University of Wollongong

Research Online

Faculty of Informatics - Papers (Archive)

Faculty of Engineering and Information
Sciences

5-4-2005

On Adaptive and Online Data Integration

J. R. Getta

University of Wollongong, jrg@uow.edu.au

Follow this and additional works at: <https://ro.uow.edu.au/infopapers>

Part of the [Physical Sciences and Mathematics Commons](#)

Recommended Citation

Getta, J. R.: On Adaptive and Online Data Integration 2005.
<https://ro.uow.edu.au/infopapers/466>

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

On Adaptive and Online Data Integration

Abstract

The recent works on integration of large database systems distributed over wide-area networks concentrate on the adaptive and online techniques. Online property of data integration means continuous integration of transmitted data with the already available results. Adaptivity materializes in a form of dynamic adjustments to the data integration plans in a response to the recent characteristics of data transmission. Implementation of adaptive and online data integration needs the specialized systems of operations and transformations of integration plans. This paper describes a new class of elementary operations on increments and/or decrements of data and shows how to express data integration plans as sequences of elementary operations. We demonstrate that class of operations proposed in the paper is sufficient for implementation of online and adaptive data integration systems and we discuss the operational properties of such systems.

Disciplines

Physical Sciences and Mathematics

Publication Details

This paper was originally published as: Getta, JR, On Adaptive and Online Data Integration, 21st International Conference on Data Engineering Workshops, 5-8 April 2005, 1212. Copyright 2005 IEEE.

On Adaptive and Online Data Integration

Janusz R. Getta

School of Information Technology and Computer Science

University of Wollongong

Wollongong, NSW 2522, Australia

email:jrg@uow.edu.au

Abstract

The recent works on integration of large database systems distributed over wide-area networks concentrate on the adaptive and online techniques. Online property of data integration means continuous integration of transmitted data with the already available results. Adaptivity materializes in a form of dynamic adjustments to the data integration plans in a response to the recent characteristics of data transmission. Implementation of adaptive and online data integration needs the specialized systems of operations and transformations of integration plans. This paper describes a new class of elementary operations on increments and/or decrements of data and shows how to express data integration plans as sequences of elementary operations. We demonstrate that class of operations proposed in the paper is sufficient for implementation of online and adaptive data integration systems and we discuss the operational properties of such systems.

1. Introduction

Advances in the technologies of persistent storage and wide-area networks allow for the relatively inexpensive implementations of unified and integrated views of data located at the remote and heterogeneous database systems. A central problem in the development of such systems is ad hoc integration of data transmitted over the networks. Efficiency of data integration depends on the advanced algorithms for merging the partial results of queries computed at the remote database sites. The recent trends in data integration lead towards online and adaptive algorithms. *Online algorithms* [7] process the incomplete sets of input data and continuously improve the solutions while the new data items are available for processing and the old data items are discarded. A typical example of an online algorithm is a virtual memory manager that operates on a *window* of theoretically unlimited sequence of tasks. *Adaptive algorithms* ad-

just their integration strategies to the external events, e.g. an arrival of a new packet of data or completion of transmission from a particular site. It is anticipated that data integration will soon emerge as an autonomous research area from the distributed computing and financial data processing triggered by the freely available distributed data sets and fast wide-area networks [8], [23].

Data integration has its roots in the processing of queries in the distributed and heterogeneous database systems, often called as multidatabase or federated database systems [25, 22]. The unpredictable behavior of data transmission systems and strong autonomy of remote database systems make the precise estimation of subquery processing time hard and imprecise. This is where the reactive query processing techniques show superiority over the classical proactive techniques commonly used for query processing in distributed database systems [1].

The early data integration systems looked for the solutions in the partitioning [6, 19] and dynamic modification of query processing plans [5, 10, 9]. Partitioning means that query execution plan is divided into subplans at a point when the further computations are no longer possible due to lack of data. Dynamic modification technique finds a plan equivalent to the original one plan and such that it can be partially computed with the available sets of data.

Another group of ideas addresses the optimization of individual elementary operations used for data integration. The specialized operations include the pipelined join operator XJoin [26], ripple join [14], double pipelined join [16], and hash-merge join [21].

The approaches based on scheduling change an order in which the operations are executed while preserving the semantics of data integration plan. The scheduling based techniques include query scrambling [28, 1] and dynamic scheduling of operators [27].

The techniques based on the redundant computations simultaneously execute a number of data integration plans leaving the plan that provides the most advanced results [2].

The solutions based on data partitioning integrate different components of integrated arguments accordingly to different plans. The Eddies are able to process each tuple accordingly to a different plan [3]. A concept of state modules described in [24] allows for concurrent processing of the tuples and dynamically divides data integration task among different plans and executes the plans sequentially or in parallel. Adaptive data partitioning [17] technique processes different partitions of the same argument using different data integration plans.

The recently developed data stream processing processing techniques [20, 11] also contribute to online data integration, e.g..

The works [4, 13, 15, 18] review the major solutions proposed so far. A more up-to-date and more detailed overview of the past works on adaptive data integration can be found in [12].

The approaches listed above adopt the relational model as a target data integration model and express the integration plan in the language of relational algebra. Majority of the works is limited to the plans exclusively formed from join operations and use dynamic query transformation and query scrambling techniques to migrate from one integration plan to another. The works on adaptive data partitioning [17] and optimizations of data stream processing [11] are the first attempts to use the associativity of join operation to integrate the different partitions of the same arguments accordingly to the different integration plans. It seems to us that relational algebra in its standard form is not the best language to describe the processes of online and adaptive data integration and that we need a new system of more elementary operations. The basic idea behind the online and adaptive computations is to restart the computations each time the processing of recently arrived data is possible and to reformulate an integration plan each time it is blocked by missing data. A data integrator processes a bit, waits, again processes a bit, again waits, and from time to time it adjusts a plan to the available data. A typical feature of online integration is that it never operates on a complete set of data. When the relational model is applied as a target integration mode, a data integrator must operate on the increments and decrements of relational tables and already integrated contents of the remaining relational tables. An increment is a collection of the most recently arrived and not yet processed packets of data. The decrements are created by non-monotonic operations like set difference operation where an increment of right hand side argument of the operation produces a decrement of the previous result of the set difference. As a consequence, the elementary operations of online data integrator should process the increments and/or decrements against the fixed size relational tables. Then, a data integration plan is a sequence of elementary operations whose arguments are the modifications of data containers and other data contain-

ers. The results of one elementary operation are passed to the next operation in a sequence. Adaptability of the system is achieved through a collection of rules that transform the plans blocked by unavailable data into the equivalent ones whose further execution is possible.

The main objective of this work is to propose a system of elementary operations for online and adaptive integration of data and to show how such system can be applied in practice. In particular, we show that it is possible to derive such a system from a given collection of base operations, i.e. the operations on data containers like for instance relational algebra operations, or aggregation operations. Then, we define a data integration plan as a collection of local integration plans formed from the sequences of elementary operations and we discuss the plan transformations rules needed for the implementation of adaptive features of a sample data integration system.

The paper is organized in the following way. Section 2 describes a data integration model used throughout the paper. The system of elementary operations and data flow expressions are defined in the Sections 3 and 4. Section 5 shows how the formal data integration model proposed in the previous sections can be used in implementation of a sample data integration system. Section 6 summarizes and concludes the paper.

2. Data integration model

Consider a distributed multidatabase system that integrates a number of remote and heterogeneous database systems such that remote database sites are entirely transparent at a central site. A middleware that integrates the databases provides the users with a single view of a homogeneous database. Then, a query $q(r_1, \dots, r_k)$ on a subset $r_1, \dots, r_k$ of the view is decomposed into k subqueries $q_{r_1}, \dots, q_{r_k}$ that encapsulate the computations performed at the remote systems. Two generic strategies of distributed query processing either optimize an overall amount of time spend on the computations or optimize the total amount of data transmitted over a network. Query processing time is minimized when the queries $q_{r_1}, \dots, q_{r_k}$ are submitted and processed simultaneously at the remote sites. Processing of subqueries one at a time and applying the results of one subquery to modify the remaining subqueries minimizes the amounts of transmitted data. The entire continuum of hybrid strategies is contained between these two extremes. Selection of the best strategy is a hard problem and it is beyond a scope of this paper.

We adopt a strategy that minimizes query processing time through the simultaneous computations at the remote database sites. The results obtained from the remote sites are transmitted back to the central site. Next, the results are transformed into the containers $r_1, \dots, r_k$ structurally con-

sistent with a data model at the central site, i.e. into the relational tables. Finally, the results are integrated into the final answer accordingly to a *global data integration plan* $\mathcal{P}(r_1, \dots, r_k)$ derived from the original query q and built from the base operations on the data containers e.g. the relational algebra operations on the relational tables. A simple and rather ineffective approach would be to delay the integration until all partial results are fully transmitted to the central site. Contrary, an "impatient" approach that "wakes up" a data integrator each time a new packet of data arrives, would need too much time spent on the organizational aspects of the process. In this work we consider a strategy where a data integrator "wakes up" at the fixed intervals of time and starts integration only if there is enough data transmitted since the last integration cycle. If it is so, the recently arrived packets of data are integrated with the already available results. Such approach invalidates an idea of single global data integration plan because it may happen that partial results required to follow the plan are unavailable at the moment. On the other hand a global plan cannot be completely rejected because it represents the semantics of a database application. A solution is to transform the global plan into a set of local plans describing the actions performed when a new increment of data should be integrated with the already available partial results. The actions are expressed as elementary operations on the increments and/or decrements of data containers and other static data containers. The local integration plans are expressed as the sequences of elementary operations.

3. Elementary operations

Let r and s be data containers, e.g. relational tables. A *base operation* $A(r, s)$ is an operation whose arguments are data containers and result of the operation is a data container as well.

A *modification* δ_r of a data container r is a pair of containers $\langle \delta_r^-, \delta_r^+ \rangle$ such that both elements of the pair have the same structure (schema) as r . The first element δ_r^- of the pair represents the data items that should be removed from r to implement the first stage of the modification. The second element δ_r^+ of the pair represents the data items that should be added to r to implement the second stage of the modification.

An operation that integrates a container r with a modification $\delta_r = \langle \delta_r^-, \delta_r^+ \rangle$ is denoted by $r \oplus \delta_r$ and it is called as *data integration operation*. In the relational model a data integration operation is defined by an expression $(r - \delta_r^-) \cup \delta_r^+$.

An *incremental/decremental operation (id-operation)* for the first argument r of a base operation $A(r, s)$ is denoted by $\alpha_A(\delta_r, s)$ and its result is a pair of the smallest and disjoint sets $\langle \delta_\alpha^-, \delta_\alpha^+ \rangle$ that should be integrated with

the result of $A(r, s)$ to obtain the result of $A((r \oplus \delta_r), s)$ i.e.

$$A(r, s) \oplus \alpha_A(\delta_r, s) = A((r \oplus \delta_r), s) \quad (1)$$

An *incremental/decremental operation (id-operation)* for the second argument s of a base operation $A(r, s)$ is denoted by $\beta_A(r, \delta_s)$ and its result is a pair of the smallest and disjoint sets $\langle \delta_\beta^-, \delta_\beta^+ \rangle$ that should be integrated with the result of $A(r, s)$ to obtain the result of $A(r, (s \oplus \delta_s))$, i.e.

$$A(r, s) \oplus \beta_A(r, \delta_s) = A(r, (s \oplus \delta_s)) \quad (2)$$

A base operation $A(r, s)$ always has two id-operations $\alpha_A(\delta_r, s)$ and $\beta_A(r, \delta_s)$, one for processing δ_r and other one for processing δ_s . If a base operation is commutative then its id-operations are the same. If a base operation $A(r, s)$ is monotonic for an argument r , i.e. $A(r, s) \subseteq A(r \oplus \delta_r, s)$ then a negative component of modification computed by $\alpha_A(\delta_r, s)$ is always empty.

Id-operations process the modifications of data containers and produce the modifications that can be integrated with the previous results of the respective base operation to obtain the new results of the base operation without its full re-computation. This is what is precisely needed for data integration. A modification of an argument in a global data integration plan is processed by an appropriate id-operation. The id-operation produces a modification which is processed by the next id-operation and so on until the final modification is integrated with the previous partial answer to provide a new partial answer.

An interesting problem is how to find id-operations for a given base operation. If for a particular system of the base operations and data integration operation it is possible to express $A((r \oplus \delta_r), s)$ as a combination of an old result of base operation $A(r, s)$ and modification δ_r then it is possible to find the respective id-operations as the smallest solutions of the equations (1) and (2). In this paper we consider the relational model with the base operations of *union* ($\cup$), *join* ($\bowtie$), and *antijoin* ($\sim$) and data integration operation operation defined as $r \oplus \delta_r = (r - \delta_r^-) \cup \delta_r^+$. We ignore the unary operations of *selection* (σ) and *projection* (π) as they can always be attached to the inputs or outputs of the binary operations. To solve the equation (1) we have to separately consider the negative and positive components of δ_r and data integration operation. It leads to the equations:

$$A(r, s) - \alpha(\delta_r^-, s) = A(r - \delta_r^-, s) \quad (3)$$

$$A(r, s) \cup \alpha(\delta_r^+, s) = A(r \cup \delta_r^+, s) \quad (4)$$

We are looking for the smallest solutions of the equations (3) and (4). The first equation is of type $A - x = A - B$ where A, B, x are sets. To find the smallest solution we transform the equation into an equivalent fixed point equation $x = x \cup ((A - x) - (A - B)) \cup ((A - B) - (A - x))$. The solution of the fixed point equation is obtained

through a sequence of iterations starting from $x' = \emptyset$. In the second iteration the fixed point reached and it is equal $x'' = A \cap B$. Hence, the solution of equation (3) is $\alpha(\delta_r^-, s) = A(r, s) \cap \delta_r^-$. For example if $A(r, s) = r - s$ then $\alpha(\delta_r^-, s) = (r - s) \cap \delta_r^- =$. Note, that if δ_r^- denotes the rows removed from r then $\delta_r^- \subseteq r$. Finally, we get $\alpha(\delta_r^-, s) = \delta_r^- - s$. It is possible to derive in the same way all id-operation for the remaining base operations.

Id-operations for the arguments of join ($\bowtie$) are defined as follows:

$$\alpha_{\bowtie}(\delta_r, s) = \langle (\delta_r^- \bowtie s), (\delta_r^+ \bowtie s) \rangle \quad (5)$$

$$\beta_{\bowtie}(r, \delta_s) = \langle (\delta_s^- \bowtie r), (\delta_s^+ \bowtie r) \rangle \quad (6)$$

Id-operations for the arguments of antijoin ($\sim$) are defined as follows:

$$\alpha_{\sim}(\delta_r, s) = \langle (\delta_r^- - s), (\delta_r^+ - s) \rangle \quad (7)$$

$$\beta_{\sim}(r, \delta_s) = \langle (r \cap \delta_s^+), (r \cap \delta_s^-) \rangle \quad (8)$$

Finally, id-operations for the arguments of union ($\cup$) are defined as follows:

$$\alpha_{\cup}(\delta_r, s) = \langle (\delta_r^- - s), (\delta_r^+ - s) \rangle \quad (9)$$

$$\beta_{\cup}(r, \delta_s) = \langle (\delta_s^- - r), (\delta_s^+ - r) \rangle \quad (10)$$

As a sample application of id-operations, consider a global data integration plan $q(r, s, t) = t \bowtie (r - s)$ and modification $\delta_s = \langle \emptyset, \delta_s^+ \rangle$ of an argument s . Then, (8) and (5) contribute to a formula for processing δ_s . Application of $\beta_{\sim}$ to $\langle \emptyset, \delta_s^+ \rangle$ provides $\langle r \cap \delta_s^+, \emptyset \rangle$. Next, application of $\alpha_{\bowtie}$ to the previous result provides $\langle (r \cap \delta_s^+) \bowtie t, \emptyset \rangle$. Finally, the modifications should be integrated with the partial result of q as follows $q := q - (r \cap \delta_s^+) \bowtie t$. A formula for processing the modifications of r can be derived in a similar way using (7) and (5) $q := q \cup (\delta_r^+ - s) \bowtie t$.

Processing the modifications of argument t requires the transformation of $q(r, s, t)$ into an equivalent expression $(t \bowtie r) - s$. Then, application of (5) and (7) provides $q := q \cup (\delta_t^+ \bowtie r) - s$. A problem what to do when the transformation performed above is impossible is discussed in the next sections.

As another example consider a system of operation $\mathcal{F} = \{agg, \cup\}$ where $\cup$ is a set union operation and agg is defined as follows. The operation $agg_{x,a}(r, s)$ replaces the second argument s with the result of SQL statement:

```
SELECT x, sum(a)
FROM r
GROUP BY x;
```

The id-operations of $\cup$ are the same as in the previous system. An id-operation $\alpha_{agg}(\delta_r, s)$ combines δ_r^+ with s in the following way.

```
for all  $t \in \delta_r^+$ 
  if there exists  $t' \in s$ 
```

```
    such that  $t.x = t'.x$  then
      replace  $t'$  with  $t'.a := t'.a + t.a$ ;
      insert old  $t'$  into  $\delta_{agg}^-$  and
      insert new  $t'$  into  $\delta_{agg}^+$ ;
    else
      add  $t$  to  $s$  and add  $t$  to  $\delta_{agg}^+$ ;
  end if;
```

```
for all  $t \in \delta_r^-$ 
  if there exists  $t' \in s$ 
    such that  $t.x = t'.x$  then
      replace  $t'$  with  $t'.a := t'.a - t.a$ ;
      insert old  $t'$  into  $\delta_{agg}^-$  and
      insert new  $t'$  into  $\delta_{agg}^+$ ;
  end if;
```

Finally, an id-operation $\beta_{agg}(r, \delta_s) = \langle \delta_s^-, \delta_s^+ \rangle$.

4. Data flow expressions

A *data flow expression* is a sequence $r_0: \alpha_1(r_1) \dots \alpha_n(r_n)$ where r_0 is a data container and each $\alpha_i(r_i)$, $i = 1, \dots, n$ is either an abbreviation of id-operation $\alpha(\delta_{r_j}, r_i)$ or abbreviation of data integration operation $\delta_{r_j} \oplus r_i$. The adjacent id-operations in a data flow expression are connected such that modification generated by α_i is used as an argument δ_{α_i} of its successor α_{i+1} . The evaluation of an expression starts from the first id-operation $\alpha_1(\delta_{r_0}, r_1)$. A modification δ_{α_1} produced by the first id-operation becomes an argument of the next id-operation $\alpha_2(\delta_{\alpha_1}, r_2)$. For example, $r: \alpha_{\bowtie}(s) \alpha_{-}(t) \oplus (w)$ is a data flow expression where a modification δ_r of argument r is joined with s . Then, t is deducted from the results of the join, and the results of the difference are integrated with w .

A data flow expression related to an argument r_i of an expression $E(r_1, \dots, r_i, \dots, r_n)$ is constructed through traversal of a syntax tree of E from a leaf node labeled with r_i to the root node. Initially, at a leaf node r_i , we start from an empty expression r_i . Next, we move one level up to a base operation operation $A(E_1, E_2)$ where E_1 and E_2 are subexpressions (subtrees in a syntax tree) bound with a base operation A . If a subexpression E_1 is on the path being traversed then we append id-operation $\alpha_A(w_{E_2})$ to the data flow expression expression. Otherwise, if E_2 is on the path being traversed then we append $\beta_A(w_{E_1})$ to the expression. Next, we move one level up to the next base operation and we repeat the actions listed above. At the end when all paths from the leaf nodes to the root node are traversed and data flow expression generated then we insert into the expressions data integration operations that produce the intermediate results. For example, application of the procedure described above to a relational algebra expression $r - (s \bowtie_b t)$ provides the following data flow expressions:

```
 $r: \alpha_{-}(w_{st}) \oplus (w)$ 
 $s: \alpha_{\bowtie}(t) \oplus (w_{st}) \beta_{-}(r) \oplus (w)$ 
```

$$t: \beta_{\bowtie}(s) \oplus (w_{st}) \beta_{-}(r) \oplus (w)$$

Data flow expressions represent the sequences of operations performed on the recently arrived modifications at a data integration stage.

Like in the traditional query processing, optimization of data integration expressions is performed through the transformations of data flow expressions. One group of transformations moves the most restrictive id-operations towards the left hand side of an expression in order to eliminate at the early stages of data integration as many data items as it is possible. The other group removes the intermediate data containers created and modified during the integration in order to reduce the total number of operations on persistent storage. Consider a data flow expression p which contains two adjacent id-operations $\alpha_A(r_i) \alpha_B(r_j)$. A data flow expression p' obtained from p by the order of id-operations $\alpha_B(r_i) \alpha_A(r_j)$ is equivalent to p if the respective base operations are associative, i.e. $B(A(r, s), t) = A(B(r, t), s)$. Associativity of adjacent operations allows for the elimination of intermediate data containers. As an example consider the following system of data flow expressions.

$$\begin{aligned} r: & \alpha_A(s) \oplus (w_{rs}) \alpha_B(t) \oplus (w) \\ s: & \beta_A(r) \oplus (w_{rs}) \alpha_B(t) \oplus (w) \\ t: & \beta_B(w_{rs}) \oplus (w) \end{aligned}$$

where w_{rs} is always equal to the result of $A(r, s)$. Hence, the third data flow expression can be expressed as $t: \beta_B(A(r, s))$. It is equivalent to two relational algebra expressions $\beta_B^-(A(r, s), \delta_t^-)$ and $\beta_B^+(A(r, s), \delta_t^+)$. If the base operations A and B are associative then the expressions can be transformed into $A(\beta_B^-(r, \delta_t^-), s)$ and $A(\beta_B^+(r, \delta_t^+), s)$. Taking the expressions together and replacing a base operation A with an id-operation α_a we obtain $\alpha_A(\beta_B(r, \delta_t), s)$ and in the consequence a data flow expression $t: \beta_B(r) \alpha_A(s) \oplus (w)$. Now, a temporary container w_{rs} can be removed from the remaining dataflow expressions:

$$\begin{aligned} r: & \alpha_A(s) \alpha_B(t) \oplus (w) \\ s: & \beta_A(r) \alpha_B(t) \oplus (w) \end{aligned}$$

5. Data integration

Let $r_1, \dots, r_k$ be the results of k subqueries $q_1, \dots, q_k$ computed at the remote database sites and transmitted to the central site. A *global data integration plan* $\mathcal{P}(r_1, \dots, r_k)$ is an expression build over the data containers $r_1, \dots, r_k$ and the base operations, e.g. relational algebra operations. In the traditional approaches data integration is delayed until the arguments bound by the base operations in $\mathcal{P}$ are available at the central site. Adaptive and incremental strategies allows for data integration while the arguments are still transmitted over a network. Implementation of incremental strategy needs the translation of a global integration plan into

a set of *local integration plans*. A set of local integration plans for $\mathcal{P}(r_1, \dots, r_k)$ is equivalent to set of data flow expressions $\{p_1, \dots, p_k\}$ where each p_i represents a way how the increments of an argument r_i are integrated with the intermediate results. An individual data integration plan p_i is a sequence of id-operations performed by the system in order to process an increment δ_{r_i} .

Consider a logical data integration plan $(r \bowtie s) \bowtie t$. An incremental integration strategy transforms the plan into the following individual data integration plans:

$$\begin{aligned} r: & \alpha_{\bowtie}(s) \alpha_{\bowtie}(t) \oplus (w) \\ s: & \alpha_{\bowtie}(r) \alpha_{\bowtie}(t) \oplus (w) \\ t: & \alpha_{\bowtie}(s) \alpha_{\bowtie}(r) \oplus (w) \end{aligned}$$

In another example elimination of union operation from a logical data integration expression $r(ab) - (s(ab) \cup t(ab))$ leads to expression with two occurrences of an argument r , i.e. $(r(ab) - s(ab)) \cap (r(ab) - t(ab))$. Then an individual integration plan for an argument r consists of two data flow expressions:

$$\begin{aligned} r': & \alpha_{-}(t) \oplus (v_{rt}) \\ r'': & \alpha_{-}(s) \oplus (v_{rs}) \alpha_{\cap}(v_{rt}) \oplus (w) \end{aligned}$$

The remaining individual integration plans are as follows

$$\begin{aligned} s: & \beta_{-}(r) \oplus (v_{rs}) \alpha_{\cap}(v_{rt}) \oplus (w) \\ t: & \beta_{-}(r) \oplus (v_{rt}) \alpha_{\cap}(v_{rs}) \oplus (w) \end{aligned}$$

A global data integration plan $\mathcal{P}$ implemented as a set of local data integration plans allows for a correct and adaptive integration of the partial results. The local data integration plans are created such that each argument of the respective logical data integration expression gets its local plan. If, like in the example above, the same argument used used more than one time then we get more than one plan as well. All plans associated with a given argument are activated when an increment of the argument has to be processed. Each local plan is a data flow expression constructed and optimized in a way described in the previous section.

A process of incremental and adaptive data integration "wakes up" at the regular intervals of time, verifies the amounts of data transmitted since the last integration, and if there is enough data, prepares and implements the local integration plans.

An algorithm that constructs the data flow expressions from a global data integration plan $\mathcal{P}$ is used to formulate a set of initial local integration plans. Next, the optimizations of the data flow expressions described in the previous section move the most selective operations towards the beginning of each local plan and try to eliminate the integrations with the intermediate results. The optimization of the local plans assumes the most optimistic case of the initial availability and continuous transmissions of all arguments. In the reality the initializations of transmissions are frequently delayed or the transmissions cannot be completed for a longer period of time. This is why some the local plans have to be either suspended or reduced to the id-operations that can be

executed in a given moment of time followed by the integrations with the temporary data containers.

The first run of the data integrator transforms the local plans obtained from the optimizer in way that takes under the consideration availability of the arguments and optimal integration of the available data. Each next invocation, adjusts the plans used in the pervious run to reflect the availability of the new arguments. When all arguments are partially available at the central site the local plans return to their optimized form.

The run time transformations of local plans include the addition and elimination of integrations with the temporary data containers, elimination of subexpressions that can be totally evaluated and replaced with a constant data container, changing the order elimination of the local plans.

Addition of the integration with a temporary data container is need when the computations of a plan $r : \alpha_1(r_1), \dots, \alpha_{i-1}(r_{i-1})\alpha_i(r_i), \dots$ cannot be completed because a container r_i is not available at the moment. Then, the plan is computed partially and integration with an intermediate container v_i is inserted in front of α_i in the following way $r : \alpha_1(r_1), \dots, \oplus(v_i)\alpha_i(r_i), \dots$. Moreover, a sequence of id-operations $\alpha_1(r_1), \dots, \alpha_{i-1}(r_{i-1})$ is replaced with $\beta_i(v_i)$ in all other local plans. A temporary data container is removed from the local plan r when an argument r_i is not empty. Then, $\oplus(v_i)$ is removed from the plan and $\beta_i(v_i)$ is replaced with the original sequence of operations in all other plans whenever it occurs.

When a data integrator is invoked for the first time then some of the transmissions from the remote sites may already be completed. If both arguments of a base operation in a global integration plan are available then such operation can be computed in a traditional way and its results can be incorporated as a constant argument into the plan. Consider the local plans $r_i : \alpha_A(r_j), \alpha_B(r_k) \dots$ and $r_j : \beta_A(r_i), \alpha_B(r_k) \dots$ and assume that both r_i and r_j are available for integration. Then, the respective base operation $A(r_i, r_j)$ is computed and its result r_{ij} obtains a new local integration plan $r_{ij} : \alpha_B(r_k) \dots$ and the plans r_i and r_j are removed. In all other plans a sequence $\alpha_A(r_j)\alpha_B(r_k)$ is replaced with $\beta_B(r_{ij})$. Elimination of subexpression in a way described above is possible only if completely unavailable argument at one stage of integration is totally available. What if in the same situation transmission of some of the arguments is completed but no base operations can be computed? Consider a plan $r_i : \alpha_A(r_j)\alpha_B(r_k) \dots$ and assume that transmission of data container r_i is completed. Then, a status of r_i is changed to "ready" and its plan r_i is removed from a set of local plans.

Each of the arguments involved in data integration has its status recorded and maintained by the system. At the very beginning of data integration all arguments obtain a status

Figure 1. The transitions of argument states

missing. Next, when an argument arrives and its transmission is completed the status changes to ready. If only a part of argument arrives its status is active and after the part is integrated a status changes to idle.

The state transitions given in Figure 1 occur when a data integrator completes an integration cycle. When the data integrator "wakes up" for the first time the only local integration plans are those directly constructed and optimized from a global plan. First, data integrator considers the arguments that changed their status from missing to ready. The subexpressions of a global integration plan are computed in a way described above. The local plans for the arguments that have status "ready" are removed from a set of local plans. Next, data integrator considers the arguments that changed their status from missing to active, i.e. only some of the components of these arguments have arrived. The local plans related to these arguments are computed as far as it is possible and whenever the computations do not reach integration with the final results then integration with a temporary relational table is performed, inserted into the plan, and the related local plans are modified in a way described above. No other state transitions are possible at the first integration stage.

When the data integrator "wakes up" on any other time than the first time any transition of the argument states is possible. First, the data integrator considers the arguments that changed their status from missing to ready. The local integration plans for these arguments are removed from a set of local plans and the related plans are modified in a way described above.

Next, the data integrator considers the arguments whose status has changed from active to ready. The local plans for these arguments are computed as far as possible and then the plans are removed from a set of local plans.

Next, the data integrator considers the arguments that changed their status from missing to active. The local plans for these arguments are computed as far as possible and whenever the computations do not reach integration with the final results then integration with a temporary table is performed, inserted into the plan and the related plans are updated in a way described above. Whenever an argument is used in the computations then its plan

is made inactive for this cycle. If the computation of a local plan use a temporary relational table created earlier then the temporary table is removed from the plan and all other plans are updated in a way described above.

Next, the data integrator considers the arguments whose status remained active and whose local plans have not been deactivated in this cycle. These arguments are processed in the same way as above when a status have changed from missing to active.

In all other cases, the integrator remains idle.

6. Summary and future work

This paper considers the online and adaptive integration of large data sets distributed over the wide-area networks. We argue that traditional approach where the global integration plans are expressed as the relational algebra expressions is not appropriate to precisely describe the integration processes at a level where the individual packets of data are assembled into the final results. In contrast, we define a concept of id-operation as an elementary operation on the modifications (increments and/or decrements) of data containers and the partial results. Next, we show how to construct a data integration plan as a collection of data flow expressions composed of id-operations and data integration operations. Finally, we describe the operational principles of a sample system capable of online and adaptive data integration.

A number of interesting problems remains to be solved. These include a wider system of id-operations, investigations of the properties of dataflow algebra and further investigations on more advanced data integration algorithms

References

- [1] L. Amsaleg, J. Franklin, and A. Tomasic. Dynamic query operator scheduling for wide-area remote access. *Journal of Distributed and Parallel Databases*, 6:217–246, 1998.
- [2] G. Antoshenkov and M. Ziauddin. Query processing and optimization in oracle rdb. *VLDB Journal*, 5(4):229–237, 2000.
- [3] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 261–272, 2000.
- [4] L. Bouganim, F. Fabret, and C. Mohan. A dynamic query processing architecture for data integration systems. *Bulletin of the Technical Committee on Data Engineering*, 23(2):42–48, June 2000.
- [5] J. Chudziak and J. R. Getta. On efficient query evaluation in multidatabase systems. In *Second International Workshop on Advances in Database and Information Systems, ADBIS'95*, pages 46–54, 1995.
- [6] R. L. Cole and G. Graefe. Optimization of dynamic query evaluation plans. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, 1994.
- [7] A. Fiat and G. J. Woeginger. *On Line Algorithms, The State of the Art*. Springer Verlag, 1998.
- [8] I. Foster and R. L. Grossman. Data integration in a bandwidth-rich world. *Communications of the ACM*, 46(11):51–57, November 2003.
- [9] J. R. Getta. Query scrambling in distributed multidatabase systems. In *11th Intl. Workshop on Database and Expert Systems Applications, DEXA 2000*, 2000.
- [10] J. R. Getta and S. Sedighi. Optimizing global query processing plans in heterogeneous and distributed multi database systems. In *10th Intl. Workshop on Database and Expert Systems Applications, DEXA 1999*, pages 12–16, 1999.
- [11] J. R. Getta and E. Vossough. Optimization of data stream processing. *SIGMOD record*, 33(3):34–39, 2004.
- [12] A. Gounaris, N. W. Paton, A. A. Fernandes, and R. Sakellariou. Adaptive query processing: A survey. In *Proceedings of 19th British National Conference on Databases*, pages 11–25, 2002.
- [13] G. Graefe. Dynamic query evaluation plans: Some course corrections? *Bulletin of the Technical Committee on Data Engineering*, 23(2):3–6, June 2000.
- [14] P. J. Haas and J. M. Hellerstein. Ripple joins for online aggregation. In *SIGMOD 1999, Proceedings ACM SIGMOD Intl. Conf. on Management of Data*, pages 287–298, 1999.
- [15] J. M. Hellerstein, M. J. Franklin, S. Chandrasekaran, A. Deshpande, K. Hildrum, S. Madden, V. Raman, and M. A. Shah. Adaptive query processing: Technology in evolution. *Bulletin of the Technical Committee on Data Engineering*, 23(2):7–18, June 2000.
- [16] Z. G. Ives, D. Florescu, M. Friedman, A. Y. Levy, and D. S. Weld. An adaptive query execution system for data integration. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 299–310, 1999.
- [17] Z. G. Ives, A. Y. Halevy, and D. S. Weld. Adapting to source properties in processing data integration queries. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, 2004.
- [18] Z. G. Ives, A. Y. Levy, D. S. Weld, D. Florescu, and M. Friedman. Adaptive query processing for internet applications. *Bulletin of the Technical Committee on Data Engineering*, 23(2):19–26, June 2000.
- [19] N. Kabra and D. J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, 1998.
- [20] S. Madden, M. A. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. In *Proceedings of the 2002 ACM SIGMOD Intl. Conf. on Management of Data*, 2002.
- [21] M. F. Mokbel, M. Lu, and W. G. Aref. Hash-merge join: A non-blocking join algorithm for producing fast and early join results, 2002.
- [22] F. Ozcan, S. Nural, P. Koksall, C. Evrendilek, and A. Dogac. Dynamic query optimization in multidatabases. *Bulletin of the Technical Committee on Data Engineering*, 20:38–45, March 1997.

- [23] A. Pan and A. Vina. An alternative architecture for financial data integration. *Communications of the ACM*, 47(5):37–40, May 2004.
- [24] V. Raman, A. Deshpande, and J. M. Hellerstein. Using state modules for adaptive query processing. In *Proceeding of International Conference on Management of Data*, 2003.
- [25] V. Srinivasan and M. J. Carey. Compensation-based on-line query processing. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, pages 331–340, 1992.
- [26] T. Urhan and M. J. Franklin. Xjoin: A reactively-scheduled pipelined join operator. *IEEE Data Engineering Bulletin* 23(2), pages 27–33, 2000.
- [27] T. Urhan and M. J. Franklin. Dynamic pipeline scheduling for improving interactive performance of online queries. In *Proceedings of International Conference on Very Large Databases, VLDB 2001*, 2001.
- [28] T. Urhan, M. J. Franklin, and L. Amsaleg. Cost based query scrambling for initial delays. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 130–141, 1998.