**DEBS**

**I. Query Optimization in Distributed Systems**

Distributed query optimization refers to producing a plan for processing a query in a distributed database system.A query execution plan consists of operators and their allocation to servers. Some operators process data and consolidate results and others realize the transfer of data.The query optimizer attempts to determine the most efficient way to execute a query.

Usually the query optimization is an automated process but some database engines allow hints from the user for the optimization.

The needed data can be obtained in different ways, with different data structures, different queries, different orders. Trying to obtain the exact optimal solution can be costly, if not impossible. Because of this, an approximation of the optimal solution is attempted, by comparing several good alternatives in order to find a good enough solution, in a reasonable period of time. In this respect, the current solutions are confronted with the trade-off between the quality of the solution and the amount of time needed to find it.

Most query optimizers represent query plans as a tree of "plan nodes". A plan node encapsulates a single operation. The partial results go from the bottom of the tree to the top. Each child node produces an output that is entered as input to the parent node.

In a distributed database system, a query is optimized at a global level as well as at a local level.The data required for a single query can be distributed across multiple sites and the local databases know only about the local data. The controlling site is the one that can reconstruct the data based on a global data dictionary.

If there is no replication involved, the global optimization gathers the data from the sites where each fragment is stored. If there is replication, the best source for the data must be selected, based on server speed, communication cost, and workload.

At the global level optimization, an execution plan is generated for the queries, in order to run the query with a minimum cost. The plan describes the location of the fragments, order of steps for running the query and a plan for transferring the intermediate results.

The local queries are then optimized by the local sites. After this step, the local results are merged through union in case of horizontal fragmentation or join in case of vertical fragmentation.

The evaluation is made between large numbers of query trees that produce the required results. The goal is find not necessarily the best solution, but the optimal one.

The main issues for distributed query optimization are:

1. Utilization of resources
2. Query trading
3. Reduction of solution space
4. Utilization of resources

The approaches that can be used for optimizing the resources used are:

* Operation Shipping

In operation shipping, the query is run at the local site and the results are transferred to the controlling site.

* Data Shipping

In data shipping the data is transferred to the controlling site, where the query is run. This can be used especially in cases where the cost of communication is low and the local servers are slow.

* Hybrid Shipping

This is a combination between operation shipping and data shipping. In this strategy, the data fragments are transferred to the servers that have high computational power and after this the results are sent to the controlling site.

1. Query Trading

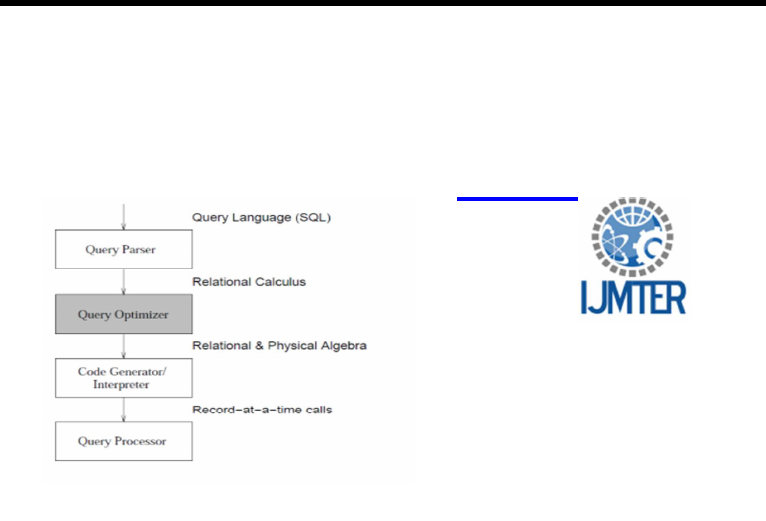
The controlling site is called the buyer and the local sites are called sellers. The purpose is to select the optimal solution. The buyer assigns parts of the queries to the seller sites and they propose local query optimizations. The plan combines these local optimization proposals and the cost of combining the results.

1. Reduction of Solution Space

Reducing the solution space can be done following a set of rules, such as:

* Perform selection operations as early as possible in order to minimize the communication between the sites.
* Eliminate selection conditions not relevant to a site.
* In case of join and union, move fragments of data where most of the data is present and perform there the operation.

A basic flow for the process of query optimization can be represented as follows:



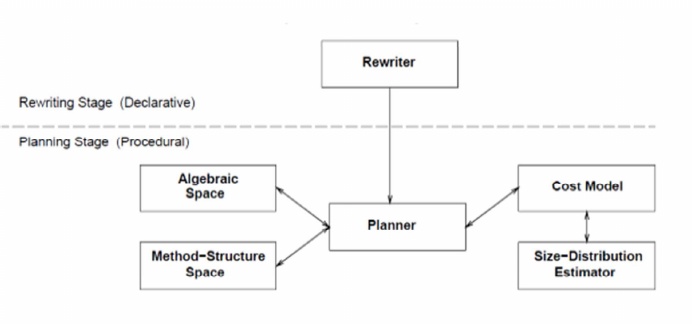
The Query checks the validity of the query and then translates it into an internal form.

The Query Optimizer examines all algebraic expressions that are equivalent to the given query and chooses the one that is estimated to be the cheapest.

The Code Generator or the Interpreter transforms the access plan generated by the optimizer into calls to the query processor.

The Query Processor actually executes the query.

**Query Optimizer Architecture**

****

The query optimization process has two stages: rewriting and planning. Only one module is in the first stage, the Rewriter, all other modules being in the second stage.

Rewriter module applies transformations to a given query and produces equivalent queries that are hopefully more efficient.

Planner module is the main module of the ordering stage. It examines all possible execution plans for each query produced in the previous stage and selects the overall cheapest one to be used to generate the answer of the original query.

Method-Structure Space module determines the implementation choices that exist for the execution of each ordered series of actions specified by the Algebraic Space.

Cost Model module specifies the arithmetic formulas that are used to estimate the cost of execution plans.

Size-Distribution Estimator module specifies how the sizes of database relations and indices as well as query results are estimated. These estimates are needed by the Cost Model.

Algebraic Spacemodule determines the action execution orders that are to be considered by the Planner for each query sent to it. All such series of actions produce the same query answer, but usually differ in performance.

**II. About DEBS**

Over the past decade, the ACM International Conference on Distributed and Event-based Systems (DEBS) has become the premier venue for contributions in the fields of distributed and event-based systems. The objectives of the ACM International Conference on Distributed and Event-Based Systems (DEBS) are to provide a forum dedicated to the dissemination of original research, the discussion of practical insights, and the reporting of experiences relevant to distributed systems and event-based computing. The conference aims at providing a forum for academia and industry to exchange ideas through industry papers and demo papers.

DEBS 2016 will be held in Irvine, California, US from June 20 to June 24, 2016, at the Beckman Center of the National Academies of Sciences & Engineering. [4]

**III. Scope of DEBS 2016**

Starting this year DEBS is extending its scope to embrace a broader set of topics related to distributed systems and event-based computing. Topics of particular interest may include (but are not limited to) models, architectures and paradigms of distributed and event-based systems, middleware systems and frameworks, and applications, experiences and requirements. The scope of the DEBS conference covers all topics relevant to distributed and event-based computing ranging from those discussed in related disciplines (e.g., software systems, distributed systems, distributed data processing, data management, dependability, knowledge management, networking, programming languages, security and software engineering), to domain-specific topics of event-based computing (e.g., real-time analytics, mobile computing, social networking, pervasive, green computing and ubiquitous computing, sensors networks, user interfaces, big data processing, spatio-temporal processing, cloud computing, the Internet of things, peer-to-peer computing, embedded systems and stream processing), to enterprise-related topics (e.g., complex event detection, enterprise application integration, real-time enterprises and web services).

In addition to these traditional topics, the scope of DEBS 2016 will include the increasingly important area of Internet of Things. New advances in distributed and event-based systems pose a great potential for a major contribution in this area. For further information, please refer to the call for contributions in the track of your choice.

**IV. Introduction of DEBS 2016**

The ACM DEBS 2016 Grand Challenge is the sixth in a series [6, 7, 8, 9] of challenges which seek to provide a common ground and uniform evaluation criteria for a competition aimed at both research and industrial event-based systems. The goal of the 2016 DEBS Grand Challenge competition is to evaluate event-based systems for real-time analytics over high volume data streams in the context of graph models.

The underlying scenario addresses the analysis metrics fora dynamic (evolving) social-network graph. Specifically, the 2016 Grand Challenge targets the following problems: identification of the posts that currently trigger the most activity in the social network, and identification of large communities that are currently involved in a topic. The corresponding queries require continuous analysis of a dynamic graph under the consideration of multiple streams that reflect updates to the graph. [5]

**V. Input Data Streams**

The input data is organized in four separate streams, each provided as a text file. Namely, we provide the following input data files:

**friendships.dat**: <ts, user\_id\_1, user\_id\_2>

|  |  |
| --- | --- |
| **ts** | is the friendship's establishment timestamp |
| **user\_id\_1** | is the id of one of the users |
| **user\_id\_2** | is the id of the other user |

**posts.dat**: <ts, post\_id, user\_id, post, user>

|  |  |
| --- | --- |
| **ts** | is the post's timestamp |
| **post\_id** | is the unique id of the post |
| **user\_id** | is the unique id of the user |
| **post** | is a string containing the actual post content |
| **user** | is a string containing the actual user name |

**comments.dat**: <ts, comment\_id, user\_id, comment, user, comment\_replied, post\_commented2>

|  |  |
| --- | --- |
| **ts** | is the comment's timestamp |
| **comment\_id** | is the unique id of the comment |
| **user\_id** | is the unique id of the user |
| **comment** | is a string containing the actual comment |
| **user** | is a string containing the actual user name |
| **comment\_replied** | is the id of the comment being replied to (-1 if the tuple is a reply to a post) |
| **post\_commented** | is the id of the post being commented (-1 if the tuple is a reply to a comment) |

**likes.dat**: <ts, user\_id, comment\_id>

|  |  |
| --- | --- |
| **ts** | is the like's timestamp |
| **user\_id** | is the id of the user liking the comment |
| **comment\_id** | is the id of the comment |

**VI. First Query**

The goal of query 1 is to compute the top-3 scoring active posts, producing an updated result every time they change.

The total score of an active post P is computed as the sum of its own score plus the score of all its related comments. Active posts having the same total score should be ranked based on their timestamps (in descending order), and if their timestamps are also the same, they should be ranked based on the timestamps of their last received related comments (in descending order). A comment C is related to a post P if it is a direct reply to P or if the chain of C's preceding messages links back to P.

Each new post has an initial own score of 10 which decreases by 1 each time another 24 hours elapse since the post's creation. Each new comment's score is also initially set to 10 and decreases by 1 in the same way (every 24 hours since the comment's creation). Both post and comment scores are non-negative numbers, that is, they cannot drop below zero. A post is considered no longer active (that is, no longer part of the present and future analysis) as soon as its total score reaches zero, even if it receives additional comments in the future.

**Input Streams:** *posts*, *comments*

***Output specification:***

<ts,top1\_post\_id,top1\_post\_user,top1\_post\_score,top1\_post\_commenters,  
top2\_post\_id,top2\_post\_user,top2\_post\_score,top2\_post\_commenters,  
top3\_post\_id,top3\_post\_user,top3\_post\_score,top3\_post\_commenters>

**ts:** the timestamp of the ~~tuple~~ event that triggers a change in the top-3 scoring active posts appearing in the rest of the tuple  
**topX\_post\_id:** the unique id of the top-X post  
**topX\_post\_user:** the user author of top-X post  
**topX\_post\_commenters:** the number of commenters (excluding the post author) for the top-X post

Results should be sorted by their timestamp field. The character "-" (a minus sign without the quotations) should be used for each of the fields (post id, post user, post commenters) of any of the top-3 positions that has not been defined. Needless to say, the logical time of the query advances based on the timestamps of the input tuples, not the system clock.

Sample output tuples for the query

2010-09-19 12:33:01.923+0000,25769805561,Karl Fischer,115,10,25769805933,Chong Liu,83,4,-,-,-,-  
2010-10-09 21:55:24.943+0000,34359739095,Karl Fischer,58,7,34359740594,Paul Becker,40,2,34359740220,Chong Zhang,10,0  
2010-12-27 22:11:54.953+0000,42949673675,Anson Chen,127,12,42949673684,Yahya Abdallahi,69,8,42949674571,Alim Guliyev,10,0

**VII. Second Query**

This query addresses the change of interests with large communities. It represents a version of query type 2 from the 2014 SIGMOD Programming contest. Unlike in the SIGMOD problem, the version for the DEBS Grand Challenge focuses on the dynamic change of query results over time, i.e., calls for a continuous evaluation of the results.

**Goal of the query:**  
 Given an integer k and a duration d (in seconds), find the k comments with the largest range, where the range of a comment is defined as the size of the largest connected component in the graph defined by persons who (a) have liked that comment (see likes, comments), (b) where the comment was created not more than d seconds ago, and (c) know each other (see friendships).

**Parameters:** k, d

**Input Streams:** likes, friendships, comments

**Output:**   
 The output includes a single timestamp ts and exactly k strings per line. The timestamp and the strings should be separated by commas. The k strings represent comments, ordered by range from largest to smallest, with ties broken by lexicographical ordering (ascending). The k strings and the corresponding timestamp must be printed only when some input triggers a change of the output, as defined above. If less than k strings can be determined, the character “-” (a minus sign without the quotations) should be printed in place of each missing string.

The field ts corresponds to the timestamp of the input data item that triggered an output update. For instance, a new friendship relation may change the size of a community with a shared interest and hence may change the k strings. The timestamp of the event denoting the added friendship relation is then the timestamp ts for that line's output. Also, the output must be updated when the results change due to the progress of time, e.g., when a comment is older that d. Specifically, if the update is triggered by an event leaving a time window at t2 (i.e., t2 = timestamp of the event + window size), the timestamp for the update is t2. As in Query 1, it is needless to say that timestamps refer to the logical time of the input data streams, rather than on the system clock.

In summary, the output is specified as follows:

**ts**: the timestamp of the~~tuple~~ event that triggers a change in the output.  
**comments\_1,...,comment\_k**: top k comments ordered by range, starting with the largest range (comment\_1).

#### *Sample output tuples for the query with k=3 could look as follows:*

2010-10-28T05:01:31.022+0000,I love strawberries,-,-  
2010-10-28T05:01:31.024+0000,I love strawberries,what a day!,-  
2010-10-28T05:01:31.027+0000,I love strawberries,what a day!,well done  
2010-10-28T05:01:31.032+0000,what a day!,I love strawberries,well done

**Bibliography**

1. J.Janki.“An Analysis on Query Optimization in Distributed Database”. International Journal of Modern Trends in Engineering and Research, 2014
2. https://www.tutorialspoint.com/distributed\_dbms/distributed\_dbms\_query\_optimization\_distributed\_systems.htm
3. https://en.wikipedia.org/wiki/Query\_optimization
4. DEBS website - <http://www.debs2016.org/>
5. V. Gulisano, Z. Jerzak, S. Voulgaris, and H. Ziekow. “The debs 2016 grand challenge”. DEBS 2016 proceedings, June 2016
6. Zbigniew Jerzak, Thomas Heinze, Matthias Fehr, Daniel Grober, Raik Hartung, and Nenad Stojanovic. “The DEBS 2012 grand challenge”. Proceedings of the Sixth ACM International Conference on Distributed Event-BasedSystems, DEBS 2012, Berlin, Germany, July 16-20, 2012, pages 393–398. ACM, 2012.
7. Christopher Mutschler, Holger Ziekow, and Zbigniew Jerzak. “The DEBS 2013 grand challenge”. The 7th ACM International Conference on Distributed Event-Based Systems, DEBS ’13, Arlington, TX, USA - June 29 - July 03, 2013, pages 289–294. ACM, 2013.
8. Zbigniew Jerzak and Holger Ziekow. The DEBS 2014 grand challenge. “The 8thACM International Conference on Distributed Event-Based Systems”, DEBS ’14, Mumbai, India, May 26-29, 2014, pages 266–269. ACM, 2014.
9. Zbigniew Jerzak and Holger Ziekow. “The DEBS 2015 Grand Challenge”. Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, DEBS ’15, Oslo, Norway, June 29 - July 3, 2015, pages 266–268. ACM, 2015.