A Scalable XSLT Processing Framework based on MapReduce

Ren Li
College of Computer Science, Chongqing University, Chongqing 400044, China
Email: renli@cqu.edu.cn

Jianhua Luo, Dan Yang, Haibo Hu and Ling Chen
School of Software Engineering, Chongqing University, Chongqing 400044, China
Email: {jianhua.luo, dyang and hbhu}@cqu.edu.cn, rainylily22@163.com

***Abstract*—The eXtensible Stylesheet Language Transformation (XSLT) is a de-facto standard for XML data transforming and extracting. Efficient processing of large amounts of XML data brings challenges to conventional XSLT processors, which are designed to run in a single machine context. To solve these data-intensive problems, MapReduce paradigm in the cloud computing domain has received a comprehensive attention in both academia and IT industry recently. In this paper, a novel MapReduce-based XSLT distributed processing framework named CloudXSLT is proposed to implement efficient and scalable XML data transforming. First, the architecture of CloudXSLT framework is outlined. Subsequently, several XML data and XSLT rule representation models which are suitable for MapReduce paradigm are defined, and several MapReduce-based XSLT distributed processing algorithms are proposed. Finally, an experiment on a simulation environment with real XML datasets shows our framework is more efficient and scalable than conventional XSLT processors when processing large size of XML data.**

***Index Terms*—XML transformation, XSLT, MapReduce, cloud computing**

I. Introduction

Until now, the eXtensible Markup Language (XML) is widely used in various domains, such as Web Services and Semantic Web, and it has become a de-facto standard for data exchange and representation [1]. To implement automatic XML data transforming and extracting, the eXtensible Stylesheet Language Transformation (XSLT) technology is proposed and recommended by the W3C [2]. However, the increasing size of XML data brings challenges to these conventional XSLT processors, such as SAXON and Xalan-Java, which are designed to run in a single machine context. These tools load the entire DOM or DOM-like models in memory where the size of memory can be larger than that of the original XML file [14]. Therefore, the memory space will be exhausted with the growing size of XML data, and the transformation will be failed unavoidably. Considering all above, an XSLT processing framework with high-performance computing capability and scalability is in demand.

Recent years, the cloud computing technologies have received a comprehensive attention in both IT industry and academia recently [3]. MapReduce [4], which is first proposed by Google in 2004, has become the dominant distributed parallel programming paradigm to solve the data-intensive problems in the cloud computing domain. MapReduce [21] paradigm is built on the top of the Goolge File System (GFS) [5]. It processes the data with the form of key-value pairs in a cluster of commodity machines. There are three different types of computing nodes in MapReduce named the master, map and reduce node, respectively [6]. Among them, the master node takes charges of the partition and locality of data, fault tolerance, schedule of tasks, and management of process communications, etc. The map nodes accept the data chunks from the master and generate a set of key-value pairs intermediate output according to the user-defined map function. All the intermediate data with the same key are merged and processed in one reduce node according to the user-defined reduce function [7]. Developers just need to define the key-value pairs based data models and implement the computing logic in a map function, a partition function, a combine function and a reduce function. Currently, Apache Hadoop platform is the most popular open source MapReduce implementation. Within this platform, the GFS has been implemented as the Hadoop Distributed File System (HDFS), which is a scalable and reliable data-storage system for storing large amounts of file in a distributed file environment [8].

Several research literatures about utilizing MapReduce to solve XML [22] related data-intensive problems have been proposed. For example, paper [9] proposed a MapReduce based framework to implement XML structural similarity searching on large clusters. In order to analysis large-scale of XML data in a MapReduce environment, a novel query language called MRQL was presented in paper [10]. To analyze the large amounts of XML data in science workflow, an approach for exploiting data parallelism in XML processing pipelines through novel compilation strategies within the MapReduce framework was presented in paper [11]. However, to the best of our knowledge, an efficient and scalable framework, which combines the MapReduce and

© 2013 ACADEMY PUBLISHER
doi:10.4304/jcp.8.9.2175-2181

XSLT technologies to implement large-scale XML data transformation, is still in blank [15].

In this paper, we propose a novel MapReduce-based XSLT parallel processing framework named CloudXSLT to implement efficient and scalable XML data transformation. Instead of using streaming processing solution, the distributed processing approach is adopted not only because the former one can't handle special XML [16], but also cloud computing is widely used nowadays and processing on a cluster of machines in parallel can achieve better performance and scalability.

The reminder of this paper is organized as follows. In section 2, the architecture and workflow of CloudXSLT framework will be presented. In section 3, several XML and XSLT representation models suitable for MapReduce paradigm will be defined. Some MapReduce-based XSLT parallel processing algorithms will be given in section 4, followed with the experimental evaluations in section 5. The conclusions come out in section 6.

II. The Architecture and Workflow of CloudXSLT

In this section, we first describe the architecture of the framework, and then introduce the workflow in detail.

Based on the MapReduce and HDFS technologies in Hadoop platform, a novel XSLT distributed processing framework named CloudXSLT is designed as shown in Fig. 1. The entire framework consists of two logic layers:

1. The Parallel Data Processing Layer. It is the core component of the framework and consists of the Central Control Module (CCM), the XSLT Rule Parsing Module (XRPM), the XML Parsing Module (XPM) and the XSLT Parallel Processing Module (XPPM). The first three sub-modules are designed to run in the master node. The CCM is responsible for monitoring the workflow. The XRPM and XPM are used to convert the XSLT file and XML data into defined data models. The XPPM works in a cluster of map and reduce nodes and produces the results fragments.
2. The Distributed Data Storage Layer. We utilize the HDFS as the repository to support distributed storage of large-scaled XML data. HDFS creates multiple replicas of data blocks and distributes them on computing nodes throughout network to enable reliable and rapid computations.

As shown in Fig. 2, the workflow of the proposed framework consists of three steps:

1. First, in the preprocessing phase, the XSLT stylesheet file is parsed into MR-XSLT models (defined in section 3) in the XRPM. And then, the CCM processes on the root node of each XML data file and calls the XPM to convert the corresponding XML nodes into the MR-XML models before uploading them into HDFS via an input stream.
2. Second, the XPPM takes the MR-XML models from HDFS as input unit, and transforms them into the MR-XML-Output models in parallel, according to the map and reduce algorithms.
3. At last, the CCM outputs the final result files by assembling the MR-XML-Output models .

Before discussing the MapReduce-based parallel XSLT processing algorithms in detail, we will define the fundamental data representation models mentioned above in the following section 3.

III. The Fundamental Data Models

As discussed, the key-value pairs form the basic data structure in MapReduce. Hence, the tree structure-based XML and XSLT files should be parsed and converted into a novel data model which is suitable for the MapReduce paradigm. In this section, we first propose the XSLT and XML data models named MR-XSLT and MR-XML, respectively.

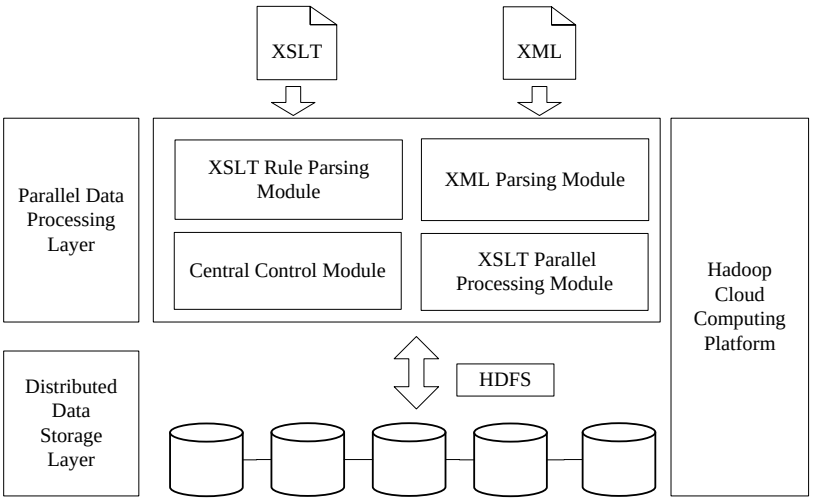

Figure 1. Architecture of the CloudXSLT Framework

© 2013 ACADEMY PUBLISHER

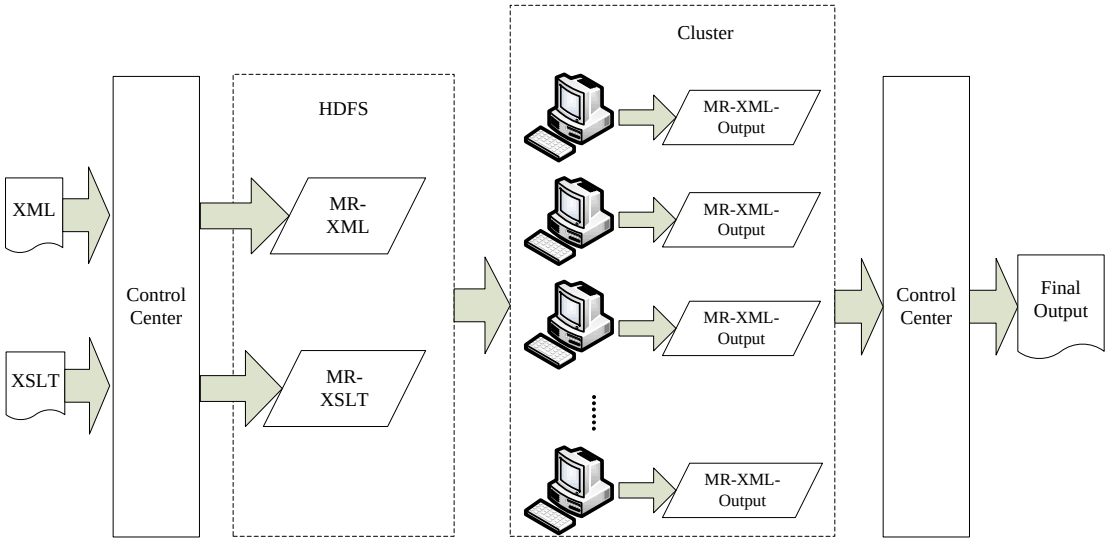

Figure 2. Workflow of the CloudXSLT Framework

A. MapReduce based XSLT Rule Model

As mentioned in the first step of the workflow, in the XRPM, each XSLT template rule in the stylesheet file will be converted into the MR-XSLT model, according to the Def. 1.

Definition 1 (MR-XSLT). Each XSLT template rule in an XSLT stylesheet file is defined as a MR-XSLT model, following the form of: MR-XSLT = < *FN*, *ID*, *TTag*, *CommList* >.

Within the MR-XSLT model, the first parameter *FN* is used to specify the source XSLT file name. Parameter *ID* denotes the sequence number of the template rule in *FN* based on the depth-first strategy; *TTag* records the value of Xpath pattern, which will be used to match the target XML nodes. Especially, the parameter *CommList* stores the template command of this template rule, and it consists of some MR-XSLT-Comd model, as showed in Def. 2.

Definition 2 (MR-XSLT-Comd). MR-XSLT-Comd = <*CID*, *Op*, *TLoc*, *Cont*> is a model to present each XSLT template command in an XSLT template rule.

CID is the sequence number of current command in a XSLT template rule. We use the parameter *Op* as the symbol of command, like Insert, Replace and Foreach. *TLoc* denotes the target location of current command to be executed. The parameter *Cont* records the new content of the corresponding *Op*. The mapping relationship between MR-XSLT-Comd model and some common used XSLT commands are summarized in Table.1.

B. MapReduce based XML Data Model

Generally, the XML data [23] to be processed often consist of a number of XML documents with different size, and each document can be modeled as rooted ordered trees since it has a root node and some nested child nodes. Rather than the root node, the secondary nodes are more important to us, so for the secondary XML node and its child nodes, the XSLT template commands are executed to generate new contents. Furthermore, although some simple data models and APIs about XML data splitting and parallel computing has been provided by MapReduce framework, they cannot be easily applied to the special situation of XSLT based XML data transforming. Therefore, we propose a MR-XML model to represent the tree structure of an XML fragment in the form of key-value pair, as shown in Def. 3.

Definition 3 (MR-XML). For a given XML node *R* and its child nodes (C_1,C_2, …, C_N) in an XML file *F*, the key-value pair based MR-XML model is used to represent the tree structure of this XML fragment, which follows the form of: MR-XML = < <*FN*, *ID*> , <*Tag*, *AL*, *DL*, *CList*> >.

Among the parameters of this key-value pair model, the key is <*FN*, *ID*>, <*Tag*, *AL*, *DL*, *CList*> denotes the corresponding value. We use *FN* to represent the file name of *F*; *ID* is the sequence number of node *R* in *F* follows the depth-first strategy; *Tag* denotes the tag name of *R*. *AL* is a list of attributes of *R*, and each attribute is

TABLE1
MAPPING RELATIONSHIP BETWEEN MR-XSLT-COMD AND XSLT TEMPLATE COMMAND

Operator	TargetLocation	Content	XSLT Template Command	Example
Insert	Local	Command data	Null	<newtext>text</newtext>
Get	Value of select attribute	null	xsl:value-of	<xsl:value-of select=”.”/>
Loop	Value of select attribute	null	xsl:for-each	<xsl:for-each select=”a”>
Call	Value of select attribute	null	xsl:apply-templates	<xsl:apply-template select=”*”>
Insert	Local	Command data	xsl:text	<xsl:text>text</text>

© 2013 ACADEMY PUBLISHER

defined as a key-value pair <*AttributeName*, *Value*>.

DL is a string collection which records the data content of *R*. The parameter *CList* denotes a collection of child nodes (C_1,C_2, …, C_N) of *R* and *CN* is represented as a MR-XML-Node model, as defined in Def.4.

Definition 4 (MR-XML-Node). One XML child node *C* in a MR-XML model *M* is defined as MR-XML-Node = < *NodeID*, *Tag*, *AL*, *DL*, *FatherID*, *CList*>.

In the MR-XML-Node model, *NodeID* denotes the unique sequence number of node *C* in *M* following the depth-first strategy; Tag records the tag name of *C*. *AL* is a collection of key-value pairs for recording the attributes of *C*; *DL* is a string array to record the data content. *FatherID* points to the father XML node of current node *C*. *CList* is a collection of its child nodes with the same definition.

After the XML nodes being transformed in the map and reduce phase, a new XML node representation model, as defined in Def. 5, is adopted to represent the output result.

Definition 5 (MR-XML-Output). The model MR-XML-Output=<*F*, *ID*, *NewContent*> represents one XML node in the output files. *F* is the output file name; *ID* is the sequence number of the MR-XSLT to identify the position where this result fragment should be placed. *NewContent* is the output content.

IV. MapReduce based XSLT Parallel Processing Algorithms

Given a XSLT stylesheet file, the XRPM is called first to convert XSLT into a collection of MR-XSLT models before uploading them to HDFS and waiting for being fetched by the computing nodes, the pseudo-code is shown in Alg.1.

Algorithm 1. Convert_XSLT (*F*)

Input: *F*, Output: *Output_XSLT*.

F is a source XSLT stylesheet file; *TR* is a template rule in *F* with the tag "<xsl:template>"; *MX* is a MR-XSLT model; *Cmd* is a template command node in *TR*; *MXC* is a MR-XSLT-Comd model;

Output_ XSLT is an output text file consists of a collection of MR-XSLT models.

```
1  seq_num=1
2  Foreach(TR in F)
3     Initialize a MX model and assign it with the attributes of TR;
4     Foreach(Cmd in TR)
5        Initialize a MXC model and assign it with the attributes of Cmd;
6        MX.Insert(MXC);
7     EndFor
8     Output_XSLT.Add(MX);
9     seq_num++;
10 EndFor
11 Upload the Output_XSLT to HDFS.
```

In Alg.1, a MR-XSLT model *MX* is created firstly for every template rule node of XSLT files, before assigning the sequence number, the file name and the match attribute to it. And then, for each template command node of current template rule, a MR-XSLT-Comd model *MXC* is initialized and assigned with the parameter Operator, TargetLocation and Content based on the mapping rules in Table.1. In line 6 of Alg.1, the new command model is inserted into the CommandList of MR-XSLT. At last, all MR-XSLT models are outputted to the Output_XSLT text file, and then this file will be uploaded to HDFS to wait for being fetched by every computing node of the framework via a data stream.

After the XSLT file being converted, the CCM parses the corresponding XML data nodes into MR-XML models by using a stream based on the XML parsing technology. The pseudo-code executed in the Central Control Module is shown in Alg.2.

Algorithm 2. Central_Control (*X*)

Input: *X*, Output: *Output_XML*.

X is the collection of source XML data files; *x* is one XML file; *N* is a node in XML; *MX* is a MR-XML model; *MXN* is a MR-XML-Node model; *Output_XML* is an output text file consists of a collection of MR-XML models.

```
1  Foreach(x in X)
2     Foreach(N in x)
3        if(N is the secondary layer node in x)
4           Initialize a MX and assign it with the attributes of N;
5           Foreach(child in N)
6              Initialize a MXN and assign it with the attributes of Cmd;
7              MX.AddChild(MXN);
8           EndFor
9        Output_XML.Add(MX);
10       EndIf
11    EndFor
12 EndFor
13 Upload the Output_XML to HDFS.
```

At the beginning of Alg. 2, whether the node is a secondary layer node in a XML document is checked. For the matched node, current node is converted into one MR-XML model, and its children nodes will be added in the form of MR-XML-Node model, correspondingly. An Output_XML model is used to collect all the MR-XML models and uploaded to HDFS finally as the input of MapReduce program.

And then, the XDPM is called to transform these input models into MR-XML-Output models in parallel. The pseudo-code of the processing algorithms are shown in Alg.3 and Alg. 4, respectively.

Within each job, the map function takes a batch of MR-XML models as input. And if the matched MR-XSLT model exists, the intermediate key-value pairs are constructed with the key as a combination with the file name of MR-XML and the ID of MR-XSLT, the value is defined as shown in Line 6 of Alg.3. Otherwise, nothing will be exported as the XSLT rules can filter the context. At last, all the intermediate key-value pairs will be sorted in the MapReduce framework, and values have the same key are emitted to a reduce function.

© 2013 ACADEMY PUBLISHER

The reduce function receives the intermediate key-value pairs have been sorted by key with the key as the input unit. And then, a MR-XML-Output model is created and the collection of values is assembled as the content, before it is outputted to the result files in HDFS. At last, the CCM combines those output fragments of Alg.4 to the final output XML files according to both the XSLT definitions and the attributes-F and ID-in the MR-XML-Output, because MR-XML-Output has recorded the mark of locations connected with XSLT.

Algorithm 3. Map(*MR-XML*)

Input: *MR-XML*; Output: intermediary <key, value> pairs.

MR-XML is an input <key, value> pair in HDFS; *MX* is a MR-XSLT model.

```
1  Foreach(MR-XML)
2    Foreach(MR-XSLT)
3      If (Tag in MR-XML equals TargetLocaion in MR-XSLT )
4        key = <FN in MR-XML, ID in MX>;
5        Foreach (MR-XSLT-Commd in MR-XSLT)
6          NewContent is accumulated;
7        EndFor
8        value = NewContent;
9      EndIf
10   EndFor
11   EMIT(key, value);
12 EndFor
```

Algorithm 4. Reduce(key, value)

Input: *key*, *value*; Output: *Output_XML*.

key and *value* is the result of map function; *Output_XSLT* is the output text file based on the MR-XML-Output model.

```
1  Foreach (key)
2    Initialize a MR-XML-Output model Output_XSLT;
3    Output_XSLT.setF(FN in key);
4    Output_XSLT.setID(ID in key);
5    Output_XSLT.setContent(the combine of all values with this key);
6    OutputToHDFS(Output_XSLT);
7  EndFor
```

V. EXPERIMENTS

A. Experimental Setup

To simulate the computing performance and scalability, we have constructed a prototype CloudXSLT framework in a cloud computing environment which consists of nine 100M/s Ethernet connected commodity machines. The master node, with 2 cores of Intel Pentium 4 CPU, 1.5 GB of main memory, and 80 GB of hard disk space, is configured as the Central Control node. While the other 8 machines are working as computing nodes, each has 2 cores of CPU, 1.5 GB of main memory and 80 GB of hard disk space each. The operating system Ubuntu 12.04 and Hadoop 1.0.3 platform are configured in each machine.

To compare with the prototype framework, two well-known XSLT processors, SAXON 9.4 and Xalan-Java 2.7.1 are deployed on a powerful single machine with Intel i5 2.50 GHz dual core processor, 8 GB main memory, and 4 TB disk space.

DBLP, which use XML format to store bibliographic information on major computer science journals and proceedings, is chosen as the benchmark dataset [12]. To make our experiments convincible, a group of datasets with different sizes of 50MB, 100MB, 200MB, 400MB and 800MB are allocated. However, as there are no standard XSLT stylesheet files to be tested, a self-defined XSLT stylesheet file which contain some common used XSLT commands, such as <xsl:template>, <xsl:apply-template> and <xsl:value-of>, is defined to transform DBLP XML data mentioned above. By using this XSLT, a DBLP XML document will be transformed into another schema: for each conference paper, a HTML table row is generated, listing the paper's key attribute, followed by the author and the title of the paper. The details of our self-defined XSLT are given in Fig. 3. This rule extracts the specified part of information, and then presents them is a different kind of way from the source document.

B. Experimental Results

In this section, we report the experimental results. For evaluation purposes, two experiments are conducted. The first one is a scalability test on CloudXSLT, while the second is a comparative study between those processors mentioned in previously.

In the first group, we investigate the scalability of CloudXSLT by transforming large XML documents range from 50MB to 800MB. CloudXSLT being as a parallel processing framework considers the number of machine as one of the main factors, which affects the performance, so different numbers of computing nodes are tested to observe the changes in performance. Fig. 4 gives the corresponding results, and at least three results can be concluded.

1. First, two main factors affect the performance and response times: the size of the test data and the number of computing nodes.
2. Second, when processing a same size of DBLP data, the more computing nodes participate in, the CloudXSLT runs faster, especially when the number of node increases from 2 to 4. However, the run times couldn't decrease continuously for the resource consumption in the framework.
3. At last, when processing various sizes of DBLP data in a specified number of nodes, the response time rises with the growing size of the XML data, obviously. What is more important is that the speed of the increase of response time is slower than the speed of the increase of data. That is to say, CloudXSLT has much more superiority when processing large-scaled of XML data.

In the second experiment, to compare the performance with our framework, the conventional single-machine-based XSLT processors SAXON and Xalan-Java are tested by processing the same datasets and XSLT rules. Fig. 5 illustrates the experimental results, where response time is in seconds, and 4999s denotes out-of-memory errors.

© 2013 ACADEMY PUBLISHER

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html" indent="yes" />
    <xsl:template  match="/">
      <html>
        <body>
          <table>
            <xsl:apply-templates select="/dblp/mastersthesis"> </xsl:apply-templates>
            <xsl:apply-templates select="/dblp/article"> </xsl:apply-templates>
            <xsl:apply-templates select="/dblp/incollection"> </xsl:apply-templates>
            <xsl:apply-templates select="/dblp/www"> </xsl:apply-templates>
            <xsl:apply-templates select="/dblp/inproceedings"> </xsl:apply-templates>
            <xsl:apply-templates select="/dblp/proceedings"> </xsl:apply-templates>
          </table>
        </body>
      </html>
    </xsl:template>
    <xsl:template match="incollection">
      <tr>
        <td>
          <xsl:value-of select="@key"></xsl:value-of>
          <xsl:value-of select="author"></xsl:value-of>
          <xsl:value-of select="title"></xsl:value-of>
        </td>
      </tr>
    </xsl:template>
    <xsl:template match="mastersthesis">
      <tr>
        <td>
          <xsl:value-of select="@key"></xsl:value-of>
          <xsl:value-of select="author"></xsl:value-of>
          <xsl:value-of select="title"></xsl:value-of>
        </td>
      </tr>
    </xsl:template>
      ……
</xsl:stylesheet>
```

Figure 3. Part of self-defined XSLT

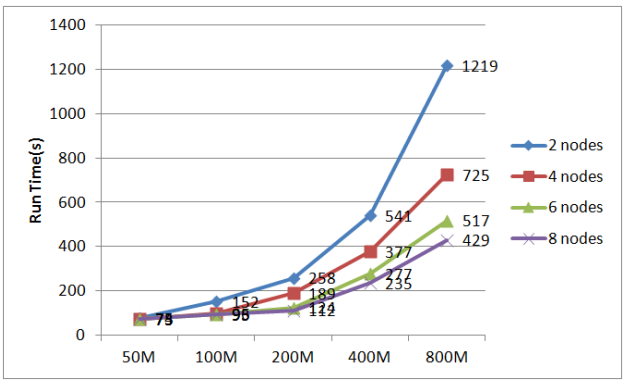

Figure 4. Experimental Results of CloudXSLT with Additional Compute Nodes

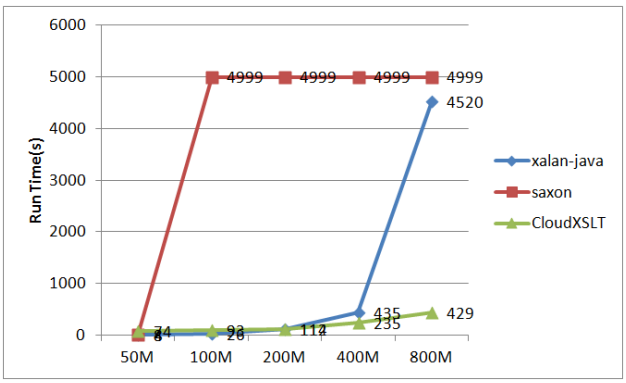

Figure 5. Experimental Results-Comparison of CloudXSLT, SAXON and Xalan-Java

SAXON shows a higher performance when dealing with small size of DBLP data for 50MB, where only 4 seconds are needed. However, when the size of input XML data increased to 100MB and larger, out-of-memory errors are occurred. Therefore, we can conclude that there is a memory limitation for SAXON since all the input data are parsed and loaded into memory, and large-scale XML data cannot be processed.

Xalan-Java achieves better performance than SAXON in processing larger XML data, because it can handle all sizes of datasets. When the size of dataset is 50MB, Xalan-Java takes 8 seconds which is slower than SAXON but faster than our framework with 74 seconds. When the size of dataset increases to 200MB, the response time is almost equal to that of our implementation with 114 seconds and 112 seconds, respectively. Since then, the performance of Xalan-Java is getting worse, especially when processing 800MB of DBLP XML data with an unacceptable 4520 seconds. The scalability of Xalan-Java is closely related to the size of main memory as the input data to be processed will be loaded and processed within it.

Our CloudXSLT achieves better performance when processing large-scaled DBLP data. Based on the results of Fig.5, CloudXSLT has a stable performance. As the

© 2013 ACADEMY PUBLISHER

size of the data increases, the time to process the data increases sublinearly.

Based on the experimental results, we can conclude that when the size of input XML data is small, the conventional XSLT processors achieve better computing performance as they execute locally without any network communication consumption. But when the size of input data becomes larger, conventional XSLT processors show limitations in finishing the job normally in a commodity machine. By contrary, the MapReduce-based CloudXSLT framework can efficiently process large size of XML data in an acceptable time and has better performance and scalability with the growing size of data. Furthermore, the scalable CloudXSLT framework can gain a better storage capability and computing performance by adding more computing nodes.

VI. CONCLUSION

In this paper, we present a novel MapReduce based XSLT processing framework named CloudXSLT, which provides efficient and scalable XML data transformation services. The logic structure of this framework and some novel MapReduce suitable XML data and XSLT rule models are defined. And then, several parallel processing algorithms are proposed according to the workflow. Through comparing with the existing XSLT processors, the proposed framework shows superior performance and more scalable when processing on large size of XML data.

In the future, we plan to integrate the remaining XSLT template commands in the CloudXSLT framework. In addition, the strategy for storing XML data in the HBase distributed database will be researched as well to provide more flexible data-management service.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China (91118005, 61103114 and 51005260), the Natural Science Foundation of Chongqing City in China (2011BA2022), and the Fundamental Research Funds for the Central Universities in China (CDJXS11181162). We also wish to thank the reviewers for their valuable comments and suggestions.

REFERENCES

[1] L.O. Moreira, F.R.C Sousa, and J.C. Machado, “A distributed concurrency control mechanism for XML data,” Journal of Computer and System Sciences, vol.77 (6), pp.1009-1022, 2011.

[2] W3C. XSL Transformations (XSLT). http://www.w3.org /TR/xslt, 1999.

[3] P. Mika and G. Tummarello, “Web Semantics in the Clouds,” IEEE Intelligent Systems, vol.23(5), pp.82-87, 2008.

[4] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” OSDI’04: Proceeding of the 6th conference on Symposium on Operating Systems Design & Implementaion, pages 10,2004.

[5] S. Ghemawat, H. Gobioff, and S.T. Leung, “The google file system,” SOSP'03: Proceedings of the 19th ACM Symposium on Operating Systems Principles, pp.29-43, 2003.

[6] R. Mutharaju, F. Maier, and P. Hitzler, “A mapreduce algorithm for EL+,” Proceeding of the 23rd International Workshop on Description Logics, pp.464-474, 2010.

[7] J. Urbani, S. Kotoulas, E. Oren, and F.V. Harmelen, “Scalable distributed reasoning using mapreduce,” In Processdings of the 8th International Semantic Web Conference,2009.

[8] Apache, Hadoop. http://hadoop.apache.org/.

[9] P.S. Yuan, C.F. Sha, X.L. Wang, B. Yang, A.Y Zhou, and S. Yang, “XML structural similarity search using MapReduce,” WAIM 2010:11th International Conference on Web-Age Information Management, Lecture Notes in Computer Science, 6184, pp.169-181, 2010.

[10] L. Fegaras, C.K. Li, U. Gupta, and J.J. Philip, “XML Query Optimization in Map-Reduce,” In Processdings of Fourteenth International Workshop on the Web and Databases, 2011.

[11] D. Zinn, S. Bowers, S. Kohler, and B.Ludascher, “Parallelizing XML data-stream workflows via MapReduce”, Journal of Computer and System Sciences, vol.76(6), pp.447-463, 2010.

[12] The DBLP Computer Science Bibliography. http://dblp.uni-trier.de/xml/

[13] XML Data Repository. http://www.cs.washington.edu/ research/xmldatasets/

[14] Guo, Z. M., M. Li, Wang, X, Zhou, A. Y, “Scalable XSLT evaluation,”Advanced Web Technologies and Applications 3007: pp.190-200, 2004.

[15] Zavoral, F., J. Dvorakova, Ieee, “Perfomance of XSLT Processors on Large Data Sets,” 2009 Second International Conference on the Applications of Digital Information and Web Technologies (Icadiwt 2009): pp.110-115, 2009.

[16] Dvorakova, J. “Automatic streaming processing of XSLT transformations based on tree transducers,” Advances in Intelligent and Distributed Computing 78: pp.85-94, 2008.

[17] Thomas Bosch, Brigitte Mathiak, “XSLT Transformation Generating OWL Ontologies Automatically Based on XML Schemas,” 6th International Conference on Internet Technology and Secured Transactions, 11-14 December 2011.

[18] Dong-Hoon Shin, D. H. and K. H. Lee, "Towards the faster transformation of XML documents," Journal of Information Science, vol.32(3), pp.261-276, 2006.

[19] Gou, G. and R. Chirkova, "Efficiently querying large XML data repositories: A survey," Ieee Transactions on Knowledge and Data Engineering, vol.19(10), pp.1381-1403, 2007.

[20] Y., Sun, T., Li, Q., Zhang, J., Yang, and S., Liao, “Parallel XML Transformations on Multi-Core Processors”, IEEE International Conference on e-Business Engineering, 2007.

[21] S., Ren, and D., Muheto, “A Reactive Scheduling Strategy Applied On MapReduce OLAM Operators System”, Journal of Software, vol.7(11), pp.2649-2656, Nov 2012.

[22] H., Zhao, W., Xia, and J., Zhao,“The Research on XML Filtering Model using Lazy DFA”, Journal of Software, vol. 7(8), pp.1759-1766, Aug 2012.

[23] X., Li, Z., Li, Q., Chen and N., Li, “XIOTR: A Terse Ranking of XIO for XML Keyword Search”, Journal of Software, vol. 6(1), pp.156-163, Jan 2011.

© 2013 ACADEMY PUBLISHER