Computer Science and Information Systems 11(1):69–88 DOI: 10.2298/CSIS120825063Y

Optimizing Resource Allocation for Approximate Real-Time Query Processing*

Anna Yarygina and Boris Novikov

Saint Petersburg University
Universitetsky prosp. 28
198504 Saint Petersburg, Russia
anya_safonova@mail.ru, b.novikov@spbu.ru

Abstract. Query optimization techniques are proved to be essential for high performance of database management systems. In the context of new querying paradigms, such as similarity based search, exact query evaluation is neither computationally feasible nor meaningful, and approximate query evaluation is the only reasonable option.
In this paper a problem of resource allocation for approximate evaluation of complex queries is considered. An approximate algorithm for a near-optimal resource allocation is presented, providing the best feasible quality of the output subject to a limited total cost of a query. The results of experiments have shown that the approximate resource allocation algorithm is accurate and efficient.

Keywords: query optimization, approximate query evaluation, resource allocation.

1. Introduction

Declarative query languages for database management systems are both effective as high level tools for specification of data processing needs and computationally efficient due to availability of powerful optimizers.

Heterogeneous autonomous information resources, as well as user needs may require diverse querying paradigms to be used in a declarative query. For example, the user might need to combine data on company products extracted from relational database with sentiments from customer tweets on these products obtained by means of natural language processing techniques. For some of data management models, such as probabilistic and similarity-based, the traditional exact queries are neither computationally feasible nor pragmatically meaningful. Hence an approximate query evaluation is the only option.

Query evaluation is approximate if the output is, in a certain sense, incomplete or imprecise. There are different types of approximate query evaluation: based on nature of the query (similarity based processing) or algorithms providing imprecise answer (aggregation based on sampling). Obviously, approximate evaluation makes sense only if it requires less resource (e.g. processing time) than an exact one. Informally, the result is expected to be better if more resources are spent on the query evaluation.

In this research we consider a controllable approximate query evaluation based on approximate algorithms implementing algebraic operations. We estimate how good the

* This research is supported by HP Labs.

result is with a numeric value which is called data quality. Similarly the operation quality shows how the operation affects the quality of processed data.

A user query is translated into an algebraic expression (query evaluation plan) in terms of a certain set of operations. Typically these operations constitute a variation or extension of the relational algebra; however, the semantics of algebraic operations is not essential for the algorithms presented in this paper, the consideration is restricted to the properties of approximate execution. For each approximate operation the relative quality of its output depends on the amount of resources allocated. Usually the critical resource is time. Further we use the terms resource, time, and cost interchangeably.

Thus the limited amount of resources has to be allocated among operations in query evaluation plan to balance between the query evaluation cost and quality. Further when we talk about exact (in contrast to approximate) query evaluation plan we consider the one with unlimited amount of resources for query evaluation.

Let us illustrate the above with an example. Consider a query:

Find least rated company products based on retailer ratings.

The idea is to find products with low ratings to analyze them. At the first glance, the answer to this query may be obtained from any retailer site. However, to make results more reliable, we might want to combine the output of several services. An appropriate query evaluation plan should include data extraction from different sources followed by a join on product name, with subsequent ranking.

```
1   rank and sort
2       join on product name
3           get ratings for company products from reatiler1
4           get ratings for company products from reatiler2
```

The exact evaluation of this query requires exhaustive extraction of data from both sources, might be time consuming and, most likely, is not needed as the user is interested in the few products which need detailed analysis. Thus the input from data sources should be limited to certain abounds, resulting in a partial loss of results and hence affecting the quality of final output. Just for this example, the quality may be estimated as a percentage of correctly returned objects in the query output.

Let us suppose that two data sources respond with a series of objects, at average rates of 300 and 500 per second, respectively, and also suppose the user expects an answer in 200 milliseconds and the best feasible output contains 50 objects.

The available amount of resources (200 msec in this example) may be distributed between operations differently, resulting in different output quality. To possible allocation of resources for example query are demonstrated in figure 1.

Consider a resource allocation where the first data source utilizes 150 msec and returns 45 objects, the second source utilizes 30 msec and returns 15 objects, and remaining 20 msec are needed for join and ranking (see figure 1(a)). The final output will contain at most 15 objects and hence the quality estimation cannot exceed 30%.

Much better results may be obtained if the first data source will utilize 100 msec, second - 60 msec, leaving 40 msec for subsequent join and ranking (see figure 1(b)). Both data sources will produce 30 objects and the output quality may reach up to 60%.

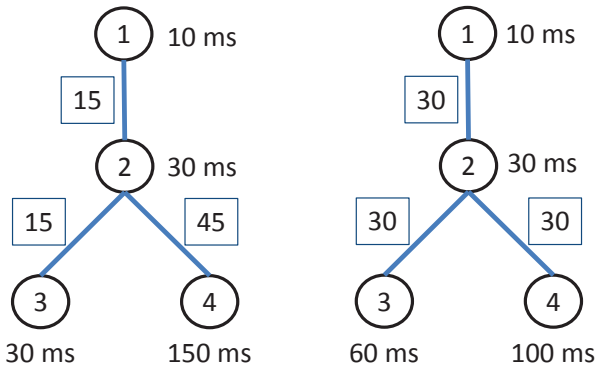


Fig. 1. Resource allocation for example query

To make this example simple, we ignore impact of parallel execution and used over-simplified estimations for both cost and quality.

The example above suggests that the response time can be significantly improved if the extraction of data from primary sources is restricted to a small number of items and approximate algorithms to calculate join predicates are applied. This restriction will affect the quality of the result, as some potentially good items will be excluded. Further, it does not make sense to restrict some operations but leave others untouched, as the quality of the result will be affected by the input of the worse quality. In other words, allocation of resources to operations should be balanced.

Optimization techniques should be re-considered in the context of approximate evaluation. As soon as data quality is included into consideration, the optimization problem becomes multi-objective, and hence trade-offs between objectives appear on the stage. Possible options are either “provide the minimal cost yielding at least specified quality”, or “provide the best possible quality for at most given amount of resources”. The remaining part of this paper considers the latter problem only.

The restriction on the amount of resources to be spent on the query evaluation effectively means that the response time is predictable. Consequently, the query optimization problem addressed in our research is suitable for real-time systems, where predictability of response time is one of the most essential requirements.

In this research a user query is translated into an algebraic expression in terms of a certain set of operations. Typically these operations constitute a variation or extension of the relational algebra; however, the semantics of algebraic operations is not essential for the algorithms presented in this paper, the consideration is restricted to the properties of approximate execution. For each operation the relative quality of its output depends on the amount of resources allocated to the operation. Usually the critical resource is time. Further we use the terms resource, time, and cost interchangeably.

The contribution of this paper is an algorithm for near-optimal allocation of limited resources between operations in query evaluation plan providing the best possible quality of the output.

We define an extended abstract cost model providing trade-offs between quality and cost for all operations and proceed with the formal statement of the resource allocation

problem. We then provide a solution to the problem for some special cases and proceed with an algorithm for a general case.

As far as we know a research in the area of query optimization and approximate query evaluation do not cover the problem of optimal resource allocation for approximate query evaluation. Approximate algorithms for operations like join, top k, aggregation are a hot topic in literature. However, the problem of distribution of limited amount of time among controllable approximate algorithms implementing different operations in a query is still open.

This paper is an extended and substantially revised version of [17]. We included all proofs omitted in the short version, relaxed the restriction on arity of operations, and provided additional explanations and clarifying examples.

The rest of the paper is structured as follows. Section 2 describes a formal model of operations, including the extended cost model and a formal problem statement. The model overview and auxiliary lemmas are followed by an algorithm specification in section 3. The experiments and the analysis of their results are presented in section 4. Section 5 outlines the related work.

2. Abstract Model

2.1. Query Evaluation Plans

In this work we consider the query processing in the distributed environment which includes autonomous primary sources of data and operation processing units which constitute the query processing facility and run under control of the query processor. We assume that all information needed for query planning and optimization is accessible to a single coordinating service and do not consider distributed optimization for autonomous processing units. That is, a user query is translated into a single execution plan consisting of operations which may be executed on several units in a distributed system.

The query processor operates with a query evaluation plan represented as a tree. Our main target is the set of operations outlined in [12], for example similarity join, fusion, and top k; although the results of this research are applicable to any set of operations admitting approximate execution.

Let P be an execution plan, that is, a set of operation calls organized as a tree. In this tree vertices are operation calls, and edges connect them with their arguments. We use terms node, operation, and operation call interchangeably, because in this paper we do not consider operations independently from its call in query evaluation plan. It is also important to note that in our query tree leaves are operations which receive data from primary sources, for example, streams, relations, files, and so on. Further the set $args(l) \subset P$ will denote arguments of an operation $l \in P$ or its child nodes in the corresponding query tree. For any node $l \in P$ except the root, $parent(l)$ denotes its parent node. For an operation $l \in P$ the subquery rooted in l will be denoted as $\bar{l}$.

An example query execution plan tree is shown in figure 2. For this tree P includes all nodes, $P = \{l_0, l_1, l_2, l_3, l_4, l_5\}$, a sub-tree rooted in l_2 is $\bar{l_2} = \{l_2, l_4, l_5\}$, $parent(l_5) = l_2$, and $args(l_0) = \{l_1, l_2, l_3\}$.

We assume that data may be imprecise or uncertain, and operations admit approximate evaluation which consumes less resource but may affect the quality of the output. For

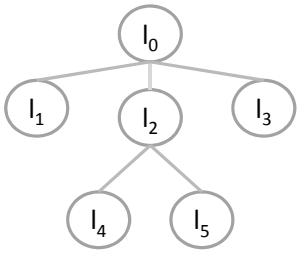


Fig. 2. An example of query tree

example, an approximate calculation of an average value of a certain attribute based on a sampling returns imprecise value but is usually faster than a full scan.

An actual (absolute) quality of data may be defined differently depending on data type. The quality of calculated aggregate values mentioned above might be based on the accuracy, while the quality of information retrieval results may be defined in terms of precision and recall. Another dimension of quality is trustfulness of information sources. In several cases the estimation of actual quality involves human assessment.

Although the semantics of data quality is complex [11, 2], it is neither elaborated nor defined in this research. Instead we assume that the quality of a data set is estimated with a single numeric value and the quality of different data sets is expressed in a comparable way. Further, (most of) operations in our model are approximate and may produce different quality of the output depending on the quality of the input and amount of allocated resources. For example, the quality of an average value might be based on its accuracy, and a larger size of sample is more expensive but, in general, provides better quality.

To estimate the relative impact of a multi-argument operation on the output quality, an estimation of the overall quality of all inputs is needed. In this paper the we use the minimal quality of arguments as an overall estimation.

We define the relative quality of an operation or a plan as a ratio of the achieved result quality to the best one for given arguments.

Consequently, the quality of the output can be controlled with amount of resources assigned to an operation. Our goal is to distribute available resources in such a way that the overall quality of output for the query plan is maximized.

We use *quality function* which returns estimated relative quality for any operation and amount of resources allocated to this operation. In this section we derive certain necessary conditions for an optimal resource allocation. These conditions are stated in a series of lemmas.

To prove that a condition is necessary for an optimal allocation, we start from an allocation which does not satisfy the condition and then build another resource allocation for the same plan which yields better quality (although still is not necessarily optimal).

Subsequent paragraphs describe the approach more formally.

Let us consider unary operation $l \in P$ which receives input data with absolute quality a_{in}. Even the best possible implementation of operation l may reduce the quality of the result data a_{out}. Thus, the quality of operation l equals $\frac{a_{out}}{a_{in}}$. If the operation $l \in P$ receives the unlimited resources for the fixed input data it produces data with the best feasible absolute quality $a_{out}(\infty)$. If l receives amount of resources t, which is less than

needed for the best quality, the absolute quality of its output is $a_{out}(t)$; thus, the relative quality of operation l is $\frac{a_{out}(t)}{a_{out}(\infty)}$. Further we work only with relative quality of operations in query evaluation plans.

For each operation the relative quality of its output depends on the amount of resources allocated to it. Therefore, the behavior of each operation $l \in P$ from the query tree can be expressed with the relative quality function $q(l) : \mathfrak{R} \to \mathfrak{Q}$, where $\mathfrak{R}$ and $\mathfrak{Q}$ represent resources and quality respectively.

Thus, if operation $l \in P$ received the amount of resources $t_l \in \mathfrak{R}$ it achieves quality $\bar{q}(l) = q(l)(t_l)$. Further, we use the notation $\bar{q}(l)$ to indicate the achieved relative quality.

The relative quality $Q(\bar{l})$ of a (sub-)query represents the achieved quality of the whole (sub-)query, where $l \in P$ is a root operation.

The quality $Q(\bar{m})$ of each subquery $\bar{m}$ in the execution plan rooted in $m \in P$ depends on the quality of its child subqueries and properties of the root operation. Say, for a binary operation it depends on 3 parameters: the quality of the left subquery $Q(\bar{l})$, the quality of the right subquery $Q(\bar{r})$, and the quality which the root operation produces $\bar{q}(m)$, where $l, r \in args(m)$.

For multi-argument operations we assume that the impact of the worst argument (in terms of quality) dominates.

More formally, the output quality is estimated as a product of a relative quality for an operation and overall (minimal) quality of its arguments:

$$Q(\bar{l}) = \bar{q}(l) \cdot \min_{m \in args(l)} Q(\bar{m}).$$

2.2. Problem Statement

Informally, the problem is to find the amount of resources $t_l \in \mathfrak{R}$ for any operation $l \in P$ such that the estimated quality of the final query result is maximized for any given total resource specified.

Let us state the problem in exact terms.

We have a query tree organized as described above and fixed amount of time $T \in \mathfrak{R}$.

The set $\bar{t}_P = \{t_l, l \in P\}$ is called a distribution of resources, where $t_l \in \mathfrak{R}$ is the amount of resources allocated for operation $l \in P$. The amount of resources allocated to the subquery rooted in $l \in P$ is denoted as $T_{\bar{l}} \in \mathfrak{R}$.

Problem 1 (Resource Allocation Problem). Given a query execution plan P with root operation $l \in P$ and amount of resources $TT \in \mathfrak{R}$, find a distribution of resources $\bar{t}_P$ such that the quality function $Q(\bar{l})$ is maximized subject to constraints $\sum_{m \in P} t_m \leq T$ and $t_m \geq 0$ for all $m \in P$.

The distribution of resources which solves problem 1 will be called optimal distribution or optimal resource allocation.

2.3. Assumptions

We assume that the relative quality is a non-decreasing continuous bounded function of the allocated amount of resources, i.e. the following conditions hold:

- For each operation $l \in P$ a minimal amount of resources needed to complete the operation is known $t_{min}(l) \in \mathfrak{R}$. This amount yields a certain level of the output quality, and the operation cannot be completed for any smaller amount of resources. Further, we will simply use t_{min} where it is clear from the context what the target is operation.
- For each operation $l \in P$ an amount of time $t_{max}(l) \in \mathfrak{R}$ is known, such that any additional resource does not improve the quality.
- For any amount of resources allocated between the minimal and maximal amount, the quality estimation for the output is a non-decreasing function of the amount allocated, i.e. $\forall l \in P, \forall t_1, t_2 \in \mathfrak{R} : t_1 < t_2\ q(l)(t_1) \leq q(l)(t_2)$.

For the average aggregate evaluation, the minimal amount might be the cost for a sample of cardinality 1, while maximum is the cost of a full scan.

If an approximate evaluation for a given operation is not available or is meaningless, then the minimal and maximal values for this operation are equal to the cost of the exact evaluation and resource allocation is trivial.

In this research we additionally assume that each operation quality function can be approximated with a continuous piecewise linear function. That is, we assume that

$$\forall l \in P\ \ q(l)(t) = \begin{cases} 0, & t < t^0_{min} \\ u^i(l) + s^i(l)(t - t^i_{min}), & t^i_{min} \leq t < t^i_{max} \\ u^i(l) + s^{I_l}(l)(t^{I_l}_{max} - t^{I_l}_{min}), & t^{I_l}_{max} \leq t \end{cases}$$

where $i \in [0, I_l]$ is the number of linear segment, $t \in [t^i_{min}, t^i_{max}]$, $s^i(l)$ is the slope of the corresponding linear segment, and $u^i(l) + s^i(l)(t^i_{max} - t^i_{min}) = u^{i+1}(l)$. The linearity assumption is not over-restrictive, as the quality functions return only estimations. We also assume that each linear segment has non-zero slope, i.e. $s^i(l) > 0$. Feasibility of this kind of extended cost models supporting quality functions is demonstrated in [18], where cost models for selected operations defined in [12] are elaborated.

In the remaining part of this paper we will work with resource increments and consider only one linear segment at each moment of time: $\forall l \in P\ q(l)(\tau) = u(l) + s(l)\tau$, where $t \in [t^i_{min},\ t^i_{max}],\ t_{max} = t^i_{max} - t,\ \tau \in [0, t_{max}]$, and $u(l) = q(l)(t) = u^i(l) + s^i(l)(t - t^i_{min})$.

2.4. Critical Subquery

As soon as the minimal needed resource is allocated to each operation in a query, additional resource may be allocated to some operations to improve the quality of the final result.

In order to simplify the notation, in the rest of the paper we consider only the amounts to be allocated in addition to the amounts already allocated. As soon as an incremental amount is allocated to an operation, the quality function is re-calculated accordingly. After this re-calculation the value of the quality function $q(l)(0)$ always equals the relative quality achieved for this operation.

Obviously, the resource should be allocated only to operations which have an impact on the output quality. These operations constitute a *critical sub-query* C. Any algorithm solving the resource allocation problem has to operate with the critical subquery only.

The critical subquery is a set of nodes in the query tree $C \subseteq P$ which can be constructed recursively:

- if $l \in P$ is root of P then $l \in C$;
- if $Q(\bar{l}) = \min_{m \in args(parent(l))}(Q(\bar{m})$, where $parent(l) \in C$ then $l \in C$.

In other words operation $l \in P$ belongs to the critical subtree C if the quality of the subtree rooted in l equals the minimum among the quality of the sibling subtrees and $parent(l)$ is also critical. Edges in the critical subtree are those for the query tree which connect critical operations.

Now we are ready to prove the optimality of the resource allocation along the critical subtree.

Lemma 1. *For each $l \in P \setminus C$ $t_l = 0$ for any optimal resource allocation.*

Proof. Assume node $l \notin C$ has sibling nodes. The quality of the whole query depends on $\min_{n \in args(o)}(Q(\bar{n})(T_{\bar{n}})) = Q(\bar{m})(T_{\bar{m}})$, where $args(o) \ni l, m$; and $Q(\bar{l}) < Q(\bar{m})$. We have to allocate resource to the subquery rooted in node m to increase this component, that is $T_{\bar{m}} > 0$. In case when $T_{\bar{l}} > 0$ resource allocated to the subquery rooted in operation l does not improve the result quality. Therefore, for each operation p in the corresponding non-critical subquery $t_p = 0$.

Assume node $l \notin C$ has no sibling nodes. In this case l has an ancestor node which meets this conditions and therefore $t_l = 0$ as well.

It is important to mention that the quality of the whole query is equal to the quality of its critical subquery.

2.5. Resource Allocation along Paths

In case when the critical subquery is a single path the quality function of the query is a product of the linear quality functions of its operations.

The following lemma 2 provides a necessary condition for resource allocation on a vertical path for a special case when quality functions are linear.

Lemma 2. *Let C be a critical path, T the amount of resources for allocation, for each $n \in C$ the quality function is linear, that is $q(n)(t) = u(n) + s(n)t$, where t is within the limits for the quality function. For the optimal resources allocation the following two conditions hold:*

- *if $t_m = 0$ and $t_l > 0$ then $\frac{u(l)}{s(l)} + t_l \leq \frac{u(m)}{s(m)}$,*
- *if $t_l > 0$ and $t_m > 0$ then $\frac{u(l)}{s(l)} + t_l = \frac{u(m)}{s(m)} + t_m$,*

where $m, l \in C$

Proof. The objective is to maximize the value of the function

$$F(\bar{t_C}) = \prod_{o \in C}(s(o)) \prod_{o \in C}(\frac{u(o)}{s(o)} + t_o)$$

subject to constraints $T = \sum_{o \in C} t_o$ and $t_o \geq 0$, where $\bar{t}_C = \{t_o, o \in C\}$ is a resource distribution. We skip the constant factor in the objective function for the rest of the proof.

Let $r(o) = \frac{u(o)}{s(o)}$ for any $o \in C$, and let $\bar{x}_C$ be any distribution of resources satisfying the constraints, that is, $x_o \geq 0$ and $T = \sum_{o \in C} x_o$.

We first prove that if $r(l) + x_l < r(m) + x_m$ and $x_m > 0$ then the distribution x_C is not optimal. Let $d = min(x_m, \frac{r(m)+x_m-r(l)-x_l}{2})$ and $g = \frac{r(m)+x_m+r(l)+x_l}{2}$. We build a distribution $\bar{y}_C$ such that $y_l = x_l + d, y_m = x_m - d$, and $y_o = x_o, o \notin \{l, m\}$. Let us show that in this case the value of the objective function is larger, i.e. $F(\bar{y}_C) > F(\bar{x}_C)$.

If $y_m = 0$ then $r(m) + x_m - r(l) - x_l \geq 2x_m$ or $r(m) - r(l) - x_l \geq x_m$.

$$\frac{F(\bar{y}_C) - F(\bar{x}_C)}{\prod_{i \in C \setminus \{l,m\}}(r(i) + x_i)} =$$

$$(r(l) + y_l)(r(m) + y_m) - (r(l) + x_l)(r(m) + x_m) =$$

$$(r(l) + x_l + d)r(m) - (r(l) + x_l)(r(m) + x_m) =$$

$$(r(l) + x_l)r(m) + dr(m) - (r(l) + x_l)r(m) - (r(l) + x_l)x_m =$$

$$x_m r(m) - (r(l) + x_l)x_m \geq x_m^2 > 0.$$

Alternatively, if $y_m > 0$ then $d = \frac{r(m)+x_m-r(l)-x_l}{2}$.

$$(r(l) + y_l)(r(m) + y_m) = g^2$$

$$(r(l) + x_l)(r(m) + x_m) = (g - d)(g + d) = g^2 - d^2.$$

Hence $F(\bar{y}_C) > F(\bar{x}_C)$.

Consequently, an optimal distribution $\bar{t}_C$ has the following properties:

- if $t_l > 0$ and $t_m > 0$ then $r(l) + t_l = r(m) + t_m$,
- if $t_m = 0$ and $t_l > 0$ then $r(l) + t_l \leq r(m)$.

The following lemma 3 provides a necessary condition similar to that of previous lemma under additional constraints on the ranges of arguments for quality functions of operations on the path.

Lemma 3. *Let C be a critical path, T the amount of resources for allocation, for each $l \in C$ the quality function is linear, that is $q(l)(t) = u(l) + s(l)t$, where t is within the limits for the quality function. For any optimal resource allocation $\exists C^+ \subseteq C : t_m > 0$ iff $m \in C^+$; and*

$$t_l = \max\left(\frac{T}{n} + \frac{1}{n}\sum_{m \in C^+}\frac{u(m)}{s(m)} - \frac{u(l)}{s(l)}, 0\right),$$

where $l \in C$, and $n = |C^+|$.

Proof. The notations of lemma 2 will be used in this proof.

The first statement in lemma 2 suggests that resources should be allocated to operations l with relatively small $r(l)$, and we define the subset of operations C^+ for which resource will be allocated as follows.

To simplify the notation, assume that operations are ordered in the ascending order of $r(l)$ and let n be the greatest number such that $T \geq nr(n) - \sum_{i=1}^{n} r(i)$. Note that the expression on the right side of the inequality equals to 0 when $n = 1$, hence such n exists for any T.

According to the second statement of lemma 2 resources should be allocated to operations with $r(i) \leq r(n)$ to make the corresponding factors of the objective function equal. The first step is to allocate the amount of resources $r(n) - r(i)$ to each $i \in C^+$. Thus, the total of $\sum_{i=1}^{n}(r(n) - r(i))$ will be distributed. Finally, leftovers $T + \sum_{i=1}^{n} r(i) - nr(n)$ are evenly distributed between these nodes. Formally:

$$t_l = r(n) - r(l) + \frac{T + \sum_{i=1}^{n} r(i) - nr(n)}{n} = \frac{T + \sum_{i=1}^{n} r(i)}{n} - r(l),$$

where $l \in C^+$.

Finally, we have to substitute all $r(i)$ with a ratio $\frac{u(i)}{s(i)}$.

2.6. Resource Allocation between Siblings

The following lemma 4 suggests how to split resource between critical siblings.

The necessary condition for an allocation to be optimal is that amout of resources allocated to each of siblings is proportional to invers of the slope of its quality function.

Lemma 4. *Let C be a critical subtree; $l_i \in C$ are root nodes of sibling subtrees with linear quality functions, that is $Q(\bar{l_i})(t) = U(\bar{l_i}) + S(\bar{l_i})t$, where t is in the range for the quality functions; $T \in \mathfrak{R}$ is the amount of resources for allocation between these siblings. For any optimal resource allocation $T_{\bar{l_i}} = \frac{1/S(\bar{l_i})}{\sum_j 1/S(\bar{l_j})}T$.*

Proof. The quality of the whole query depends on

$$\min_j\{Q(\bar{l_j})(T_{\bar{l_j}})\} = \min_j\{U(\bar{l_j}) + S(\bar{l_j})T_{\bar{l_j}}\},$$

where $\sum_j T_{\bar{l_j}} = T$. To maximize the query quality we have to maximize this component. Therefore, we have to allocate resources such that $\sum_j T_{\bar{l_j}} = T$ and for all i, k

$$U(\bar{l_k}) + S(\bar{l_k})T_{\bar{l_k}} = U(\bar{l_i}) + S(\bar{l_i})T_{\bar{l_i}}.$$

The following equalities show that the quality of critical siblings remains equal after the allocation of the increment

$$U(\bar{l_k}) + S(\bar{l_k})T_{\bar{l_k}} = U(\bar{l_k}) + S(\bar{l_k})\frac{1/S(\bar{l_k})}{\sum_j 1/S(\bar{l_j})}T = U(\bar{l_k}) + \frac{1}{\sum_j 1/S(\bar{l_j})}T =$$

$$U(\bar{l_i}) + \frac{1}{\sum_j 1/S(\bar{l_j})}T = U(\bar{l_i}) + S(\bar{l_i})\frac{1/S(\bar{l_i})}{\sum_j 1/S(\bar{l_j})}T = U(\bar{l_i}) + S(\bar{l_i})T_{\bar{l_i}},$$

and, finally, the constraint on the total allocated amount is satisfied

$$\sum_j T_{\bar{l_j}} = \sum_j \frac{1/S(\bar{l_j})}{\sum_i 1/S(\bar{l_i})} T = \frac{\sum_j 1/S(\bar{l_j})}{\sum_i 1/S(\bar{l_i})} T = T.$$

3. Resource Allocation Algorithm

3.1. The Algorithm and Data Structures

This section describes an approximate algorithm for optimal resource allocation for a fixed query execution plan. To obtain an optimal plan for query evaluation under constrained resources, we start from a plan obtained with conventional optimization techniques, which do not account for the amount of resources, and apply the resource allocation algorithm to this plan. Of course, this plan is not necessarily optimal globally. However, our approach provides the optimal distribution for any fixed tree.

According to the lemmas 3 and 4 in section 2 we are able to optimally distribute resources among operations for two types of the plan structure, namely, for plans consisting of a single path from root to leaf (vertical path) or between leaf siblings of a single parent node. As the lemmas are valid only within linear ranges of the quality functions, several distribution steps might be needed.

In order to exploit these lemmas for a general tree, we build virtual hypernodes from certain parts of the query execution plan and then apply an appropriate lemma inside a hypernode.

A node is called a splitting point if it has multiple child nodes. Virtual hypernodes are constructed for the critical tree according to the following rules:

- If the critical plan contains sibling leaf (hyper-)nodes, these siblings are retracted into a *horizontal hypernode*, which becomes a single child of the parent node.
- Any path from a leaf to a splitting point is retracted into a *vertical hypernode*, which becomes a leaf child of the splitting point.

Obviously, at least one of two steps listed above is applicable if a plan contains at least two nodes. The process stops when the whole plan is retracted into a single hypernode which is always vertical as a root cannot have siblings.

An example of a hypernode structure is shown in figure 3.

The quality functions for original tree nodes are provided from cost models for corresponding operations. Quality functions for hypernodes are constructed when the hypernodes are created.

The lemmas 3 and 4 require that the quality of each operation (node) is a linear function of the allocated resource. Unfortunately, the quality of a hypernode cannot be always expressed as a linear function and it is replaced with a linear approximation, making our algorithm approximate (except for few special cases). The quality functions for horizontal (sibling) hypernodes are linear by lemma 4. Functions for vertical nodes are not linear in general, and are replaced with tangent approximations, valid on a certain range of the argument values.

The range for a hypernode is constructed in such a way that the corresponding amount of resources can always be distributed to the sub-nodes of this hypernode without violation of the constraints inside the hypernode.

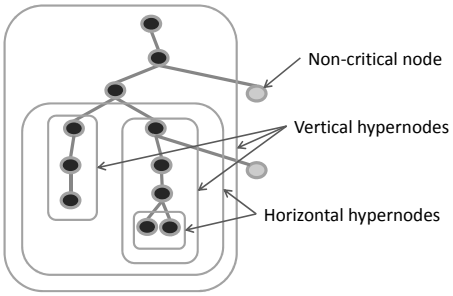


Fig. 3. An example of hypernodes

The overall resource allocation control flow is displayed in the following pseudo code:

Input: Query tree P, resources for allocation T.
Output: For each operation $l \in P$, an allocated amount of resources t_l.
Initialization(P, T)
while $T > 0$ & !isMaxQualityReached(P) **do**
QualityEstimation(P)
C=CriticalSubtreeConstruction(P)
H=HypergraphConstruction(C)
ResourceAllocation(H, T)
end while

The algorithm can now be outlined as follows.

During the initialization using the recursive tree descent the amount of resources equal to the minimal needed is allocated to each operation in P and the quality functions are adjusted to accept increments. If the sum of minimal resources exceeds T, the algorithm fails, as the execution of the plan is not possible with available resources.

Those nodes which received maximum resources needed for operation processing are marked as saturated and are not considered in the further resource distribution process, however are taken into account during the hypernode quality functions construction.

After initial allocation of the minimally required resources the remaining amount of resources is distributed in incremental amounts. The increments are subject to the following constraints:

- boundaries of linear segments in piecewise linear operation quality functions are not exceeded for any node, and
- the quality of a critical sub-query should not exceed the quality of its non-critical siblings.

The incremental allocation is repeated until either all available resources are exhausted or the maximal possible quality for the query is reached.

At the quality estimation step we apply the recursive tree descent to recalculate the achieved quality for all subtrees in the query evaluation plan.

For each incremental step, the critical sub-tree and hypernodes are re-calculated and then allocation proceeds recursively into all hypernodes, using the appropriate lemma for allocation inside each hypernode.

The pseudo code presented below demostrates the hypernodes construction step.

Input: Operation node $r \in P$.
Output: Hypernode $h(r)$.

```
tmp=r
loop
    if |children(tmp)| > 1 then
        for all child ∈ children(tmp) do
            child=Hypernode(child)
        end for
        children(tmp)= HorisontalHypernode(children(tmp))
    else if |children(tmp)| = 0 then
        break
    else
        tmp=children(tmp)[0]
    end if
end loop
r=VerticalHypernode(r)
```

3.2. Horizontal Hypernodes

According to lemma 4, the amount of resources allocated to the horizontal hypernode is distributed between the siblings so that the increase of the quality function is same for all siblings in the hypernode. Let H be a hypernode with operations l_i, and s_{l_i} be the slopes for the quality functions, that is for each l_i the quality function is $q(l_i)(t) = u(l_i) + s(l_i)t$. Let

$$S = \sum_i 1/s_{l_i}$$

for brevity. If the total amount of resources allocated to H is T, then $t_{l_i} = \frac{T}{s_{l_i} S}$.

The construction of the quality function for a horizontal hypernode is straightforward: the initial quality is the same as that of siblings, and the slope is calculated from the slopes of siblings and is equal to $\frac{1}{S}$. Let $t^{l_i}_{max}$ be the right boundaries for the linear segments of the quality functions of the siblings, then the right boundary for the linear segment of the hypernode is $T_{max} = \min\{t^{l_i}_{max} S s_{l_i}\}$.

3.3. Vertical Hypernodes

Handling of vertical hypernodes is more complex.

Let V be a vertical hypernode with nodes l_i as its elements, and let $r(l_i) = u(l_i)/s(l_i)$, where u and s are defined as in lemma 3. Also let $S = \sum_{l \in V} r(l)$. According to this lemma, an optimal allocation of resources T is reached when

$$t_{l_i} = \begin{cases} (T+S)/n - r(l_i), & \text{if } (T+S)/n - r(l_i) > 0 \\ 0 & \text{otherwise} \end{cases}$$

where S and n are the sum and count, respectively, of those $r(l_i)$ for which the

$$(T + S)/n - r(l_i) > 0.$$

Obviously, this expression will be positive for minimal values of $r(l_i)$, as S/n is the arithmetic mean of $r(l_i)$.

To construct the quality function for a hypernode the algorithm selects the subset of sub-nodes with the minimal value of $r(p)$. The resource will be allocated to these nodes only. The allocated amounts cannot exceed the ranges for the quality functions of respective nodes. In addition, the increased value of the quality functions should not exceed that of the node with the second minimal $r(l_i)$, and the quality of non-critical siblings also should not be exceeded.

The exact quality function for V is a product $\prod_{l \in V}(u(l) + s(l)t_l)$. This function is replaced with a linear tangent approximation. The range is calculated from the constraints. As soon as the amount of resources allocated to a hypernode is calculated, it is divided between the selected sub-nodes evenly and this completes the incremental step for a hypernode.

Our algorithm is approximate because linear approximations are used instead of precise quality functions. However, the smaller are time increments allocated at each iteration the smaller error we obtain and more iterations are spent for resource allocation. This fact enables us to balance the distribution cost with the accuracy of the output.

3.4. The Complexity

We provide a very rough estimation for the complexity of the proposed algorithm, leaving more precise estimations for a future work.

The complexity of the algorithm described above depends on the number N of operations in the query and the number of linear segments of the quality functions for each node. In addition to operation nodes, the algorithm operates with hypernodes. As each hypernode includes at least one edge from the original tree, the number of hypernodes cannot exceed $N - 1$, and the total number of nodes and hypernodes can be estimated as $2N$.

During each incremental iteration a number of tree traversals is needed. The worst case complexity of the tree traverse is estimated as a number of nodes, that is, $2N$. This estimation applies to calculation of sub-query quality values, adjustment of quality functions, and allocation of increments. This also applies to the initial allocation of minimal resources, which is executed before the main loop and hence can be neglected.

Further, the calculation of constraints related to non-critical siblings requires a traverse of sub-queries for each node in a vertical hypernode. The worst case complexity can be estimated as the number of subqueries $N - 1$ multiplied by the number of all nodes $2N$. Consequently, the overall complexity of one iteration can be estimated as $3(2N) + (2N)(N - 1) = O(N^2)$.

Let M be the maximum of numbers of linear segments for quality functions. The rough worst case estimation for the number of linear segments is MN, additional break points are coming from constraints on non-critical siblings and the cardinality of vertical hypernodes, both having upper bound of N.

Each iteration increment cannot span multiple linear segments or break points. Hence the upper bound for the number of iterations is MN^3, and finally the overall complexity of the algorithm is not worse than $O(MN^5)$.

Actually some of the values estimated above can never reach the upper bound at the same time. Consider, for example, the length of a vertical path. If it is equal to N, then all nodes of the tree are included into this path and hence there are no siblings at all, neither critical nor non-critical, reducing corresponding multiplier to 1. Similarly, if the critical sub-tree includes all N nodes, there are no non-critical siblings and hence no need to check these constraints.

Thus, the proposed algorithm is polynomial and can provide acceptable performance for queries containing tens of operations.

4. Experiments

To analyze the behavior of the proposed approximate resource allocation algorithm we have performed a set of experiments. All experiments are based on Java implementation, running on i5 2.67 GHz CPU, 8 Gb RAM, 64 bit operating system.

We evaluated both the efficiency and the effectiveness of the developed technique. The performance is measured in terms of the amount of time needed for resource allocation and the number of iterations in the main loop of the algorithm.

To analyze the accuracy of the algorithm, a value of the optimal relative quality for a plan is needed. As an optimal resource allocation algorithm is not available, we use an approximation obtained as described in the following paragraph.

The error of our algorithm is caused by use of linear approximations instead of precise quality functions. Obviously, this error becomes negligible for sufficiently small increments. To obtain an estimation of the optimal resource allocation, the algorithm was executed with additional restriction on the size of the increment at each iteration. The results were used as a substitute for the optimal solution.

It is important to emphasize that the algorithm under investigation deals with estimations obtained from the cost and quality models, rather than actual executions of the operations or properties of data sets. The characteristics of the algorithm to be measured depend on the number of operations in the query, the topology of the query plan, and the number of linear segments of the quality functions. To ensure a comprehensive analysis, all experiments were run on synthetic query plans and quality functions.Use of any real data would not add any value. Of course, real data sets would be a must for evaluation of cost or quality models, however, such evaluation is not in the scope of this paper.

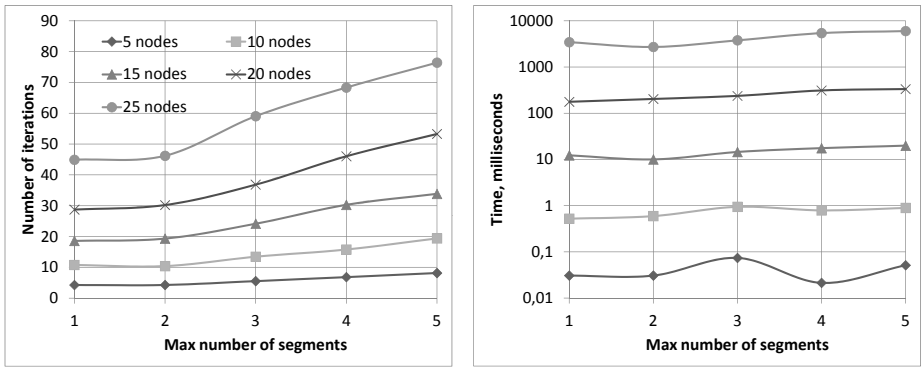
A number of random query trees were generated with N nodes, where

$$N \in \{5, 10, 15, 20, 25\}.$$

Quality functions for each operation in the query were generated with the number of linear segments varying from 1 to l, with $l \in \{1, 2, 3, 4, 5\}$. Minimum and maximum amounts of resources for each operation were also chosen randomly from the range [0, 100]. The topology of the query tree was defined randomly and then the number of splits was calculated in order to estimate how this parameter affects the approximate algorithm.

The most interesting findings from our experiments are described below.

Figure 4(a) shows how the number of iterations needed to allocate all available resources depends on the query tree: the number of operations and the number of linear segments in the operation quality functions. For each pair (N, l) 300 trees were constructed for the experiment. For each tree we allocated by the approximate algorithm resource $T = T_{min} + p(T_{max} - T_{min})$, where $p \in \{0, 0.25, 0.5, 0.75, 1\}$. Similar results with absolute time measurements are presented in figure 4(b). One may see that the average number of iterations and the average absolute time needed for resource allocation goes up with the increase of the number of operations in the query and the number of linear segments in each node.



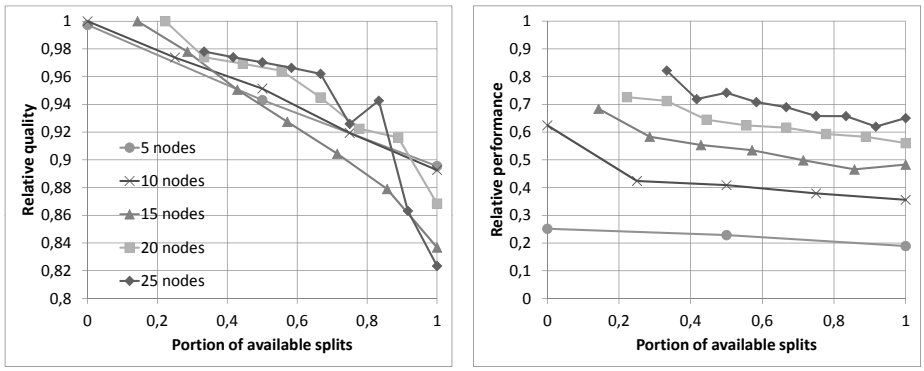
(a) (b)

Fig. 4. Average number of iterations and absolute time for various queries

The results of experiments on accuracy of the proposed approximate resource allocation algorithm are presented in figure 5. For each pair (N, l) 100 trees were randomly constructed for the experiment, and for each tree 10 quality functions were generated in order to collect enough number of examples for trees with rare topologies. We allocated $T = T_{min} + 0.5 * (T_{max} - T_{min})$ resource. Figure 5(a) shows on the Y-axis the relative quality produced by the proposed approximate algorithm compared to the one with artificially restricted resource increments allocated on each iteration to be equal 5% of the maximum amount of resources needed for the query. The X-axis shows the "bushiness" of trees, which is calculated as a ratio between the number of splits in the randomly generated tree and the maximum number of splits in binary tree with N nodes. One may see the relative quality decreases when the number of splits grows, that is, a tree becomes bushy. In this case trees contain enough paths to use tangent approximation of the sub-query quality functions and enough splits to use them in the resource allocation process.

Figure 5(b) shows the ratio of the number of iterations used in the approximate algorithm and in the nearly exact one. The number of iterations in the approximate algorithm decreases as the portion of available splits grows. The reason is that horizontal hypernodes accept larger increments of resources compared to the vertical ones.

The results of the experiments have shown that the approximate resource allocation algorithm is quite precise and efficient.



(a) (b)

Fig. 5. Cost-quality trade-off

5. Related Work

Query optimization became both required and enabled since the advent of high-level declarative query languages, mostly in the context of the relational database model. A brief overview of classical query optimization techniques can be found in [10]. The optimization techniques for distributed systems are summarized in [9]. An optimizer for distributed heterogeneous systems is proposed in [13]. The query optimization techniques based on hypergraphs are analyzed in [14].

The algorithm proposed in this research uses the output of a traditional optimizer as its starting point.

The approximate query evaluation techniques were considered in the context of very large data warehouses and mobile networks [1, 4, 3, 7]. The approximation is typically based on sampling, wavelets, or synopsis. However, in both cases the query optimization was not extensively studied. For data warehouses, the dominating cost is produced by accesses to a single huge table, hence the performance depends mostly on the performance of a single operation, namely, data extraction from this table. For mobile networks, the queries are typically very simple and the optimization is not an issue. Note that the critical resource might depend on the context: in contrast with large databases and data warehouses, where time is the most important, the energy might be more valuable for mobile or sensor devices. Our approach does not depend on the nature of the resources to be allocated and might be applicable in both contexts.

Handling of time constraints on complex SQL queries is proposed in [6]. The authors distinguish an approximate (based on sampling) and partial (top k) query evaluation. The proposed approach is based on extendibility of an optimizer in the commercial DBMS and does not consider the resource allocation problem.

The quality and performance trade-offs for stream processing are discussed in [19, 8].

Approximate query evaluation of the web is discussed in [15].

Optimal resource allocation was studied in the context of distributed query evaluation and load balancing [20, 16, 5]. Although our algorithm resembles load balancing, it is not

specifically related to a distributed system and allocates resource to operations rather than to processing units.

The approximate query evaluation is a must in contexts such as information retrieval, however, in these contexts the primary objective is the quality of result (usually measured as precision and recall), while resources are considered less important. Further, complex queries and hence query optimization are not common in these contexts. In contrast, our work is focused on complex queries with mixed querying paradigms and efficiency of query evaluation.

The extensive research on data and information quality is summarized in [11]. Although the nature of data quality is essential, in our research we rely on a quantitative estimation of quality expressed as a single number and abstract from any specific aspects of data quality. Further, our goal is to estimate relative quality, that is, how the operations and the whole query affects and depends on the quality of data sources and final output, rather than the absolute quality.

A lot of research is done in the area of query optimization and approximate query evaluation. However, as far as we know, the problem of optimal resource allocation for approximate query evaluation was not considered.

6. Conclusion

High-level declarative querying facilities both require and enable sophisticated query optimization. In the context of data spaces and/or heterogeneous information resources an approximate query evaluation turns out to be a dominating paradigm.

To address the need in controlled trade-offs between quality and performance, optimization techniques should be augmented with additional capabilities.

We presented an approximate polynomial algorithm for optimal resource allocation suitable for real-time systems where predictability of response time is critical. The resource allocation algorithm provides best possible quality of the result subject to the constrained total amount of resources for a query.

The experimental evaluation shows that the algorithm provides good approximation, especially on queries with small number of operations, which are likely to dominate in real applications.

There are still some unresolved issues concerning the resource allocation problem in the context of approximate query evaluation and optimization. First of all, the selected exact query evaluation plan may not support approximate execution and in this case the subsequent application of the resource allocation technique is not applicable. In future work we are planning to consider the approach which solves the query optimization and resource allocation problems simultaneously. Another challenge is adaptive resource allocation capable to dynamically re-allocate resources during the query evaluation.

References

1. Babcock, B., Chaudhuri, S., Das, G.: Dynamic sample selection for approximate query processing. In: Proceedings of the 2003 ACM SIGMOD international conference on Management of data. pp. 539–550. SIGMOD '03, ACM, New York, NY, USA (2003), http://doi.acm.org/10.1145/872757.872822

2. Batini, C., Cappiello, C., Francalanci, C., Maurino, A.: Methodologies for data quality assessment and improvement. ACM Comput. Surv. 41(3), 16:1–16:52 (Jul 2009), http://doi.acm.org/10.1145/1541880.1541883
3. Chaudhuri, S., Das, G., Narasayya, V.: Optimized stratified sampling for approximate query processing. ACM Trans. Database Syst. 32 (June 2007), http://doi.acm.org/10.1145/1242524.1242526
4. Dell'Aquila, C., Di Tria, F., Lefons, E., Tangorra, F.: Accuracy estimation in approximate query processing. In: Proceedings of the 14th WSEAS international conference on Computers: part of the 14th WSEAS CSCC multiconference - Volume II. pp. 452–458. ICCOMP'10, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA (2010), http://dl.acm.org/citation.cfm?id=1984366.1984374
5. Epimakhov, I., Hameurlain, A., Dillon, T., Morvan, F.: Resource scheduling methods for query optimization in data grid systems. In: Proceedings of the 15th international conference on Advances in databases and information systems. pp. 185–199. ADBIS'11, Springer-Verlag, Berlin, Heidelberg (2011), http://dl.acm.org/citation.cfm?id=2041746.2041765
6. Hu, Y., Sundara, S., Srinivasan, J.: Supporting time-constrained sql queries in oracle. In: Proceedings of the 33rd international conference on Very large data bases. pp. 1207–1218. VLDB '07, VLDB Endowment (2007), http://dl.acm.org/citation.cfm?id=1325851.1325989
7. Jermaine, C., Arumugam, S., Pol, A., Dobra, A.: Scalable approximate query processing with the dbo engine. ACM Trans. Database Syst. 33, 23:1–23:54 (December 2008), http://doi.acm.org/10.1145/1412331.1412335
8. Jiang, Q.: A framework for supporting quality of service requirements in a data stream management system. Ph.D. thesis, Arlington, TX, USA (2005), aAI3181900
9. Kossmann, D.: The state of the art in distributed query processing. ACM Comput. Surv. 32(4), 422–469 (Dec 2000), http://doi.acm.org/10.1145/371578.371598
10. Kossmann, D., Stocker, K.: Iterative dynamic programming: a new class of query optimization algorithms. ACM Trans. Database Syst. 25(1), 43–82 (Mar 2000), http://doi.acm.org/10.1145/352958.352982
11. Madnick, S.E., Wang, R.Y., Lee, Y.W., Zhu, H.: Overview and framework for data and information quality research. J. Data and Information Quality 1(1), 2:1–2:22 (Jun 2009), http://doi.acm.org/10.1145/1515693.1516680
12. Novikov, B., Vassilieva, N., Yarygina, A.: Querying big data. In: Proceedings of the 13th International Conference on Computer Systems and Technologies. pp. 1–10. CompSysTech '12, ACM, New York, NY, USA (2012), http://doi.acm.org/10.1145/2023607.2023609
13. Pentaris, F., Ioannidis, Y.: Query optimization in distributed networks of autonomous database systems. ACM Trans. Database Syst. 31(2), 537–583 (Jun 2006), http://doi.acm.org/10.1145/1138394.1138397
14. Scarcello, F., Greco, G., Leone, N.: Weighted hypertree decompositions and optimal query plans. J. Comput. Syst. Sci. 73(3), 475–506 (May 2007), http://dx.doi.org/10.1016/j.jcss.2006.10.010
15. Tran, T., Ladwig, G., Wagner, A.: Approximate and incremental processing of complex queries against the web of data. In: Proceedings of the 22nd international conference on Database and expert systems applications - Volume Part II. pp. 171–187. DEXA'11, Springer-Verlag, Berlin, Heidelberg (2011), http://dl.acm.org/citation.cfm?id=2033546.2033567
16. Yang, R., Bhulai, S., van der Mei, R., Seinstra, F.: Optimal resource allocation for time-reservation systems. Perform. Eval. 68, 414–428 (May 2011), http://dx.doi.org/10.1016/j.peva.2011.01.003
17. Yarygina, A., Novikov, B.: Optimizing the resource allocation for approximate query processing. In: ADBIS 2012 (to appear). pp. 1–12. Advances, Springer-Verlag (2012)
18. Yarygina, A., Novikov, B., Dolmatova, O.: Cost models for approximate query evaluation. In: Baltic DBIS 2012 (to appear). pp. 1–12. Short communications, Lithuanian Computer Society (2012)

19. Zhang, R., Koudas, N., Ooi, B.C., Srivastava, D., Zhou, P.: Streaming multiple aggregations using phantoms. The VLDB Journal 19, 557–583 (August 2010), http://dx.doi.org/10.1007/s00778-010-0180-z
20. Zhao, H.C., Xia, C.H., Liu, Z., Towsley, D.: A unified modeling framework for distributed resource allocation of general fork and join processing networks. In: Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems. pp. 299–310. SIGMETRICS '10, ACM, New York, NY, USA (2010), http://doi.acm.org/10.1145/1811039.1811073

Anna Yarygina is a PhD student at the Department of Analytical Information Systems at Saint-Petersburg University, Russia. She obtained her M.S. in Computer Science in June 2011 from Saint-Petersburg University, Russia. Her research interests include advanced information management and retrieval, fuzzy query optimization and processing, approximate query evaluation.

Boris Novikov is a professor and chair, Analytical Information systems at St.Petersburg University. Boris received his PhD in Comp.Sci.in 1981, and Dr. Sci. degree in 1994 from St. Petersburg University. His research interests include design of database systems, transactions, indexing and data structures, query processing and optimization, performance tuning, and management of distributed heterogeneous information resources.

Received: September 26, 2012; Accepted: August 4, 2013.