

Online Integration of Fragmented XML Documents

Handoko¹(✉) and Janusz R. Getta²

¹ Electronic and Computer Engineering Department,
Satya Wacana Christian University, Salatiga, Indonesia
`handoko@staff.uksw.edu`

² School of Computer Science and Software Engineering,
University of Wollongong, Wollongong, NSW, Australia
`jrg@uow.edu.au`

Abstract. Online data integration of large XML documents provides the most up-to-date results from the processing of user requests issued at a central site of heterogeneous multi-database system. The fragments of large XML documents received from the remote sites are continuously combined with the most current state of integrated documents. Online integration of fragmented XML documents has a positive impact on performance of entire online data integration system.

This paper presents the online integration procedures for the fragments of large XML documents. We propose a new model of data for fragmented XML documents and we define a set of operations to manipulate the fragments. A new optimisation procedure presented in the paper finds the smallest core of each new fragment that can be integrated with the documents available at a central site. We show that processing of the smallest cores of XML fragments significantly reduces overall processing time.

Keywords: Data integration · Online algorithm · Fragmented XML documents · Semistructured data

1 Introduction

The recent growth of wide area networks allows for data integration across many diverse systems with various data formats at the remote endpoints. In a global data model approach, a central site in a data integration system breaks down user requests and sends a number of sub-requests to the remote sites to get the results. Then, such results are processed by the data integration system accordingly to the earlier prepared data integration plans.

Online integration is a continuous consolidation of data transmitted over a network with data already available at a central site. It provides a user with the most up-to-date results of a query being processed by the system. Online integration applies online processing algorithms where a smallest unit of data increment is instantly processed without having entire set of data available [5].

In a distributed and heterogeneous multi-database system, the large chunks of data retrieved at the remote sites are decomposed and sent to a central site as XML fragments. Then, the incoming XML fragments are combined at a central site to form the original documents for further processing. In fact, integration of the fragments can be started immediately if the fragments available at a central site have enough data to complete the integration process. Hence, it is important to identify the minimum requirements for processing of XML fragments to allow processing of a smaller cores of fragments rather than the complete ones. Then, a modified version of the algorithms described in [5] allows for more efficient processing of the fragments of XML documents.

The structure of this paper is the following. Section 2 covers the previous work in an area of integration of fragmented XML documents. The principles of processing XML fragments are described in Sect. 3. Section 4 covers fragmentation of XML documents, and Sect. 5 describes XML algebraic operations on fragmented XML documents. In Sect. 6 we describe an online integration algorithm for fragmented XML documents, and Sect. 7 concludes the paper.

2 Previous Work

XML fragmentation improves the performance of query processing through the decompositions of queries into smaller sub-queries operating in a parallel mode on the fragments of XML documents. It is achieved as either *ad-hoc* or *structured* fragmentation [7]. Hole-filler is the most popular *ad-hoc* fragmentation technique. In this approach, every XML fragment has a unique *filler* ID and a set of *holes* which represent empty places where other fragments could be connected to. A structure of original document and its fragmentation schema is presented in a simple DTD named *Tag Structure* [2]. On the other hand, *structured* fragmentation follows the rules of the relational model fragmentation [3,6].

XML data stream is a theoretically infinite sequence of XML documents. Processing of XML data stream is a challenging process because it requires very efficient algorithms to integrate the most recently received increments with the already completed results of processing. Most of query processing techniques on XML data streams are based on XQuery and XPath streaming evaluation [9]. Some of XML stream processing techniques based on XQuery evaluation refer to XML fragments in the concept of *hole-filler* [1,2,8]. In addition to *hole-filler* model, Bose [2] proposed a query algebra for XQuery on XML stream data.

Fegaras [4] proposed an incremental query processing for a large-scale database called MRQL Streaming. A streaming query is expressed as $q(\bar{S})$ where $S_i \in \bar{S}$ and $S_i : i = 0, \dots, n$ is a streaming data source. S_i contains an initial dataset and followed by a continuous incremental stream ΔS_i in a time interval Δt . Incremental processing is performed by combining the result query at time t with results of query on ΔS_i , such that $h(\bar{S} \uplus \Delta \bar{S}) = h(\bar{S}) \otimes h(\Delta \bar{S})$. \otimes is a merge function which is implemented as a partitioned *join*. Online integration system proposed in [5] performs continuous integration where an incoming and complete XML document triggers the computations of a data integration expression.

3 Principles and Assumptions

We consider a model of data integration where the data increments are transmitted to a central site as the fragments of XML documents. We found in an earlier work [5] that it is possible to increase performance of an online integration system when the processing is performed on the smaller cores of XML fragments. Integration of semistructured data described in this work is based on the following assumptions. (1) The remote sites have the ability to disassemble XML documents into XML fragments with the characteristics described in the next section. (2) XML fragments retrieved at the remote sites are received by a central site in a random order. (3) Due to a high level of autonomy of the remote sites, a central site has no impact on the priorities with which the fragments are retrieved at and sent by the remote sites.

Based on these assumptions, we adopt the following more specific principles for *online data integration* process.

1. A fragmented XML document is a set of XML fragments.
2. Every data container located at a central site contains fragmented XML documents.
3. At a pre-processing stage described in [5], we generate online integration plans for every user request received by the central site. First, we transform a user query into a *global query expression* ($f(q_1, \dots, q_k)$). Then, we transform it into a *data integration expression* ($f(D_1, \dots, D_k)$) by systematic replacement of symbol q_1, \dots, q_k with the data container D_1, \dots, D_k . In the next step, we generate the *increment expressions* which allow for instant computation of new data increments of the particular data containers at a central site. We use the increment expressions obtained earlier to generate the online integration plans for each data container. The main difference between an approach described in [5] and this work is online integration performed on XML fragments instead of complete XML documents.
4. The attributes used in the filter expressions related to data containers are stored in the adequate lists to find when a set of XML fragments is ready for processing.

4 Fragmented XML Document

The operations on XML fragments allow us to process incomplete XML documents and to append their missing parts at the end of integration. Based on an assumption that every node in an XML document can be identified by a unique path (i.e. path and index), we define an XML fragment in the following way.

Definition 1. An XML fragment is a tuple $\langle x_i(m_i), o, p, H \rangle$ where $x_i(m_i)$ is an XML document that represents a body of the XML fragment. A component o is the identity of parent XML document the fragment comes from, and p (hook) is a unique path where the root of XML fragment is located in the parent XML document. A component H is a set of paths that represents the missing XML fragments (holes) in $x_i(m_i)$.

An XML fragment $(\langle x_i(m_i), o, p, H \rangle)$ has the following characteristics. (1) An XML fragment body $(x_i(m_i))$ is a well-formed XML document. (2) It has a component (o) to store id of its parent XML document. (3) It has a *hook* (p) and a *hole* component (H) that allow for reconstruction of XML fragments into the parent XML document. (4) A *hook* component (p) is represented by a path, and determines a location of XML fragment in its parent XML document. (5) There is exactly one XML fragment that includes a root node of the parent XML document, i.e. its *hook*="xml". (6) A *hole* (H) component is a set of paths to the roots of missing fragments. If $H = \emptyset$ then a fragment is complete.

Definition 2. A *fragmented XML document* is a set of XML fragments $\{\langle x_i(m_i), o_i, p_i, H_i \rangle : i = 1, \dots, n\}$.

Note, that o_i does not need to be the same for all fragments in a fragmented XML document.

Definition 3. Let $x(m) = \{\langle x_i(m_i), o_i, p_i, H_i \rangle : i = 1, \dots, n\}$ be a fragmented XML document. A *complete XML document* is defined as a fragmented XML document where $\bigcup_i p_i - \{\text{"xml"}\} = \bigcup_i H_i$.

In the other words, a set of XML fragments is a *complete XML document* if for every *hole* the set includes a respective fragment with a matching *hook* and the set also includes a *root fragment* with a path "xml" where a virtual node "xml" plays a role of a root node of every XML document.

Example 1. Let $x(m)$ be a fragmented XML document disassembled into six XML fragments visualised in Fig. 1. The details of the fragments are listed in Table 1.

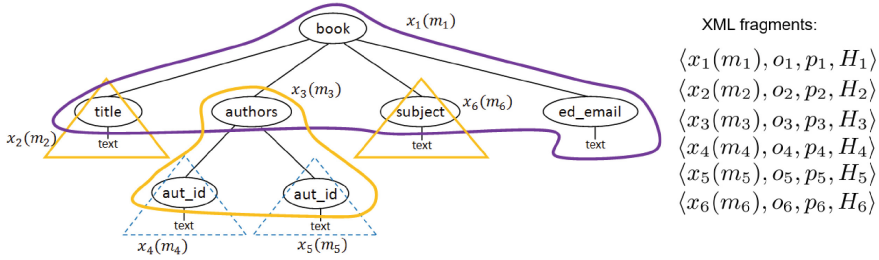


Fig. 1. Fragmented XML document.

5 Operations of XML Algebra

The operations on XML algebra belong to one of the following three types depending on an abstraction level of an operation: (1) operations on XML fragments, (2) operations on fragmented XML documents, and (3) operations on data containers with fragmented XML documents. A *fusion* of fragmented XML

Table 1. The components of XML fragments in Fig. 1

XML fragment	Origin id	Hook	Holes
(a)	$o_1 = "1"$	$p_1 = "xml"$	$H_1 = \{ "xml/book/title", "xml/book/authors", "xml/book/subject" \}$
(b)	$o_2 = "1"$	$p_2 = "xml/book/title"$	$H_2 = \{ \}$
(c)	$o_3 = "1"$	$p_3 = "xml/book/authors"$	$H_3 = \{ "xml/book/authors/aut_id[1]", "xml/book/authors/aut_id[2]" \}$
(d)	$o_4 = "1"$	$p_4 = "xml/book/authors/aut_id[1]"$	$H_4 = \{ \}$
(e)	$o_5 = "1"$	$p_5 = "xml/book/authors/aut_id[2]"$	$H_5 = \{ \}$
(f)	$o_6 = "1"$	$p_6 = "xml/book/subject"$	$H_6 = \{ \}$

documents is needed to combine two XML documents where one of them is a part of the other, see Definition 4 below. The definition assumes that a structure of each XML document is represented by an *Extended Tree Grammar (ETG)*[5].

Definition 4. Let $G = (N_g, T_g, A_g, S_g, P_g)$ and $H = (N_h, T_h, A_h, S_h, P_h)$ be ETGs, $N_g \cap N_h \neq \emptyset$, Y be a non terminal symbol, and $y \in (N_g \cap N_h)$. Let $S \rightarrow xml[id](Y)$ be a production rule for start symbol in H . Let $p_g \in P_g$ and $p_h \in P_h$ be production rules for a non terminal symbol Y in both ETGs. A fusion operation on two ETGs is denoted as $F = G \oplus H$ and is an operation that combines G and H , such that $F = (N, T, A, S, P)$ is an ETG where $N = N_g \cup N_h$, $T = T_g \cup T_h$, $A = A_g \cup A_h$, and $P = P_g \cup P_h - \{p_g\}$.

A *hook* operation combines two XML fragments which have matching *hook* and *hole* components to form a more complete XML fragment. The operation creates a new XML fragment where one of the *holes* in the first argument is filled with the second argument, see Definition 5 below.

Definition 5. Let $\langle x_i(m_i), o_i, p_i, H_i \rangle$, $\langle x_j(m_j), o_j, p_j, H_j \rangle$ be XML fragments with ETG G and H respectively. Let $o_i = o_j$ and $p_j \in H_i$. A *hook operation* on two XML fragments is defined as $\langle x_i(m_i), o_i, p_i, H_i \rangle \leftarrow \langle x_j(m_j), o_j, p_j, H_j \rangle = \langle x_r(m_r), o_r, p_r, H_r \rangle$. $x_r(m_r)$ is an XML fragment result after a hook operation, $o_r = o_i = o_j$, and $p_r = p_i$. $H_r = H_i \cup H_j - \{p_j\}$ is a set of holes after a hook operation. The XML fragment result has an ETG $F = G \oplus H$.

A *union* of two sets of XML fragments is possible when the sets do not contain the fragments with the same *hook* and it is defined in a usual way as a theoretical set union. At some points of processing, we might want to apply *defragmentation* procedure that first uses *union* operation to group all relevant fragments in one set and later on it uses *hook* operation to combine the XML fragments which have the matching values of *hook* and *hole*.

To speed up defragmentation, we sort the XML fragments by their o and p components. The sorted XML fragments shows that the position of XML fragments in a fragmented XML document represents their location at the origin XML document. If the XML fragments are sorted, then for two XML fragments

$\langle x_i(m_i), o_i, p_i, H_i \rangle$ and $\langle x_j(m_j), o_j, p_j, H_j \rangle$ where i, j are the element indexes and $i < j$, we can perform a *hook* operation $\langle x_i(m_i), o_i, p_i, H_i \rangle \leftarrow \langle x_j(m_j), o_j, p_j, H_j \rangle$ but not the opposite.

Next, we need an efficient algorithm that processes the incoming XML fragments. The algorithm must determine where to place an incoming fragment, when to combine a fragment into any of existing fragmented documents in the data containers on input to data integration and materializations containing the intermediate results of data integration, and how to perform defragmentation process on the fragmented XML documents.

A *minion* operation integrates two data containers with fragmented XML documents, see Definition 6 below.

Definition 6. Let $D(\mathcal{G}), D(\mathcal{H})$ be data containers of fragmented XML documents. Let $x(m) \in D(\mathcal{G}), x(m) = \{\langle x_i(m_i), o_i, p_i, H_i \rangle : i = 1, \dots, s\}$ and $y(n) \in D(\mathcal{H}), y(n) = \{\langle y_j(n_j), o_j, p_j, H_j \rangle : j = 1, \dots, t\}$. Let $O_i = \{o_i : \exists \langle x_i(m_i), o_i, p_i, H_i \rangle \in x(m) : i = 1, \dots, s\}$, $O_j = \{o_j : j = 1, \dots, t\}$ and $O_i \cap O_j \neq \emptyset$. A *minion* (merge-union) operator is defined as $D(\mathcal{G}) \uplus D(\mathcal{H}) = \{z(l) : \exists x(m) \in D(\mathcal{G}), y(n) \in D(\mathcal{H}), z(l) = x(m) \cup y(n)\}$.

We organise the input data containers and processing of incoming fragments in the following way.

1. Every data container (D) is divided into a *bounded* data container (D^b) and a *rover* data container (D^r) ($D = D^b \uplus D^r$). D^b is used to store fragmented XML documents which are ready for processing. Meanwhile, not processed fragmented XML documents are placed in a *rover* data container (D^r).
2. A new incoming XML fragment is placed as an element of a fragmented XML document in a data container D^r according to its original XML document.
3. When a fragmented XML document in D^r satisfies the minimal requirements for processing, it is transferred to a *bounded* data container D^b , and its processing is started.
4. We need an operation on two data containers of fragmented XML document to combine the fragmented XML documents which have the same identity in both data containers. For example, such operation is needed at the end of processing in order to add the unprocessed XML fragments to fragmented XML document results.

Since we use a concept of fragmented XML document to replace a complete XML document, most of the XML algebra operators described in [5] are applicable. Nevertheless, XML algebraic operations have to be re-defined. Some XML algebraic operators require examination of a condition expression (φ) on the path expressions. For complete XML documents, path expressions refer to navigation paths from the root node of XML documents. Meanwhile, path evaluation on a fragmented XML document requires a procedure to discover a particular node in XML fragments which may not have its root element. Below, we redefine the concepts of *selection*, *join*, and *antijoin* for fragmented XML documents.

Definition 7. Let $D(\mathcal{G})$ be a data container of fragmented XML documents, $x(m) \in D(\mathcal{G})$, $x(m) = \{\langle x_i(m_i), o_i, p_i, H_i \rangle : i = 1, \dots, n\}$, and $x_f = \langle x_i(m_i), o_i, p_i, H_i \rangle$. Selection on $D(\mathcal{G})$ is a unary operator denoted by $\sigma_\varphi(D(\mathcal{G})) = \{x(m) : \exists x_f \in x(m) (f(x_f, \varphi) = \text{true})\}$, where $f(x_f, \varphi) \in \{\text{true}, \text{false}\}$, φ is a condition expression.

Definition 8. Let $x(m) = \{\langle x_i(m_i), o_i, p_i, H_i \rangle : i = 1, \dots, n\}$ and $y(n) = \{\langle y_j(n_j), o_j, p_j, H_j \rangle : j = 1, \dots, m\}$ be fragmented XML documents. Let $x_f = \langle x_i(m_i), o_i, p_i, H_i \rangle$ and $y_f = \langle y_j(n_j), o_j, p_j, H_j \rangle$. Join operation on fragmented XML documents is defined as $x(m) \bullet_\varphi y(n) = x(m) \cup y(n) : \exists x_f \in x(m) \exists y_f \in y(n) \text{ and } f(x_f, y_f, \varphi) = \text{true}$. φ is a condition expression and f is an evaluation function such that $f(x_f, y_f, \varphi) \in \{\text{true}, \text{false}\}$.

Definition 9. Let $D(\mathcal{G}), D(\mathcal{H})$ be data containers of fragmented XML documents. Join operation is defined as $D(\mathcal{G}) \bowtie_\varphi D(\mathcal{H}) = \{z(o) : \exists x(m) \in D(\mathcal{G}), y(n) \in D(\mathcal{H}), z(o) = x(m) \bullet_\varphi y(n)\}$.

Definition 10. Let $D(\mathcal{G}), D(\mathcal{H})$ be data containers of fragmented XML documents. Antijoin operator is defined as $D(\mathcal{G}) \sim_\varphi D(\mathcal{H}) = \{x(m) : x(m) \in D(\mathcal{G}) \text{ and } \forall y(n) \in D(\mathcal{H}) \neg \exists (x(m) \bullet_\varphi y(n))\}$, where φ is a condition expression.

Let D_i, D_j, D_k be data containers for fragmented XML documents. A *minion* (merge-union) operation has the following properties.

1. A *minion* operation is commutative, i.e. $(D_i \uplus D_j) = (D_j \uplus D_i)$.
2. A *minion* operation is associative, i.e. $D_i \uplus (D_j \uplus D_k) = (D_i \uplus D_j) \uplus D_k$.
3. If a condition φ can be evaluated in D_i , then *minion* operation is distributive over selection operation, i.e. $\sigma_\varphi(D_i \uplus D_j) = \sigma_\varphi(D_i) \uplus D_j$.
4. If a property for join operation condition φ exists in D_j , then *minion* operation is distributive over join operation: $D_i \bowtie_\varphi (D_j \uplus D_k) = (D_i \bowtie_\varphi D_j) \uplus D_k$.
5. A *minion* operation is left and right distributive over union operation, i.e. $D_i \cup (D_j \uplus D_k) = (D_i \cup D_j) \uplus D_k$ and $(D_i \uplus D_j) \cup D_k = (D_i \cup D_k) \uplus D_j$.
6. If a condition φ can be evaluated in D_i , then *minion* operation is right distributive over antijoin operation, i.e. $(D_i \uplus D_j) \sim_\varphi D_k = (D_i \sim_\varphi D_k) \uplus D_j$.
7. A *minion* operation can reduce the antijoin operation, i.e. $D_i \sim_\varphi (D_j \uplus D_k) = (D_i \sim_\varphi D_j)$, if XML fragment D_j contains elements in operation condition (φ) $D_i \sim_\varphi (D_j \uplus D_k) = (D_i \sim_\varphi D_k)$ and if XML fragment D_k contains elements in operation condition φ .

6 Online Integration of XML Fragments

Initially, online integration of XML fragments is similar to integration described in [5]. An incoming XML fragment is placed in a *rover* data container (D^r) and it is added to a corresponding fragmented XML document accordingly to its identity. If an incoming XML fragment does not match to any existing fragmented XML document, we create a new fragmented XML document and place the incoming XML fragment into it. If a fragmented XML document in the

rover data container has enough properties for processing then it is transferred to a corresponding *bounded data container*. Incoming data at the *bounded* data containers triggers the following processing of a data integration expression.

The online integration system adjusts the pre-processing phase to deal with XML fragments. The central site is responsible to determine all condition (φ) attributes for all operations, all elements involved in the conditions for every data container, and all available elements in the fragmented XML documents. Then, the existing nodes/elements are applied to decide whether a fragmented XML document has enough properties for further processing.

It may happen that a data container is used several times in a data integration expression. Therefore, a list of adequate properties is applied.

Example 2. Let $f(D_1, D_2, D_3) = (D_1 \bowtie D_2) \cup (D_3 \sim D_1)$ be a data integration expression. A *join* operation $D_1 \bowtie_{(\varphi_1 \vee \varphi_2)} D_2$ has two condition expressions as follows: $\varphi_1 = \text{xml/book/authors/aut_id}[1]=//\text{aut_id}$ and $\varphi_2 = \text{xml/book/authors/aut_id}[2]=//\text{aut_id}$.

XML paths `xml/book/authors/aut_id[1]` and `xml/book/authors/aut_id[2]` are unique locations of node elements of a fragmented XML document. Meanwhile, `//aut_id` is a path of a node element of a fragmented XML document in the data container D_2 . Hence, we generate an adequate list as in the Table 2.

Table 2. An adequate lists of data containers for a data integration expression $f(D_1, D_2, D_3) = (D_1 \bowtie D_2) \cup (D_3 \sim D_1)$

Data container	Operation	Path	Opr	Path or value
D_1	Operation 1	<code>xml/book/authors/aut_id[1]</code>	=	<code>//aut_id</code>
D_1	Operation 1	<code>xml/book/authors/aut_id[2]</code>	=	<code>//aut_id</code>
D_2	Operation 1	<code>//aut_id</code>	=	<code>xml/book/authors/aut_id</code>

Processing of fragmented XML documents may create fragmented XML documents at data containers, materializations (M_j), where the fragmented XML documents have been computed, and remove lists (L_d), where they do not meet criteria for evaluation of the required condition (φ).

6.1 Data Integration Expression

We use a *bounded* D_i^b and *rover* D_i^r input data containers to save the incoming fragmented XML documents. As a consequence, a data integration expression should be transformed in the following way.

1. We replace all data containers D_i in a data integration expression with $D_i^b \uplus D_i^r$, for $i = 1 \dots k$.

2. Next, we transform the data integration expression by multiple applications of *minion* properties such that all *rover* data containers are moved to the end of computation process.

Example 3. A data integration expression for XML fragments.

Consider the following input data containers D_i represented as *minion* of respective *bounded* and *rover* containers $D_i^b \uplus D_i^r$, for $i = 1 \dots 6$. A data integration expression $(D_1 \bowtie (D_2 \sim D_3)) \cup ((D_4 \bowtie D_5) \sim D_6)$ is transformed through systematic replacement of the containers D_i with the expressions $D_i^b \uplus D_i^r$ and distribution of *minion* operation over *join*, *antijoin* and *union* operations.

$((D_1^b \uplus D_1^r) \bowtie (D_2 \sim D_3)) \cup ((D_4 \bowtie D_5) \sim D_6)$; D_1 is replaced with $D_1^b \uplus D_1^r$
 $((D_1^b \bowtie (D_2 \sim D_3)) \uplus D_1^r) \cup ((D_4 \bowtie D_5) \sim D_6)$
 $((D_1^b \bowtie (D_2 \sim D_3)) \cup ((D_4 \bowtie D_5) \sim D_6)) \uplus D_1^r$; D_6 is replaced with $D_6^b \uplus D_6^r$
 $((D_1^b \bowtie (D_2 \sim D_3)) \cup ((D_4 \bowtie D_5) \sim (D_6^b \uplus D_6^r))) \uplus D_1^r$
 $((D_1^b \bowtie (D_2 \sim D_3)) \cup (((D_4 \bowtie D_5) \sim D_6^b) \uplus D_6^r)) \uplus D_1^r$
 $((D_1^b \bowtie (D_2 \sim D_3)) \cup ((D_4 \bowtie D_5) \sim D_6^b)) \uplus D_1^r \uplus D_6^r$
 and so on.

At the end we obtain an expression given below.

$$((((((D_1^b \bowtie (D_2 \sim D_3)) \cup ((D_4^b \bowtie D_5^b) \sim D_6^b) \uplus D_1^r) \uplus D_2^r) \uplus D_4^r) \uplus D_5^r) \uplus D_6^r$$

6.2 Increment Expression

In the next step, we transform a data integration expression into an increment expression for every data container by applications of XML algebra rules as described in [5]. A *defragmentation* process can be performed at the end of processing because the *rover* data containers have been moved to the end of data integration expression.

Example 4. An increment expression generation.

Let δ_1 be an increment data at a *bounded* data container D_1^b . The data integration expression

$$f(D_1, \dots, D_6) = (((((D_1^b \bowtie (D_2 \sim D_3)) \cup ((D_4^b \bowtie D_5^b) \sim D_6^b)) \uplus D_1^r) \uplus D_2^r) \uplus D_4^r) \uplus D_5^r$$

can be transformed as follows:

$$\begin{aligned} & (((((((D_1 \cup \delta_1) \bowtie (D_2 \sim D_3)) \cup ((D_4^b \bowtie D_5^b) \sim D_6^b)) \uplus D_1^r) \uplus D_2^r) \uplus D_4^r) \uplus D_5^r \\ & (((((((D_1^b \bowtie (D_2 \sim D_3)) \cup (\delta_1 \bowtie (D_2^b \sim D_3^b))) \cup ((D_4^b \bowtie D_5^b) \sim D_6^b)) \uplus D_1^r) \uplus D_2^r) \uplus D_4^r) \uplus D_5^r \\ & (((((((D_1^b \bowtie (D_2 \sim D_3)) \cup ((D_4^b \bowtie D_5^b) \sim D_6^b)) \cup (\delta_1 \bowtie (D_2^b \sim D_3^b))) \uplus D_1^r) \uplus D_2^r) \uplus D_4^r) \uplus D_5^r \\ & (((f(D_1^b, \dots, D_6^b) \cup (\delta_1 \bowtie (D_2^b \sim D_3^b))) \uplus D_1^r) \uplus D_2^r) \uplus D_4^r) \uplus D_5^r \\ & (((f(D_1^b, \dots, D_6^b) \cup (\delta_1 \bowtie M_1)) \uplus D_1^r) \uplus D_2^r) \uplus D_4^r) \uplus D_5^r M_1 = ((D_2^b \sim D_3^b) \uplus D_2^r) \end{aligned}$$

Using the same transformation procedures, we obtain a set of increment expressions for the rest of data containers.

$$\begin{aligned}
\delta_1 &: (((f(D_1^b, \dots, D_6^b) \cup (\delta_1 \bowtie M_1)) \uplus D_1^r) \uplus D_2^r) \uplus D_4^r) \uplus D_5^r \\
\delta_2 &: (((f(D_1^b, \dots, D_6^b) \cup (D_1 \bowtie (\delta_2 \sim D_3))) \uplus D_1^r) \uplus D_2^r) \uplus D_4^r) \uplus D_5^r \\
\delta_3 &: (((f(D_1^b, \dots, D_6^b) \sim (\delta_3 \sim M_4)) \uplus D_1^r) \uplus D_2^r) \uplus D_4^r) \uplus D_5^r \\
\delta_4 &: (((f(D_1^b, \dots, D_6^b) \cup ((\delta_4 \bowtie D_5) \sim D_6^b)) \uplus D_1^r) \uplus D_2^r) \uplus D_4^r) \uplus D_5^r \\
\delta_5 &: (((f(D_1^b, \dots, D_6^b) \cup ((D_4^b \bowtie \delta_5) \sim D_6^b)) \uplus D_1^r) \uplus D_2^r) \uplus D_4^r) \uplus D_5^r \\
\delta_6 &: (((f(D_1^b, \dots, D_6^b) \sim (\delta_6 \sim M_1))) \uplus D_1^r) \uplus D_2^r) \uplus D_4^r) \uplus D_5^r \\
\text{where } M_1 &= ((D_2^b \sim D_3^b) \uplus D_2^r); M_2 = (((D_1^b \bowtie (D_2^b \sim D_3^b)) \uplus D_1^r) \uplus D_2^r); M_3 = \\
&(((D_4^b \bowtie D_5^b) \uplus D_4^r) \uplus D_5^r); M_4 = (((D_4^b \bowtie D_5^b) \sim D_6^b) \uplus D_4^r) \uplus D_5^r
\end{aligned}$$

Finally, the increment expressions are the following.

$$g_1 = (\delta_1 \bowtie M_1); g_2 = (D_1 \bowtie (\delta_2 \sim D_3)); g_3 = (\delta_3 \sim M_4); g_4 = ((\delta_4 \bowtie D_5) \sim D_6); g_5 = ((D_4 \bowtie \delta_5) \sim D_6); g_6 = (\delta_6 \sim M_1)$$

6.3 Online Integration Plans for XML Fragments

The increment expressions generated from a data integration expression on fragmented XML documents are the extensions of processing on complete XML documents. Therefore, we can utilize the algorithms for online integration plan and scheduling described in [5]. However, since fragmented XML documents in the *rover* data containers have no effect to the rest of computation, we apply *minion* operations at the very end of computation process. Fragmented XML documents in the *rover* data containers are excluded from computation of data integration expression until the results are ready to send.

Example 5. Let $g_2 = (D_1 \bowtie (\delta_2 \sim D_3))$ be an increment expression generated in Example 4. We consider a data increment (δ_2) arrives at a data container D_2 . Transformation of g_2 into an online integration plan d_2 is performed as follows.

- (1) In the first step, we map an expression $(\delta_2 \sim D_3)$ into a step $\Delta_1 = (\delta_2 \sim D_3)$ and an expression $(D_1 \bowtie \Delta_1)$ into a step $\Delta_2 = (D_1 \bowtie \Delta_1)$.
- (2) Then, we append $M_e = (M_e \sim \Delta_2)$ to combine the computation results with the previous final materialization.
- (3) Next, we update a data container $D_2 : D_2 = (D_2 \cup \delta_2)$.
- (4) An intermediate materialization M_1 is identified to be affected to update. M_1 is a computation result of a data integration expression $h_1(D_2, D_3) = (D_2 \sim D_3)$. Therefore $h_1(D_1, D_2)$ is transformed into an increment expression $g_{M_1} = (\delta_2 \sim D_3)$. A plan to update M_1 is generated as follows: $d_{M_1} : \Delta_{M_1} = (\delta_2 \sim D_3); M_1 = (M_1 \cup \Delta_{M_1})$. These steps are appended to the steps produced earlier.

The complete online integration plan for increment expression g_2 is a sequence of operations $p_1 : \Delta_1 = (\delta_2 \sim D_3); p_2 : \Delta_2 = D_1 \bowtie \Delta_1; p_3 : M_e = (M_e \sim \Delta_3); p_4 : D_2 = (D_2 \cup \delta_2); p_5 : \Delta_{M_1} = (\delta_2 \sim D_3); p_6 : M_1 = (M_1 \cup \Delta_{M_1})$.

Then, we perform *minion* operations to combine *rover* data containers with the final materialization before we send the results to users. A sequence of operations includes $M_e = (M_e \uplus D_1^r); M_e = (M_e \uplus D_2^r); M_e = (M_e \uplus D_4^r); M_e = (M_e \uplus D_5^r)$. Operations to combine *rover* data containers to the intermediate materializations are not necessary since we have reached the end of computation process.

To increase performance of the online integration system, we apply *defragmentation* procedure at the very end of computation or only if it is needed. At this stage, the scheduling algorithms for online integration plans described in [5] are applicable.

7 Summary and Future Work

Online integration system proposed in this paper optimizes the integration of large size XML documents by processing the fragments of data increments. Our approach allows for the dynamic identification of a core within every data increment and for processing of each increment in a moment when a core is complete even if an increment itself is not complete yet. It reduces time an increment is waiting for the processing. Meanwhile, the defragmentation procedures are performed at the end of processing to reduce overall online data integration time. We provide the formal backgrounds to show that the online data integration system proposed in the paper is implementable.

A number of interesting research problems remain to be investigated. Online integration system proposed in this paper is applicable for data integration in Internet of Things environment, by associating sensor nodes to the data containers. The first problem is the data structure alignment when the sensors use their own internal data structures. The second problem is related to the large number of sensors involved. The last problem involves properties of usually small and frequently changed data.

References

1. Bose, S., Fegaras, L.: Data stream management for historical XML data. *SIGMOD* **99**(3), 403–422 (2004)
2. Bose, S., Fegaras, L., Levine, D., Chaluvadi, V.: A query algebra for fragmented XML stream data. In: Lausen, G., Suci, D. (eds.) *DBPL 2003*. LNCS, vol. 2921, pp. 195–215. Springer, Heidelberg (2004). doi:[10.1007/978-3-540-24607-7_13](https://doi.org/10.1007/978-3-540-24607-7_13)
3. Braganholo, V., Mattoso, M.: A survey on XML fragmentation. *SIGMOD Rec.* **43**(3), 24–35 (2014)
4. Fegaras, L.: Incremental query processing on Big Data streams. *CoRR*, abs/1511.07846 (2015)
5. Handoko, Getta, J.R.: Dynamic query scheduling for online integration of semi-structured data. In: 2015 IEEE 39th Annual Computer Software and Applications Conference (COMPSAC), vol. 3, pp. 375–380, July 2015
6. Ma, H., Schewe, K.-D.: Fragmentation of XML documents. *J. Inf. Data Manage.* **1**(1), 21–33 (2010)
7. Özsu, T.M., Valduriez, P.: *Principles of Distributed Database Systems*, 3rd edn. Springer, Heidelberg (2011)
8. Wang, G., Huo, H., Han, D., Hui, X.: Query processing and optimization techniques over streamed fragmented XML. *World Wide Web* **11**(3), 339–359 (2008)
9. Wu, X., Theodoratos, D.: A survey on XML streaming evaluation techniques. *VLDB J.* **22**(2), 177–202 (2013)

Intelligent Information and Database Systems

9th Asian Conference, ACIIDS 2017, Kanazawa, Japan,

April 3-5, 2017, Proceedings, Part I

Nguyen, N.-T.; Tojo, S.; Nguyen, L.M.; Trawiński, B. (Eds.)

2017, XLIV, 817 p. 269 illus., Softcover

ISBN: 978-3-319-54471-7