An Architecture for Approximate Real-Time Query Optimization and Processing *

Anna Yarygina Boris Novikov

Saint-Petersburg University
anya_safonova@mail.ru, b.novikov@spbu.ru

Abstract

High-level declarative scripting languages are considered to be an effective tool for specification and execution of complex analytical data processing scenarios as they can hide the complexity of underlying heterogeneous processing environment. A growing demand to process huge amounts of data under real-time requirements as well as advanced similarity-based models raises the need in approximate processing.

We discuss architecture of an extendable system for optimization and execution of approximate complex queries formulated in similarity-based declarative query language.

1 Introduction

Complex querying scenarios appear in various data integration and processing tasks. Several different information resources, such as databases, semi-structured data, streams, and uncertain data might be needed in the same data processing workflow. The systems might be heterogeneous in terms of data model, dynamics, trustfulness, and content type, as well as querying or retrieval paradigms. The need to combine data extracted from heterogeneous resources potentially based on diverse querying paradigms appears in several application contexts and application areas, including advanced search, personalization, and analytical processing.

Specifications of complex analytical scenarios can be effectively expressed in high-level declarative scripting languages hiding the complexity of underlying processing environment.

The systems integrating these ideas usually exploit intermediate algebraic languages which in addition to common features of query languages, such as filtering, set-theoretic operations, and joins incorporate their fuzzy extensions and complex processing techniques such as NLP, mining, and analytics.

1.1 Problem Statement

Any complex data workflow processing based on similarity or other uncertain querying model tends to be computationally heavy. The expectation is that a declarative algebraic approach to query evaluation over diverse set of information resources enabling query optimization is capable to address this performance issue.

Query optimization techniques for such context differ from relational due to significant differences in algebraic properties and cost models for extended similarity-based operations.

The uncertain nature of the queries implies a need in approximate query evaluation because in such context exact query evaluation sometimes is not reasonable or meaningful.

The approximate algorithms implementing operations of the algebra in combination with traditional exact query evaluation can be applied to reduce the query evaluation time. Some algorithms may produce different quality of the output depending on the amount of resources allocated for execution. This leads to the analysis of speed/quality trade-offs in the context of query optimization and evaluation.

The problem of traditional query optimization, namely "find an execution plan with the minimal cost" is replaced with either "find the best plan yielding at least specified quality", or "provide the best possible quality for at most given amount of resources".

We have to maximize the quality having the limited resources for query evaluation. Thus, compared to traditional query optimization the system incorporates the resource allocation step in the whole query evaluation pipeline.

Our primary objective is to build an architecture which is able to process complex data processing workflows in real-time, that is ensure predictable response time controlling approximate evaluation.

1.2 Querying Examples

The example from model implementation is based on data from OpinRank dataset with judgments. The dataset contains hotels from 10 cities over the world. For each hotel its name, address, and scores on cleanliness, room, service, etc. are available.

Let us consider a query: find best 10 hotels in London according to service and cleanliness. The query is formulated in terms of high-level declarative query language similar to one proposed in [8] and then is translated into expression in terms of algebraic operations (query evaluation plan) shown in figure 1(a).

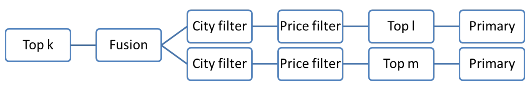
In this work we further do not distinguish operation

* This work was partially supported by HP Labs under contract No. CW317234.

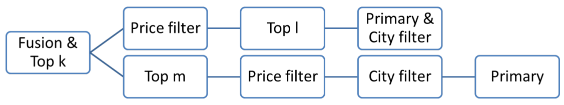
Proceedings of the Ninth Spring Researcher's Colloquium on Database and Information Systems, Kazan, Russia, 2013

and algorithm its implementing because most modules in the architecture process algorithms rather than logic operations.

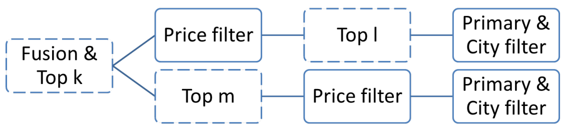
The primary operations retrieve hotels with scores from primary sources based on provided score types. Subsequent top operations are applied to reduce the number of objects for further analysis. The next price filter retrieves the price for double room for specific date. The city filter keeps only hotels in London. The fusion operation combines scores for hotels based on provided rule. Finally, top operation is applied to retrieve only 10 best hotels according to new input scores.



(a) Example query evaluation plan



(b) Alternative query evaluation plan



(c) Approximate query evaluation plan

Figure 1: Example scenario

One may see that there are several opportunities for optimization. For example, we can swap unary operations. The system also supports implementation of fusion operation which simultaneously retrieves best objects and integrated implementation of primary operation and city filter.

Figure 1(b) demonstrates how the query evaluation plan can be transformed (not necessarily making it optimal).

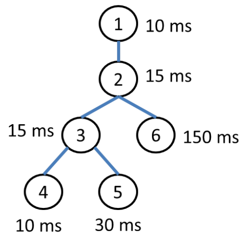
The quality of the query evaluation result depends on the amount of time specified by the user.

Thus, as soon as the query optimization step produces a reasonably good query evaluation plan the system has to distribute fixed amount of resources specified by the user. Figure 1(c) shows some query evaluation plan with operations admitting approximate implementation marked by dashed borders. Approximate algorithms for them and corresponding cost models are described in details in [13].

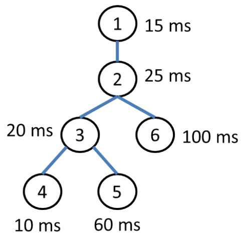
1.3 Analytics Example

Let us suppose that we analyze social sources (such as ratings from retailer web site and customer tweets) to find out how customer satisfaction depends on price, for each product group. Thus, we probably formulate the following query: count average values of ratings from retailer and from sentiments, grouped by price range, for each product group. The corresponding query evaluation plan is shown in figure 2 with the following description of nodes:

```
1: group by PriceRange
                 nest (ProdGroup, avg(Rating),avg(Sent))
2;   group join on ProductModel
3:       group join on ProductModel
4:           get database product table
5:           get retailer ratings for products
6:       get sentiments from tweets for products
```



(a) Possible resource allocation



(b) Another resource allocation

Figure 2: Analytics example

Let us as well suppose that external primary sources produce results as output streams of objects with different speed: 500 objects per second for retailer ratings and 300 objects per second for sentiments extracted from tweets. As the data sources are actually (potentially unlimited) streams, an exact evaluation is impossible. We assume that an approximate result is expected in at most 230 ms. This implies an ultimate need in approximate evaluation which restricts the quantity of data items processed by each of operations.

The limited amount of resources can be distributed among operations in the plan in different ways shown in figures 2(a), 2(b). One may see that the resource allocation strategy shown in figure 2(b) is preferable because we do not retrieve excessive tweets when have not enough retailer ratings.

1.4 Contributions

We propose an architecture for an extendable system for optimization and execution of approximate complex queries formulated in similarity-based declarative query language. Based on the proposed architecture we demonstrate the integration of techniques proposed in previous research and connect together several parts: similarity-based algebra, extended cost/quality models, and resource allocation techniques. We provide the detailed discussion of several aspects of the architecture focusing on its extensibility.

The rest of the paper is structured as follows. Section 2 describes the architecture including underlying data model, intermediate algebraic query language, and main modules. The architecture details on specific modules are followed by a detailed discussion of classes of operations from the system extensibility point of view in section 3. Section 4 outlines the related work.

2 The Architecture

This section outlines high-level architecture of a middleware query processing system acting on top of existing query evaluation and data processing systems, and below some user interface for query formulation and results exploration.

The system supports different types of data sources and applies the uniform representation of different types of data to enable their integration in one query. This results in a compromise between uniform integration of different types of sources and processing units and necessity to control the query correctness and validity at the query formulation step.

2.1 The Language and Data Model

The central concept of our model is a q-set defined in [23] as a tripe (q,B,S) where

- q is a query (no matter how represented),
- B is a base set of objects,
- S is a scoring function for objects in B.

Actually q-sets are represented as sets of objects with scores indicating relevance of object to the underlying query.

The algebra on q-sets is characterized in [23] and includes extended relational operations taking q-sets as input arguments and return q-sets as results as well. Operations can be configured with different parameters, for example *top k* operation has parameter representing the number of objects to be returned; *selection filter* may have parameter with selection predicate; and *primary* operation extracting q-set from database is configured with a SQL query.

2.2 Query Optimization and Execution

Query processing engine contains modules shown in figure 3: parser, optimizer which interacts with transformations and cost models, resource allocation module which is based on quality models, and executor.

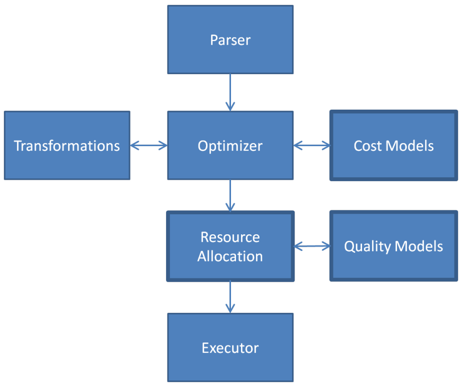


Figure 3: Model Implementation architecture

2.2.1 Parsing

The parser translates a query into the initial query evaluation plan that is the query tree with algebraic operations.

The initial query is supposed to be formulated in some high-level language: extension of SQL similar to one proposed in [8] or some graphic one. In this work we focus on intermediate level and essential part of query evaluation is processed on the algebraic level. In our model implementation the initial query is represented by XML tree with nodes and attributes describing operations and parameters.

2.2.2 Optimization

The optimizer in query evaluation system in the described environment should be adaptive and distensible.

Similar to the traditional optimization of relational queries the optimizer has to analyze the algebraic properties of underlying operations. The associativity and distributive properties enable the rewriter to form the space of query plans based on the transformations of the query tree.

We have to support algebra extensibility and develop optimizer tunable to new specific operations which may possess new algebraic equivalences and expand the space of possible query plans.

The optimizer is based on extensible and tunable set of possible transformations of the query representing the space of query evaluation plans. It is important to note that in the proposed architecture the set of transformations includes all possible reorganizations of the query evaluation (sub) plan: based on algebraic equivalencies or different implementations of algebraic operations. This fact enables an easy way to introduce new operations, algorithms, and corresponding transformations.

In comparison with traditional relational data bases the space of transformations is not as homogeneous as in extended algebra: for example, non-associative joins and complex transformations with *top k* operation are there.

A special attention on the implementation of transformations in the system is needed because of its extensibility: the space of transformations should not be deeply coded in the optimizer, thus uniform interfaces are needed to make transformations easily pluggable in.

Any high-quality optimizer should be cost-based, hence cost models for all operations should be provided, including user-defined extensions. In order to be able to select a good query evaluation plan the optimizer for complex query processing should be constructed based on cost models of operations in the queries. The architecture should support an integration of new cost models for operations and tuning of existing ones. The cost models are supposed to be tunable and adaptive, that is, be able to use data statistics when available.

In the model implementation an optimizer is based on limited descent by plan cost.

Further the adaptive query optimization technique [12] should be integrated into the system, as it compensate for the lack of statistics inherent for heterogeneous environment, especially with user-defined operations.

2.2.3 Resource Allocation

The architecture supports the class of approximate algorithms with the following properties:

- with controllable quality (allocated resources define the quality of result);
- with fixable parameters (the fixed amount of allocated resources is transferred into fixed operation call parameters).

The restricted class of supported approximate algorithms contains quite large number of specific algorithms: for example, an approximate algorithm for aggregation operation proposed in [29], approximate join algorithms discussed in [6], or any algorithm based on sampling.

Approximate algorithms with fixable parameters should be distinguished from any-time techniques when user is able to stop operation evaluation based on estimations of already achieved result quality, for example proposed in [4, 14].

Approximate pre-configurable algorithms are more restrictive as the estimation of result quality should be done in advance. On the other hand, this property is less restrictive as there is no need to support correct approximate result of operation evaluation at any time of execution.

As the focus of this work is on the approximate query evaluation the cost models are extended with the notion of quality and form the quality models for algebraic operations. Few examples of such extended cost models are presented in [13]. Quality models describe the speed/quality trade-off for approximate algorithms that is, the models show how the result quality depends on the amount of resources allocated for operation processing. In model implementation only simple non-adaptable cost and quality models are implemented.

Although the semantics of data quality is complex [21] and may be considered in several dimensions, we assume that the quality of a data set is estimated with a single numeric value. 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. In other case we can assume that the result quality of approximate aggregation operation based on sampling can be estimated as relative sample size.

It is important to note that cost/quality models are based on relative operation quality. Let us consider unary operation which receives input data with absolute quality a_{in}. Even the best possible implementation of operation may reduce the quality of the result data a_{out}. Thus, the quality of operation equals $\frac{a_{out}}{a_{in}}$. If the operation receives the unlimited resources for the fixed input data it produces data with the best feasible absolute quality $a_{out}(\infty)$. If it 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 is $\frac{a_{out}(t)}{a_{out}(\infty)}$.

Another concept essential for our extended cost/quality models is a resource. Different types of resources may be used to restrict the query execution, some of them, like CPU time, CPU cycles, and I/O, depend mostly on the complexity of the plan, while others like memory size or elapsed time may also depend on available configuration (e.g. number of processors allocated for the query).To treat all kinds of resources uniformly, we consider configuration to be a part of the plan.

In this work the amount of resources is expressed with single numeric value. This value may represent either the most important type of resources, e.g. elapsed time for real-time requirements, or a combination of different resource types.

In order to construct the query evaluation plan which operates in the limited amount of time and provides the best possible quality the resource allocation module should interact with the optimizer. The optimization problem for approximate query evaluation seems to be much harder than commonly known exact query optimization: in addition to selection of one of equivalent execution plans, the optimizer has to distribute available processing resources between operations of the query execution plan.

Model implementation uses an approximate solution described and analyzed in [32]: the limited amount of resources is allocated to the best execution plan for unlimited resource yielding the best possible quality. The rationale is separation of resource allocator from an optimizer, enabling the use of any of well-known optimization techniques.

Obviously, the above solution may result in suboptimal plan. For example, approximate evaluation of operations may limit the cardinality of intermediate results making the selected query evaluation plan inefficient. Another limitation is in the inability to support approximate algorithms implementing operations without controllable quality but still relevant for approximate real-time query evaluation. An integrated implementation of an optimizer and resource allocator is planned for future work.

2.2.4 Execution

The executor is based on commonly accepted pipeline query evaluation augmented with provisions for extendibility. Thus intermediate results of evaluation are transferred between operations without materialization; however, some implementations of specific operation still may materialize intermediate results internally.

The set of operations can be extended by other generic or user operations; new exact and approximate algorithms implementing operations also can be smoothly introduced to the system together with new algebraic equivalences and cost/quality models.

3 Detailed Discussion

The main requirements to the system are extensibility and approximation, thus we should take them into account in the system architecture.

3.1 Transformations

A notable feature of the environment in consideration is the need to provide for configurable transformations, in contrast with usually hard-coded transformations for relational optimizers.

Transformations are described in the system by 2 functions and are registered in the list of available transformations.

- *Check function* which evaluates the applicability of transformation to the root of the query evaluation (sub) plan.

- *Apply function* which constructs a new query evaluation (sub) plan from input one applying the transformation.

The system has base implementation of some universal query transformations such as algorithm change transformation and distributive property. This enables easy implementation of additional transformations just configuring base ones.

3.2 Cost Models

As any effective query optimization is based on cost models of operations the system should support user-defined cost models for operations to be able to improve applied cost models for basic operations and add new ones for extensions.

A cost model is a function with the following interface:

Input: Operation call parameters; input data statistics.

Output: Operation cost; output data statistics.

3.3 Quality Models

At the resource allocation step the system applies quality models which describe the trade-off between cost of operation execution and quality of result for approximate algorithms.

A quality model is a function with the following parameters:

Input: Input data statistics; non-changeable operation call parameters.

Output: Piecewise-linear representation of dependence between resources allocated for operation execution and result quality.

In addition, a function mapping allocated resources and quality into operation parameters is needed for correct work of the system. This function returns operation call parameters based on specified operation execution conditions (resources, quality or both):

Input: Input data statistics; non-changeable operation call parameters; resources allocated for operation execution; expected operation result quality.

Output: Operation call parameters.

3.4 Operations

Discussion of operation library is followed by architecture details for specific classes of operations: primary, unary, and binary focusing on algebra extensibility.

3.4.1 Operation Library

Operation library stores the set of algebraic operations and all attendant structures. For each operation the library stores the following:

- Mapping of operation call in the query evaluation plan into the direct call;
- Cost model;
- Quality model;
- Relevant transformations using the operation.

Each algorithm, for example join based on nested loops or sort merge, is registered in the operation library as a separate operation. To add new operation to the operation library one should register the corresponding cost model, quality model, and transformations.

From the architecture point of view the set of algebraic operations can be divided into: primary operations, unary operations, binary operations.

During the algebra extension operations from all these groups can be implemented supporting predefined interface discussed below or constructed based on universal basic implementations for each class.

Parameters of data processors and functions used to configure basic implementations are passed from operation parameters: map of parameter names into values. In this case the interface of basic implementation is universal to some extent and different data processors can work with different sets of specific parameters.

All operations support the following interface:

Input: Operation call parameters, pipelined argument(s)

Output: Pipelined output

The number of arguments depends, of course, on the arity of the operation.

3.4.2 Primary Operations

Primary sources of data are implemented as operations without input arguments.

For example, if the primary source is a database, it is implemented as an operation with input parameters representing SQL query to retrieve data and data needed to setup the connection.

A built-in basic implementation of primary operation is provided. It can be configured by data retrieval processor to construct different real primary operations.

Data retrieval processor returns an object from data set for each call until all data is returned. Basic implementation of primary operation calls data retrieval processor while the latter returns objects and transfer collected objects to the output pipe. To extend the algebra by new primary operation one may implement a data retrieval processor and register it in the system. After that a new primary operation may be implemented as basic primary operation configured by new data retrieval processor can be registered in the operation library.

3.4.3 Unary Operations

The architecture supports the basic implementation of the unary operation which can be configured by unary data processor. Basic implementation of unary operation reads objects from input pipe one by one, asynchronously calls unary data processor, and when results of processing are available it forwards objects to the output pipe.

Unary data processor implements 2 methods: put object, get object. Put method retrieves object and processes it in specific way, for example adds new attribute or inserts object into the internal list of object sorted by score. Get method returns the results of processing (objects) one by one if they are available in the moment of method call.

Let us consider the architecture of unary operation based on aggregation. Aggregation process can be implemented as 4 unary operations:

- grouping operation which defines how objects are grouped, that is adds group label to each object;
- aggregating operation, which constructs objects representing groups based on input objects with group labels;
- group score calculation operation, which retrieves input objects representing groups and returns them with scores calculated based on specific rule;
- aggregate calculation operation, which retrieves input objects representing groups and returns them with additional attribute (aggregate) calculated based on group of objects, for example sum on some attribute.

All these operations can be implemented using the basic one configured by corresponding unary data processors.

3.4.4 Binary Operations

Binary operations retrieve objects from 2 input pipes and return result objects when possible to the output pipe.

The system architecture supports the basic implementation of the binary operation based on construction of cross product of operation arguments. Basic implementation is configured by predicate function which takes two objects as arguments and returns the object constructed based on input pair of objects with corresponding score. Basic implementation of binary operation reads objects from input pipes, calls predicate function, and puts to the output pipe new objects with scores from predicate function.

Thus the extensibility of algebra by new binary operations can be based on implementation of new predicate functions. It is important to note that basic configurable predicate implementations can be applied: predicate function comparing attribute value with constant takes attribute name, constant value, and sing of comparison as parameters.

New binary operations can be implemented without predicate function supporting interface used by basic implementation. It is especially important in case of implementation of approximate algorithms for binary operations. One can extend the core of the system with other configurable basic implementations of binary operation, for example approximate algorithms based on partial consideration of cross product pairs.

4 Related Work

A comprehensive discussion of the related work can be found in [31]. However, some main works should be outlined there make the presentation consistent.

4.1 Systems

Several data processing systems are developed to process Big Data [3, 7, 11]. High-level declarative querying languages were proposed [24, 30] to decrease the complexity of data processing workflows specification and benefit from query optimization.

The system for optimization and procession of complex analytical workflows is discussed in [10, 28].

System for approximate evaluation of SQL aggregation queries is proposed in [2]. Approximate queries can be annotated with either an error bound, or a time constraint, based on which the system selects an appropriate sample to evaluate the query. In [27] authors developed a framework for data exploration based on approximately evaluated SQL queries that gives precise control over runtime and quality of result. The proposed system is based on data samples, called impressions, selected to produce low statistical error of a query evaluation result within strict time bounds.

Systems presented in [2, 27] enable evaluation of specific classes of queries in limited resources and focus on approximate query execution based on sampling. The aim of the architecture proposed in this research is to support different types of approximate real-time data analytics in uniform way.

4.2 Extended Models

Traditional data querying means are reconsidered and extended because of variety of data under processing. Proposed declarative querying languages are mostly the extensions of relational algebra [1, 18, 22, 26] and support new approaches to data mining, for example based on similarity and probabilistic models. This fact enables their natural integration into traditional relational data management systems and application of some known query optimization techniques.

Some extensions of querying languages are based on extension of relational data model: attribute representing rank or score of object is marked out from the set of attributes [1, 9, 22].

Based on the extended data model new algebraic operations can be introduced. Several types of querying language extension can be outlined:

- languages supporting user-defined weights representing the importance of data source or sub-query [22, 26];
- extensions of relational algebra [22] based on notion of fussy sets with fixed interpretation of object scores (scores of tuple represents the degree of its membership in relation);
- specific algebras processing fixed type of data, for example images [5, 8].

The ideas and approaches for querying systems with different data processing paradigms, such as relational database and information retrieval in collection of documents, were considered in [9].

4.3 Query Optimization

A brief overview of classical query optimization techniques can be found in [17]. The optimization techniques for distributed heterogeneous systems are discussed in [16, 25]. The query optimization to generate efficient distributed query execution plans for MapReduce workflows were proposed in [20].

4.3.1 Algebraic Equivalencies

Extended querying languages usually constrain the optimization because underlying open and extensible similarity-based algebras support less amount of algebraic equivalencies compared to relational algebra [9]. For example, extended operations changing object scores do not commute with sorting operation.

Authors of [9] proposed 2 alternative ways of extension of the set of equivalent query evaluation plans transformations: based on special limitations on underlying algebra or support of approximate algebraic equivalencies.

Extended querying languages support some algebraic equivalencies considered in [1, 22]. For example, authors of [22] discuss exact transformation of algebraic expression with join and following selection by threshold on score value into the equivalent one with preliminary selection. Such algebraic equivalencies demonstrate that the set of possible query transformation have not decreased but have changed.

4.3.2 Cost Models

The development of cost models is base for optimization of queries specified in extended high-level language [1, 18]. The cost models for specific extended operations and corresponding algorithms should be constructed and analyzed. The technique for estimation of operation evaluation result based on sampling is proposed in [18]. Cost models based on selectivity estimation of extended operations are developed in [1].

Authors of [15] proposed system fully integrating rank-join operation into relational database. The probabilistic model to estimate the size of input arguments for rank-join operation and corresponding cost model are developed in [15].

4.4 Positioning

The proposed system architecture is based on the authors previous research and connects together several parts: similarity-based algebra [23], extended cost/quality models [13], resource allocation approach [32]. The detailed discussion of some separate ideas, concepts, and techniques relevant for implementation of extendable system for optimization and execution of approximate complex queries formulated in similarity-based declarative query language was presented and model implementation was developed to check them. In this work a step forward implementation of full integrated prototype of such system is done: main modules and corresponding interfaces are discussed.

5 Conclusion

The proposed architecture of an extendable system for optimization and execution of approximate complex queries formulated in similarity-based declarative query language demonstrates the possible integration of techniques proposed in previous research and connects together several parts: similarity-based algebra, extended cost/quality models, resource allocation approach.

The future work includes full prototype development supporting adaptable cost and quality models, adaptive query evaluation, enhanced optimizer for approximate query evaluation, and support for multiple execution platforms.

References

[1] S. Adali, P. Bonatti, M. L. Sapino, and V. S. Subrahmanian. A multi-similarity algebra. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, SIGMOD '98, pages 402–413, New York, NY, USA, 1998. ACM.

[2] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: queries with bounded errors and bounded response times on very large data. In Z. Hanzálek, H. Härtig, M. Castro, and M. F. Kaashoek, editors, *EuroSys*, pages 29–42. ACM, 2013.

[3] S. Alsubaiee, Y. Altowim, H. Altwaijry, A. Behm, V. R. Borkar, Y. Bu, M. J. Carey, R. Grover, Z. Heilbron, Y.-S. Kim, C. Li, N. Onose, P. Pirzadeh, R. Vernica, and J. Wen. Asterix: An open source system for "big data" management and analysis. *PVLDB*, 5(12):1898–1901, 2012.

[4] B. Arai, G. Das, D. Gunopulos, and N. Koudas. Anytime measures for top-k algorithms. In C. Koch, J. Gehrke, M. N. Garofalakis, D. Srivastava, K. Aberer, A. Deshpande, D. Florescu, C. Y. Chan, V. Ganti, C.-C. Kanne, W. Klas, and E. J. Neuhold, editors, *VLDB*, pages 914–925. ACM, 2007.

[5] S. Atnafu, L. Brunie, and H. Kosch. Similarity-based algebra for multimedia database systems. In *ADC*, pages 115–122, 2001.

[6] D. Braga, A. Campi, S. Ceri, and A. Raffio. Joining the results of heterogeneous search engines. *Inf. Syst.*, 33(7-8):658–680, 2008.

[7] N. Bruno, S. Jain, and J. Zhou. Continuous cloud-scale query optimization and processing. *PVLDB*, 6(11):961–972, 2013.

[8] P. Budíková, M. Batko, and P. Zezula. Query language for complex similarity queries. In T. Morzy, T. Härder, and R. Wrembel, editors, *ADBIS*, volume 7503 of *Lecture Notes in Computer Science*, pages 85–98. Springer, 2012.

[9] S. Chaudhuri, R. Ramakrishnan, and G. Weikum. Integrating db and ir technologies: What is the sound of one hand clapping? In *CIDR*, pages 1–12, 2005.

[10] U. Dayal, M. Castellanos, A. Simitsis, and K. Wilkinson. Data integration flows for business intelligence. In M. L. Kersten, B. Novikov, J. Teubner, V. Polutin, and S. Manegold, editors, *EDBT*, volume 360 of *ACM International Conference Proceeding Series*, pages 1–11. ACM, 2009.

[11] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150. USENIX Association, 2004.

[12] A. Deshpande, Z. Ives, and V. Raman. Adaptive query processing. *Found. Trends databases*, 1:1–140, January 2007.

[13] O. Dolmatova, A. Yarygina, and B. Novikov. Cost models for approximate query evaluation algorithms. In A. Caplinskas, G. Dzemyda, A. Lupeikiene, and O. Vasilecas, editors, *Databases and Information Systems. Tenth International Baltic Conference on Databases and Information Systems. Local Proceedings, Materials of Doctoral Consortium.*, pages 20–28. Vilnius: Zara, 2012.

[14] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In J. Peckham, editor, *SIGMOD Conference*, pages 171–182. ACM Press, 1997.

[15] I. F. Ilyas, R. Shah, W. G. Aref, J. S. Vitter, and A. K. Elmagarmid. Rank-aware query optimization. In G. Weikum, A. C. König, and S. Deßloch, editors, *SIGMOD Conference*, pages 203–214. ACM, 2004.

[16] D. Kossmann. The state of the art in distributed query processing. *ACM Comput. Surv.*, 32(4):422–469, Dec. 2000.

[17] D. Kossmann and K. Stocker. Iterative dynamic programming: a new class of query optimization algorithms. *ACM Trans. Database Syst.*, 25(1):43–82, Mar. 2000.

[18] C. Li, K. C.-C. Chang, I. F. Ilyas, and S. Song. Ranksql: Query algebra and optimization for relational top-k queries. In F. Özcan, editor, *SIGMOD Conference*, pages 131–142. ACM, 2005.

[19] F. Li, M. M. Moro, S. Ghandeharizadeh, J. R. Haritsa, G. Weikum, M. J. Carey, F. Casati, E. Y. Chang, I. Manolescu, S. Mehrotra, U. Dayal, and V. J. Tsotras, editors. *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*. IEEE, 2010.

[20] H. Lim, H. Herodotou, and S. Babu. Stubby: A transformation-based optimizer for mapreduce workflows. *PVLDB*, 5(11):1196–1207, 2012.

[21] S. E. Madnick, R. Y. Wang, Y. W. Lee, and H. Zhu. Overview and framework for data and information quality research. *J. Data and Information Quality*, 1(1):2:1–2:22, June 2009.

[22] D. Montesi, A. Trombetta, and P. A. Dearnley. A similarity based relational algebra for web and multimedia data. *Inf. Process. Manage.*, 39(2):307–322, 2003.

[23] B. Novikov, N. Vassilieva, and A. Yarygina. Querying big data. In *Proceedings of the 13th International Conference on Computer Systems and Technologies*, CompSysTech '12, pages 1–10, New York, NY, USA, 2012. ACM.

[24] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In J. T.-L. Wang, editor, *SIGMOD Conference*, pages 1099–1110. ACM, 2008.

[25] F. Pentaris and Y. Ioannidis. Query optimization in distributed networks of autonomous database systems. *ACM Trans. Database Syst.*, 31(2):537–583, June 2006.

[26] I. Schmitt and N. Schulz. Similarity relational calculus and its reduction to a similarity algebra. In D. Seipel and J. M. T. Torres, editors, *FoIKS*, volume 2942 of *Lecture Notes in Computer Science*, pages 252–272. Springer, 2004.

[27] L. Sidirourgos, M. L. Kersten, and P. A. Boncz. Sciborq: Scientific data management with bounds on runtime and quality. In *CIDR*, pages 296–301. www.cidrdb.org, 2011.

[28] A. Simitsis, K. Wilkinson, M. Castellanos, and U. Dayal. Optimizing analytic data flows for multiple execution engines. In K. S. Candan, Y. Chen, R. T. Snodgrass, L. Gravano, and A. Fuxman, editors, *SIGMOD Conference*, pages 829–840. ACM, 2012.

[29] M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, editors, *VLDB*, pages 648–659. Morgan Kaufmann, 2004.

[30] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, N. Zhang, S. Anthony, H. Liu, and R. Murthy. Hive - a petabyte scale data warehouse using hadoop. In Li et al. [19], pages 996–1005.

[31] A. Yarygina. Execution and optimization techniques for approximate queries in heterogeneous systems. *Programming and Computer Software*, pages 309–317, 2013.

[32] A. Yarygina and B. Novikov. Optimizing resource allocation for approximate real-time query processing. *Computer Science and Information Systems*, 11:69–88, January 2014.