to calculate SLCA, is proposed and realized under MapReduce programming model. At last, a series of experiments are implemented on 7 datasets of different sizes. The obtained results indicate that the proposed algorithm has high execution efficiency and is applicable to keyword search of massive XML data

Keywords: Extensible markup language; keyword search; MapReduce; parallelization

1. INTRODUCTION

Extensible Markup Language (XML) is the standard for data exchange on Web. Massive data on Web are stored and transmitted by means of XML, so it is a hot research topic, at present, to store and search XML data in an efficient and accurate way. Wide application of XML makes it worthy to research extraction of information from XML data. In order to obtain desired information from XML data, the users may turn to two query mechanisms: one is structured query language, which grammar is relatively complex and is not conducive to usage by many users; and the other is keyword search through which the information can be obtained.

Aiming at XML keyword search, the main research direction is the smallest lowest common ancestor (SLCA) [1, 2] and its series of variation query semantics, e.g., VLCA [3, 4] and Exclusive LCA (ELCA) [5, 6]. Lots of researches have been made thereon: Xu *et al.* [1] proposed 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 the ILE algorithm is superior to other algorithms. Sun *et al.* [2] proposed Multi-SLCA corresponding to ILE algorithm to enhance algorithm efficiency through jumping many redundancies based on anchor node. Li *et al.* [3] proposed the stack-based algorithm *VLCAStack* according to VLCA semantics and meaning Dewey code.Liu *et al.* [4] analyzed XML data structure and keyword matching model and designed algorithm return generation result. Xu *et al.* [5] proposed the new semantic ELCA based on the deficiency in SLCA return result and proposed index stack algorithm based on stack structure. Bao *et al.* [7] proposed XML keyword search based on the related guide order according to SLCA semantics and in combination with XReal and XSeek, and made the quality analysis of searching return result. Chen *et al.* [8] proposed a kind of join-based algorithm and returned the first K results in XML keyword search in combination with top-K algorithm. Zhou *et al.* [9] proposed the computation process of FastSLCA and FastELCA based on crossing set operation and put forward the FwdSCLA algorithm and BwdSLCA algorithm to generate SLCA and ELCA. Zhao *et al.* [10] made node classification of probabilistic XML file by means of extreme learning machine method. Based on the previous researches and SLCA and ELCA semantics, Dimitriou*et* *al.* [11] proposed the stack-based algorithm which returned *k* optimum results.

However, XML keyword search algorithms above only focus on small XML data, and are inadequate to deal with the large-scale XML data. The parallelization is the most effective and feasible solution to settle the problem [12]. At present, some scholars have studied the parallel XML keyword query. Zhang *et al.* [13] gave a simple solution to process XML query using MapReduce framework. Camacho-Rodríguez *et al.* [14] considered the problem of parallelizing the execution of XQuery and proposed a solution under MapReduce framework. Li *et al.* [15] realized IMS algorithm parallel in Hadoop programming framework. Zhang *et al.* [16] proposed a kind of distributed keyword search algorithm based on SLCA. However, it does not optimize the original serial algorithm but performs the parallel on the basis of the original algorithms. According to the feature of XML, this paper proposes an efficient keyword search algorithm for massive XML data based on MapReduce framework.

The main contributions of this paper are as follows:

- We demonstrate four properties and propose an *Intelligent Indexed Lookup Eager* (*IILE*) algorithm to optimize and improve the classical ILE algorithm.
- We implement the parallel algorithm of IILE based on MapReduce programming model to solve the problem in low efficiency on massive XML data.
- The simulation results demonstrate that the proposed *IILE* algorithm can effectively and efficiently deal with keyword search on massive XML data.

The remaining of the paper is organized as follows. SLCA and MapReduce framework are simply reviewed in Section 2. Four properties and the proposed IILE algorithm are presented in detail in Section 3. Section 4 analyses and evaluates the results through a series of experiments. Finally, the conclusion and future work are presented in the last Section.

2. RELATED WORK

2.1 SLCA

2.1.1 SLCA semantics

SLCA is the important concept for XML keyword query. It is used mainly for defining the significant result returned for the given keyword query, which is the core problem in keyword query research [1, 2, 6, 7, 17-20]. The concrete definition of SLCA is to solve the subtree root node meeting the two conditions below:

- The subtree includes all keyword sequences.
- The subtree includes no smaller subtree but all keyword sequences.

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.

2.1.2 SLCA solution

ILE and SE [1] are the classical algorithms for solving SLCA. The main principle of ILE is to design a kind of data format of B+ tree structure, to facilitate *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 is applicable to the keyword set containing low-frequency keyword. However, SE, as a variation of ILE, is applicable to all conditions with little frequency fluctuation of keyword. The two 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 node sets are required as the input for each SLCA calculation before generation of an intermediate result set.

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. 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 $i - 1^{\text{th}}$ keyword must be completed before processing the i^{th} keyword [4], which greatly reduces the algorithm speed. Therefore, the traditional ILE algorithm is not efficient for large-scale data.

2.2 Hadoop and MapReduce Framework

As an open-source framework, Hadoop mainly consists of the distributed file system (HDFS) and MapReduce. HDFS can realize the efficient storage and management of data on the cloud composed of computer cluster [21]. As a kind of parallel programming model, MapReduce is intended for parallel computing and highly abstracts the parallel computing process and develops it into Map function and Reduce function. Therefore, the compilation of application program in the MapReduce framework is the process of mapper and reducer customization. Such computing model requires that the input, output and intermediate data are present as a key-value pair <key, value>. Map function accepts an input in <k1,v1> form and also generates an intermediate input in <k2,v2> form. Hadoop will integrate v2 set with the same intermediate k2 value and then transfer it to Reduce function. Reduce function will accept an input in <k2, list of value> form and process the value set. Each Reduce output is in <k3,v3> form. Finally, the result will be returned to the application program.

3. INTELLIGENT INDEXED LOOKUP EAGER ALGORITHM BASED ON SLCA SEMANTICS

3.1 Optimization and Improvement of ILE Algorithm

Aiming at the above mentioned issues emerged in ILE algorithm, an improved ILE algorithm was proposed for data optimization, called *Intelligent Indexed Lookup Eager* (*IILE*). The optimization and improvement of ILE algorithm are described from such four aspects as follows:

To begin with, the complexity of the classical ILE can be expressed as $O(|S_1|\sum_{i=2}^{k} d\log|S_i|+|S_1|^2)$, where $|S_1|$ denotes the minimum size of keyword lists. k and d indicate the number of keywords and the maximum depth of XML tree, respectively. It is obvious that 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 $k)$.

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.

Example 1: 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. 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.

Next, if S_1 is split into ⌈|S₁|/|P|⌉ subsets (subgroups) [1], where P is the buffer of fixed size and subset is expressed by B, then SLCA result can be presented faster by solving $slca(B,S_2,\ldots,S_k)$.

Property 2: $slca(S_1,S_2)$=$removeAncestor(slca(B_1,S_2)\bigcup\ldots\bigcup$ $slca(B_i,S_2)\bigcup\ldots\bigcup slca(B_k,S_2))$, where B_i satisfies $S_1 = \bigcup_{i=1}^{k} B_i$ and $B_i \cap B_{i+1} = \emptyset$ $(1\le i < k)$.

Supposing $slca(S_1,\ S_2)\ \neq removeAncestor(slca(B_1,S_2)\bigcup$ $\ldots\ \bigcup\ slca(B_i,S_2)\bigcup\ldots\ \bigcup slca(B_k,S_2))$, then there exists B_i which must make $slca(B_i,S_2)\ \not\subseteq slca(S_1,S_2)$. $S_1 = \bigcup_{i=1}^{k} B_i$ and $B_i \cap B_{i+1} = \emptyset$ $(1\le\ i\ <\ k)$, so $B_i \subset S_1$ and $slca(B_i,\ S_2)\ \not\subseteq slca(S_1,\ S_2)$ is false. Therefore, $slca(S_1,S_2)$=$removeAncestor(slca(B_1,S_2)\bigcup\ldots\bigcup slca(B_i,$ $S_2)\bigcup\ldots\bigcup slca(B_k,S_2))$ is correct.

Then, it is proven that SLCA solution meets requirements of associative law. The algorithm parallelization will be realized in the following part of this paper and here is a parallel point for realization of algorithm parallelization. Therefore, S_1 can be split and correct SLCA solving is the foundation and guarantee for realization of algorithm parallelization.

Property 3: $slca(S_1,\ldots,S_i,\ldots,S_k)$=$slca(slca(S_1,S_2),\ \ldots,$ $slca(S_i,S_{i+1}),\ \ldots,\ slca(S_{k-1},S_k))$, 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 $v_1 \in S_1$ and $v_2 \in S_2$), so $v_{12} = lca(_1, v_2)$ [9]; Similarly, v_{34}=$lca(v_3,\ v_4)$.

Supposing 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 $v_{12} \in A$ and $v_{34} \in B$). So, $v_{AB} = lca(v_{12},\ v_{34})$.

$v_{AB} = lca(v_{12},\ v_{34})$, $_{12} = lca(v_1,\ v_2)$ 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, $slca(S_1,\ldots,S_i,\ldots,S_k) = slca(slca(S_1,\ S_2),\ \ldots,\ slca(S_i,\ S_{i+1}),\ldots,\ slca(S_{k-1},\ S_k))$.

Finally, in ILE and SE algorithms, SLCA is obtained by removing the ancestor node in SLCAs candidate set directly or indirectly, so the intermediate SLCA candidate set SLCAs is in fact generated based on a massive calculation process. See Example 2 below for intuitive description of the problem.

Example 2: Supposing the corresponding Dewey sets of two keywords w_1 and w_2 are S_1 = {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, 0.2.1.0.0, 0.2.2.0.0} and S_2 = {0.0.1.5, 0.1.0.9.0, 0.1.1.2.0, 0.1.2.1.0, 0.2.0.0.1, 0.3.0.0.0, 0.3.1.0.0, 0.3.2.0.0}. We calculate SLCAs between keywords w_1 and w_2 (corresponding to node set S_1 and S_2) based on ILE algorithm. The value of P is set to be 2. According to ⌈|S₁|/|P|⌉ principle, S_1 is divided equally into five groups of S_{11} = {0.0.0, 0.1.1.0.1}, S_{12} = {0.1.1.1.0, 0.1.1.2.1.0}, S_{13} = {0.1.1.3.0, 0.1.2.0.0}, S_{14} = {0.2.0.0.0, 0.2.1.0.0} and S_{15} = {0.2.2.0.0}, so we can obtain SLCA = $removeAncestor(slca(S_{11},\ S_2) \cup slca(S_{12},\ S_2) \cup slca(S_{13},\ S_2) \cup slca(S_{14},\ S_2)) \cup slca(S_{15},\ S_2)$, i.e., SLCAs= *removeAncestor*({{0.0,0.1.1}, {0.1.1.2}, {0.1.1,0.1.2}, {0.2.0.0}, {0.2}}). At last, we can obtain SLCAs={0.0, 0.1.1.2, 0.1.2, 0.2.0.0}. Two redundant SLCAs will be generated if the result set with only four SLCAs is obtained. In case of large XML document and many keywords, it is a heavy task to remove the ancestor node and the redundant SLCAs such generated is a huge waste of computing cost.

Property 4: Principle of intelligent grouping. If $lca(v_i,\ v_{i+1}) = lca(v_{i+1},\ v_{i+2})$, then v_i and v_{i+1} are in the same group; if $lca(v_i,\ v_{i+1}) > lca(v_{i+1},\ v_{i+2})$, then v_i and v_{i+1} are in the same group but v_{i+1} and v_{i+2} are not in the same group; and if $lca(v_i,$ $v_{i+1}) < pre(lca(v_{i+1},\ v_{i+2}))$, then v_i and v_{i+1} are not in the same group $(1 \le i \le k-2)$.

For the same Dewey code (node) v_{i+1}, $lca(v_i,\ v_{i+1}) \ge lca(v_{i+1},\ v_{i+2})$ indicates that v_i is more closer to v_{i+1}than v_{i+2}, that is, v_i has the longer common prefix than $v_{i+2}$with respect to v_{i+1}. Then, in the process of calculating SLCA, $slca(v_n,\ \ldots,\ v_i,\ v_{i+1},\ v_{i+2},\ \ldots,\ v_k)$ can be divided into $slca(v_n,\ldots,v_i)$ and $slca(v_{i+1},\ v_{i+2},\ldots,v_k)$, or $slca(v_n,\ldots,v_i,\ v_{i+1})$ and $slca(v_{i+2},\ldots,v_k)$. The number of ancestor contained in the SLCA candidate set in the first decomposition condition must not be smaller than that in the second decomposition condition. v_i and v_{i+1} are "closer", so we suppose that the ancestor relationship exists between v_i and v_{i+1} and SLCA is generated by a certain node v in S_2. We have to remove the ancestor node generated right now after calculating SLCA, to obtain the accurate SLCA under the first condition. However, the judgment can be made for removing the ancestor node in the solution process, without waiting for the completion of decomposition and solution. Therefore, Example 2 can be solved again by the grouping strategy. Now, let's refer to Example 3 below.

Example 3: According to Property 4, lca(0.0.0, 0.1.1.0.1)<lca(0.1.1.0.1, 0.1.1.1.0), so "0.0.0" and "0.1.1.0.1" are not in the same group, and S_{11} = {0.0.0}. Moreover, lca(0.1.1.0.1, 0.1.1.1.0)= 0.1.1 and lca(0.1.1.1.0,

Table 1 Basic information of the testing datasets

dataset	The size of dataset	The number of nodes	The number of Keywords
data1	10.7 MB	320,714	19,393
data2	125.2 MB	3,737,432	92,633
data3	168.8 MB	5,954,571	128,824
data4	226.3 MB	6,999,749	1,050,132
data5	247.1 MB	7,389,164	176,431
data6	1.1 GB	35,852,619	3,951,735
data7	1.6 GB	69,427,983	15,082,852

0.1.1.2.1.0)=0.1.1, so "0.1.1.0.1" and "0.1.1.1.0" are in the same group. At last, S_1 is dynamically divided into four groups of changing size. Therefore, SLCAs = *removeAncestor*(*slca*(S_{11},S_2) ∪ *slca*(S_{12}, S_2)∪*slca*(S_{13}, S_2)∪*slca*(S_{14}, S_2)), that is, SLCAs = *removeAncestor*({{0.0}, {0.1.1.2}, {0.1.2}, {0.2.0.0}}). Four SLCA result sets are generated directly without redundant results. "0.1.1.2" in SLCA needs no screening after SLCA candidate set obtained from grouping, so the "ancestor relationship" generated in solution process will not be processed and no more computing resource will be wasted. According to *lemma*1 and *lemma*2 in Ref. [1], the logic complexity of SLCA solving is simplified. The effect is more obvious in the solution process for many keywords.

Based on such four 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 intelligently grouped and each group will be delivered to a Reduce for parallel process. At last, the results are collected and subjected to removing duplicates and removing ancestor.

3.2 Realization of Hadoop-based Algorithm

This section will present the realization of IILE algorithm under Hadoop and MapReduce framework. The algorithm realization may be divided into four major parts as shown in Fig.1, which presents the flow chart of SLCA keyword query of keyword Title, Ben and John in School.xml document tree [1] as shown in Fig. 2.

3.3 Description of Algorithm

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

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

In Algorithm 1, m keywords to be looked up are {w_1, w_2,...,w_i,...,w_m}. We select the keyword to be looked up and the corresponding Dewey code through *SelectMapper* function, and then merge its Dewey code set to save in S_i in *SelectReducer* function. The results are outputted to HDFS and will be used in the next sorting function module.

Algorithm 1. SelectMapReduce

```
1:SelectMapper (key=offset,value=(dewey, keyword)){
2: if keyword= $w_i$ then
3: output (keyword, dewey);
4: end if
5:}
6:SelectReducer (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:}
```

Algorithm 2 arranges the Dewey codes of each keyword in the ascending order.

Algorithm 2. 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:}
```

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

Then, Algorithm 3 realizes the parallel of IILE algorithm.

Algorithm 3. 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: foreach Set in S do
7: result=ileAlgorithm(Set, Ss)[1];
8: if result is not null then
9: output(size of result, result);
10: end if
11: end for
12: if ($i$=1) then
13: IGA($S_1$);
14: end if
15:}
16:SLCAReducer (key=(size of result, result),values){
17: output(key, values);
18:}
```

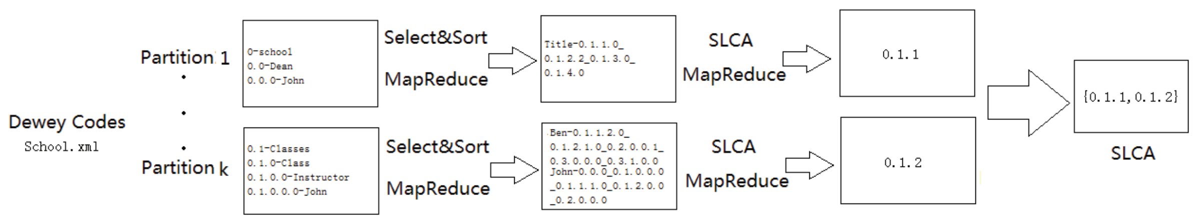


Figure 1 Flow chart of Hadoop-based SLCA keyword query.

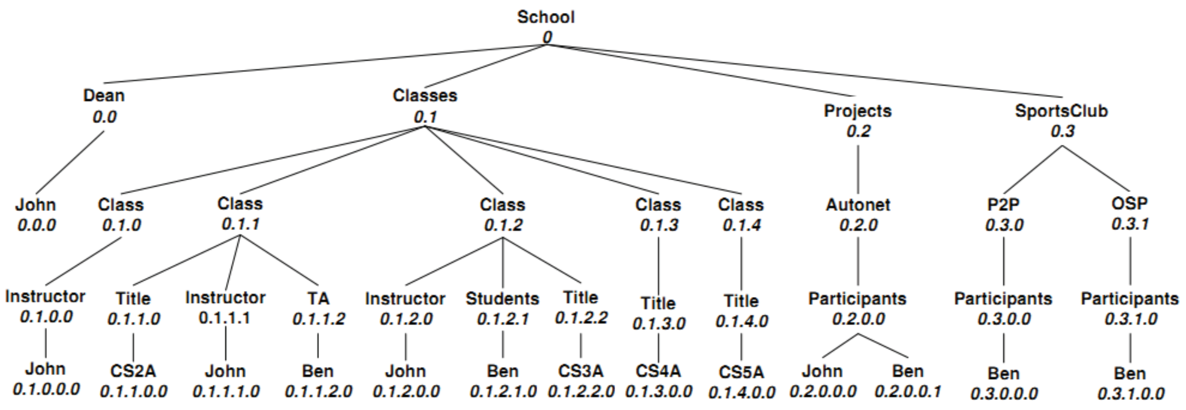


Figure 2 School.xml document tree (each node is identified by Dewey code) [1].

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. S_1 is grouped intelligently by function IGA(S_1) in Line 13 of Algorithm 3, which is implemented as follows.

Sub-function IGA indicates the intelligent grouping process. Lines 3-4 indicate no grouping is required if the quantity of Dewey code is too few. Lines 11-16 indicate if $lca(v_j, v_{j+1})$=$lca(v_{j+1}, v_{j+2})$, then v_j and v_{j+1} are in the same group. Lines 17-23 indicate if $lca(v_j, v_{j+1}) > lca(v_{j+1}, v_{j+2})$, then v_j and $v_{j+1}$are in the same group, and v_{j+1} and $v_{j+2}$are not in the same group. Lines 24–30 indicate if $lca(v_j, v_{j+1}) < lca(v_{j+1}, v_{j+2})$, then v_j and v_{j+1} are not in the same group.

4. EXPERIMENTAL ANALYSIS

The paper designs a series of experiments by combining the several factors of XML keywords query, such as number and frequency of keywords, size of XML dataset and number of parallel cluster nodes. Through the analysis of the experimental results, this section also validates the high efficiency of the XML keyword query based on Hadoop.

In our experiments of the paper, 17 nodes are involved in the cluster. The configuration information and operation environments of all nodes are the same. We have built a system platform using Ubuntu10.12, Hadoop 0.20.2 and Java 1.6.0_21. One of them is used as the master node and the rest are used as slave nodes.

4.1 Experimental Data

4.1.1 Datasets

7 XML datasets are used in the experiments as follows. Dataset 1 is FI_meta.xml, called as data1. Dataset 2 is FR_meta.xml, called as data2. Dataset 3 is ES_meta.xml, called as data3. Dataset 4 is od_bsz-tit_130516_20.xml, called as data4. Dataset 5 is DE_meta.xml, called as data5. Dataset 6 is od-up_bsz-tit_150316_01.xml, called as data6. The above data1, data2, data3 and data5 can be downloaded from https://www.monetdb.org/Downloads. Data4 and data6 can be downloaded from http://swblod.bsz-bw.de/od/. Dataset 7 is dblp.xml, called as data7, which can be downloaded from http://dblp.uni-trier.de/xml/. Table I lists their basic information.

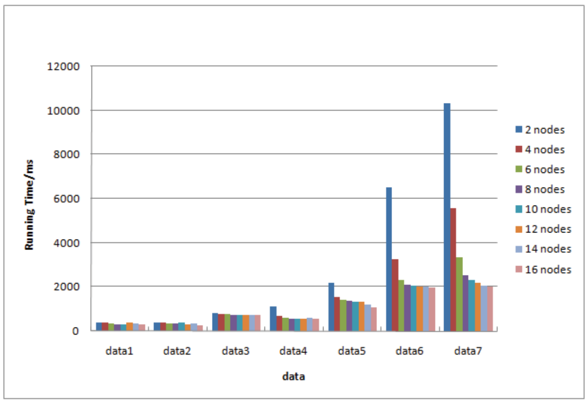
4.2 Basic Information of the Testing Datasets

4.2.1 Queried keyword information

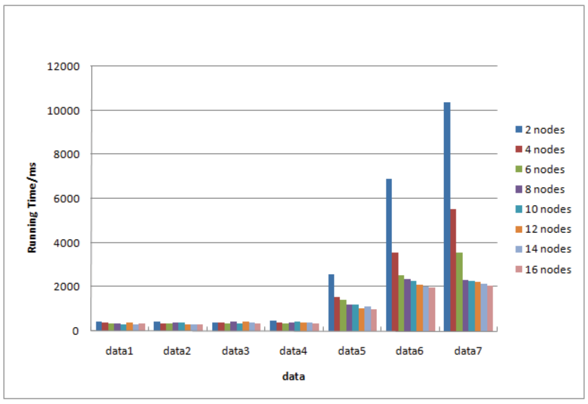
Based on the need of the experiment, 7 sets were selected randomly from each dataset in accordance with the number and frequency to search for the keyword groups. As shown in Table 2, 49 keyword groups were available in total, which were numbered from Q1-Q49. Each group is separated by commas, and the number and frequency of the keywords were separated by "–" [7]. For example, 4-100 means 4 keywords and the frequency of each keyword is 100 (1±10%), i.e. the frequencies of all the keywords that appear 90–110 times are all 100.

Table 2 Information of keyword groups

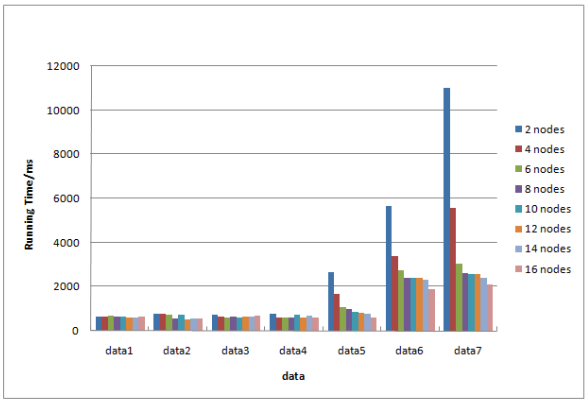
dataset	ID	Keywords	query condition
data1	Q1	1.750, 1988_01_01, 98.105, C6H6	4-10
	Q2	station_local_code, station_name, station_latitude_dms, 97.260	4-100
	Q3	0,1, data_file, data_type	4-1000
	Q4	Mean, maximum, P98, Max	4-5000
	Q5	97.260, station_name	2-100
	Q6	station_local_code, station_name, station_latitude_dms, 97.260, 7, 9, 4.000, sabe_unit_name	8-100
	Q7	station_local_code, station_name, station_latitude_dms, 97.260, 7, 9, 4.000, sabe_unit_name, 101, station_info, 366, urban, station_city, station, population, 3.000	16-100
data2	Q8	90.217, 1988_01_02, MQ, Villeurbanne	4-10
	Q9	Organization, 90.164, 16, person_first_name	4-100
	Q10	Station_city, street_name, population, NO2	4-1000
	Q11	Data_set, UNKNOWN, Max8, 1.000	4-5000
	Q12	90.164, 16	2-100
	Q13	Organization, 90.164, 16, person_first_name, 1999_06_01, organization_city, organization_name, unknown	8-100
	Q14	Organization, 90.164, 16, person_first_name, 1999_06_01, organization_city, organization_name, unknown, 12, FR035A, person_last_name, street_type, person, Temperature, 1976_01_01, organization_address	16-100
data3	Q15	Cu, Fe, 90.205, 2010_08_01	4-10
	Q16	1988_01_01, 34.500, Madrid, CH4	4-100
	Q17	3, station_latitude_dms, station_european_code, station_city	4-1000
	Q18	60,102, Max25, Max26	4-5000
	Q19	34.500, CH4	2-100
	Q20	3, station_latitude_dms, station_european_code, station_city, regional, 1992_05_01, hour, 3.600	8-100
	Q21	3, station_latitude_dms, station_european_code, station_city, regional, 1992_05_01, hour, 3.600, T_VOC, C6H5_C2H5, natural, chemiluminescence, station_distance_to_source, DESCONOCIDO, near city, rural, background	16-100
data4	Q22	C, M, DVD ROM, Growth	4-10
	Q23	Boon, Duncker, Ratgeber, 780	4-100
	Q24	Jpn, spa, lat, trl	4-1000
	Q25	Aut, rvk, prf, pup	4-5000
	Q26	Ratgeber, 780	2-100
	Q27	Boon, Duncker, Ratgeber, 780, UB, Jena, aui, grc	8-100
	Q28	Boon, Duncker, Ratgeber, 780, UB, Jena, aui, grc, fin, dbp, cmm, pdf, Foreign_relations, Film, Autoren, Germany	16-100
data5	Q29	K, 90.209, 1993_03_031, Magdeburg	4-10
	Q30	As, Ni, 1983_07_01, station_description	4-100
	Q31	1988_01_01, NO, Background, SPM	4-1000
	Q32	361, 362, 363, 364	4-5000
	Q33	1983_07_01, station_description	2-100
	Q34	As, Ni, 1983_07_01, station_description, 81.500, 7018, Conductimetry, light_scattering	8-100
	Q35	As, Ni, 1983_07_01, station_description, 81.500, 7018, Conductimetry, light_scattering, DE012A, meadow, permuation_tube, Direct_solar_radiation, month, 2008_12_30, G, 26	16-100
data6	Q36	Set, sankt, UQ_1225, UB_2780	4-10
	Q37	Tests, Tempel, Testmaterial, UY	4-100
	Q38	1957, Stuttgart, DE_21_32a, DE_25_75	4-1000
	Q39	Politologie, adp, ad18, De_1033	4-5000
	Q40	Tests, Tempel	2-100
	Q41	Tests, Tempel, Testmaterial, UY, UF_1500, UF_4000, UVK_Lucius, AF_02000	8-100
	Q42	Tests, Tempel, Testmaterial, UY, UF_1500, UF_4000, UVK_Lucius, AF_02000, AE_55000, Aeschylus, Adressat, BC_1100, BB_1850, BC_2205, Commercial_law, Country_report	16-100
data7	Q43	Elvis C. S. Chen, Emanuele toscano, Embedded Systems, Emily F. Conant	4-10
	Q44	NLPKE, NLPRS, NMA, JFIADSMA	4-100
	Q45	HPDC, HRI, ISM, ISPA	4-1000
	Q46	UAI, ACC, AAAI, AMIA	4-5000
	Q47	NLPKE, NLPRS	2-100
	Q48	NLPKE, NLPRS, NMA, JFIADSMA, ADS, ADCS, SUTC, SwSTE	8-100
	Q49	NLPKE, NLPRS, NMA, JFIADSMA, ADS, ADCS, SUTC, SwSTE, VIP, VRIC, TES, TAPOS, ZEUS, WISES, WAIFI, WEB	16-100



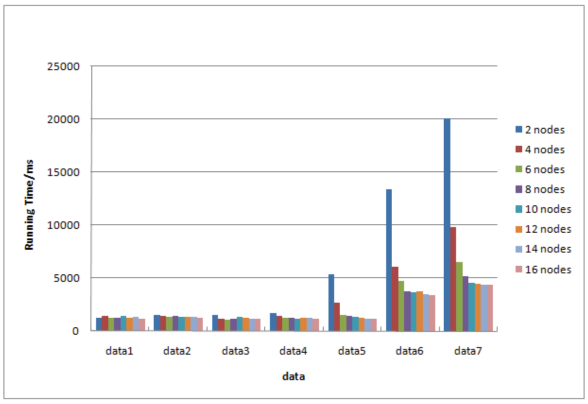
(a) Running status at query condition of 4-10



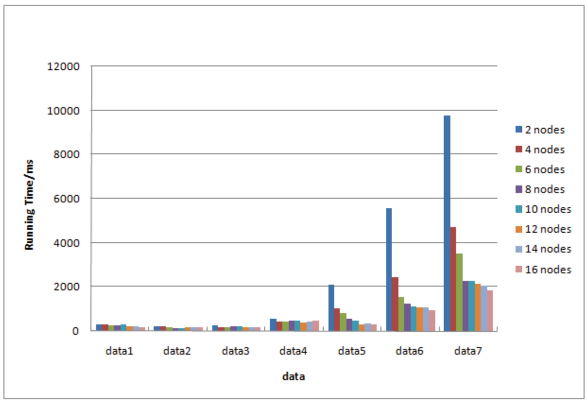
(b) Running status at query condition of 4-100



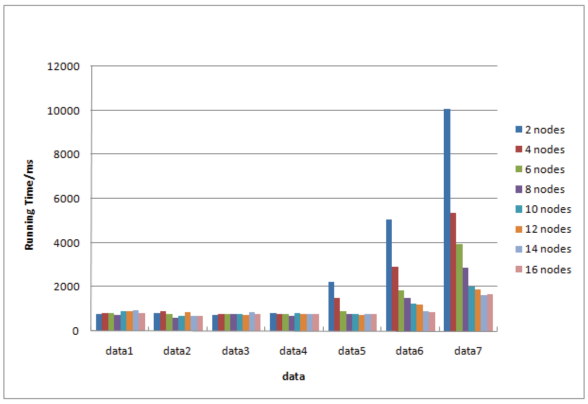
(c) Running status at query condition of 4-1,000



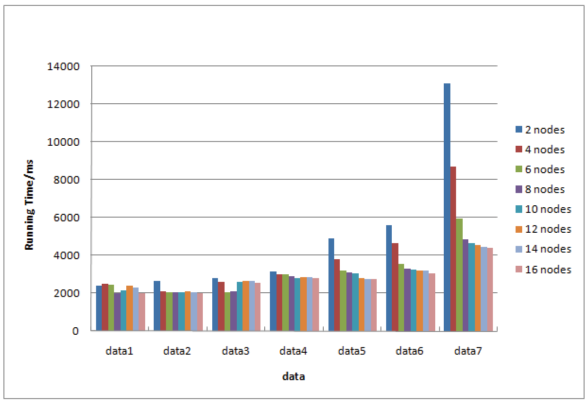
(d) Running status at query condition of 4-5,000



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

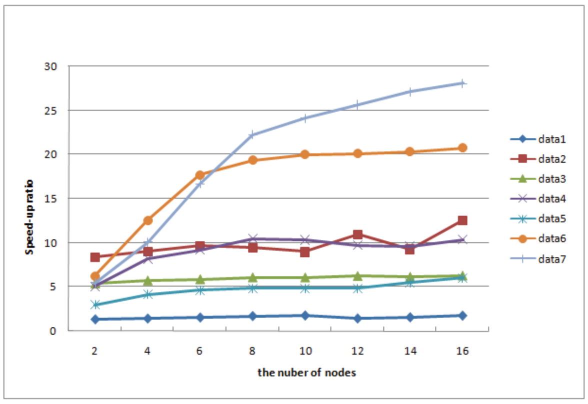


(f) Running status at query condition of 8-100

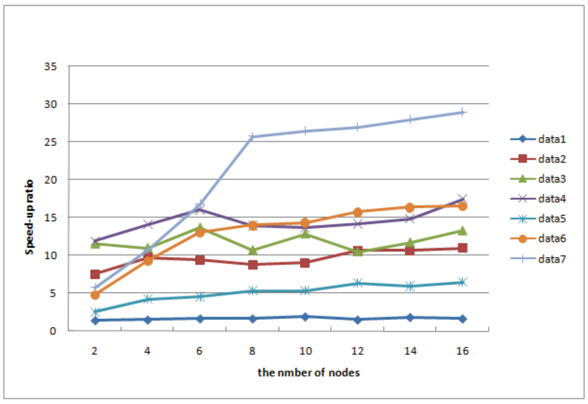


(g) Running status at query condition of 16-100

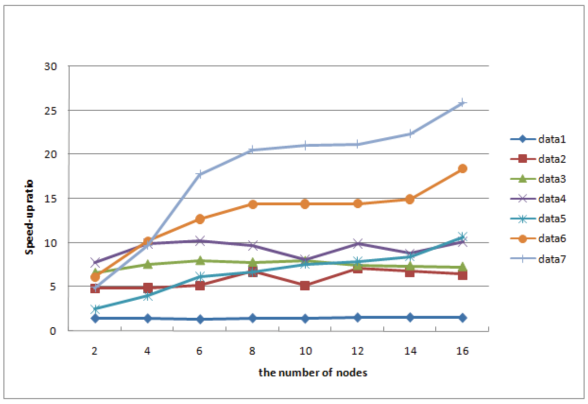
Figure 3 Bar graph of running time.



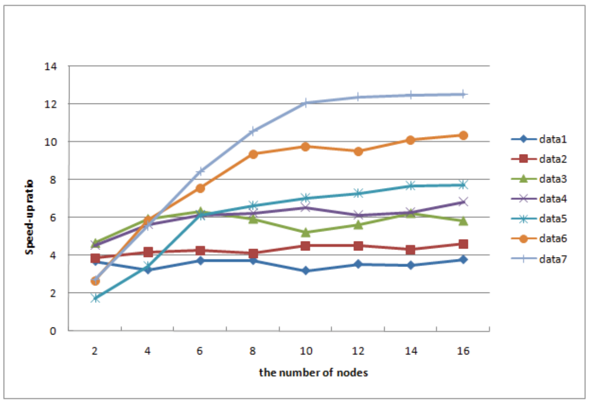
(a) Running status at query condition of 4-10



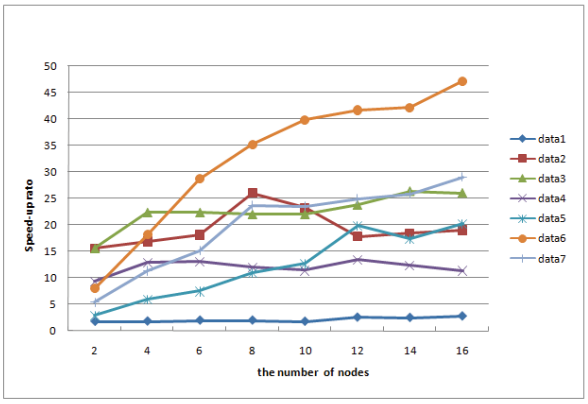
(b) Running status at query condition of 4-100



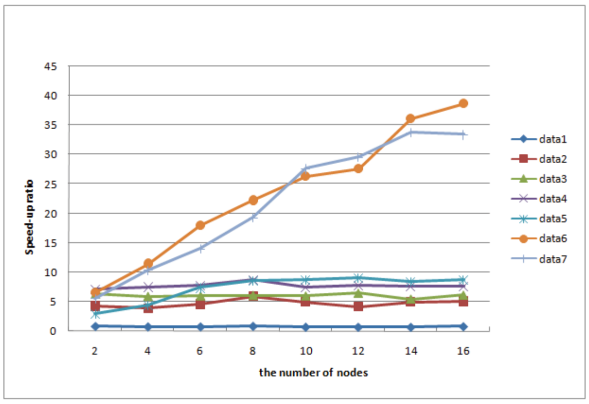
(c) Running status at query condition of 4-1,000



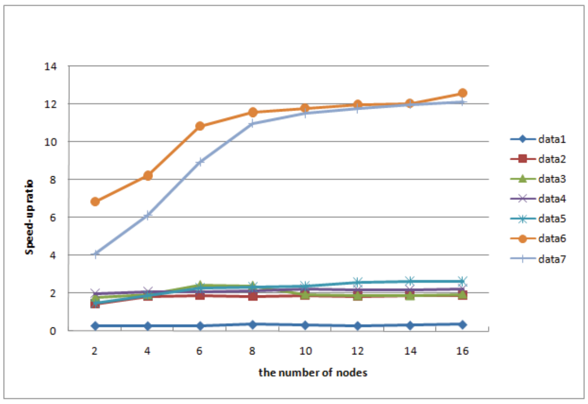
(d) Running status at query condition of 4-5,000



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

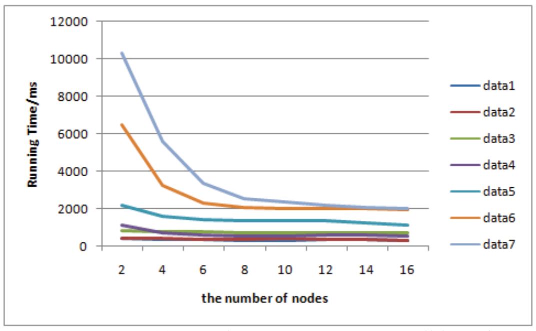


(f) Running status at query condition of 8-100

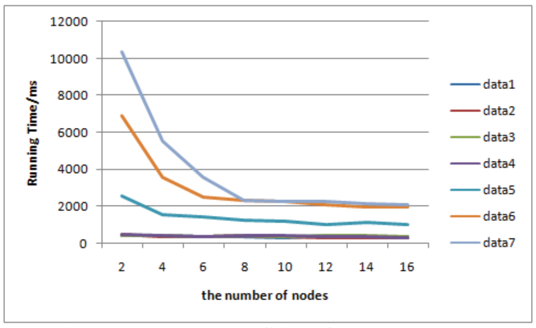


(g) Running status at query condition of 16-100

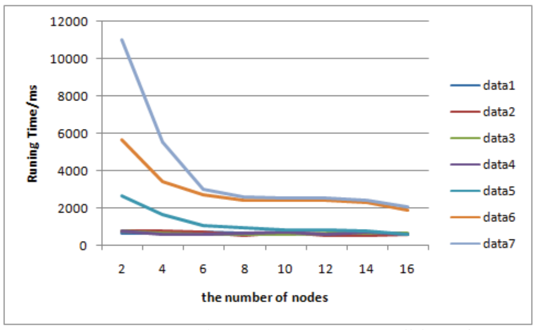
Figure 4 Speed-up ratio curve.



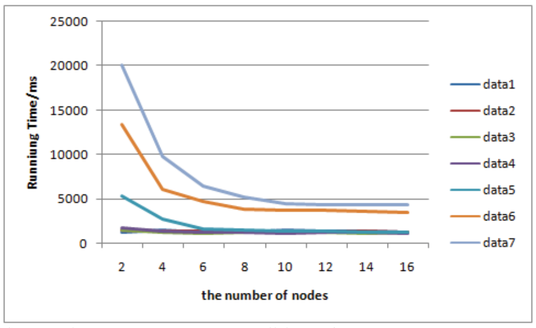
(a) Running status at query condition of 4-10



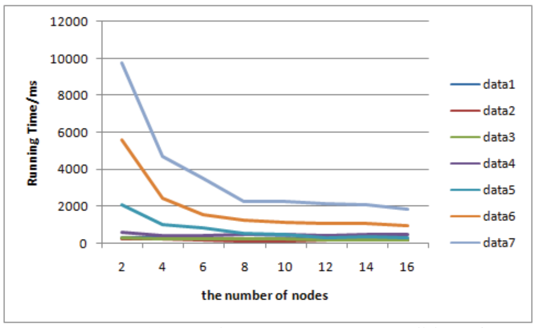
(b) Running status at query condition of 4-100



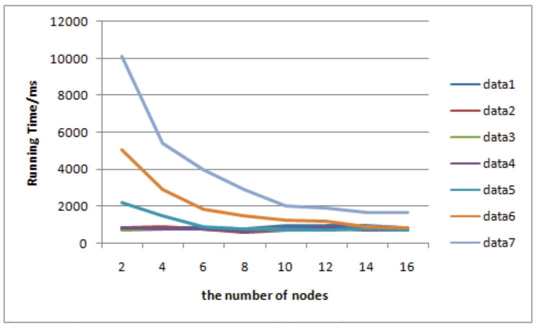
(c) Running status at query condition of 4-1,000



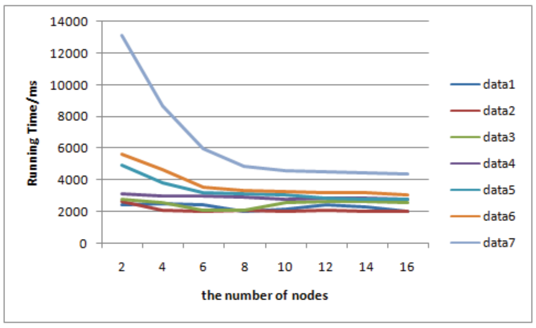
(d) Running status at query condition of 4-5,000



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



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



(g) Running status at query condition of 16-100

Figure 5 Curve chart of the optimal size of nodes in the cluster.

5. INFORMATION OF KEYWORD GROUPS

5.1 Experimental Process

5.1.1 Experiment 1: change the query characteristic and compare the running time

To test the running time of proposed IILE algorithm, we change the size of nodes in the cluster from 2 to 16 in the first experiment. Fig. 3 illustrates corresponding results.

It is obvious that the running efficiency on small-scale dataset is not ideal, such as data1-data4, as shown in Fig. 3. The running time changes not obviously and even more slowly while the size of nodes gradually increases. However, for large-scale datasets, such as data5-data7, it shows obvious advantages. Fig. 3 (e, f, g) indicates that the running time obviously decreases when the size of nodes changes from 2 to 4, while it changes slowly when the size of nodes reaches 8. The query condition has small effect on the running time. For the same dataset, the running time in the query conditions of 4-5000 and 16-100 is longer than that in other query conditions. It indicates that the size and the frequency of

Sub-function. Intelligent Grouping Algorithm (IGA)

Input: S_1

Output: S={*every Set from* S_1 },*Set* is a group ofS_1

```
1: $S$={};
2: $Set$={};
3: if ($|S_1| <= 2$) then
4: {$Set$=$S_1$; $S = S \cup \{Set\}$;}
5: else
6: {Flag=true;
7: for ($j$=1 to $|S_1| - 2$) do
8: if (Flag) then
9: {$Set$=$Set \cup \{v_j\}$; Flag=false;}
10: end if
11: if (lca($v_j$, $v_{j+1}$)=lca($v_{j+1}$, $v_{j+2}$)) then
12: {$Set$=$Set \cup \{v_{j+1}\}$;
13: if ($j = |S_1| - 2$) then
14: {$Set$=$Set \cup \{v_{j+2}\}$; $S = S \cup \{Set\}$;}
15: end if
16: }
17: else if (lca($v_j$, $v_{j+1}$) >lca($v_{j+1}$, $v_{j+2}$)) then
18: {$Set$=$Set \cup \{v_{j+1}\}$; $S = S \cup \{Set\}$; $Set$={};
19:if ($j = |S_1| - 2$) then
20: $Set$=$Set \cup \{v_{j+2}\}$;
21: end if
22: $j$++; Flag=true;
23: }
24: else if (lca($v_j$,$v_{j+1}$) <lca($v_{j+1}$, $v_{j+2}$)) then
25: {$S = S \cup \{Set\}$; $Set$={};
26: if ($j$=$|S_1| - 2$) then
27: {$Set$=$Set \cup \{v_{j+1}\}$,$Set$=$Set \cup \{v_{j+2}\}$; $S = S \cup \{Set\}$;}
28: end if
29: Flag=true;
30: }
31: end if
32: end for
33: }
34:end if
35:return $S$
```

keyword will affect the running time of IILE algorithm. The experimental results have verified that IILE algorithm is more suitable for large-scale dataset, which makes it possible to query massive XML keywords.

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

We have introduced the speed-up ratio to analyze the experimental results and measure the performance and efficiency of parallel system. The speed-up ratio is defined as *speed-up ratio = Running time on single machine/Running time on clusters*.

From the speed-up ratio curve in Fig. 4, we can clearly see that, the speed-up ratio shows a tendency of increase as the size of nodes increases. However, it is not obvious in the case of small-scale datasets. We can also see that after 8 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 dataset, 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.

5.1.3 Experiment 3: change the size of cluster nodes and compare the optimal number of nodes

Fig. 5 shows the curve chart of the optimal size of nodes in the cluster. We can see from Fig. 5 that, for a specific sized dataset, the running efficiency will increase when the size of nodes increases. But it is not the more the better. As shown in Fig. 5, we can see that for the same data file, the reducing tendency of the running time is slowing down as the size of nodes increases. This is determined by the block mechanism of Hadoop itself whose default data block size is 64 MB, just like the data of 128 MB divided into 2 data blocks and the data of 256MB divided into 4 data blocks. The figure shows that data7 has the best efficiency in the cluster with 8 nodes, while the reducing tendency of running time tends to be gentle if the number of nodes increases.

6. CONCLUSION AND FUTURE WORK

In this paper, parallelization of MapReduce-based XML keyword search algorithm is further researched to process massive XML data. The grouping-based IILE algorithm is proposed and realized, and then the parallel is carried out by means of Hadoop. The experimental results show that our proposed algorithm can process SLCA-based keyword search of large-scale XML data.

However, some gaps still remain in this paper: (1) the algorithm does not achieve the complete separation of relationship between groups and some ancestor relationships still exist. The reason for these problems may be further analyzed or studied in the next phase. (2) Limited by unavailable Hadoop iteration and other factors, these functional modules shall be written by block. The previous research was conducted from the perspective of repeated starting of Hadoop program. (3) Parallelization of XML Dewey code has not yet been implemented in place. In light of these, emphasis will be placed on these gaps to study efficient algorithm for keyword search and applications of the same in cloud computing environment.

Acknowledgements

This work is partly supported by National Natural Science Foundation of China (No. 61373127).

REFERENCES

1. Y. Xu and Y. Papakonstantinou, "Efficient keyword search for smallest LCAs in XML databases," Proceedings of SIGMOD, 2005, pp. 537–538.
2. 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, pp. 1043–1052.
3. G. L. L, 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 Information and Knowledge Management, ACM Press, 2007, pp. 31–40.
4. Z. Liu Z and Y. Chen, "Identifying meaning return information for XML keyword search," Proceedings of SIGMOD, 2007, pp. 329–340.
5. Y. Xu and Y. Papakonstantinou, "Efficient LCA based keyword search in XML data," Proceedings of EDBT, 2008, pp. 535–546.
6. J. Li, C. Liu, R. Zhou, and J. Yu, "Quasi-SLCA based keyword query processing over probabilistic XML data," IEEE Transactions on Knowledge and Data Engineering, 2014, 26(4): 957–969.
7. 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.
8. 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, pp. 689–700.
9. 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," Proceedings of IEEE 28th International Conference on Data Engineering (ICDE), 2012, pp. 905–916.
10. Y. Zhao, Y. Yuan, and G. Wang, "Keyword search over probabilistic XML documents based on node classification," Mathematical Problems in Engineering, 2015, Article ID 210961, 11 pages.
11. A. Dimitriou, D. Theodoratos, and T. Sellis, "Top-k-size keyword search on tree structured data," Information Systems, 2015, 47: 178–193.
12. N. Bidoit, D. Colazzo, N. Malla, F. Ulliana, M. Nolé, and C. Sartiani, "Processing XML queries and updates on Map/Reduce clusters," Proceedings of the 16th International Conference on Extending Database Technology (EDBT), 2013, pp. 745-748.
13. Y. Zhang, Q. Li, and B. Liu, "MapReduce implementation of XML keyword search algorithm," Proceedings of International Conference on Big Data Intelligence and Computing, 2015, pp. 721-728.
14. J. Camacho-Rodríguez, D. Colazzo, and I. Manolescu, "PAXQuery: efficient parallel processing of complex XQuery," IEEE Transactions on Knowledge and Data Engineering, 2015, 27(7): 1977-1991.
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. Zhang, Q. Ma, X. Wang, and A. Zhou, "Distributed SLCA-based XML keyword search by Map-Reduce," Proceedings of the 15th International conference on Database systems for Advanced Applications (DASFAA'10), Springer Berlin Heidelberg, 2010, pp. 386–397.
17. Y. Shen and L. Feng, "Evaluation of XPath queries with predicates: an Eulerian cycle theory based sequencing approach," International Journal of Computer Systems Science & Engineering, 2011, 26(4): 241–257.
18. S. Böttcher, R. Hartel, and J. Rabe, "Efficient XML keyword search based on DAG-compression," Proceedings of DEXA, Springer International Publishing, 2014, pp. 122–137.
19. M. Mataoui and M. Mezghiche, "A distance based approach for link analysis in XML information retrieval," International Journal of Computer Systems Science & Engineering, 2015, 30(3): 173–183.
20. R. R. Lin, Y. H. Chang, and K. M. Chao, "Locating Valid SLCAs for XML keyword search with NOT semantics," ACM SIGMOD Record, 2014, 43(2): 29–34.
21. X. Zheng, J. Li, Y. Zhang, and Q. Liu, "An optimization model of Hadoop cluster performance prediction based on Markov process," International Journal of Computer Systems Science & Engineering, 2016, 31(2): 127–136.