ISSN 1846-6168 (Print), ISSN 1848-5588 (Online)
https://doi.org/10.31803/tg-20210205094356

Preliminary communication

An Efficient Query Optimizer with Materialized Intermediate Views in Distributed and Cloud Environment

Archana Bachhav*, Vilas Kharat, Madhukar Shelar

Abstract: In cloud computing environment hardware resources required for the execution of query using distributed relational database system are scaled up or scaled down according to the query workload performance. Complex queries require large scale of resources in order to complete their execution efficiently. The large scale of resource requirements can be reduced by minimizing query execution time that maximizes resource utilization and decreases payment overhead of customers. Complex queries or batch queries contain some common subexpressions. If these common subexpressions evaluated once and their results are cached, they can be used for execution of further queries. In this research, we have come up with an algorithm for query optimization, which aims at storing intermediate results of the queries and use these by-products for execution of future queries. Extensive experiments have been carried out with the help of simulation model to test the algorithm efficiency.

Keywords: Conventional SQL; Database Management in Cloud Environment; Distributed Databases; Intermediate Views; Materialized Views; Query Execution Time; Query Optimization

1 INTRODUCTION

In cloud environment, bulk of applications are based on data which are managed by Database Management Systems and that forms a crucial issue on cloud platform. So, service oriented computing in cloud, however is also extended to Database as a Service (DBaaS) [1]. Database management in cloud and efficient data access for cloud users become cumbersome task for cloud service providers. Cloud computing has limit of sharing processing time and storage space for various applications in databases. The proliferation found in diverse applications which impacted exceptional cloud stages bringing about massive growth inside the length of the information created just as devoured through such applications [2]. The most effective method to put together and deal with those large databases in order to get the required information for the clients is found to be the new research area in distributed and cloud environment. The data modelling in cloud platform is the basis of cloud applications and the key issue is the searching algorithms applied [3]. The most effective method to get the information convenient, precisely and dependably; assumes a significant function in the achievement of database model in cloud platform.

In cloud platform, users can hire huge resources for short period of time to execute complex queries more efficiently on large database using group of virtual machines [4]. The hire charges of resources for users can be reduced using better query optimization technique [5]. Thus there is a need of exploring efficient techniques for query execution that would decrease runtime and response time. It will likewise upgrade optimum use of resources in cloud data centers. This paper is in continuation with [6-8], where an extensive survey on various query optimization approaches as well the novel architecture of an intelligent query optimizer for distributed database has been presented.

Contributions - As such, the researchers have designed and developed a technique for query optimization in distributed and cloud environment. The contributions of this research paper are as follows.

- Devise an architecture of query optimizer for distributed databases which is integrated with materialized views that are reused for evaluation of further queries in the system.
- Formulate a query optimization technique that results in a better optimization of database queries.
- Introduce an optimization strategy that may reduce bandwidth requirement in cloud environment by satisfying SLA between customer and cloud service provider.
- Demonstrate the developed model with TPC-H benchmark dataset and series of benchmark queries to test performance of devised query optimization technique.
- Improve resource utilization in cloud by reducing query evaluation time and response time.

The remainder of this research paper is structured as follows. Section 2 presents an outline of related work. The proposed work with cost model is elaborated in section 3. Section 4 describes an experimental setup with workload of benchmark dataset and series of queries for testing performance of query optimizer as well as results for evaluation of the proposed system. Section 5 concludes the paper.

2 RELATED WORK

Optimization of database queries is performed through two phases – search space generation and optimal plan selection from the search space [5]. Researchers have discovered different approaches on query optimization those deals with reduction of communication cost, reduction of execution time and appropriate utilization of system resources.

Execution time of queries can be reduced by eliminating common sub-expressions used. Start-fetch wrapper using request window mechanism is used by Lee R et al. [9] to develop an IGNITE system that eliminates common sub-expressions. However, the communication traffic generated in IGNITE is reduced by Chen G et al. [10] and Dokeroglu et al. [4] by using efficient sets of query execution plans in their research work. The response time of queries is reduced with the help of parallel query processing systems in research works proposed by Garofalakis et al. [11] and Dokeroglu et al. [12]. Giannikis et al. [13] proposed an architecture based on sharing of computation, storage and cache memory by creating batch of queries. Similar concept of resource sharing for query optimization is presented in [14].

Iterative processing method can also be used for query optimization in which actual runtime statistics is collected by continuously monitoring execution of queries [15]. However, based on intermediate results, it would be cumbersome to collect required statistics. Hence Cole and Grafe [16, 17] addressed a multiple plans generation technique at compile time. POP is the progressing query optimization approach invented in [18, 19], where cardinality approximation errors are detected in mid execution of queries. DB2L learning optimizer based on use of misestimates to learn and adjust the statistics to enhance better optimization of further queries [20]. Wang et al. [21] presented adaptive query optimization approach for cloud database system based on execution time as well as monetary costs.

The technique of caching intermediate results is one of the widely used query optimization technique [22], extended by Safaeei A et al. [23] based on multiple sliding windows to improve execution of overlapping queries with common sub-expressions. Laptev et al. [24] presented EARL system and Agarwal et al. [25] proposed the BlinkDB, those iteratively works for collecting larger samples to reach at the desired accuracy. The Shark, presented in [26] caches inter query data with the help of shared memory concept. In distributed cloud environment, CHive [27] and NOVA [28] are the query optimization techniques based on incremental processing of continuous data. Logothetis et al. [29] invented CBP (Continuous Bulk Processing) system in which working state is preserved during query processing to reuse it for future queries. DBaaS can offer an assistance to cloud users that ready to get results with more fragile quality in return of lower cost [30].

Query processing time can be reduced by materializing views generated from select-project join operations [31]. The policies of protecting materialized views cache are introduced by WATCHMAN [32] and DynaMat [33] systems to increase the hit ratio. Ivanova et al. [34] devised an architecture that optimizes query processing by maintaining cache to avoid repetitions of physical operations. The results from MapReduce jobs can be reused by describing them in the form of analytical query languages [35-37]. Perez et al. [38] presented history aware optimizer that archives intermediate results of queries so as to reuse them during future query execution. Intermediate results or views that are to be cached are determined by Cache-on-Demand [39] as well as MQT technique [40] and relations that may be useful in execution of future queries are identified by Kossmann et al. [41].

MapReduce technique results in higher processing cost for queries with more join operations, which can be reduced using pipeline approach where results of a query can be referred by next queries to continue processing. Anyanwu K et al. [42] proposed a data model to minimize the number of MapReduce cycles using pipeline approach. Automatic Query Analyser (AQUA) developed by Wu et al. [43] works for MapReduce in two phases, first phase for minimizing number of MapReduce cycles and second phase for joining intermediate results. MapReduce online system avoids materialization by pipelining midway results from map jobs to reduce jobs [44].

After navigating various approaches on database query optimization, it has been perceived that there is a need of suitable technique to reduce the runtime for database queries that minimizes resource requirement. Conventional SQL cannot predict the future requirements, therefore it reduces the performance than MapReduce technique. MapReduce uses pipeline approach to reduce query processing time, however it require higher processing cost for queries with more join operations. Hence to bridge the gap between MapReduce and Conventional SQL, there is a demand of efficient query optimization technique in distributed and cloud environment.

3 PROPOSED WORK

The primary aim of this research work is to design and develop an intelligent query optimizer that improves resource utilization in distributed and cloud environment by reducing query evaluation time and response time. As such, the proposed query optimizer materializes intermediate views during query evaluation so as to reuse them for execution further queries. Hence, it will reduce I/O operations, communication cost as well as execution time.

3.1 An Architecture

An architecture of the proposed query optimizer presented in Fig. 1 comprises of various components.

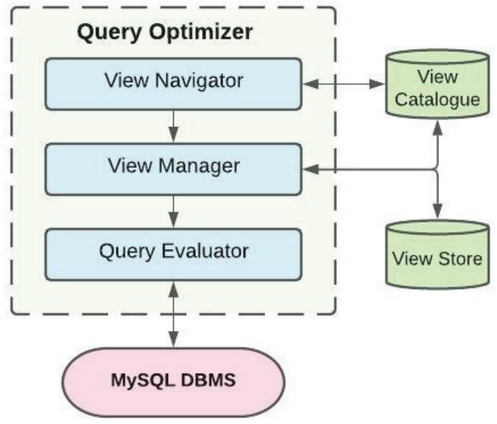


Figure 1 Architecture of the Proposed Query Optimizer

View Navigator searches for appropriate views during query evaluation by navigating View Catalogue with the help of view matching algorithm presented in Tab. 1. It navigates through each catalogue entry and tries to find the longest match view name stored in view store and returns the location of view store where match is found. Partial intermediate results are also used for evaluation of future queries. This algorithm applies commutative rule for natural join.

View Manager maintains generated views in View Store and also updates View Catalogue. It also keeps track of view reference count as well as read/write timestamp values for every view. These values are used for decision making of view deletions from View Store and corresponding updates in View Catalogue.

Query Evaluator performs the query evaluation task after substituting matching views.

View Store materializes views those are generated from previously evaluated queries.

View Catalogue maintains the catalogue of view names those are materialized in view store. View name matched by View Manager from View Catalogue mapped into View Store.

Table 1 Algorithm for View Matching

```
    Algorithm ViewMatching
    Input :
    - Catalog of View names
    - Viewpattern to be searched
    Output:  blocation (Location of matched view name as best location)
1   Begin
2     bvlength ← 0; //initialize Length of longest matched Viewname as best length to 0
3     For each catalogue entry i do
4         Viewname ← catalogue[i].viewname;
5         If (Viewname is substring of Viewpattern) OR
          (Reverse(Viewname) is substring of Viewpattern)
6             If (length(Viewname) > bvlength) //select the longest
              //viewname as best viewname
7                 bViewname ← Viewname;
8                 bViewlength ← length(Viewname);
9                 blocation ← catalogue[i].location
10                If length(Viewname)=length(Viewpattern)) //Exact match
                  //found
11                    break For; //Terminate searching
12                End If
13            End If
14        End If
15    End For
16    Return blocation;
17  End
```

3.2 Cost Model

In order to measure the performance of proposed query optimization technique, a cost model has been developed based on various parameters presented in Tab. 2.

Let Q be the query from workload has been divided into n subqueries $\{Q_1, Q_2, \dots, Q_n\}$ during evaluation and trying to match with k number of materialized views $\{V_1, V_2, \dots, V_k\}$ in View Store.

Total execution cost of query Q includes execution time, memory requirement, IO cost as well as total view matching time from View Store during evaluation of query Q, as shown in Eq. (1). Total view matching time for query Q is calculated as in Eq. (2).

$$CE(Q) \approx \left(\sum_{i=1}^{n} ET(Q_i) + M(Q_i) + IO(Q_i) \right) + VMT(Q) \tag{1}$$

$$VMT(Q) = \sum_{i=1}^{n} VMT(Q_i) \tag{2}$$

where maximum and minimum view matching time from View Store of any subquery Q_i is shown in Eq. (3) and (4) respectively.

Table 2 List of Parameters and symbols used in the cost model

Parameter/ Symbol	Purpose
N	Number of subqueries in query Q
k	Number of materialized views
$CE(Q)$	Total execution cost of query Q
$ET(Q_i)$	Execution time for query Qi, where i = 1,..., n
$M(Q_i)$	Memory requirement for execution of query Q_i
$IO(Q_i)$	Input-output cost require for execution of query Q_i
$VMT(Q)$	Total view matching time require for query Q
$VMT(Q_i)$	View matching time for subquery Q_i, where i = 1,..., n
$VMT(V_j)$	Matching time require for particular View V_j, where j = 1,…, k
$Size(r)$	Size of relation r
$EET(Q_i)$	Execution time require for subquery Q_i
$ETMV(Q_i)$	Execution time of subquery Q_i using materialized view
$ETNMV(Q_i)$	Execution time of subquery Q_i without using materialized view
HR	Hit ratio which is the probability of matching view from View Store
MR	Miss ratio which is the probability of not matching any view from View Store

$$Max\left(VMT(Q_i)\right) = VMT(V_i) \times k \tag{3}$$

$$Min\left(VMT(Q_i)\right) = VMT(V_j) \tag{4}$$

Memory requirement $M(Q_i)$ and IO cost $IO(Q_i)$ for execution of query Q_i are directly proportional to the size of relations r those are involved in query evaluation, shown in Eq. (5) and (6). However processing and IO cost will get reduced when view is matched from View Store.

$$M(Q_i) \propto Size(r) \tag{5}$$

$$IO(Q_i) \propto Size(r) \tag{6}$$

Total effective execution time of query Q using materialized view is calculated as in Eq. (7), where effective execution time of subquery Q_i is as shown in Eq. (8).

$$EET(Q) = \sum_{i=1}^{n} EET(Q_i) \tag{7}$$

$$EET(Q_i) = HR \times ETMV(Q_i) + MR \times ETNMV(Q_i) \tag{8}$$

4 EXPERIMENTAL SETUP AND RESULTS

It is crucial to evaluate the performance of proposed algorithm on real infrastructure of large-scale database system. Hence, to ensure the large scale database system infrastructure, the simulator model for query optimization has been developed in Java which run on the top of MySQL to compare the results of the proposed technique with existing approaches.

4.1 Simulation

The simulation model contains various modules.

Frequent Query Join Holder module maintains a data structure for catalogue where it makes entries of views which are cached. Before executing any query this catalogue is to be searched. The view matching algorithm has been implemented on the top of Zql parser that parses SQL statements for matching views. If any view matches then resulting entry is searched in actual view store.

Frequent Query Join module is responsible for managing view store. It stores results of previously executed queries as views. If any view is not in use for a long period of time then it deletes it and also deletes its entry from catalogue. Hence view searching time will not dominate query execution time.

DBMS Query Runner module is responsible for execution of queries with the help of Zql Parser [45]. Zql parser parses SQL statements and generates Java structures signifying query statements and expressions. Zql parser takes SQL statements with insert, delete, update, select etc. as an input and produces data structure that represent the statements it parsed.

Example:
Select ps_supplycost, pname from part, partsupp, supplier, nation where p_partkey=ps_partkkey and ps_suppkey=s_suppkey and s_nationkey=n_nationkey and n_name='India';

Zql parser extract various parts of this query and generates ZqlQuery structure. The methods getSelect(), getFrom() and getWhere() returns Select, From and Where parts of the query in their respective data structures of Java.

Result Processor module analyses results using various parameters such as total execution time of queries, average runtime per query and number of iterations to be performed.

4.2 Workload

The proposed query optimizer has been tested by generating various workloads using the standard benchmark dataset of TPC-H [46]. The TPC-H database consist of eight individual relations as shown in Tab. 3, which represents data to exercise functionalities of complex analysis application system. The testing has been performed as small scale level, after distributing these benchmark relations randomly over two nodes after the horizontal fragmentation.

TPC-H benchmark comprised of set of 22 original queries to give realistic context that represent the activity of wholesale supplier. With the variant of these 22 base queries, the workload of 50 queries is generated to test the performance of proposed technique. In order to test the effectiveness of proposed system, four variations of workload of queries are generated as shown in Tab. 4. In each query, the selection predicates are generated at random with the help of proper range of probable values as per the benchmark.

Table 3 List of relations in TPC-H database

Sr. No.	Relation/Table Name	Number of records
1	Part	1,89,945
2	Supplier	10,000
3	Partsupp	7,43,870
4	Customer	1,37,542
5	Nation	25
6	Region	5
7	Orders	14,53,561
8	Lineitem	10,59,450

Table 4 Variations in Workload of Queries

Workload	Queries with features
W_1	Queries in which none of them having common sub-expressions in their predicates
W_2	Queries in which some of them having common sub-expressions in their predicates
W_3	Set of repeated queries which are not having common sub-expressions
W_4	Set of repeated queries having common sub-expressions

By using the cost model presented in section 3.2, efficiency of proposed query optimizer has been measured in terms of various performance parameters such as total time for query execution, average runtime per query and number of iterations.

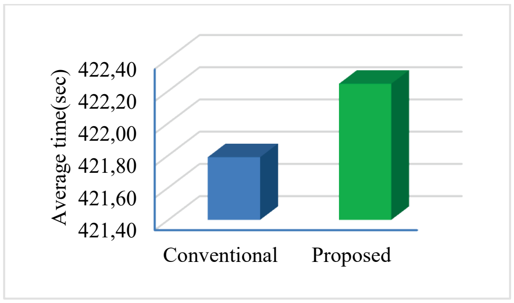
4.3 Results and Discussion

The performance of proposed query optimization technique is analysed and compared with conventional SQL system w.r.t. performance parameters viz. total execution time for workload of queries, average runtime per query and number of iterations required. Extensive experiments have been conducted with the help of all variations of query workloads specified in Tab. 4 and results are compared with conventional SQL system as presented in Tab. 5.

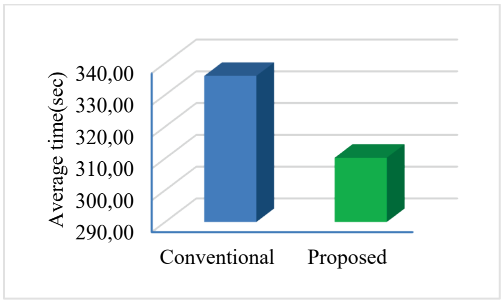
Table 5 Result analysis of Conventional SQL system vs proposed system

Workload	Parameter	Conventional	Proposed
W_1	Average time per query (Sec)	421.79	422.25
	Total time (Minutes)	70.30	70.37
	Number of Iterations	5305870	5306045
W_2	Average time per query (Sec)	336.01	310.25
	Total time (Minutes)	61.60	56.88
	Number of Iterations	4090545	4090765
W_3	Average time per query (Sec)	396.69	111.70
	Total time (Minutes)	132.23	37.23
	Number of Iterations	10611740	5306320
W_4	Average time per query (Sec)	265.09	91.77
	Total time (Minutes)	88.36	30.59
	Number of Iterations	9039020	3078470

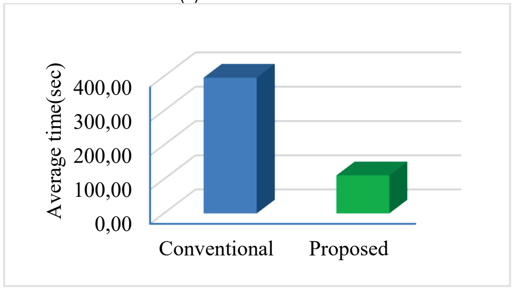
As workload W_1 contains queries without any repetition and/or common subexpression, it cannot take benefit of materialization. Hence, proposed system incurs extra overhead of matching views from view store. However, we observed the decrease in running time for queries in workloads W_2 to W_4 that are matched and recycled intermediate views from view store. Fig. 2 shows the comparative chart on average execution time per query using conventional SQL system and proposed query optimization technique.



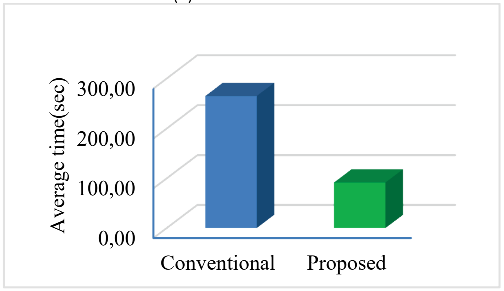
(a) For Workload W_1



(b) For Workload W_2



(c) For Workload W_3



(d) For Workload W_4

Figure 2 Comparative study on average runtime per query

5 CONCLUSION AND FUTURE WORK

In this paper the novel query optimization technique with architecture of query optimizer has been presented, which aims at recycling intermediate results of previously executed queries for execution of future queries. The queries those contain common subexpressions or repeated, can use these materialized views for their execution so that it results into decrease in the average execution time per query and increase in performance of the system. Hence using this technique resource utilization in cloud environment will also get improved. The proposed system has been tested on small scale infrastructure and it is observed that query execution time is reduced to approximately 30% as compared to the conventional query processing system due to materialization of intermediate views. As a future work, the proposed system will be evaluated on a large scale infrastructure in cloud environment to test its efficiency.

Notice

This paper was presented at IC2ST-2021 – International Conference on Convergence of Smart Technologies. This conference was organized in Pune, India by Aspire Research Foundation, January 9-10, 2021. The paper will not be published anywhere else.

6 REFERENCES

[1] Hacigümüs, H., Tatemura, J., Hsiung, W. P., Moon, H. J., Po, O., Sawires, A., et al. (2010, July). CloudDB: One size fits all revived. In *2010 6th World Congress on Services* (pp. 148-149). IEEE. https://doi.org/10.1109/SERVICES.2010.96

[2] Abadi, D. J. (2009). Data management in the cloud: Limitations and opportunities. *IEEE Data Eng. Bull.*, 32(1), 3-12.

[3] Zhou, L., He, K., Sheng, X., & Wang, B. (2013). A survey of data management system for cloud computing: models and searching methods. *Research Journal of Applied Sciences, Engineering and Technology*, 6(2), 244-248. https://doi.org/10.19026/rjaset.6.4064

[4] Dokeroglu, T., Sert, S. A., & Cinar, M. S. (2014). Evolutionary multiobjective query workload optimization of Cloud data warehouses. *The Scientific World Journal*, 2014. https://doi.org/10.1155/2014/435254

[5] Doshi, P., & Raisinghani, V. (2011, April). Review of dynamic query optimization strategies in distributed database. In *2011 3rd International Conference on Electronics Computer Technology* (Vol. 6, pp. 145-149). IEEE. https://doi.org/10.1109/ICECTECH.2011.5942069

[6] Bachhav, A., Kharat, V., & Shelar, M. (2017). Query optimization for databases in cloud environment: a survey. *International Journal of Database Theory and Application, 10*(6), 1-12. https://doi.org/10.14257/ijdta.2017.10.6.01

[7] Bachhav, A., Kharat, V., & Shelar, M. (2018). Novel Architecture of an Intelligent Query Optimizer for Distributed Database in Cloud Environment. *Journal of Advanced Database Management & Systems*, 5(2), 28-32.

[8] Bachhav, A., Kharat, V., & Shelar, M. (2018). Processing Distributed Internet of Things Data with Query Optimization in Cloud. *International Journal of Research and Analytical Reviews (IJRAR), 6*(1), 122-124 https://doi.org/10.14257/ijdta.2017.10.6.01

[9] Lee, R., Zhou, M., & Liao, H. (2007, September). Request Window: an approach to improve throughput of RDBMS-based data integration system by utilizing data sharing across concurrent distributed queries. In *Proceedings of the 33rd International Conference on Very Large Data Bases* (pp. 1219-1230).

[10] Chen, G., Wu, Y., Liu, J., Yang, G., & Zheng, W. (2011). Optimization of sub-query processing in distributed data integration systems. *Journal of Network and Computer Applications, 34*(4), 1035-1042. https://doi.org/10.1016/j.jnca.2010.06.007

[11] Garofalakis, M. N. & Ioannidis, Y. E. (1996). Multi-dimensional resource scheduling for parallel queries. *ACM SIGMOD Record, 25*(2), 365-376. https://doi.org/10.1145/235968.233352

[12] Dokeroglu, T., Bayir, M. A., & Cosar, A. (2015). Robust heuristic algorithms for exploiting the common tasks of relational cloud database queries. *Applied Soft Computing*, 30, 72-82. https://doi.org/10.1016/j.asoc.2015.01.026

[13] Giannikis, G., Alonso, G., & Kossmann, D. (2012). SharedDB: killing one thousand queries with one stone. In *Proceedings of the VLDB Endowment, 5*(6), 526-537. https://doi.org/10.14778/2168651.2168654

[14] Agrawal, P., Kifer, D., & Olston, C. (2008). Scheduling shared scans of large data files. *VLDB Endowment, ACM, 2008*. https://doi.org/10.14778/1453856.1453960

[15] Bruno, N., Jain, S., & Zhou, J. (2013). Continuous cloud-scale query optimization and processing. In *Proceedings of the VLDB Endowment, 6*(11), 961-972. https://doi.org/10.14778/2536222.2536223

[16] Cole, R. L. & Graefe, G. (1994, May). Optimization of dynamic query evaluation plans. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data* (pp. 150-160). https://doi.org/10.1145/191843.191872

[17] Graefe, G. & Ward, K. (1989, June). Dynamic query evaluation plans. In *Proceedings of the 1989 ACM SIGMOD international conference on Management of data* (pp. 358-366). https://doi.org/10.1145/66926.66960

[18] Kabra, N. & DeWitt, D. J. (1998, June). Efficient mid-query re-optimization of sub-optimal query execution plans. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data* (pp. 106-117). https://doi.org/10.1145/276305.276315

[19] Markl, V., Raman, V., Simmen, D., Lohman, G., Pirahesh, H., & Cilimdzic, M. (2004, June). Robust query processing through progressive optimization. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data* (pp. 659-670). https://doi.org/10.1145/1007568.1007642

[20] Stillger, M., Lohman, G. M., Markl, V., & Kandil, M. (2001, September). LEO-DB2's learning optimizer. In *Proceedings of the 27th VLDB Conference* (pp. 19-28).

[21] Wang, C., Arani, Z., Gruenwald, L., & d'Orazio, L. (2018, December). Adaptive Time, Monetary Cost Aware Query Optimization on Cloud Database Systems. In *2018 IEEE International Conference on Big Data* (pp. 3374-3382). IEEE. https://doi.org/10.1109/BigData.2018.8622401

[22] Roy, P., Seshadri, S., Sudarshan, S., & Bhobe, S. (2000, May). Efficient and extensible algorithms for multi query optimization. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (pp. 249-260). https://doi.org/10.1145/335191.335419

[23] Safaeei, A. A., Kamali, M., Haghjoo, M. S., & Izadi, K. (2007, May). Caching intermediate results for multiple-query optimization. In *2007 IEEE/ACS International Conference on Computer Systems and Applications* (pp. 412-415). IEEE. https://doi.org/10.1109/AICCSA.2007.370914

[24] Laptev, N., Zeng, K., & Zaniolo, C. (2012). Early accurate results for advanced analytics on mapreduce. In *Proceedings of the VLDB Endowment, 5*(10), 1028-1039. https://doi.org/10.14778/2336664.2336675

[25] Agarwal, S., Iyer, A. P., Panda, A., Madden, S., Mozafari, B., & Stoica, I. (2012). Blink and it's done: interactive queries on very large data. In *Proceedings of the VLDB Endowment, 5*(12), 1902-1905. https://doi.org/10.14778/2367502.2367533

[26] Engle, C., Lupher, A., Xin, R., Zaharia, M., Franklin, M. J., Shenker, S., & Stoica, I. (2012, May). Shark: fast data analysis using coarse-grained distributed memory. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (pp. 689-692). https://doi.org/10.1145/2213836.2213934

[27] Theeten, B. & Janssens, N. (2015). Chive: Bandwidth optimized continuous querying in distributed clouds. *IEEE Transactions on cloud computing*, 3(2), 219-232. https://doi.org/10.1109/TCC.2015.2424868

[28] Olston, C., Chiou, G., Chitnis, L., Liu, F., Han, Y., Larsson, M., et al. (2011, June). Nova: continuous pig/hadoop workflows. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data* (pp. 1081-1090). https://doi.org/10.1145/1989323.1989439

[29] Logothetis, D., Olston, C., Reed, B., Webb, K. C., & Yocum, K. (2010, June). Stateful bulk processing for incremental analytics. In *Proceedings of the 1st ACM symposium on Cloud computing* (pp. 51-62). https://doi.org/10.1145/1807128.1807138

[30] Lang, W., Nehme, R. V., & Rae, I. (2015). Database Optimization in the Cloud: Where Costs, Partial Results, and Consumer Choice Meet. In *7th Biennial Conference on Innovative Data Systems Research (CIDR '15)* (pp. 1-8).

[31] Goldstein, J. & Larson, P. Å. (2001). Optimizing queries using materialized views: a practical, scalable solution. *ACM SIGMOD Record, 30*(2), 331-342. https://doi.org/10.1145/376284.375706

[32] Scheuermann, P., Shim, J., & Vingralek, R. (1996). Watchman: A data warehouse intelligent cache manager. In *Proceedings of the 22nd VLDB Conference* (pp. 1-12).

[33] Kotidis, Y. & Roussopoulos, N. (1999). DynaMat: a dynamic view management system for data warehouses. *ACM Sigmod Record, 28*(2), 371-382. https://doi.org/10.1145/304181.304215

[34] Ivanova, M. G., Kersten, M. L., Nes, N. J., & Gonçalves, R. A. (2010). An architecture for recycling intermediates in a column-store. *ACM Transactions on Database Systems (TODS), 35*(4), 1-43. https://doi.org/10.1145/1862919.1862921

[35] Elghandour, I., & Aboulnaga, A. (2012, May). ReStore: reusing results of MapReduce jobs in pig. *In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (pp. 701-704). https://doi.org/10.1145/2213836.2213937

[36] Olston, C., Reed, B., Srivastava, U., Kumar, R., & Tomkins, A. (2008, June). Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (pp. 1099-1110). https://doi.org/10.1145/1376616.1376726

[37] Lim, H., Herodotou, H., & Babu, S. (2012). Stubby: A transformation-based optimizer for mapreduce workflows. In *Proceedings of the VLDB Endowment, 5*(11), 1196-1207. https://doi.org/10.14778/2350229.2350239

[38] Perez, L. L., et al. (2014, March). History-aware query optimization with materialized intermediate views. In *2014 IEEE 30th International Conference on Data Engineering* (pp. 520-531). IEEE. https://doi.org/10.1109/ICDE.2014.6816678

[39] Tan, K. L., Goh, S. T., & Ooi, B. C. (2001, April). Cache-on-demand: Recycling with certainty. In *Proceedings 17th International Conference on Data Engineering* (pp. 633-640). IEEE.

[40] Phan, T. & Li, W. S. (2008, April). Dynamic materialization of query views for data warehouse workloads. In *2008 IEEE 24th*

International Conference on Data Engineering (pp. 436-445). IEEE. https://doi.org/10.1109/ICDE.2008.4497452

[41] Kossmann, D., Franklin, M. J., Drasch, G., & Ag, W. (2000). Cache investment: integrating query optimization and distributed data placement. *ACM Transactions on Database Systems (TODS), 25*(4), 517-558. https://doi.org/10.1145/377674.377677

[42] Anyanwu, K., Kim, H., & Ravindra, P. (2012). Algebraic optimization for processing graph pattern queries in the cloud. *IEEE Internet Computing, 17*(2), 52-61. https://doi.org/10.1109/MIC.2012.22

[43] Wu, S., Li, F., Mehrotra, S., & Ooi, B. C. (2011, October). Query optimization for massively parallel data processing. In *Proceedings of the 2nd ACM Symposium on Cloud Computing* (pp. 1-13). https://doi.org/10.1145/2038916.2038928

[44] Condie, T., Conway, N., Alvaro, P., Hellerstein, J. M., Elmeleegy, K., & Sears, R. (2010, April). MapReduce online. In *Nsdi, 10*(4).

[45] See http://zql.sourceforge.net/

[46] See http://www.tpc.org/tpch/

Authors' contacts:

Archana Bachhav, Assistant Professor
(Corresponding author)
Department of Computer Science,
KSKW Arts, Science and Commerce College,
Jawahar Road Trimbykeshwar Ta: Tryambakeshwar,
Nashik, Maharashtra 422212, India
Savitribai Phule Pune University,
Ganeshkhind Road, Pune, Maharashtra 411007, India
77.archana@gmail.com

Vilas Kharat, Senior Professor
School of Mathematical and Computing Sciences,
Savitribai Phule Pune University,
Pune, India
laddoo1@yahoo.com

Madhukar Shelar, Associate Professor
Department of Computer Science,
KRT Arts, BH Commerce and AM Science (KTHM) College,
Nashik, India
mnshelar70@gmail.com