University of Wollongong
Research Online

Faculty of Informatics - Papers (Archive)

Faculty of Engineering and Information Sciences

1-1-2011

Optimization of task processing schedules in distributed information systems

Janusz 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, Janusz R.: Optimization of task processing schedules in distributed information systems 2011, 333-345.
https://ro.uow.edu.au/infopapers/1534

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

Optimization of task processing schedules in distributed information systems

Abstract

The performance of data processing in distributed information systems strongly depends on theefficient scheduling of the applications that access data at the remote sites. This work assumes atypical model of distributed information system where a central site is connected to a number ofremote and highly autonomous remote sites. An application started by a user at a central site isdecomposed into several data processing tasks to be independently processed at the remote sites.The objective of this work is to find a method for optimization of task processing schedules at acentral site. We define an abstract model of data and a system of operations that implements thedata processing tasks. Our abstract data model is general enough to represent many specific datamodels. We show how an entirely parallel schedule can be transformed into a more optimal hybridschedule where certain tasks are processed simultaneously while the other tasks are processedsequentially. The transformations proposed in this work are guided by the cost-based optimizationmodel whose objective is to reduce the total data transmission time between the remote sites and acentral site. We show how the properties of data integration expressions can be used to find moreefficient schedules of data processing tasks in distributed information systems.

Keywords

Optimization, task, processing, schedules, distributed, information, systems

Disciplines

Physical Sciences and Mathematics

Publication Details

Getta, J. R. (2011). Optimization of task processing schedules in distributed information systems. Proceeding of International Conference on Informatics Engineering and Information Science, ICIEIS 2011 (pp. 333-345). New York: Springer.

This journal article is available at Research Online: https://ro.uow.edu.au/infopapers/1534

Optimization of Task Processing Schedules in Distributed Information Systems

Janusz R. Getta

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

Abstract. The performance of data processing in distributed information systems strongly depends on the efficient scheduling of the applications that access data at the remote sites. This work assumes a typical model of distributed information system where a central site is connected to a number of remote and highly autonomous remote sites. An application started by a user at a central site is decomposed into several data processing tasks to be independently processed at the remote sites. The objective of this work is to find a method for optimization of task processing schedules at a central site. We define an abstract model of data and a system of operations that implements the data processing tasks. Our abstract data model is general enough to represent many specific data models. We show how an entirely parallel schedule can be transformed into a more optimal hybrid schedule where certain tasks are processed simultaneously while the other tasks are processed sequentially. The transformations proposed in this work are guided by the cost-based optimization model whose objective is to reduce the total data transmission time between the remote sites and a central site. We show how the properties of data integration expressions can be used to find more efficient schedules of data processing tasks in distributed information systems.

Key words: Distributed information system, data processing, scheduling, data integration, optimization

1 Introduction

The rapid growth in the number of distributed applications and the users of these applications creates an ever increasing pressure on the performance of data processing in distributed information systems. To satisfy the increasing performance requirements we investigate more sophisticated and more efficient algorithms for distributed data processing. A factor, that has a significant impact on the performance of distributed data processing is scheduling of the individual data processing tasks over the remote sites. In a typical approach a central site decomposes a task submitted by a user into a number of individual tasks, to be processed at one of the remote sites. A partial order in which the individual tasks

are submitted to the remote sites and a way how their results are assembled into the final result is called as a *task processing schedule*.

Two generic task processing schedules are either *entirely sequential* or *entirely parallel* schedules. In an *entirely sequential* schedule the tasks $t_1, \ldots, t_n$ are processed one by one in a way where a task t_i can be processed only when all results of the tasks $t_1, \ldots, t_{i-1}$ are available at a central site. Accordingly to an *entirely parallel* schedule all tasks $t_1, \ldots, t_n$ are simultaneously submitted for processing at the remote sites. When looking at the performance, our intuition always favor an entirely parallel schedule over a sequential one because processing of several tasks is done in the same time at many remote sites. An entirely parallel schedule attempts to save time on processing of all tasks. However, if we consider time spent on transmission of the results from the remote sites then in some cases a sequential schedule is more appropriate than a parallel one because the intermediate results received so far can be used to reduce the size of the other results. For example, if an individual task t_i returns a lot of data then processing of t_i and transmission of its results to a central site may take more time than parallel processing of the tasks $t_1, \ldots, t_{i-1}$, modification of task t_i with the results $r_1, \ldots, r_{i-1}$, processing of updated t_i, and transmission of its results. In such a case simultaneous processing of the tasks $t_1, \ldots, t_{i-1}$ followed by simultaneous processing of the tasks $t_{i-1}1, \ldots, t_n$ may provide better performance than entirely parallel schedule.

An *entirely sequential* schedule attempts to minimize data transmission time of the results while an entirely parallel schedule minimizes the total processing time of the individual tasks. As the efficiency of both methods depend on a number of factors like for instance the computational complexity of the individual tasks, computational power of local systems, data transmission speed, etc, then usually a *hybrid schedule* where some of the individual tasks are processed sequentially while the others simultaneously, provides the best results.

The objectives of this work are the following. We consider a model of distributed information system where a user application running at a central site submits a data processing task T against a global view of the system. An abstract model of data containers represents a data component of a distributed system and a system of operations on data containers is used to implement the data processing tasks. The task is decomposed into a number of individual tasks $t_1, \ldots, t_n$ to be processed at the local sites of the distributed system. A *data integration expression* $e(t_1, \ldots, t_n)$ determines how the results $r_1, \ldots, r_n$ of the individual tasks suppose to be assembled into the final result of a task T. A starting point for the optimization is an entirely parallel schedule where the individual tasks are in the same moment submitted for the simultaneous processing at the remote sites. We show how an entirely parallel task processing schedule can be transformed into a hybrid schedule that minimizes the total amount of time spent on transmission of data from the local sites. To minimize total transmission time we estimate the amounts of time needed for transmission of the results $r_1, \ldots, r_n$ and we find if it is possible to reduce the amounts of transmission if some of the tasks are processed before the others. Then, we find

how the results of the tasks processed earlier can be used to transform the tasks processed later.

The paper is organized in the following way. An overview of the works related to an area of optimization of data processing in distributed systems is included in the next section. A section 3 introduces an abstract data model used in this work. A method used for the estimation of the costs of alternative data processing schedules is presented in a section 4. Transformation and optimization of data processing schedules is described in the sections 5 and 6. Finally, section 7 concludes the paper.

2 Previous works

The previous works concentrated on three aspects of distributed data processing: optimization of query processing in distributed systems, estimation of processing time at the remote site and transmission time, and optimization of data integration.

Optimization of data processing in distributed systems has its roots in optimization of query processing in multidatabase and federated database systems [17, 15]. Due to the syntactical and semantic heterogeneities of the remote systems [16] optimization of distributed query processing is conceptually different from optimization of query processing in homogeneous and centralized systems [14]. One of the recent solutions to speed up distributed query processing in distributed systems considers the contents of cache in the remote systems and prediction of cache contents [10]. Wireless networks and mobile devices triggered research in mobile data services and in particular in location-dependent queries that amalgamate the features of both distributed and mobile systems. The existing literature related to location-dependent query processing is reviewed in [7]. A framework for distributed query scheduling has been proposed in [13]. The framework allows for the dynamic information gathering across distributed systems without relying on a unified global data model of the remote systems. [20] introduces an adaptive distributed query processing architecture where fluctuations in selectivities of operations, transmission speeds, and workloads of remote systems, can change the operation order distributed query processing.

Optimization of data processing schedules in distributed systems strongly depends on the precise estimation of data processing time at the remotes sites and on the amounts of data transmitted from the remote sites. Due to the strong autonomy of the remote sites a central site has no impact on processing of subqueries there and because of that the estimation of the local performance indicators is pretty hard [21]. A solution proposed in [5] categorizes the local databases into three groups and uses such classification to estimate the cost functions for data processing at the remote sites. In [21] the query sampling methods is used to estimated the query processing costs at the local systems. [11] proposes a clustering algorithm to classify the queries and to derive the cost functions. Query scheduling strategy in a grid-enabled distributed database proposed in [4] takes under the consideration so called "site reputation" for ranking response time of

the remote systems. A new approach to estimation of workload completion time based on sampling the query interactions has been proposed in [1] and in [2]. Query monitoring can be used to collect information about expected database load, resource allocation, and expected size of the results [9].

Efficient integration of the partial results obtained from the remote sites is one of subproblems in optimization data processing schedules in distributed systems. Data integration combines data stored at the remote sites and provides a single unified view of the contents of remote sites. The reviews of research on data integration are included in [8],[22]. The implementations of experimental data integration systems systems based on application of ontologies and data sharing are described in [19],[18], [12]. A distributed and open query processor that integrates Internet data sources was proposed in [3].

3 Basic concepts

To remain at a high level of generality we define an abstract data model where a data component of an information system is a set $\mathcal{D}$ of *data containers*. A data container $d \in \mathcal{D}$ includes *data objects*. A data object is either a *simple data object* or a *composite data object*. A *simple data object* includes the pairs of data items $(name, value)$ where *name* is a name of data item and *value* is a value of data item. An *internal data structure* can be used to "assemble" the data items into a simple data object. At an abstract level we do note refer to any particular internal data structure. In a *concrete data model* an internal data structure could be a sequence of tuples of data items, a hierarchical structure of data items, a graph of data items, a vector of data items etc.

A *composite data object* is a pair (o_i, o_j) where o_i and o_j are either simple data objects or composite data objects.

An operation of *composition* on data containers r_i and r_j is defined as

$$r_i +_f r_j = \{(o_i, o_j) : o_i \in r_i \; and \; o_j \in r_j \; and \; f(o_i, o_j)\} \tag{1}$$

where f is an *evaluation function* $f : r_i \times r_j \rightarrow \{true, false\}$.

An operation of *semicomposition* on data containers r_i and r_j is defined as

$$r_i \dashv_f r_j = \{(o_i : o_i \in r_i \; and \; \exists o_j \in r_j f(o_i, o_j)\} \tag{2}$$

where f is an *evaluation function* $f : r_i \times r_j \rightarrow \{true, false\}$.

An operation of *elimination* on data containers r_i and r_j is defined as

$$r_i -_f r_j = \{o_i : o_i \in r_i \; and \; not \; \exists o_j \in r_j f(o_i, o_j)\} \tag{3}$$

where f is an *evaluation function* $f : r_i \times r_j \rightarrow \{true, false\}$.

An operation of *union* on data containers r_i and r_j is defined as

$$r_i \cup_f r_j = \{o_i : (o_i \in r_i \; or \; o_i \in r_j) \; and \; f(o_i)\} \tag{4}$$

were f is an *evaluation function* $f : r_i \rightarrow \{true, false\}$.

An operation of *elimination* on a data container r_i is defined as

$$\sigma_f(r_i) = \{o_i : o_i \in r_i \ and \ f(o_i)\} \tag{5}$$

where f is an *evaluation function* $f : r_i \to \{true, false\}$.

Like for an internal structure of data objects, a precise syntax of an evaluation function is not determined in the abstract data model. Selection of the particular internal structures for the simple and composite data objects and a syntax for an elimination function defines a concrete data model. For example, a choice of n-tuples as a unified internal structure of all data objects and a syntax of formulas of prepositional logic for an elimination function defines a relational data model with the operations of join, semijoin, antijoin, union, and selection.

A *query* is an expression whose arguments are simple and composite data objects and all its operations belong to a set $\{+_f, \dashv_f, -_f, \cup_f, \sigma_f\}$.

Let $f(x, y)$ be an evaluation function $f : r_i \times r_j \to \{true, false\}$. A *signature* of f is a pair (r_i, r_j).

A *Projection of function* $f(x, y)$ on an object $o_j \in r_j$ is denoted as $f(x|o_j)$ and it is defined as $f(x|o_j) : r_i \to \{true, false\}$. A projection of a function $f(x, y)$ on an object $o_j \in r_j$ is obtained through a systematic replacement of an argument y with a constant object o_j. For example if an evaluation function $f(x, y)$ is implemented as `return((x.a+y.b)>5)` then projection of the function on an object o_j such that $o_j.b = 3$ is a function $f(x|o_j)$ implemented as `return((x.a+3)>5)`.

Let T denotes a task submitted at a central site of a distributed information system and let $t_1, \ldots, t_n$ be its decomposition into the individual tasks to be processed at the remote sites of the system. Let $S = \{\top, \bot, t_1, \ldots, t_n\}$ be a set where $\top$ is a *start of processing* symbol, $\bot$ is an *end of processing* symbol. Then, a partial order $P \subseteq S \times S$ such that $< S, P >$ is a lattice where $sup(S) = \top$ and $inf(S) = \bot$ and any pair $(t_i, t_j) \in P$ is called as a *task processing schedule*. For instance, a lattice given in a Figure 1 represents a task processing schedule where the system starts from the simultaneous submission of the tasks t_1, t_2, t_3. When the results of t_2 are available, the system submits t_4. When both results of t_2 and t_3 are available the system submits t_5.

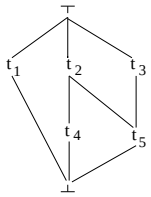

Fig. 1. A sample task processing schedule

Let $r_1, \ldots, r_n$ denote the results of the tasks $t_1, \ldots, t_n$. An expression that determines how to combine $r_1, \ldots, r_n$ into the final answer is called as a *data integration expression* and it is denoted as $e(r_1, \ldots, r_n)$.

4 Evaluation of integration strategies

Consider a task processing schedule $S \subseteq T \times T$ where $T = \{\top, \bot, t_1, \ldots, t_n\}$ The cost of a schedule S is measured as the total amount of time required to transmit the results $r_1, \ldots, r_n$ to a central site. The total transmission time depends on the amounts of transmitted data and transmission speed of a network. With an entirely parallel processing schedule the total transmission time is equal to

$$max(|r_1|/\tau_1, \ldots, |r_n|/\tau_n) \tag{6}$$

where τ_i is a transmission speed from a remote system i and $|r_i|$ is the total amount of data transmitted from a remote system i. When one of $|r_i|/\tau_i$ is significantly bigger then the others then it is beneficial to delay the processing of t_i until the results of $r_1, \ldots, r_{i-1}, r_{i+1}, \ldots, r_n$ are available at a central site and to use these results to modify t_i to t'_i such that its result r'_i is smaller than r_i. Then, the total transmission time is equal to

$$max(|r_1|/\tau_1, \ldots, |r_{i-1}|/\tau_{i-1}, |r_{i+1}|/\tau_{i+1}, \ldots, |r_n|/\tau_n) + |r'_i|/\tau_i, \tag{7}$$

When a value of (7) is smaller than a value of (6) then a hybrid task processing schedule, that delays processing of a task t_i and transforms it to reduce transmission time is better than entirely parallel schedule.

An important problem in the evaluation of alternative task processing schedules is estimation of the sizes $|r_i|$ and $|r'_i|$. In the database systems where the query processors use cost based optimization techniques it is possible to get information about an estimated total amount of data returned by a query. For example the cost based query optimizers in relational database systems use histograms on columns of relational tables to estimate the total number of rows returned by a query and SQL statement `EXPLAIN PLAN` can be used find a query execution plan and estimated amounts of data processed accordingly to the plan. Then, it is possible to estimate the values $|r_1|, \ldots, |r_n|$ before the queries are processed. These results can also be used to estimate the reductions of data transmission time when a task t_i is transformed into $t'_i = \sigma_{f(x|r_j)}(t_i)$ where $f(x|r_j)$ is a projection of elimination function on the results r_j. The transformations of t_i are explained in the next section. If an elimination operation removes from r_i data objects that do not satisfy a condition $f(x|r_j)$ then smaller r_j reduces r_i to a larger extent. On the other hand, if elimination removes from r_i data objects that do not have matching data items in the data objects in r_j then larger r_j reduces r_i more than smaller r_j. When it is possible to get information about the total number of data objects included in r_i and r_j then together with a known projection of elimination function $f(x|r_j)$ and known the distributions of data items in objects in r_i and r_j it is possible to estimate the size of $\sigma_{f(x|r_j)}(t_i)$ and find whether processing of t_j before t_i is beneficial.

5 Transformations of task processing schedules

In this section we consider the tasks t_i and t_j to be processed at the remote systems and we show when and how a task t_i can be transformed by the results r_j. We start from an example that explains an idea of transformations of task processing schedules.

Consider a task T submitted at a central site and decomposed into the tasks t_1, t_2, t_3, and t_4 to be processed at the remote sites. Let r_1, r_2, r_3, r_4 denote data containers with the results of the individual tasks and let a data integration expression $(r_1 +_{f_1} r_2) +_{f_3} (r_3 -_{f_2} r_4)$ determines how the results of individual tasks must be "assembled" into the final result. Assume, that an evaluation function f_3 has a signature (r_1, r_3). It means that implementation of f_3 uses only the data items from the data containers r_1 and r_3. Then it is possible to transform the data integration expression into an equivalent form $((r_1 \dashv_{f_3} r_3) +_{f_1} r_2) +_{f_3} (r_3 -_{f_2} r_4)$. The result of the transformed expression is the same as the result of the original expression because a subexpression $r_1 \dashv_{f_3} r_3$ removes from r_1 data objects which would not contribute to the result of operation $+_{f_3}$ in the transformed expression. It means that a task t_1 can be transformed into an expression $t_1 \dashv_{f_3} r_3$. Unfortunately, due to a high level of autonomy of a remote system the expression cannot be computed in its present form. A remote system does not accept any tasks that include the input data containers like for example r_3. Therefore the expression must be transformed into a form that can be processed by a remote system. We consider an evaluation function $f_3(x, y) : r_1 \times r_3 \rightarrow \{true, false\}$ and its projections $f_3(x|o_1), \ldots, f_3(x|o_n)$ on the objects $o_1, \ldots o_n \in r_3$. Next, we replace an expression $t_1 \dashv_{f_3} r_3$ with $\sigma_{f_3(x|o_1) \ or \ \ldots \ or \ f_3(x|o_n)}(t_1)$. It means that we construct a new task that filters the results of t_1 with a condition built over the values of data items in the objects $o_1, \ldots o_n \in r_3$. As a consequence, an entirely parallel task processing schedule of can be changed into a schedule where processing of t_3 precedes processing of t_1 while t_2 and t_4 are still processed simultaneously.

A problem how to transform an entirely parallel schedule can be expressed in the following way. Let T be a task submitted at a central site and decomposed into the tasks $t_1, \ldots, t_n$ to be processed at the remote systems. Let $e(r_1, \ldots, r_n)$ be a data integration expression build over the operations $\{+_f, -_f, \cup_f\}$ and the partial results $r_1, \ldots, r_n$ obtained from the processing of $t_1, \ldots, t_n$ at the remote systems. A question is when and how a task t_i can be transformed into a task t'_i using the results r_j of a task t_j such that a result of data integration expression $e(r_1, \ldots, r_i, \ldots, r_n)$ is the same as a result of expression $e(r_1, \ldots, r'_i, \ldots, r_n)$ where r'_i is the result of transformed task t'_i.

We consider a syntax tree T_e of data integration expression $e(r_1, \ldots, r_n)$ and the smallest subtree T_{ij} of T_e that contain both arguments r_i and r_j. A syntax tree T_e is constructed such that the arguments $r_1, \ldots, r_n$ are located at the leaf nodes and the operations of data integration expression are located at non-leaf nodes. Let $\alpha_f \in \{+_f, -_f, \cup_f\}$ be an operation located at the root node of a subtree T_{ij}. If a signature of an elimination function f is equal to (r_i, r_j) then

a task t_i can be transformed using a result r_j or a task t_j can be transformed using a result r_i of a task t_i.

In the example above t_1 can be reduced with the results of t_3 and the opposite because a signature of an operation $+_{f_3}$ in the root of the smallest syntax tree that contains r_1 and r_3 is equal to (r_1, r_3).

In the specific cases the condition determined above may not be satisfied and still it is possible to transform a data integration expression. For example if in expression $(r_1 +_{f_1} r_2) +_{f_3} (r_3 -_{f_2} r_4)$ a signature of f_3 is equal to (r_2, r_3) and signature of f_2 is equal to (r_3, r_4) and $f_2(x_3, x_4)$ is implemented as $\texttt{return}(x_3 = x_4)$ then it is still possible to transform a task t_2 to a form $\sigma_{not\ f_3(x|o_1)\ or\ \ldots\ or\ not\ f_3(x|o_n)}(t_2)$ where $o_1, \ldots, o_n \in r_4$. This is because an equality condition $x_3 = x_4$ in implementation of a function f_2 makes r_3 in in a signature of f_3 equal to r_4 and the second argument of an operation $+_{f_3}$ does not contain objects included in r_4. In the specific cases it is possible to transform the queries despite that signature does not satisfy a condition above.

	$+_f$	$(left) -_f$	$-_f (right)$	$\cup_f$
d	$d-$	$d-$	$-d$	$d*$
$d-$	$d-$	$d-$	$d*$	$d*$
$-d$	$-d$	$-d$	$d*$	$d*$
$d*$	$d*$	$d*$	$d*$	$d*$

Table 1. The labeling rules for syntax trees of data integration expressions

The next problem is to find a transformation that in a general case can be applied to a given task t_i to reduce transmission time of its results r_i. To discover a transformation we label a syntax tree T_e in the following way.

(i) An edge between a leaf node that represent an argument d is labeled with d.
(ii) If a node n in T_e represents an operation α that produces a result r and "child" edge of a node n is labeled with one of the symbols d, $d-$, $-d$, $d*$ then a "parent" edge of n can be labeled with a symbol located in a row indicated by a label of "child" edge and a column indicated by an operation α in Table 1.

The interpretations of the labels are the following. A label d attached to a "child" edge of composition operation at root node of the tree indicate that all d data objects are processed by the operation. A label $d-$ attached to a "child" edge of the same operation indicates that only a subset of data objects of an argument d are processed by the operation. A label $-d$ attached to the same edge indicates that none of data objects d are processed by the operation. A label $d*$ indicates that some of data objects in d and some other data objects are processed by the operation.

As an example, consider an integration expression $(r_1 -_{f_1} r_2) +_{f_2} r_3$. The "parent" edges of the nodes r_1, r_2, and r_3 obtain the labels r_1, r_2, and r_3. A left

"child" edge of the root node obtained a label r_1- indicated by a location in the first row and the second column in Table 1. Moreover, the same edge obtains a label $-r_2$ indicated by a location in the first row and the third column in Table 1. A complete labeling is given in a Figure 2.

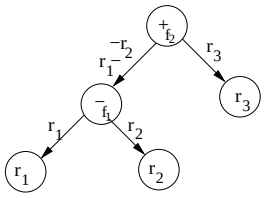

Fig. 2. A labeled syntax tree of data integration plan $(r_1 -_{f_1} r_2)) +_{f_2} r_3$

$+_f$	r_j	r_j-	$-r_j$	r_j*
r_i	$\sigma_{f(x\|r_j)}(t_i)$ $\sigma_{f(r_i\|y)}(t_j)$	$\sigma_{f(x\|r_j)}(t_i)$ $\sigma_{f(r_i\|y)}(t_j)$	$\sigma_{not\ f(x\|r_j)}(t_i)$ $\sigma_{f(r_i\|y)}(t_j)$	$\sigma_{f(r_i\|y)}(t_j)$
r_i-	$\sigma_{f(x\|r_j)}(t_i)$ $\sigma_{f(r_i\|y)}(t_j)$	$\sigma_{f(x\|r_j)}(t_i)$ $\sigma_{f(r_i\|y)}(t_j)$	$\sigma_{not\ f(x\|r_j)}(t_i)$ $\sigma_{not\ f(r_i\|y)}(t_j)$	$\sigma_{f(r_i\|y)}(t_j)$
$-r_i$	$\sigma_{not\ f(r_i\|y)}(t_j)$ $\sigma_{f(x\|r_j)}(t_i)$	$\sigma_{not\ f(x\|r_j)}(t_i)$ $\sigma_{not\ f(r_i\|y)}(t_j)$	$\sigma_{not\ f(r_i\|y)}(t_j)$ $\sigma_{not\ f(x\|r_j)}(t_i)$	$\sigma_{not\ f(r_i\|y)}(t_j)$
r_i*	$\sigma_{f(x\|r_j)}(t_i)$	$\sigma_{f(x\|r_j)}(t_i)$	$\sigma_{not\ f(x\|r_j)}(t_i)$	*none*

Table 2. The transformations of arguments in task processing schedules (1)

$-_f$	r_j	r_j-	$-r_j$	r_j*
r_i	$\sigma_{f(r_i\|y)}(t_j)$ $\sigma_{not\ f(x\|r_j)}(t_i)$	$\sigma_{f(r_i\|y)}(t_j)$	$\sigma_{f(r_i\|y)}(t_j)$	$\sigma_{f(r_i\|y)}(t_j)$
r_i-	$\sigma_{f(r_i\|y)}(t_j)$	$\sigma_{f(r_i\|y)}(t_j)$	$\sigma_{f(r_i\|y)}(t_j)$	$\sigma_{f(r_i\|y)}(t_j)$
$-r_i$	$\sigma_{not\ f(r_i\|y)}(t_j)$ $\sigma_{not\ f(x\|r_j)}(t_i)$	$\sigma_{not\ f(r_i\|y)}(t_j)$	$\sigma_{not\ f(r_i\|y)}(t_j)$ $\sigma_{not\ f(x\|r_j)}(t_i)$	$\sigma_{not\ f(r_i\|y)}(t_j)$
r_i*	$\sigma_{not\ f(x\|r_j)}(t_i)$	$\sigma_{not\ f(x\|r_j)}(t_i)$	*none*	*none*

Table 3. The transformations of arguments in task processing schedules (2)

The interpretation of the transformations included in the Tables 2 and 3 is the following. Consider the arguments r_i and r_j included in the smallest subtree of a syntax tree of data integration expression If an operation in the root of the subtree is $+_f$ then the possible transformations r_i and r_j are included in a Table 2. If an operation in the root of the subtree is $-_f$ then the possible

transformations r_i and r_j are included in a Table 3. The replacements of the arguments r_i and r_j can be found after the labeling of both paths from the leaf nodes representing the both arguments towards the root node of the subtree. The transformations of the arguments r_i and r_j 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 r_i and r_j such that an operation $+_f$ is in the root node of the subtree. If a left "child" edge of the root node is labeled with $-r_i$, and a right "child" edge of the root node is labeled with r_j* then a Table 2 indicates that it is possible to replace the contents of an argument t_j with an expression $\sigma_{not\ f(r_i|y)}(t_j)$.

As an example consider a data integration expression $(r_1 -_{f_1} r_2)) +_{f_2} r_3$ and its labeling is given in a Figure 2. The following transformations of the arguments are possible. A query t_1 can be replaced with $\sigma_{not\ f_1(x|r_2)}(t_1)$ or with $\sigma_{f_2(x|r_3)}(t_1)$. A query t_2 can be replaced with $\sigma_{f_1(r_1|y)}(t_2)$ or with $\sigma_{f_2(x|r_3)}(t_2)$. A query t_3 can be replaced with $\sigma_{not\ f_2(r_2|y)}(t_3)$ or with $\sigma_{f_2(r_1|y)}(t_3)$. It is possible to apply both transformations. For example, if we plan to process both t_1 and t_2 before t_3 then t_3 can replaced with $\sigma_{not\ f_2(r_2|y)}(\sigma_{f_2(r_1|y)}(t_3))$.

6 Optimization of task processing schedules

At an early stage of data processing at a central site a task T is decomposed into the tasks $t_1, \ldots, t_n$ to be submitted for processing at the remote sites and a data integration expression $e(r_1, \ldots, r_n)$ determines how to combine the results $r_1, \ldots, r_n$ into the final answer. Optimization of a task processing schedule finds an order in which the individual tasks $t_1, \ldots, t_n$ are submitted for processing to minimize the total data transmission time from the remote systems to a central site. The initial task processing schedule is an entirely parallel schedule where all tasks $t_1, \ldots, t_n$ are submitted for processing in one moment in time and processed simultaneously at the remote systems. Optimization of an entirely parallel task processing schedule consists of the following steps.
For all pairs of results (r_i, r_j) perform the following actions:

(1) Find in a syntax tree T_e of a data integration expression $e(r_1, \ldots, r_n)$ the smallest subtree that contain both arguments r_i and r_j. Find an operation α_f in the root node of the subtree. If a signature of an elimination function f is (r_i, r_j) then progress to the next step, otherwise consider the next pair of arguments (r_i, r_j).
(2) Use a Table 1 to label the paths from the leaf nodes r_i and r_j to the root node α_f of the subtree.
(3) Use the Tables 2 and 3 to find the transformations of t_i by a result r_j and t_j by a result r_i.
(4) Compare the costs of the following data integrations plans: (i) t_i processed simultaneously with t_j, (ii) t_i processed before a transformed t_j, (iii) t_j processed before a transformed t'_i and record the best processing order, i.e.

a pair (t_i, t_j) or a pair (t_i, t_j) or nothing if simultaneous processing of t_i and t_j provides the smallest costs.

Next, we use the pairs of queries obtained from a procedure above to construct a scheduling lattice. The queries $t_1, \ldots, t_n$ are the labels of the nodes in the lattice and each pair (t_i, t_j) contributes to an edge from node t_i to a node t_j where t_i is located "above" t_j in the lattice. Finally the nodes labeled with $\top$ and $\bot$ are added to the scheduling lattice.

As an example consider a data integration expression $(r_1 -_{f_1} r_2)) +_{f_2} r_3$ and its labeling is given in a Figure 2. The following transformations of the arguments are possible. A query t_1 can be replaced with $\sigma_{not\ f_1(x|r_2)}(t_1)$ or with $\sigma_{f_2(x|r_3)}(t_1)$. A query t_2 can be replaced with $\sigma_{f_1(r_1|y)}(t_2)$ or with $\sigma_{f_2(x|r_3)}(t_2)$. A query t_3 can be replaced with $\sigma_{not\ f_2(r_2|y)}(t_3)$ or with $\sigma_{f_2(r_1|y)}(t_3)$. It is possible to apply both transformations. For example, if we plan to process both t_1 and t_2 before t_3 then t_3 can replaced with $\sigma_{not\ f_2(r_2|y)}(\sigma_{f_2(r_1|y)}(t_3))$. If estimation of the processing times indicated that the results r_2 and r_2 used to transformation of task t_3 into $t'_3 = \sigma_{not\ f_2(r_2|y)}(\sigma_{f_2(r_1|y)}(t_3))$ reduce the transmission of the results d'_3 such that $max(|r_1|/\tau_1, |r_2|/\tau_2) + |d'_3|/\tau_3| < max(|r_1|/\tau_1, |r_2|/\tau_2, |r_3|/\tau_3|)$ then simultaneous processing of the tasks t_1 and t_2 followed processing of t_3 is more efficient than entirely parallel processing of t_1, t_2, and t_3.

7 Summary and open problems

In this work we consider optimization of task processing schedules in distributed information system. A task submitted for processing at a central site of the system is decomposed into a number of individual tasks to be processed at the remote sites. A parallel processing schedule of the individual tasks does not always minimize data transmission time and its transformation into a sequential or hybrid schedule may provide shorter response time. This work shows how to transforms entirely parallel task processing schedules into more optimal hybrid schedules where certain tasks are processed simultaneously while the other tasks are processed sequentially. The transformations are guided by the cost-based optimizations whose objective is to reduce the total data transmission time. We show that the properties of data integration expressions can be used to find more efficient schedules. We propose a technique of labeling of syntax trees of data integration expressions to find the coincidences between the arguments. Different types of coincidences between the arguments determine possible transformations of data processing tasks. We show how to use the results of the tasks processed earlier to transform the tasks still waiting for processing in a way that reduce transmission time of their results.

The avenues for further research in this area include the analysis of previous results to estimate the amounts of time needed to transfer the results of individual tasks, and binding optimization of data processing schedules with optimization of processing of data integration expressions. An important factor in optimization of task processing schedules is the ability to precisely predict the amounts of data transmitted from the remote sites by the individual tasks.

Recording the characteristics of data processing tasks and the respective amount of data would provide statistical information that later on can be used to more precisely estimate the future transmission size. At the moment processing of data integration expression is resumed only whenever the complete partial results of task processing are available at a central site. An interesting idea would be to process a data integration expression in an online mode where an increment of the partial results would trigger the computations of data integration expression. Such technique would better utilize the available computing resources and it will more evenly spread processing load in time. The other interesting problems include an extension of cost based optimization on both task processing time at a remote site and data transmission time and investigation of an impact of different types of elimination function on transformations of data processing tasks.

References

1. Ahmad, M., Aboulnaga, A., Babu, S., Query interactions in database workloads.In: Proceedings of the Second International Workshop on Testing Database Systems, 1–6, (2009)
2. Ahmad, M., Duan, S., Aboulnaga, A., Babu, S., Predicting completion times of batch query workloads using interaction-aware models and simulation. In: Proceedings of the 14th International Conference on Extending Database Technology, 449–460, (2011)
3. Braumandl, R., Keidl, M., Kemper, A., Kossmann, D., Kreutz, A., Seltzsam, S., Stocker, K., ObjectGlobe: Ubiquitous query processing on the Internet. In: The VLDB Journal, vol. 10, no. 1, 48-71, (2001)
4. Costa, R.L-C., Furtado, P., Runtime Estimations, Reputation and Elections for Top Performing Distributed Query Scheduling. In: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 28–35, (2009)
5. Du, W., Krishnamurthy, R., Shan M-C., Query Optimization in Heterogeneous DBMS. In: Proceedings of the 18th VLDB Conference, 277–299, (1992)
6. Friedman, M., Levy, A., Millstein, T.: Navigational plans For Data Integration. In: Proceedings of the National Conference on Artificial Intelligence, 67–73, (1999)
7. Ilarri, S., Mena, E., Illarramendi, A., Location-dependent query processing: Where we are and where we are heading. In: ACM Computing Surveys, vol. 42, no. 3, 1–73, (2010)
8. Lenzerini, M., Data Integration: A Theoretical Perspective. (2002)
9. Mishra, C., Koudas, N., The design of a query monitoring system. In: ACM Transactions on Database Systems, vol. 34, no. 1, 1–51, (2009)
10. Nam, B., Shin, M., Andrade H., Sussman, A.,Multiple query scheduling for distributed semantic caches. In: Journal of Parallel and Distributed Computing, vol.70, no. 5, 598–611, (2010)
11. Harangsri, B., Shepherd, J., Ngu. A.: Query Classification in Multidatabase Systems. In: Proceedings of the 7th Australasian Database Conference, 147–156, (1996)
12. Ives, Z.G., Green, T.J., Karvounarakis, G., Taylor, N.E., Tannen, V., Talukdar, P.P., Jacob, M., Pereira F., The ORCHESTRA Collaborative Data Sharing System. In: SIGMOD Record, (2008)

13. Liu L., Pu, C., A Dynamic Query Scheduling Framework for Distributed and Evolving Information Systems. In: Proceedings of the 17th International Conference on Distributed Computing Systems, (1997)
14. Lu, H., Ooi, B-C., Goh, C-H.: Multidatabase Query Optimization: Issues and Solutions, In: Proceedings RIDE-IMS'93, Research Issues in Data Engineering: Interoperability in Multidatabase Systems, 137–143, April (1993)
15. Ozcan, F., Nural, S., Koksal, P., Evrendilek, C., Dogac, A.: Dynamic Query Optimization in Multidatabases. In: Bulletin of the Technical Committee on Data Engineering, vol. 20, no. 3, 38–45 (1997)
16. Sheth, A.P., Larson, J.A.: Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. In: ACM Computing Surveys, vol.22, no. 3, 183–236, (1990)
17. Srinivasan, V., Carey, M.J.: Compensation-Based On-Line Query Processing. In: Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, 331–340 (1992)
18. Thain, D. Tannenbaum, T., Livny, M., Distributed computing in practice: the Condor experience: Research Articles. In: Concurrency Computing: Practice and Experience. vol. 17, no. 2-4, 323–356, (2005)
19. Wache, H., Vogele T., Visser, U., Stuckenschmidt, H., Schuster, G., Neuman H., Hubner, S., Ontology-Based Integration of information - A Survey of Existing Approaches. (2001)
20. Zhou, Y., Ooi, B.C., Tan, K-L., Tok, W. H., An adaptable distributed query processing architecture. In: Data and Knowledge Engineering,vol. 53, no. 3, 283–309, (2005)
21. Zhu, Q., Larson, P.A.: Solving Local Cost Estimation Problem for Global Query Optimization in Multidatabase Systems. Distributed and Parallel Databases, vol. 6, no. 4, 373–420 (1998)
22. Ziegler, P., Three Decades of Data Integration - All problems Solved ? In: 18th IFIP World Computer Congress, vol. 12, (2004)