Lower Bounds on the Communication of XPath queries in MapReduce[*]

Foto Afrati
National Technical University of Athens, Greece

Matthew Damigos
Ionian University, Greece

Manolis Gergatsoulis
Ionian University, Greece

ABSTRACT

We present two algorithms, each depending on a different data fragmentation of the XML tree. They both compute XPath queries in MapReduce, by first computing subqueries and then combining their results. We compute the replication rate of each algorithm and show it is less than 2.

1. INTRODUCTION

In this paper, we study how to use MapReduce to compute XPath queries on large XML files. We focus on optimizing the communication cost. It is known that the sequential complexity of evaluating XPath queries on XML trees falls into lower complexity classes with high parallelizable problems [3]. The tree structure of both the data and the query facilitate the low sequential complexity. However, when it comes to using a distributed computational environment to evaluate such queries, and especially when using the MapReduce framework, rigorous work that optimizes the significant performance measures is missing. In starting such an investigation, first we note that, unlike relational databases, XML files have a hierarchical structure that makes distribution to compute-nodes special, in that chunks of data in HDFS are already structured. This structure can be used already in the mappers to compute partial answers to the query [9, 10, 6, 7, 5]. Another approach (which is not discussed in this work) would be to view the data as a collection of one binary relation and a set of unary relations which are distributed to the compute-nodes (mappers) randomly, thus the tree structure of the data cannot be used. This approach however does not seem to have an obvious advantage – although it may be worth being investigated rigoursly in order to figure out its limits.

Communication cost is the size of data transferred among the compute-nodes during a MapReduce job and it affects performance. Communication cost per input is the *replication rate* [4] (one node counts as one input).

In this paper, we give lower bounds on the replication rate for XPath queries on XML trees, taking into account a limit on the size of each compute-node. The size of a compute-node is the number of XML tree nodes stored. The lower bounds we derive for each round are smaller than 2. Actually we give proof of the validity of these lower bounds, by providing an algorithm that achieves this replication rate.

The algorithms presented here use more than one rounds of MapReduce. In the first round, the data are distributed to the compute nodes and subqueries are computed. The next round(s) combine the partial results of the subqueries to compute the final result. For each of the algorithms we assume a different fragmentation of the XML tree. We do not discuss how to implement this fragmentation, which would be necessary for these algorithms to derive also upper bounds on the replication rate.

In particular we present two algorithms, one in Sec. 3, where we do the assumption that a path from the root to any leaf of the XML tree fits in one compute-node, so descendant edges of the query are accommodated. The other algorithm (Sec. 4) accommodates descendant edges in the next rounds after partial descendant-free subqueries are evaluated.

2. PRELIMINARIES

2.1 XML trees and XPath Queries

Consider a directed, rooted, labeled tree t, where its labels come from an infinite set Σ. We denote $\mathcal{N}(t)$ and $\mathcal{E}(t)$ the set of nodes and edges, respectively, of t, and we write $label(n)$ to denote the label of a node n of t. The number d of edges of the unique path through which n is reachable from root of t is said to be the *depth* of n. We define children and descendants of a node if there is an edge or a path, respectively.

We consider two types of trees, those that represent XML documents and those that represent XPath queries. An XML document is represented by a tree (also called *XML tree*) having labels from Σ on its nodes. XPath queries are different from XML trees in three aspects. First, the labels of a query come from the set $\Sigma \cup \{*\}$, where $*$ is the "wildcard" symbol. Second, a query P has two types of edges: $\mathcal{E}_{/}(P)$ is the set of child edges (represented by a single line) and $\mathcal{E}_{//}(P)$ is the set of descendant edges (represented by a double line). Third, a non-Boolean query P has an *output node*,

[*]This research was supported by the project "Handling Uncertainty in Data Intensive Applications", co-financed by the European Union (European Social Fund - ESF) and Greek national funds, through the Operational Program "Education and Lifelong Learning", under the research funding program THALES.

denoted by $out(P)$, and is represented by a circled node. A *Boolean XPath query* does not have any output node. Without loss of generality, we will only consider Boolean queries here. A *subquery* of Q is a single XPath query having a subset of both the nodes and the edges of Q. Furthermore, given an XML tree t and a node n of t, we say that the tree rooted at n is a *subtree* of t. A subquery is Boolean or has the same output as the query if the output node is in the subquery.

The result of applying a query Q on an XML tree t is based on a set of mappings from the nodes of Q to the nodes of t, called embeddings. An *embedding* from Q to t is a mapping $e : \mathcal{N}(Q) \to \mathcal{N}(t)$ with the following properties: (1) Root preserving: $e(root(Q)) = root(t)$, (2) Label preserving: For all nodes $n \in \mathcal{N}(Q)$, either $label(n) = *$ or $label(n) = label(e(n))$, (3) Child preserving: For all edges $(n_1, n_2) \in \mathcal{E}_/(Q)$, we have that $(e(n_1), e(n_2)) \in \mathcal{E}(t)$, and (4) Descendant preserving: For all edges $(n_1, n_2) \in \mathcal{E}_{//}(Q)$, the node $e(n_2)$ is a proper descendant of the node $e(n_1)$.

The result $Q(t)$, now, of applying a non-Boolean query Q on a tree t is formally defined as follows:

$$Q(t) = \{e(out(Q)) | e \textit{ is an embedding from } Q \textit{ to } t\}.$$

If Q is a Boolean query then the result $Q(t)$ is "*true*", only if there is an embedding from Q to t. A *partial embedding* of the query is an embedding of a subtree of the query on the data tree.

According to Dewey encoding system [1], a unique identifier of the form $x_0.x_1.x_2.\ldots.x_d$ can be assigned to each node n of an XML tree. These labels help to decide whether one node is descendant of another (if and only if the Dewey label of the latter is a prefix of the Dewey label of the former), or what is the distance between nodes on the XML tree.

2.2 MapReduce

We will assume that the reader is familiar with MapReduce (details can be found in [2]). However, we need to explain our setting. Typically, each MapReduce job has a map phase and a reduce phase. If we have a sequence of such jobs, then the reducers of the first job send their data to the mappers of the second job, etc. However, the reducers of the first job may act also as mappers of the second job (if it is convenient for the problem at hand) and thus, distribute the data themselves to the reducers of the second job. This is the approach we take here. Hence, we will talk about compute-nodes, instead of distinguishing between mappers and reducers. There is another unconventionality we adopt. Since we use the algorithms we present to argue for lower bounds on the replication rate, we assume that the mappers of the first job have the ability to send any subtree of the XML to the first reducers. This is not totally unrealistic, since many experiments on XML data do a similar fragmentation as ours, because it is a natural way to obtain XML data from HDFS.

3. XML TREE OF SHORT DEPTH

In this section, we consider XML trees where the root-to-leafs paths fit into main memory of compute-nodes; i.e., the size of each compute-node is larger than the depth of the XML tree.

3.1 Data Fragmentation

The fragmentation of the XML tree is done so that in each compute-node we include one subtree of the data tree. Each subtree is rooted in some data node u and all its leaves are leaves of the data tree. We also include the path from the root of the XML tree to u. As we will prove later, including this path adds little extra cost to the replication rate – while, apparently, prunes more nodes.

3.2 Computing and Combining Subqueries

We name the nodes of the query tree by $n_i, i = 1, 2, \ldots$. E.g., in Figure 1, the tree on the left is an XPath query with 23 nodes.

Definition 1. *If there is a partial embedding from the query to the XML tree that maps node n_i of the query to node u of the data tree such that all the descendants of n_i participate (are mapped on some data node) in the partial embedding, then we say that node u is a n_i-*node.

Note that the same node can be both a n_i-node and n_j-node, for distinct i and j. Thus, by considering partial embeddings, we say that we create *adorned nodes*, where the adornment is a nonempty set of nodes from the query tree. Hence, if n_i is in the adornment set of a data node m_j then m_j is a n_i-node.

After distributing the data, each compute-node calculates partial embeddings of the query and finds *maximal* n_i-nodes, for all i, i.e., the parent of a maximal n_i-node is not a n_j-node where n_j is the parent of n_i on the query tree.

We only distribute to the compute-nodes of second round a) the adorned nodes which have at least one maximal adornment in their adornment set and b) all their ancestors (remember they are in the same compute-node). If we can afford to send all such data nodes to one compute-node, then we begin to adorn more nodes as follows: If a node u with a non maximal adornment n_i has children, each child with adornment n_{i_j}, for all the $n_{i_j}, j = 1, 2, \ldots$ children of the query node n_i, then we maximally adorn u with n_i. We terminate this procedure when we find no more nodes to maximally adorn.

If we decide to apply multiple rounds to combine the partial results from the first round, then use the following observation:

- We call a node *candidate n_i-node* if some of its children are adorned accordingly maximally.
- If a data node u is a candidate n_i-node then all its maximally adorned children must meet in the same compute-node in the next round (otherwise "progress" is not made).

The above multi-round distribution is feasible because we do in each compute-node a special kind of deduplication, so that it never emits two siblings with the same adornment. Now, in the following subsection we calculate the replication rate that results from the kind of data fragmentation we do. This calculation applies to both the data fragmentation method in this section and to each of the next necessary rounds that combine the subqueries, since in all cases we distribute similarly structured data (only less, when non-adorned nodes are not distributed).

3.3 Analysis of replication rate

We examine the replication rate of the phase where we distribute the data to the compute-nodes. We analyze in detail two special cases in this section.

3.3.1 Two level XML tree with high degree

Here, we assume that the XML tree has a root with m_0 children and each child c_i of the root has qm_i children itself, where q is the size of a compute-node. These are all leaves of the XML tree. Thus the XML tree T has $n = 1 + m_0 + q\Sigma_1^{m_0} m_i$ nodes in total. For convenience in the calculations below, we assume that each compute-node has size $q + 2$.

Each compute-node is identified by a number from 1 to $M = \Sigma_1^{m_0} m_i$. We send each child of the root c_i to m_i compute-nodes and each leaf to one compute-node. We send the root to all the compute-nodes. The total number of compute-nodes we use is $\Sigma_1^{m_0} m_i$.

In particular child c_i is sent to a number of compute-nodes with identifiers (here i can be thought of as the second dot in the Dewey label):

$$x + \Sigma_{j=1}^{i-1} m_j, \quad x = 1, 2, \ldots, m_i$$

Each leaf l_i is sent only to one compute-node. The communication cost is:

$$C = \Sigma_1^{m_0} m_i + \Sigma_1^{m_0} m_i + q\Sigma_1^{m_0} m_i$$

The first term corresponds to the root, the second term to the children of the root and the final term to the leaves. The replication rate is $r = C/n$. Since $m_0 \leq \Sigma_1^{m_0} m_i$, it is easy to prove that $r \leq 1 + \frac{1}{q}$.

3.3.2 XML tree being a full binary tree

Here, we assume the XML tree is a full binary tree with n nodes. Since we have assumed that the size q of a compute-node is larger than the length of the path from the root to a leaf, we have here that $q > \log n$. Again for convenience in the calculations, we assume that the compute-node size is $q + \log n - \log q$ with $q > \log n$.

In this case, each compute-node gets a whole subtree (with its leaves being all leaves of the XML tree) of size q. Thus the depth of this subtree is $\log q$. The nodes in the XML tree that are closer to the root than $\log n - \log q$ are replicated a number of times. In particular, the nodes at distance $\log n - \log q - i$ ($i = 1, \ldots, \log n - \log q - 1$) from the root are replicated 2^i times. Thus communication for each level (distance from root) is:

$$2^{\log n - \log q - i} \times 2^i = 2^{\log n - \log q} = \frac{n}{q}$$

Hence the total communication cost is:

$$(n - \frac{n}{q}) + \frac{n}{q} \times (\log n - \log q)$$

The first term counts for the nodes that are replicated once. By dividing the above by n, replication rate is

$$(1 - \frac{1}{q}) + \frac{1}{q} \times (\log n - \log q)$$

This is approximately $\log n/q$. Since the assumption is that a path from the root to any leaf of the XML tree fits in one compute-node, $\log n < q$.

3.3.3 General Remarks

In order to calculate the replication rate in the general case we combine the intuition from the two cases we analyzed in detail. The calculation is based on the following remark for the case where all roots (call them *primary roots*) in the data tree that define compute-nodes are in the same level (as in the cases we studied in detail, e.g., full binary tree). We believe that this remark can be extended for the general case too.

- The total communication cost for all nodes at any level is the same.

In order to prove this remark, we consider a node in the data tree that is a parent of some primary root. This node adds as much to the communication cost as add all its children, because it is sent to exactly all compute-nodes its children are sent (and no two children are sent to the same compute-node).

4. TALL XML TREES

Here we assume that a root-leaf path may not fit in one compute-node but a neigborhood of radius d_Q in the XML tree can fit, where d_Q is the maximum acceptable depth of a descendant-free (to be defined shortly) subquery.

4.1 Data fragmentation

Consider an XML tree t of depth d_t. Since there are root-to-leaf paths that cannot fit into main memory of the compute-nodes, we aim to split the root-to-leaf paths. Considering a positive number m (which will depend on compute-node size q), we construct a set of fragments for each $i = 1, \ldots, \lceil \frac{d_t}{m} \rceil$ which contains each tree node whose depth is included in the range $[(i-1)\frac{d_t}{m}, i\frac{d_t}{m} + d_Q]$. Furthermore, notice that every two adjacent fragments overlap. In particular, the i^{th} fragment contains the top d_Q nodes from the set $i+1$, where $i = 1, \ldots, \lceil (\frac{d_t}{m}) \rceil$. This overlap ensures that each subquery given by the decomposition described in next section can be completely answered in some fragment.

4.2 Computing and Combining Subqueries

Definition 2. *Let Q be a query tree and $\mathcal{E}_{//}(Q)$ be the set of descendant edges of Q. Then the* descendant-free subqueries *of Q are the queries obtained by eliminating the descendant edges from Q. We denote the set of the descendant-free subqueries of a query Q as $\mathcal{C}(Q)$.*

It is easy to see that for each descendant edge $d = (n_1, n_2)$ in $\mathcal{E}_{//}(Q)$, there is a pair of queries Q_1, Q_2 in $\mathcal{C}_Q$ such that n_2 is the root node of Q_2 while n_1 is a leaf node of Q_1.

Here we need some more definitions. A node n of a subquery Q' in $\mathcal{C}(Q)$, such that there exists a descendant edge (n, m)

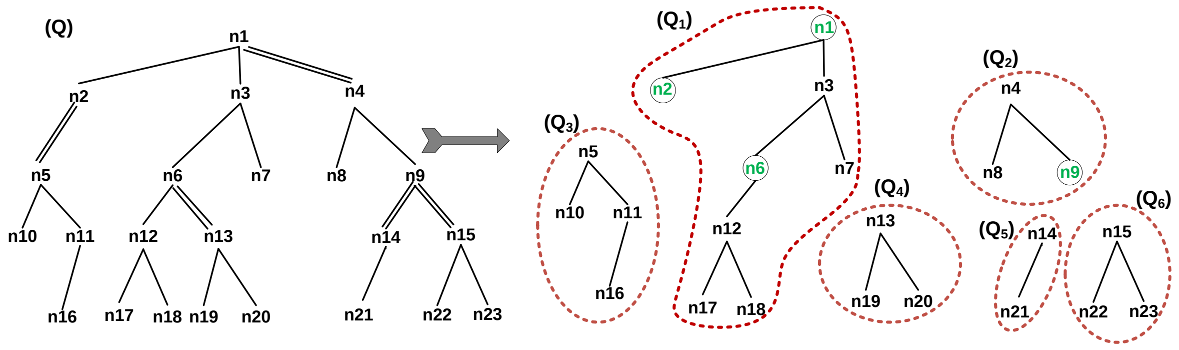

Figure 1: A query Q and its descendant-free subqueries Q_1, Q_2, Q_3, Q_4, Q_5, and Q_6.

in Q, is called a *border node* of Q'. The set of border nodes of Q' is denoted by $\mathcal{N}_{//}(Q')$. A descendant-free subquery Q' that does not contain border nodes is a *leaf subquery* while a subquery that contains a border node is said to be a *non-leaf subquery.*

Example 1. A query tree Q and the set of descendant-free trees $\mathcal{C}(Q) = \{Q_1,\ Q_2,\ Q_3,\ Q_4,\ Q_5,\ Q_6\}$, obtained by its decomposition appear in Figure 1. The set of border nodes of Q is $\mathcal{N}_{//}(Q) = \{n1, n2, n6, n9\}$. Q_3, Q_4, Q_5 and Q_6 are leaf subqueries while Q_1 and Q_2 are non-leaf subqueries.

Definition 3. *Let t be an XML tree, Q be a query and $\mathcal{C}(Q) = \{Q_1,\ Q_2\}$. Assume that $\mathcal{N}(Q_1) = q_0$ and $q_1 = root(Q_2)$. Let e_1 be an embedding from Q_1 to t such that q_0 maps on data node u and e_2 be an embedding from Q_2 to t such that q_1 maps on data node v. Suppose that v is a descendant of u. The composition of e_1 and e_2, denoted as $e_1 \circ e_2$, is a mapping e from $\mathcal{N}(Q)$ to t such that for each $n \in \mathcal{N}(Q_1)$ then $e(n) = e_1(n)$, otherwise $e(n) = e_2(n)$.*

Evaluation Strategy 1. The query evaluation strategy consists in the following three steps:

1. Decompose the query Q into a set of descendant-free subqueries $\mathcal{C}(Q)$.
2. Evaluate separately each subquery in $\mathcal{C}(Q)$.
3. Combine appropriately (pairwise as per Definition 3) the embeddings of the queries in $\mathcal{C}(Q)$ to find the embeddings of Q.

To combine appropriately the embeddings of the subqueries in $\mathcal{C}(Q)$ we can follow either a multi-round approach or a single-round approach. In the i^{th} round of the multi-round approach, we construct one compute-node for each image u of the i^{th} border node (proceeding bottom-up in that we first consider border nodes that have descendant edges to roots of trees without border nodes). We send to this compute-node all the descendants of u (Dewey label is used here). The trade-off between the two approaches is that the amount of pairs received by a compute-node may exceed the size of the compute-node; while following multi-round approach we perform iterative pruning of the intermediate pairs and we reduce the amount of the pairs sent to each compute-node in each round.

4.3 Replication rate analysis

The replication rate is less than 2 during the data fragmentation, since some of the data are replicated only once and the rest only twice. For the replication rate during the other rounds, we assume again deduplication (in a similar sense as in the first algorithm) in the first round. Thus, each compute-node emits only one (of each n_j-nodes set) descendant of a specific data node. The Dewey labels are used to recognize that. Hence we can assume again that all "relevant" descendants of a specific data node fit in one compute-node.

5. REFERENCES

[1] S. Abiteboul, I. Manolescu, P. Rigaux, M.-C. Rousset, and P. Senellart. *Web Data Management.* Cambridge University Press, 2011.

[2] J. Leskovec, A. Rajaraman, and J.D. Ullman. *Mining of massive datasets.* Cambridge University Press, 2014.

[3] G. Gottlob, C. Koch, and R. Pichler The Complexity of XPath Query Evaluation. *PODS, 179–190, 2003.*

[4] F. N. Afrati, A. D. Sarma, S. Salihoglu, J. D. Ullman. *Upper and Lower Bounds on the Cost of a Map-Reduce Computation.* PVLDB, 6(4):277–288, 2013

[5] N. Bidoit, D. Colazzo, N. Malla, F. Ulliana, M. Nolé, and C. Sartiani. Processing xml queries and updates on map/reduce clusters. In *EDBT*, pages 745–748, 2013.

[6] G. Cong, W. Fan, A. Kementsietsidis, J. Li, and X. Liu. Partial evaluation for distributed XPath query processing and beyond. *ACM Trans. Database Syst.*, 37(4):32:1–32:43, Dec. 2012.

[7] M. Damigos, M. Gergatsoulis, and S. Plitsos. Distributed processing of xpath queries using mapreduce. In *ADBIS (2)*, pages 69–77, 2013.

[8] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Jan. 2008.

[9] S. Khatchadourian, M. P. Consens, and J. Siméon. Having a chuql at xml on the cloud. In *AMW*, 2011.

[10] L. Lewandowski. Using Map and Reduce for Querying Distributed XML Data. *MSc. Thesis*, 2012. `http://www.inf.uni-konstanz.de/gk/pubsys/publishedFiles/Lewandowski12.pdf`.