STATIC OPTIMIZATION OF DATA INTEGRATION PLANS IN GLOBAL INFORMATION SYSTEMS

Janusz R. Getta
School of Computer Science and Software Engineering, University of Wollongong, Wollongong, Australia

Keywords: Data integration, Global information system, Multidatabase system, Online data integration, Integration plan, Static optimization.

Abstract: Global information systems provide its users with a centralized and transparent view of many heterogeneous and distributed sources of data. The requests to access data at a central site are decomposed and processed at the remote sites and the results are returned back to a central site. A data integration component of the system processes data retrieved and transmitted from the remote sites accordingly to the earlier prepared data integration plans.
This work addresses a problem of static optimization of data integration plans in a global information system. Static optimization means that a data integration plan is transformed into more optimal form before it is used for data integration. We adopt an online approach to data integration where the packets of data transmitted over a wide area network are integrated into the final result as soon as they arrive at a central site. We show how data integration expression obtained from a user request can be transformed into a collection of data integration plans, one for each argument of data integration expression. This work proposes a number of static optimization techniques that change an order operations, eliminate materialization and constant arguments from data integration plans implemented as relational algebra expressions.

1 INTRODUCTION

Efficient integration of data retrieved and transmitted from the remote sources is one of the central problems in the development of global information systems that provide the users with a centralized and transparent view of many heterogeneous and distributed sources of data. A data integration component of a global information system processes data retrieved from the remote sites and transmitted to a central site. A typical architecture of a global information system decomposes the user requests into the requests related to the remote source of data and submits the requests fro the processing at the remote sites. The results of processing at the remote sites are transmitted back to a central site and integrated with data already available there. A process of data integration acts upon a data integration plan which is prepared when a user's request is decomposed into the requests related to the remote sources. A data integration plan determines an order in which the individual requests are issued and a way how the results of these request are combined into the final result. The individual requests can be issued accordingly to *entirely sequential* or *entirely parallel*, or *mixed sequential and parallel strategies*. Accordingly to an *entirely sequential* strategy a request q_i can be submitted for processing at a remote site only when all results of the requests $q_1, \ldots, q_{i-1}$ are available at a central site. An entirely sequential strategy is appropriate when the results received so far can be used to reduce the complexity of the remaining requests $q_i, \ldots, q_{i+k}$. Accordingly to an *entirely parallel* strategy all requests $q_1, \ldots, q_i, \ldots, q_{i+k}$ are submitted simultaneously for the parallel processing at the remote sites. An entirely parallel strategy is beneficial when the computational complexity and the amounts of data transmitted is more or less the same for all requests. Accordingly to a *mixed sequential and parallel* strategy some requests are submitted sequentially while the others in parallel. Optimization of data integration plans is either *static* when the plans are optimized before a stage of data integration or it is *dynamic* when the plans are changed during the processing of the requests.

The problem of static optimization of data integration plans can be formulated in the following way. Given a global information system that integrates a number of remote and independent sources of data.

Consider a user request q to a global information system and its decomposition into the individual requests $q_1,\ldots,q_n$ simultaneously submitted for the processing at the remote sites. Let a request q be equivalent to an expression $\mathcal{E}(q_1,\ldots,q_n)$ over the individual requests. If the remote sites return the results $r_1,\ldots,r_n$ in the response to the individual requests $q_1,\ldots,q_n$ then the final result a user request q is equal to the result of an expression $\mathcal{E}(r_1,\ldots,r_n)$. Then, static optimization of data integration plan is equivalent to the optimization of an expression $\mathcal{E}(r_1,\ldots,r_n)$.

A naive and quite ineffective approach would be to postpone data integration until the results $r_1,\ldots,r_n$ returned from the remote sites are available at a central site. A more effective solution is to consider an individual reply r_i as a sequence of data packets $r_{i_1},r_{i_2},\ldots,r_{i_{k-1}},r_{i_k}$ and to perform data integration each time a new packet of data is received at a central site. Such approach to data integration is more efficient because there is no need to wait for the complete results when a data integration expression $\mathcal{E}(r_1,\ldots,r_n)$ is evaluated accordingly to a given order of operations. Instead, whenever a new packet of data is received at a central site it is immediately integrated into the intermediate result no matter which partial result it comes from. Then, static optimization of data integration plan finds the best processing strategy for the sequences of packets of data $r_{i_1},r_{i_2},\ldots,r_{i_{k-1}},r_{i_k}$ where $i=1,\ldots,n$. Such objective requires the transformation of data integration expression $\mathcal{E}(r_1,\ldots,r_i,\ldots,r_n)$ into the individual data integration plans for the sequences of packets $r_{i_1},r_{i_2},\ldots,r_{i_{k-1}},r_{i_k}$ where $i=1,\ldots,n$.

A starting point for the optimization is a data integration expression $\mathcal{E}(r_1,\ldots,r_i,\ldots,r_n)$ obtained from decomposition of a user request to a global information system. Processing of the individual packets means that an expression $\mathcal{E}(r_1,\ldots,r_i\oplus\delta_i,\ldots,r_n)$ must be recomputed each time a data packet δ_i is appended to an argument r_i. Of course reprocessing of the entire data integration expression is too time consuming and a better idea is to perform an incremental processing of the expression, i.e. to find how the previous result of an expression $\mathcal{E}(r_1,\ldots,r_i,\ldots,r_n)$ must be changed after δ_i is appended to an argument r_i. A data integration expression is transformed into a set of *data integration plans* where each plan represents an integration procedure for the increments of one argument of the original expression. In our approach a data integration plan is a sequence of so called *id-operations* on the increments or decrements of data containers and other fixed size containers. In order to reduces the size of arguments, static optimization of data integration plans moves the unary operation towards the beginning of a plan. Additionally, the frequently updated materializations are eliminated from the plan and constant arguments and subexpressions are replaced with the pre-computed values.

The paper is organized in the following way. First, we overview the related works in an area of optimization of data integration in distributed global information systems. Next, we derive the incremental processing of modification and we find a system of operation on modifications of data items for the system of operations included in the relational algebra. Transformation of data integration expressions into the sets of individual data integration plans is discussed in a section 4 and it is followed by presentation of static optimization of data integration plans in the next section. Finally, section 6 concludes the paper.

2 RELATED WORK

Optimization of data integration in global information systems can be traced back to optimization of query processing in multidatabase and federated database systems (Ozcan et al., 1997).

The external factors affecting the performance of query processing in multidatabase systems promote the *reactive query processing* techniques. The early works on the reactive query processing techniques are based on *partitioning* (Kabra and DeWitt, 1998) and *dynamic modification of query processing plans* (Getta, 2000). A dynamic modification technique finds a plan equivalent to the original one and such that it is possible to continue integration of the available data sets. The similar approaches dynamically change an order in which the join operations are executed depending on the arguments available at a central site. These techniques include *query scrambling* (Amsaleg et al., 1998) and *dynamic scheduling of operators* (Urhan and Franklin, 2001), and *Eddies* (Avnur and Hellerstein, 2000),

Optimization of relational algebra operations used for the data integration includes new versions of join operation customised to online query processing, e.g. *pipelined join operator XJoin* (Urhan and Franklin, 2000), *ripple join* (Haas and Hellerstein, 1999), *double pipelined join* (Ives et al., 1999), and *hash-merge join* (Mokbel et al., 2002).

A technique of *redundant computations* simultaneously processes a number of data integration plans leaving a plan that that provides the most advanced results (Antoshenkov and Ziauddin, 2000).

A concept of *state modules* described in (Raman et al., 2003) allows for concurrent processing of the tuples through the dynamic division of data integration tasks. *Adaptive data partitioning* (Ives et al., 2004) technique processes different partitions of the same argument using different data integration plans. The works on adaptive data partitioning (Ives et al., 2004) and optimization of data stream processing (Getta and Vossough, 2004) were the first attempts to use the associativity of join operation to integrate the separate partitions of the same arguments with the different integration plans. An adaptive and online processing of data integration plans proposed in (Getta, 2005) and later on in (Getta, 2006) considers the sets of elementary operations for data integration and the best integration plan for recently transmitted data.

Many of the techniques developed for the efficient processing of *data streams* (Getta and Vossough, 2004) can be applied to data integration. The reviews of the most important data integration techniques proposed so far are included in (Gounaris et al., 2002).

3 INCREMENTAL PROCESSING OF MODIFICATIONS

Initially we consider a process of data integration in the separation from any particular model of data. We adopt a general view of a database where a set of generic unary and binary operations processes a collection of data containers. We do not consider any particular internal structure of data containers and any particular system of operations on data containers. Data containers may represent relational tables, XML documents, set of persistent objects, files of records, etc.

3.1 Id-operations

Let $r_1, \ldots, r_k$ be data containers whose structure is consistent with a given model of data. A *generic operation* of the model is an operation $\mathcal{P}$ such that its arguments are the data containers r and s and whose result is another data container.

A *modification* of a data container r is denoted by δ and it is defined as a pair of disjoint data containers $<\delta^-, \delta^+>$ such that $r \cap \delta^- = \delta^-$ and $r \cap \delta^+ = \emptyset$.

An *data integration operation* that applies a modification δ to a data container r is denoted by $r \oplus \delta_i$. For example, in the relational model integration of a modification δ to a relational table r is defined as an expression $(r - \delta^-) \cup \delta^+$.

Consider a generic operation $\mathcal{P}(r,s)$ on the data containers r and s. An *incremental/decremental operation* later on called as an *id-operation* of an argument r of a generic operation $\mathcal{P}(r,s)$ is denoted by $\alpha_{\mathcal{P}}(\delta,s)$ and it is defined as the smallest modification $\delta_{\mathcal{P}}$ that should be integrated with the result of $\mathcal{P}(r,s)$ to obtain the result of $\mathcal{P}(r \oplus \delta, s)$, i.e.

$$\mathcal{P}(r,s) \oplus \alpha_{\mathcal{P}}(\delta, r) = \mathcal{P}(r \oplus \delta, s) \quad (1)$$

An *incremental/decremental operation* of an argument s of a generic operation $\mathcal{P}(r,s)$ is denoted by $\beta_{\mathcal{P}}(r,\delta)$ and it is defined as the smallest modification $\delta_{\mathcal{P}}$ that should be integrated with the result of $\mathcal{P}(r,s)$ to obtain the result of $V(r, s \oplus \delta)$, i.e.

$$\mathcal{P}(r,s) \oplus \beta_{\mathcal{P}}(r,\delta) = \mathcal{P}(r, s \oplus \delta) \quad (2)$$

If a generic operation $\mathcal{P}(r,s)$ is a component of an integration expression then *id-operations* allow for faster re-computation of $\mathcal{P}(r,s)$ when one of its arguments is integrated with data transmitted from an external data site An ineffective approach would be to integrate the transmitted data with an argument and to re-compute entire generic operation. It is represented by the right hand sides of the equations (1) and (2).

A better idea is to apply an id-operation to transmitted data and the other argument of the base operation to get a modification that can be integrated with the previous result of generic operation. It is represented by the left hand sides of the equations (1) and (2). The application of id-operations speeds up data integration because it is possible to immediately process data received at a central site. Id-operations allow for the incremental processing of data integration expressions such that an increment of one of the arguments triggers the computations of a sequence of id-operations that return a modification, which should be applied to the final result of integration.

3.2 Relational Algebra based Id-operations

Let x be a nonempty set of attribute names later on called as a *schema* and let $dom(a)$ denotes a domain of attribute $a \in x$. A *tuple* t defined over a schema x is a full mapping $t : x \to \cup_{a \in x} dom(a)$ and such that $\forall a \in x, t(a) \in dom(a)$. A *relational table* created on a schema x is a set of tuples over a schema x.

Let r, s be the relational tables such that $schema(r) = x$, $schema(s) = y$ respectively and let $z \subseteq x$, $v \subseteq (x \cap y)$, and $v \neq \emptyset$. The symbols σ_ϕ, π_z, $\bowtie_v$, $\sim_v$, $\ltimes_v$, $\cup$, $\cap$, $-$ denote the relational algebra operations of *selection*, *projection*, *join*, *antijoin*, *semijoin*, and set algebra operations of *union*, *intersection*, and *difference*. All join operations are considered to be equijoin operations over a set of attributes v.

To find the analytical solutions of the equations (1) and (2) we we assume that a data integration operation is computed in the relational model in the following way.

$$r \oplus \delta = (r - \delta^-) \cup \delta^+. \quad (3)$$

Then, the *id-operations* $\alpha_{\mathcal{P}}$ and $\beta_{\mathcal{P}}$ can be decomposed into the pairs of operations each one acting on either negative (δ^-) or positive (δ^+) component of a modification δ.

$$\alpha_{\mathcal{P}}(\delta, s) = <\alpha^-_{\mathcal{P}}(\delta^-, s), \alpha^+_{\mathcal{P}}(\delta^+, s)>, \qquad (4)$$

$$\beta_{\mathcal{P}}(r, \delta) = <\beta^-_{\mathcal{P}}(r, \delta^-), \beta^+_{\mathcal{P}}(r, \delta^+)>. \qquad (5)$$

If we separately consider the negative and positive components of a modification δ and we replace data integration operation with its relational definition then we get the following equations.

$$\mathcal{P}(r, s) - \alpha^-_{\mathcal{P}}(\delta^-, s) = \mathcal{P}(r - \delta^-, s) \qquad (6)$$

$$\mathcal{P}(r, s) \cup \alpha^+_{\mathcal{P}}(\delta^+, s) = \mathcal{P}(r \cup \delta^+, s) \qquad (7)$$

$$\mathcal{P}(r, s) - \beta^-_{\mathcal{P}}(r, \delta^-) = \mathcal{P}(r, s - \delta^-) \qquad (8)$$

$$\mathcal{P}(r, s) \cup \beta^+_{\mathcal{P}}(r, \delta^+) = \mathcal{P}(r, s \cup \delta^+) \qquad (9)$$

To find the *id-operations* we solve the equations above for the generic operations of union ($\cup$), join ($\bowtie$), and antijon ($\sim$) and we assume that selection operation is always directly applied to the arguments of binary operations and projection is applied only one time to the final result of query processing.

The analytical solutions of the equations (6) and (7) provide the following results.

$$\alpha_{\cup}(\delta, s) = <\delta^- - s, \delta^+ - s> \qquad (10)$$

$$\beta_{\cup}(r, \delta) = <\delta^- - r, \delta^+ - r> \qquad (11)$$

The derivations of *id-operation* for the generic operations of join and antijoin can be obtained in the same way.

$$\alpha_{\bowtie}(\delta, s) = <\delta^- \bowtie_v s, \delta^+ \bowtie_v s> \qquad (12)$$

$$\beta_{\bowtie}(r, \delta) = <\delta^- \bowtie_v r, \delta^+ \bowtie_v r> \qquad (13)$$

$$\alpha_{\sim}(\delta, s) = <\delta^- \sim_v s, \delta^+ \sim_v s> \qquad (14)$$

$$\beta_{\sim}(r, \delta) = <r \ltimes_v \delta^+, r \ltimes_v \delta^-> \qquad (15)$$

3.3 Application of Id-operations to Data Integration

Consider a data integration expression $\mathcal{E}(r, s, t) = t \sim_v (r \bowtie_z s)$ where r, s, t are the remote data sources and $v = schema(r) \cap schema(t)$ and $z = schema(r) \cap schema(s)$. Assume, that a new increment $\delta_s = <\emptyset, \delta^+_s>$ of an argument s has been just transmitted to a central site. We would like like to recompute a data integration expression $\mathcal{E}(r, s \oplus \delta_s, t)$ immediately after the integration of an increment δ_s with an argument s.

To avoid re-computation of entire data integration expression we first find a modification δ_{rs} that should be applied to a result of $r \bowtie_z s$ after the extension of an argument s with δ_s. Next, we find a modification δ_{rst} that should be applied to the result of $t \sim_v (r \bowtie_z s)$ after modification δ_{rs} is applied to the result of $r \bowtie_z s$ From the equation (13) we get $\delta_{rs} = <\emptyset, \delta^+_s \bowtie r>$. Next, to find δ_{rst} we find $\beta_{\sim}(t, \delta_{rs})$. From the equation (15) we get $\delta_{rst} = <t \ltimes_v \delta^+_{rs}, r \ltimes_v \emptyset>$. Finally, $\delta_{rst} = <t \ltimes_v (\delta^+_s \bowtie r), \emptyset>$. Hence, in order to get the result of $\mathcal{E}(r, s \oplus \delta_s, t)$ after the extension of s with δ_s we have to compute $\mathcal{E}(r, s, t) - (t \ltimes_v (\delta^+_s \bowtie r))$.

Next, we consider the same data integration expression $t \sim_v (r \bowtie_z s)$ and a new increment δ_t of a remote data source t. Now, processing of an increment δ_t needs either materialization of an intermediate result of a subexpression $(r \bowtie_z s)$ or transformation of the data integration expression into an equivalent one with either left (right)-deep syntax tree and with an argument t in the leftmost (rightmost) position of the tree. Materialization of an intermediate results decreases the overall performance because when one of its arguments is extended then entire subexpression of materialization must be re-computed. On the other hand it is not always possible to transform an integration expression into a left or right deep syntax tree such that a modification is located at the lowest leaf level of the tree.

If materialization $m_{rs} = r \bowtie s$ is available then from an equation (14) we get $\delta_{rst} = <\emptyset, \delta^+_t \sim_v m_{rs}>$. Hence, in order to get the result of $\mathcal{E}(r, s, t \oplus \delta_t)$ after the extension of t we have to compute $\mathcal{E}(r, s, t) \cup (\delta^+_t \sim_v m_{rs})$.

If materialization m_{rs} is not available then an interesting option is to transform an expression $\delta^+_t \sim_v (r \bowtie s))$ into $\delta^+_t \sim_v ((r \ltimes \delta^+_t) \bowtie s)$. Such transformation is correct because we do not need the entire result of $r \bowtie s$ to be computed, we only need the rows from r that can be joined with δ^+_t over the attributes in v. A subexpression $r \ltimes \delta^+_t$ will reduce the size of an argument r before join with s and its computation can be done faster because δ^+_t is small.

4 DATA INTEGRATION PLANS

In this section we introduce a concept of *data integration plan* and we show how to transform a data integration expression into a set of data integration plans. Let $\mathcal{E}(r_1, \dots r_n)$ be a data integration expression built over the generic operations and data containers $r_1 \dots, r_n$. A syntax tree T_e of an expression $\mathcal{E}$ is a binary tree such that:

(i) for each instance of argument $r_1, \dots r_n$ there is a leaf node are labelled with a name of argument,

(ii) for each subexpression $\mathcal{P}(e', e'')$ of $\mathcal{E}$ where $\mathcal{P}$ is a generic operation and e' and e'' are the subexpression of $\mathcal{E}$ there is a node labelled with $\mathcal{P}$ that has two subtrees $T_{e'}$ and $T_{e''}$.

A data integration plan is a sequence of assignment statements $s_1, \ldots, s_m$ where the right hand side of each statement is either an application of a modification to a data container ($m_j := m_j \oplus \delta_i$) or an application of left or right id-operation ($\delta_j := \alpha_j(\delta_i, m_k)$).

Consider an argument r_i of a data integration expression $\mathcal{E}$. An implementation of data integration expression for an argument r_i is constructed in the following steps.

1: Assign unique numbers to each node of a syntax tree T_e.

2: Make an implementation p_i empty.

3: Start from a leaf node of T_e labelled with r_i.

4: While not in the root node of T_e move to ancestor node of the current node and execute a procedure $moveToAncestor(i, j)$ where i is an identifier of the current node and j is an identifier of the ancestor node.

5: When in the root node i append a statement $result := result \oplus \delta_i$;

A procedure $moveToAncestor(i, j)$ consists of the following steps.

1: If i is a leaf node and it is the left descendant of a node j then append a statement that computes id-operation $p_i : \delta_j := \alpha_{operation_j}(\delta_{r_i}, m_k)$; where m_k is either a data container or a materialization in the right descendant of node j.

2: If i is a leaf node and it is the right descendant of a node j then append a statement that computes id-operation $p_i : \delta_j := \beta_{operation_j}(m_k, \delta_{r_i})$; where m_k is either a data container or a materialization in the left descendant of node j.

3: If i is not a leaf node of T_e then append a statement $m_i := m_i \oplus \delta_i$; where m_i is a materialization in a node i.

4: If i is not a leaf node and it is the left descendant of a node j then append a statement that computes id-operation $\delta_j := \alpha_{operation_j}(\delta_i, m_k)$; where m_k is either a data container or a materialization in the right descendant of node j.

5: If i is not a leaf node and it is the right descendant of a node j then append a statement that computes id-operation $\delta_j := \beta_{operation_j}(m_k, \delta_i)$; where m_k is either data container or a materialization in the left descendant of node j.

As a simple example consider a data integration expression $\mathcal{P}(r,s,t) = t \bowtie_v (r \sim_w s)$ and its syntax tree

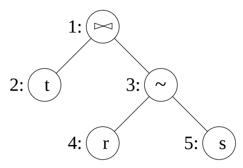

Figure 1: A syntax tree of an expression $t \bowtie_v (r \sim_w s)$.

with the numbered nodes given in Figure (1). Starting from a node 4 we get a data integration plan for the modifications of argument r:
$p_4 : \delta_3 := \alpha_{\sim}(\delta_r, s)$;
$m_3 := m_3 \oplus \delta_3$;
$\delta_1 := \beta_{\bowtie}(t, \delta_3)$;
$result := result \oplus \delta_1$;
An equivalent relational data integration plan is obtained by the substitution of id-operations with the equivalent relational algebra expressions.
$p_4 : \delta_3 := \delta_r \sim s$;
$m_3 := m_3 \oplus \delta_3$;
$\delta_1 := t \bowtie \delta_3$;
$result := result \oplus \delta_1$;
The relational algebra operations on the relational tables and modifications process both negative and positive components of the modifications. For example, $\delta_3 := \delta_r \sim s$; is equivalent to $\delta_3^- := \delta_r^- \sim s$; $\delta_3^+ := \delta_r^+ \sim s$;
The data integration plans for the arguments s and t are the following.
$p_5 : \delta_3 := \beta_{\sim}(r, \delta_s)$;
$m_3 := m_3 \oplus \delta_3$;
$\delta_1 := \beta_{\bowtie}(t, \delta_3)$;
$result := result \oplus \delta_1$;

$p_3 : \delta_1 := \alpha_{\bowtie}(\delta_t, m_3)$;
$result := result \oplus \delta_1$;

5 STATIC OPTIMIZATION OF DATA INTEGRATION PLANS

Static optimization of data integration plan means that transformation of the plans is performed before a stage of data integration while dynamic optimization of data integration plans changes the plans during a stage of data integration. In order to get more sound results we consider a language of data integration expressions to be the relational algebra and we consider data integration expressions built from the operations of *join*, *antijoin*, and *union* only. We also assume, that operation of *selection* is always processed together with an adjacent binary operation and *projection* is computed at the very end of data integration.

5.1 Preliminary Optimizations

The preliminary optimizations are performed before the transformation of data integration expression into data integration plans and it includes the standard transformations of relational algebra expressions where the selections and projections are "pushed" down the syntax trees of data integration expression and join operations are reordered such that the joins on the small arguments are performed first. Next, the selection operations are associated with the binary operations such that the rows that satisfy a selection condition are directly "piped" to the first stage of the computations of the binary operations. For example, if a join operation implemented as hash-based join follows a selection then a row that satisfies a selection condition is not saved in a temporary results of selection and instead is hashed in the first stage of the computations of a hash-based join. The same techniques is applied to the selection operations that cannot be "pushed" down below the binary operations, for example $\sigma_{a>c}(r(ab) \bowtie_b s(bc))$ are computed by "piping" the rows obtained as the results of binary operation $r(ab) \bowtie_b s(bc)$ directly to a selection operations. It is also possible to implement a selection operation as an additional comparison when the rows from the arguments are matched during the processing of binary operation, for example in the example above, testing of equality condition $r.b = s.b$ can be followed by testing of a condition $r.a > s.c$. After the preliminary stage of optimizations data integration expressions are transformed into data integration plans.

5.2 Optimization through Reordering of Operations

The following example shows how further optimization of data integration plans can be achieved through reordering of the operations. Consider a fragment of data integration plan $p_t : \delta_j := \delta_t \bowtie_v r$; $m_j := m_j \oplus \delta_j$; $\delta_k := \delta_j \sim_w s$; The following two observations lead to the transformations that may have a positive impact on the performance of data integration plans. First, if a data container r is significantly larger than a data container s then it would be more efficient to start from the computations on a data container s because the results would be smaller. Second, some of the data items in δ_j may not contribute to the results of $\delta_k := \delta_j \sim_w s$ and can be removed from δ_j before the computations of *antijoin* operation. We compute an operation $\delta_i \div s$ to partition both s and δ_j into pairs $<s^{(\delta_j+)}, s^{(\delta_j-)}>$ and $<\delta_j^{(s+)}, \delta_j^{(s-)}>$ such that only $s^{(\delta_j+)}$ and $\delta_j^{(s+)}$ have an impact on the result of $\delta_k := \delta_j \sim_w s$; The partitioning is performed such that $\delta_i^{(s+)} := \delta_i \ltimes s$, $\delta_i^{(s-)} := \delta_i \sim s$, $s^{(\delta_i+)} := s \ltimes \delta_i$, and $s^{(\delta_i-)} := s \sim \delta_i$. Then, we compute $<\delta_i^{(s+)}, \delta_i^{(s-)}> \bowtie r$ and later on the result of $\delta_i^{(s-)} \bowtie r$ is directly passed to δ_k and it is unioned with with the result of $(\delta_i^{(s+)} \bowtie r) \sim s^{(\delta_i+)}$.

The complexity of the partitioning $\delta_i \div s$ is comparable with the complexity of an ordinary join operation. The complexity of the computations of $<\delta_i^{(s+)}, \delta_i^{(s-)}> \bowtie r$ is the same as the complexity of $\delta_i \bowtie r$. It is expected that the additional computations of $\delta_i \div s$ will take less time than the difference between the computations of $(\delta_i \bowtie r) \sim s$ and the computations of $(\delta_i^{(s+)} \bowtie r) \sim s^{(\delta_i+)}$. The benefits depend on how far the partitioning reduces the size of $s^{(\delta_i+)}$ and $(\delta_i^{(s+)} \bowtie r)$.

5.3 Elimination of Materializations

An algorithm that transforms a syntax tree of data integration expression into the data integration plans creates the references to so called *materializations*. Materialization is a relational table that contains the intermediate results of processing of one of subexpression of a data integration plan. Materializations are needed when when an online processing plan is created for an argument which is not at the bottom level of a syntax tree or a syntax tree is not left-/right-deep syntax tree. Materializations require the additional integration operations in online processing plans and because of that frequently performed integrations of the partial results with materializations may consume a lot of additional time. To avoid this problem we find the ways how to remove materializations from data integration plans.

Consider an integration expression $t \bowtie_w (r \bowtie_v s)$. An integration plan for an argument t, i.e. $p_t : \delta_{trs} := \delta_t, \bowtie_w m_{rs}$; $result := result \oplus \delta_{trs}$; uses a materialization m_{rs} which contains the intermediate results of a subexpression $r \bowtie_v s$. The plans for the arguments r and s integrate the intermediate results of the same subexpression with the materialization m_{rs}, for example $p_r : \delta_{rs} := \delta_r \bowtie_v s$; $m_{rs} := m_{rs} \oplus \delta_{rs}$; $\delta_{rst} := \delta_{rs}, \bowtie t$; $result := result \oplus \delta_{rst}$; A simple solution to eliminate a materialization m_{rs} is to apply the associativity of join operation and to transform the expression into an expression which has left-/right-deep syntax tree and argument t is located at at one of the bottom leaf level nodes of the tree. To do so, we transform the integration expression into $(t \bowtie_w r) \bowtie_v s$ and we create a new integration plan $p'_t : result := result \oplus (\delta_t \bowtie r) \bowtie_v s$;

Additionally, we eliminate the integration of partial results with a materialization m_{rs} from the integration plans for r and s.

The next example shows, that associativity of the operations involved in an expression is not a necessary condition for the elimination of materialization. We consider a data integration expression $(r \sim_v s) \bowtie_w t$. The objective is to eliminate a materialization that contains the intermediate results of $r \sim_v s$ from an integration plan for the argument t, i.e. $p_t : \delta_{rst} := m_{rs} \bowtie \delta_t;\ result \oplus \delta_{rst}$. We transform an expression $m_{rs} \bowtie \delta_t$ into a form where a modification δ_t is located at the bottom level of left-/right-deep syntax tree in the following way. First we substitute m_{rs} with $r \sim_v s$. Next, we replace an argument r with an equivalent expression $(r \ltimes_y \delta_t) + (r \sim_y \delta_t)$ where operation $+$ denotes a concatenation of the disjoint results of semijoin and antijoin operations. The distributivity of concatenation operation over semijoin and join operations allows us to transform an expression $(((r \ltimes_y \delta_t) + (r \sim_y \delta_t)) \sim_v s) \bowtie_w \delta_t$ into concatenation of two subexpressions $((r \ltimes_y \delta_t) \sim_v s) \bowtie_w \delta_t + ((r \sim_y \delta_t) \sim_v s) \bowtie_w \delta_t$. The results of processing a subexpression $((r \sim_y \delta_t) \sim_v s) \bowtie_y \delta_t$ are always empty because y-values removed from r by $(r \sim_y \delta_t$ return no results when join with δ_t is performed later on. We get an expression $((r \ltimes_y \delta_t) \sim_v s) \bowtie_w \delta_t$ equivalent to an expression $m_{rs} \bowtie \delta_t$ in the original integration plan p_t. Because the transformations above eliminated a materialization $m_r s$ from p_t it is possible to eliminate it from the remaining plans p_r and p_s. Hence, the data integration plans for an integration expression $(r \sim_v s) \bowtie_w t$ are as follows.
$p_r : result := result \oplus (\delta_r \sim_v s) \bowtie_w t;$
$p_s : result := result \oplus (r \ltimes_v \delta_s) \bowtie_w t;$
$p_t : result := result \oplus (((r \ltimes_y \delta_t) \sim_v s) \bowtie_w \delta_t);$

Next, we discuss how to eliminate materialization in a more general case. Consider an argument r whose integration plan uses a materialization. If r has at least one common attribute with a modification of δ_s of another argument s of integration expression than it is always possible to replace r with $(r \ltimes_v \delta_s) + (r \sim_v \delta_s)$. Then it is possible to apply distributivity of concatenation operation and to eliminate one of the components of the expression later on like in the example above. A problem is how to find when such transformation is possible. Consider an implementation of online processing plan where an operation $p_z(\delta(x), m(y))$ acts on a modification $\delta(x)$ and materialization $m(y)$ such that $x \cap y = z$ and $z \neq \emptyset$. It is possible to eliminate materialization $m(y)$ from the online processing plan when there exists an argument $s(v)$ of subexpression of materialization $m(y)$ (see Figure 2) such that:

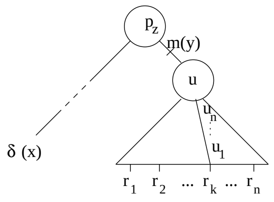

Figure 2: Elimination of materialization $m(y)$.

(i) $v \cap x \neq \emptyset$ and

(ii) $(v \cap x) \in z$.

Elimination of materialization $m(y)$ is performed by the substitution of $s(v)$ in a subexpression of the materialization with $(s(v) \ltimes_{v \cap x} \delta(x)) + (s(v) \sim_{v \cap x} \delta(x))$. Then processing of modification $\delta(x)$ triggers the computations along a path that leads from the processing of semijoin and antijoin of $s(v)$ with $\delta(x)$ to a materialization $m(y)$. Unfortunately, it does not solve the problem from performance point of view . The substitution of $s(v)$ with a concatenation of semijoin and antijoin of $s(v)$ with a modification $\delta(x)$ still provides a complete $s(v)$ and requires the reprocessing of entire materialization $m(y)$. In fact, when modification $\delta(x)$ is small then only a fraction of materialization $m(y)$ affects the result of $p_z(\delta(x), m(y))$. Then, a solution would be to recompute only such component of materialization that affect the result of operation p_z. If it is possible to eliminate one of semijoin of $s(v)$ with $\delta(x)$ or antijoin of $s(v)$ with $\delta(x)$ then only a subset of argument $s(v)$ is involved in the processing. Next, we show a formal method that finds when a materialization can be removed and what transformations of the arguments of a relational implementation of data integration plan are required to do so. Let T_e be a syntax tree of a relational algebra expression $e(r_1, \ldots, r_n)$ built over the operations of *set difference*, *join*, *semijoin*, and *antijoin*. Let a node n_p in T_e represents a binary operation $p_v(r(x), s(y))$ such that $v = x \cap y$. Labelling of T_e is performed in the following way.

(i) An edge between a leaf node that represent an argument $r(x)$ can be labeled with z_r where $z \subseteq x$ and $z \neq \emptyset$.

(ii) If a node n_p in T_e represents an operation p that produces a result $r(x)$ and "child" edge of a node n_p is labeled with one of the symbols z, $z-$, $-z$, $z*$ then a "parent" edge of n_p can be labeled with a symbol located in a row indicated by a label of "child" edge and a column indicated by an operation p_v in a Table 1.

Labelling of syntax tree is performed to discover the types of coincidences between the z-values of one or more arguments of relational algebra expression. The

Table 1: The labelling rules for syntax trees of relational algebra expressions.

	$\bowtie_v$	$(left) \sim_v$	$\sim_v (right)$	$(left) \ltimes_v$	$\ltimes_v (right)$	$(left)-$	$-(right)$
z	$z-$	$z-$	$-(z \cap v)$	$z-$	$(z \cap v)-$	$z-$	$-z$
$z-$	$z-$	$z-$	$(z \cap v)*$	$z-$	$(z \cap v)-$	$z-$	$z*$
$-z$	$-z$	$-z$	$(z \cap v)*$	$-z$	$(z \cap v)*$	$-z$	$z*$
$z*$	$z*$	$z*$	$(z \cap v)*$	$z*$	$z*$	$z*$	$z*$

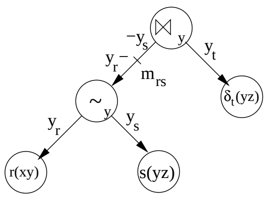

Figure 3: A labelled syntax tree of online processing plan $p_t : result := result \oplus ((r(xy) \sim_y s(yz)) \bowtie_y \delta_t(yz))$.

coincidences and their types are needed to find out if it is possible to remove the materializations and whether their elimination is beneficial.

As an example, consider an integration expression $(r(xy) \sim_y s(yz)) \bowtie_y t(yz)$ and an integration plan $p_t : result := result \oplus (m_{rs}(xy) \bowtie_y \delta_t(yz))$; for processing the increments of an argument t. A materialization is computed as $m_{rs}(xy) = r(xy) \sim_y s(yz)$. A syntax tree of the plan with the materialization m_{rs} replaced with $r(xy) \sim_y s(yz)$ is given in Figure 3. To eliminate the materialization we try to find the coincidences between y-values of $r(xy)$ and $\delta_t(yz)$ and we perform the labelling of the syntax tree in a way described above. The "parent" edges of the nodes $r(xy)$, $s(yz)$, and $\delta_t(yz)$ obtain the labels y_r, y_s, and y_t. A left "child" edge of the root node obtained a label y_r- indicated by a location in the first row and the second column in Table 1. Moreover, the same edge obtained a label $-y_s$ indicated by a location in the first row and the third column in Table 1. The final labelling of the syntax tree is given in Figure 3. The interpretations of the labels are the following. A label y_t attached to a "child" edge of join operation at root node of the tree indicate that all y-values of an argument $\delta_t(yz)$ are processed by the operation. A label y_r- attached to a "child" edge of the same operation indicates that only a subset of y-values of an argument $r(xy)$ and no other y-values are processed by the operation. A label $-y_s$ attached to the same edge indicates that none of y-values in $s(y,z)$ is included in the result of $r(xy) \sim_y s(yz)$. The above interpretation of the labels y_r- and y_t in a context of join operation over a set of attributes y means that y-values not included in the arguments $r(xy)$ and $\delta_t(yz)$ have no impact on the result of join operation. It means that $r(xy)$ can be replaced with $r(xy) \ltimes_y \delta_t(yz)$ and $\delta_t(yz)$ can be replaced with $\delta_t(yz) \ltimes_y r(xy)$ without changing the result of the expression. The interpretation of the labels $-y_s$ and y_t in a context of join operation allows for the elimination from $\delta_t(yz)$ of all y-values, which are included in $s(yz)$ because these values have no impact on join operation. It means that $\delta_t(yz) \ltimes_y r(xy)$ can be replaced with $(\delta_t(yz) \ltimes_y r(xy)) \sim_y s(yz)$ with changing the result of the expression. It is also possible to replace $s(yz)$ with $s(yz) \ltimes \delta_t(yz)$ because all y-values included in $s(yz)$ and not included in $\delta_t(yz)$ have no impact on the result of join operation. However, the last modification is questionable from a performance point of view. It definitely, speeds up antijoin operation but it also delays join operation because the results of antijoin operation are larger after the reduction of $s(y,z)$.

The labelling and the possible replacements of arguments are summarized in the Tables 2 and 3. The interpretations of the Tables are the following. Consider a relational algebra expression $e(r, r_1, \ldots, r_n, s)$ such that operation p_x is included in the root node of its syntax T_e. If an operation p_x is either join or semijoin operations then the possible replacements of the arguments r and s are included in a Table 2. If an operation p_x is either antijoin or set difference the the possible replacements are included in a Table 3. The replacements of the arguments r and s over a common set of attributes $z \subseteq x$ can be found after the labelling of both paths from the leaf nodes representing the arguments r and s towards the root node of T_e labeled with p_x. The replacements of the arguments r and s are located at the intersection of a row labeled with a label of left "child" edge and a column labeled with a label of "right" child edge of the root node. For instance, consider a subtree of the arguments s and r such that an operation $\bowtie_x$ is in the root node of the subtree. If a left "child" edge of the root node is labeled with $-z_r$, and a right "child" edge of the root node is labeled with z_s* then Table 2 indicates that it is possible to replace the contents of an argument s with an expression $s \sim_z r$. A sample justification of the replacements included in the Table 2 at the intersection of a row labeled with $-z_r$ and a column labeled with z_s* is the following. Let T_e be a syntax tree of a relational algebra expression $e(r, r_1, \ldots, r_n, s)$ built of the operations of join, semijoin, antijoin, and set difference, and such that root node of the tree is labeled with either $\bowtie_x$ or $\ltimes_x$ and

Table 2: The replacements of arguments in integration plans.

$\bowtie_x, \ltimes_x$	z_s	z_s-	$-z_s$	z_s*
z_r	n/a	$r \ltimes_z s$ $s \ltimes_z r$	$r \sim_z s$ $s \ltimes_z r$	$s \ltimes_z r$
z_r-	$r \ltimes_z s$ $s \ltimes_z r$	$r \ltimes_z s$ $s \ltimes_z r$	$r \sim_z s$ $s \sim_z r$	$s \ltimes_z r$
$-z_r$	$s \sim_z r$ $r \ltimes_z s$	$s \sim_z r$ $r \sim_z s$	*either* $s \sim_z r$ *or* $r \sim_z s$	$s \sim_z r$
z_r*	$r \ltimes_z s$	$r \ltimes_z s$	$r \sim_z s$	*none*

Table 3: The replacements of arguments in integration plans.

$\sim_x, -$	z_s	z_s-	$-z_s$	z_s*
z_r	n/a	$s \ltimes_z r$	$s \ltimes_z r$	$s \ltimes_z r$
z_r-	$s \ltimes_z r$	$s \ltimes_z r$	$s \ltimes_z r$	$s \ltimes_z r$
$-z_r$	*either* $s \sim_z r$ *or* $r \sim s$	$s \sim_z r$	*either* $s \sim_z r$ *or* $r \sim s$	$s \sim_z r$
z_r*	$r \sim_z s$	$r \sim_z s$	*none*	*none*

its left ”child” edge is labeled with $-z_r$ and its right ”child” edge is labeled with z_s*, $z \subseteq x$. Then, for any values of the arguments $r, r_1, \ldots, r_n, s$ an expression $e(r, r_1, \ldots, r_n, s) = e(r, r_1, \ldots, r_n, (s \sim_z r))$. A label $-z_r$ attached to a left ”child” edge of join of operation $\bowtie_x$ or $\ltimes_x$ means that none of z-values of an argument r is include in an argument of join or semijoin. Then, these z-values can be removed from an argument s because they will never participate in join or semijoin operation. On the other hand we cannot replace an argument r because label z_s* means that some new z-values can be added to the original set of z-values in s.

Let $e_m(r_1, \ldots, r_n)$ be an expression that defines a materialization m in an integration plan for the increments δ_s of an argument s. Elimination of materialization m from integration plan for δ_s is possible when some of the arguments $r_1, \ldots r_n$ can be replaced with the subexpressions involving δ_s such that syntax tree of $e'_m(r_1, \ldots, r_n, \delta_s)$ does not contain a subexpression that does not involves δ_s. In the other words, we try to replace some of the arguments in an expression that defines a materialization such that entire expression can be recomputed with an argument δ_s and no subexpression exists that does not involve δ_s.

An interesting problem is whether any materialization can be removed using the replacements described above. The analysis of the Tables 2 and 3 and the structural properties of relational implementations of integration plans reveal three cases when materializations cannot be removed through the replacements.

(1) An operation at the root node syntax tree of online processing plan is a join operation and its both left and right ”child” edges are labeled with z_r* and z_s* respectively, see a location in the right lower corner of Table 2.

(2) An operation at the root node syntax tree of online processing plan is either a join operation or semijoin operation, a materialization is the fist argument of the operation and right ”child” edge is labeled with z_s*. This is because all reduction in the last column of 2 are applicable to the second argument of the operation which is obtained from the processing of modification and not materialization.

(3) An operation at the root node of a syntax tree of online processing plan is either an antijoin operation or difference operation, materialization is the second argument of the operation and left ”child” edge of the node is labeled with z_r*. This is because all replacements in the last row of Table 3 are applicable to the first argument of the operation which is obtained from the processing of modification and not materialization.

6 SUMMARY, CONCLUSIONS, AND FUTURE WORK

This work addresses a problem of static optimization of data integration plans in the global information systems. The users' requests submitted at a central site are decomposed into the individual requests and simultaneously submitted for processing at the remote sites. We show how data integration plans for the increments of the individual arguments can be derived from a data integration expression and we propose a number of static optimization techniques for data integration plans implemented as relational algebra expressions.

A technique of immediate processing of data packets as they are received from the remote sites allows for better utilization of data processing resources available at a central site. The continuous processing of small portions of data transmitted from the remote sites eliminates idle time when a data integration system has to wait for the transmission of an entire argument. Decomposition of data integration expression into the individual plans allows for more precise optimization of data integration and it also allows for better scheduling of data processing on multiprocessor systems. Identification of coincidences between the arguments of data integration expression leads to elimination of materializations from data integration plans and reduction of the processing load when materializations are frequently change.

A number of problems remains to be solved. Elimination of materialization from data integration plans depends on the parameters of transmission of the arguments and a problem is how predict these parameters at static optimization phase. Another interesting problem is identification of all materializations that can be eliminated in a given moment of time and scheduling of the replacements in a process of online data integration. The other problems include the derivations of more sophisticated systems of id-operations from the systems of binary operations different from the relational algebra e.g. a system including aggregation operations, further investigations of the properties of data integration plans and more advanced data integration algorithms where the application of a particular online plan depends on what increments of data are available at the moment.

REFERENCES

Amsaleg, L., Franklin, J., and Tomasic, A. (1998). Dynamic query operator scheduling for wide-area remote access. *Journal of Distributed and Parallel Databases*, 6:217–246.

Antoshenkov, G. and Ziauddin, M. (2000). Query processing and optmization in oracle rdb. *VLDB Journal*, 5(4):229–237.

Avnur, R. and Hellerstein, J. M. (2000). Eddies: Continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 261–272.

Getta, J. R. (2000). Query scrambling in distributed multidatabase systems. In *11th Intl. Workshop on Database and Expert Systems Applications, DEXA 2000*.

Getta, J. R. (2005). On adaptive and online data integration. In *Intl. Workshop on Self-Managing Database Systems, 21st Intl. Conf. on Data Engineering, ICDE'05*, pages 1212–1220.

Getta, J. R. (2006). Optimization of online data integration. In *Seventh International Conference on Databases and Information Systems*, pages 91–97.

Getta, J. R. and Vossough, E. (2004). Optimization of data stream processing. *SIGMOD record*, 33(3):34–39.

Gounaris, A., Paton, N. W., Fernandes, A. A., and Sakellariou, R. (2002). Adaptive query processing: A survey. In *Proceedings of 19th British National Conference on Databases*, pages 11–25.

Haas, P. J. and Hellerstein, J. M. (1999). Ripple joins for online aggregation. In *SIGMOD 1999, Proceedings ACM SIGMOD Intl. Conf. on Management of Data*, pages 287–298.

Ives, Z. G., Florescu, D., Friedman, M., Levy, A. Y., and Weld, D. S. (1999). An adaptive query execution system for data integration. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 299–310.

Ives, Z. G., Halevy, A. Y., and Weld, D. S. (2004). Adapting to source properties in processing data integration queries. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*.

Kabra, N. and DeWitt, D. J. (1998). Efficient mid-query reoptimization of sub-optimal query execution plans. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*.

Mokbel, M. F., Lu, M., and Aref, W. G. (2002). Hash-merge join: A non-blocking join algorithm for producing fast and early join results.

Ozcan, F., Nural, S., Koksal, P., Evrendilek, C., and Dogac, A. (1997). Dynamic query optimization in multidatabases. *Bulletin of the Technical Committee on Data Engineering*, 20:38–45.

Raman, V., Deshpande, A., and Hellerstein, J. M. (2003). Using state modules for adaptive query processing. In *Proceedings of the 19th International Conference on Data Engineering*, pages 353–.

Urhan, T. and Franklin, M. J. (2000). Xjoin: A reactively-scheduled pipelined join operator. *IEEE Data Engineering Bulletin 23(2)*, pages 27–33.

Urhan, T. and Franklin, M. J. (2001). Dynamic pipeline scheduling for improving interactive performance of online queries. In *Proceedings of International Conference on Very Large Databases, VLDB 2001*.