A study of PosDB Performance in a Distributed Environment

George Chernishev
Saint-Petersburg State University,
JetBrains Research
Email: chernishev@gmail.com

Vyacheslav Galaktionov
Saint-Petersburg State University
Email: viacheslav.galaktionov@gmail.com

Valentin Grigorev
Saint-Petersburg State University
Email: valentin.d.grigorev@gmail.com

Evgeniy Klyuchikov
Saint-Petersburg State University
Email: evgeniy.klyuchikov@gmail.com

Kirill Smirnov
Saint-Petersburg State University
Email: kirill.k.smirnov@math.spbu.ru

***Abstract*—PosDB is a new disk-based distributed column-store relational engine aimed for research purposes. It uses the Volcano pull-based model and late materialization for query processing, and join indexes for internal data representation. In its current state PosDB is capable of both local and distributed processing of all SSB (Star Schema Benchmark) queries.**

Data, as well as query plans, can be distributed among network nodes in our system. Data distribution is performed by horizontal partitioning.

In this paper we experimentally evaluate the performance of our system in a distributed environment. We analyze system performance and report a number of metrics, such as speedup and scaleup. For our evaluation we use the standard benchmark — the SSB.

I. Introduction

Column-stores have been actively investigated for the last ten years. Many open-source [1], [2], [3], [4], [5] and commercial [6], [7], [8] products with different features and aims have been developed. The core design issues such as compression [9], [10], materialization strategy [11], [12] and result reuse [13] got significant attention. Nevertheless, distribution of data and control in disk-based column-store systems was not studied at all.

The reason for this is that none of open-source systems are truly distributed, although some of them [5] support mediator-based [14] distribution. Several commercial systems, such as Vertica [6], are distributed but closed-source. To the best of our knowledge, no investigation of distribution aspects in column-stores has been conducted.

To address this problem, we are developing a disk-based distributed relational column-store engine — PosDB. In its current state it is based on the Volcano pull-based model [15] and late materialization. Data distribution is supported in the form of horizontal per-table partitioning. Each fragment can be additionally replicated on an arbitrary number of nodes, i.e. our system is partially replicated [16]. Control (query) distribution is also supported: parts of a query plan can be sent to a remote node for execution.

In our earlier studies [17], [18] we have described opportunities offered by such a system and sketched its design. Later, an initial version of our system, PosDB, was presented and its high-level features were described [19].

In this paper, we present the results of first distributed experiments with PosDB. We evaluate system performance by studying several performance metrics, namely speedup and scaleup. For evaluation we use a standard OLAP benchmark — the Star Schema Benchmark [20].

The paper is structured as follows. The architecture of the system is described in detail in section II. A short survey of distributed technology in databases is presented in section I. In section III we discuss used metrics (scaleup and speedup). The experimental evaluation and its results are presented in section IV.

II. Related Work

There is a shortage of distribution-related studies for relational column-oriented databases [18]. The main reasons are the scarcity of research prototypes and the drawbacks in the existing ones.

Two research prototypes of distributed column-store systems are known to the authors — [5], [21]. Both studies use an in-memory DBMS, MonetDB, some of whose parts were rewritten to add distribution-related functionality. This approach cannot be considered "true" distribution, because, in general, it restricts the pool of available distributed processing techniques. Developers have to take into account the architecture of the underlying centralized DBMS in order to employ it. Unfortunately, the degree of these restrictions is unclear for the aforementioned systems.

Another distributed column-store, the ClickHouse system, is an industrial open-source disk-based system. However, there are two issues with this system. Firstly, it was open-sourced only recently, in 2016, and there are no research papers based on this system, known to the authors. Secondly, it has several serious architectural drawbacks: a very restricted partitioning [22] and issues with distributed joins [23].

At the same time, there are hundreds, if not thousands, of papers on the subject in application to row-stores [16], [14].

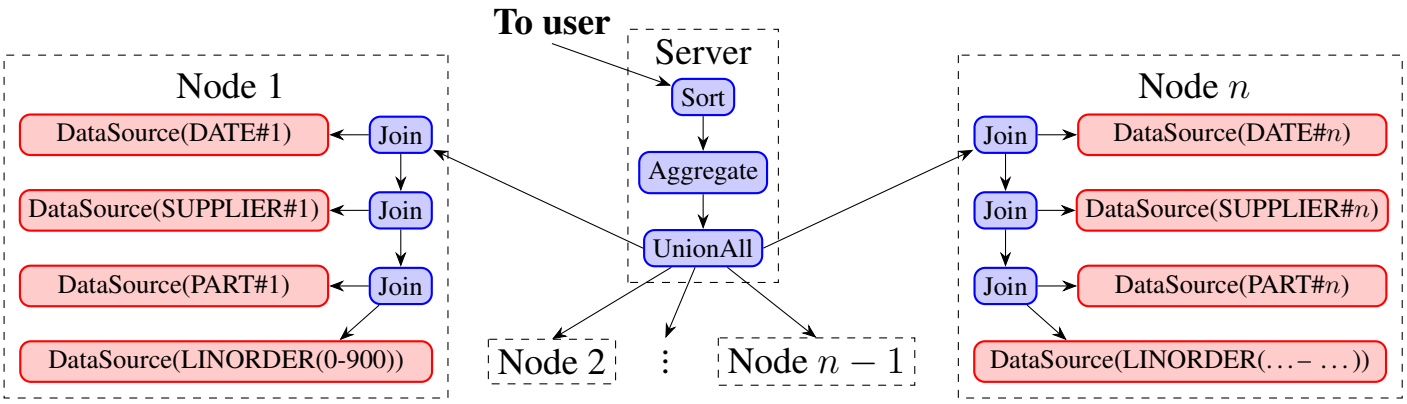

Fig. 1. Example of distributed query plan — distributed query 2.1 from SSB

III. PosDB: Architecture

PosDB uses the Volcano pull-based model [15], so each query plan is represented as a tree with operators as vertexes and data flows as edges. All operators support the "open()–getNext()–close()" interface and can be divided into two groups:

- Operators that produce blocks of positions.
- Operators that produce individual tuples.

PosDB relies on late materialization, so operators of the second type are always deployed on the top of a query tree. They are used to build tuples from position blocks and to perform aggregation. The whole tree below the materialization point consists of operators which return blocks.

Each position block stores several position vectors of equal length, one per table. This structure is essentially a join index [24], [25], which we use to process a chain of join operators.

Currently we have the following operators that produce join indexes:

- `DataSource`, `FilteredDataSource`, operators for creating initial position streams. The former generates a list of contiguous positions without incurring any disk I/O, while the latter conducts a full column scan and produces a stream of positions whose corresponding values satisfy a given predicate. These operators are the only possible leaves of a query tree in our system;
- `GeneralPosAnd`, `SortedPosAnd`, binary operators for the intersection of two position streams related to one table;
- `NestedLoopJoin`, `MergeJoin`, `HashJoin`, binary operators, which implement the join operation in different ways;
- `UnionAll` — an n-ary operator that processes its subtrees in separate threads and merges their output into a single stream in an arbitrary order;
- `ReceivePos` — an ancillary unary operator that sends a query plan subtree to a remote node, receives join indexes from it and returns them to the ancestor;
- `Asynchronizer` — an ancillary unary operator that processes its child operator in a separate thread and stores the results in the internal fixed-size buffer;

and the following that produce tuples:

- `Select`, for tuple reconstruction;
- `Aggregate`, for simple aggregation without grouping;
- `SGAggregate`, `HashSGAggregate`, for complex aggregation with grouping and sorting;
- `SparseTupleSorter`, for tuple sorting.

As can be seen, query distribution is maintained on the operator level using two ancillary operators: `ReceivePos` and `UnionAll`. It should be emphasized, that a multithreaded implementation of UnionAll is essential here, because sequential execution would definitely incur severe waiting penalties, completely negating the benefits of a distributed environment. Figure 1 presents an example of a distributed query plan for the query 2.1 from the SSB which is as follows:

```
select sum(lo_revenue), d_year, p_brand1
from lineorder, date, part, supplier
where lo_orderdate = d_datekey
and lo_partkey = p_partkey
and lo_suppkey = s_suppkey
and p_category = 'MFGR#12'
and s_region = 'AMERICA'
group by d_year, p_brand1
order by d_year, p_brand1;
```

Also, there is a notion of data readers in our system. Data reader is a special entity used for reading attribute values corresponding to the position stream. Currently, we support the following hierarchy of readers:

- `ContinuousReader` and `NetworkReader`, basic readers for accessing a local or remote partition respectively;
- `PartitionedDataReader`, an advanced reader for accessing the whole column, whose partitions are stored on one or several machines. For each partition it creates a corresponding basic reader to perform local or remote full scan. Then, using information from the catalog, a

`PartitionedDataReader` automatically determines which reader to use for a position in a join index;
- `SyncReader`, an advanced reader responsible for synchronous reading of multiple attributes. This reader maintains a `PartitionedDataReader` for each column.

Initially, a query plan does not contain readers. Each operator creates readers on demand and feeds them positions to receive necessary data. Operators that materialize tuples use `SyncReader`, others usually employ `PartitionedDataReader`. Using these advanced readers allows operators to be unaware of data distribution.

IV. General Considerations and Used Metrics

Distributing the DBMS has two important goals [16]: improving performance and ensuring easy system expansion. These goals are usually evaluated using two metrics [26]: scaleup and speedup.

Speedup reflects the dependency of system performance on the number of processing nodes under the fixed workload. Thus, it shows the performance improvement that can be achieved by using additional equipment and without system redesign.

Linear speedup is highly desired but rarely can be achieved in practice. Superlinear speed points out an unaccounted distributed system resources or poor algorithm. So, a good system should try to approximate linear dependency as well as it can.

Scaleup is a similar metric that reflects how easy it is to sustain the achieved performance level under an increased workload. The number of processing nodes and a size of the workload are increased by the same number of times. An ideal system achieves linear scaleup, but again, it is rarely achievable in practice.

Workload can be increased either by increasing the number of queries or the amount of data. The former is the transactional scaleup and the latter is the data scaleup. We do not investigate transactional scaleup, because PosDB is oriented towards OLAP processing — a kind of processing that implies long-running queries. Taniar et al. [26] argue that transactional scaleup is relevant in transaction processing systems where the transactions are small queries. On the other hand, data scaleup is very important for our system because the amount of data in OLAP environments can exhibit feasible growth.

V. Experiments

In order to conduct the experiments, we selected the following setup of data and query distributions. We designate one processing node as a server and assign it several worker nodes. The server only processes user requests, while the data is stored on worker nodes (see Figure 2). Each worker node stores a horizontal partition of the fact table (LINEORDER) along with the replicas of all other (dimension) tables. Dimension tables are always tiny compared to the fact table, so their replication incurs almost no storage overhead.

Figure 1 shows the distributed query for query 2.1 from the workload. It illustrates the general approach which we follow in this paper for each query. The server is responsible for receiving data from worker nodes and for aggregation. Note that all queries in this benchmark can be distributed in such a manner that no inter node (worker node) communication is required.

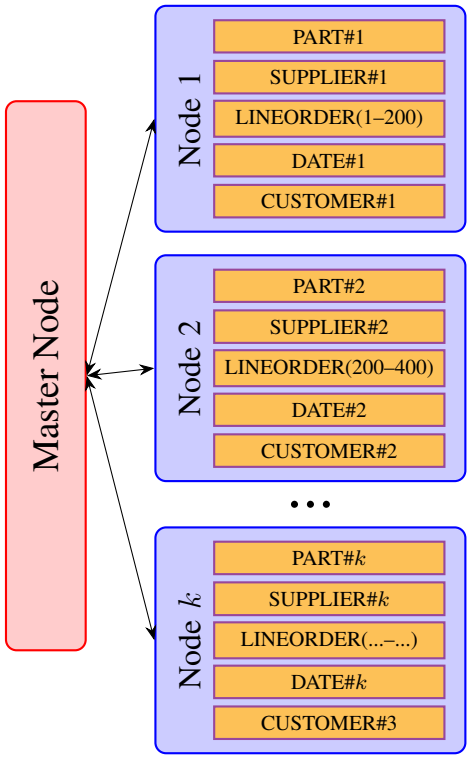

Fig. 2. Data distribution scheme in PosDB

A. Description of Experiments, Hardware and Software Experimental Setups

We consider three different experiments, all using the SSB workload:

1) The dependency of PosDB performance on SSB scale factor in a local (one node) case.
2) The speedup of PosDB, i.e. the dependency of the performance for a fixed workload (scale factor 50) on the number of nodes. The number of nodes includes server and $1, 2, 4, 6, 8$ worker nodes.
3) The scaleup of PosDB, i.e. the performance on $k = 1, 2, 4, 6, 8$ nodes for scale factor $10 * k$ workload.

These experiments are conducted on a cluster of ten machines connected by 1GB local network. Each machine has the following characteristics: Intel(R) Core(TM) i5-2310 CPU @ 2.90GHz (4 cores total), 4 GB RAM. The software used is Ubuntu Linux 16.04.1 (64 bit), GCC 5.4.0, JSON for Modern C++ 2.1.0.

B. Experiment 1

In this experiment we study PosDB behavior in a local case under the full SSB workload. We have chosen six different

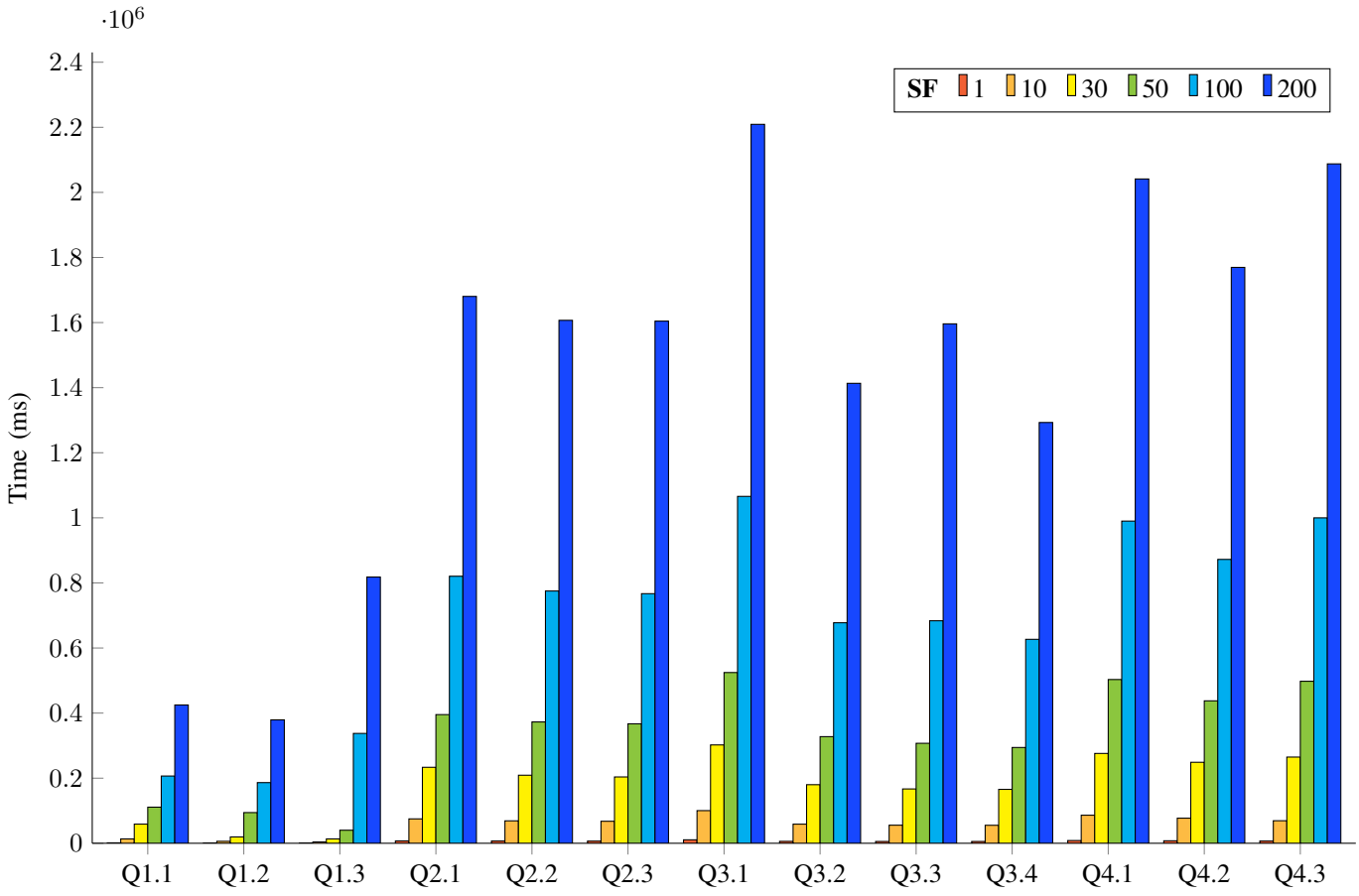

Fig. 3. Query performance from scale factor dependency in local case

scale factors: $1, 10, 30, 50, 100, 200$. The results of this experiment are presented in Figure 3. It should be emphasized that logarithmic y-axis is used here.

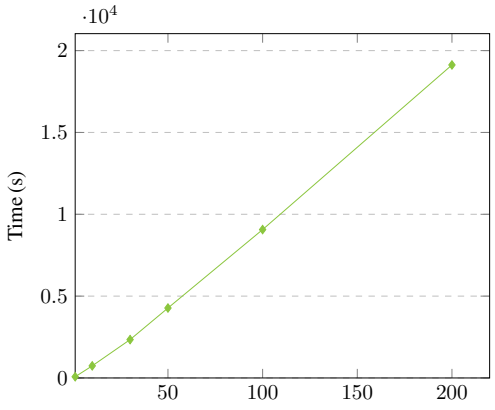

Fig. 4. Dependency of total SSB execution time from scale factor

After careful analysis, several interesting conclusions can be drawn:

- Although query 1.3 has a higher selectivity, its execution time is higher than that of the other queries of its flight. Perhaps it is due to a more expensive aggregation.
- Execution time of the queries from the second flight decreases with the increase in selectivity, as is to be expected.
- Query flight 3 reveals two interesting points. Query 3.1 is much more expensive than the others, because its first join operator returns a significantly higher number of records, thus loading the rest of the query tree. With high scale factors, query 3.3 become much more expensive than others. We suppose that it is due to intensive disk usage.
- Queries 4.1 and 4.2 behave in a very similar way, however the last join in the query 4.1 produces more results. This is the reason for the slightly extended run times for the whole query. Also, there is an anomaly in query 4.3 which still has to be explained. We plan to explore it in our further studies.

The total time of the whole workload is presented in Figure 4. In order to obtain this graph we summed up the run times of all queries described in the SSB. Essentially, this graph is just another representation of the information presented in Figure 3.

C. Experiment 2

You can see the results of the second experiment in Figure 5. Starting with 1, the number of nodes is increased by 2 with

each step. The contents of the LINEORDER table (about 11 GBs) are evenly partitioned and distributed across them. Other tables are fully replicated. The red line shows how much faster the queries are executed when the number of nodes increases. The green line represents the "ideal" case, where the speedup grows linearly.

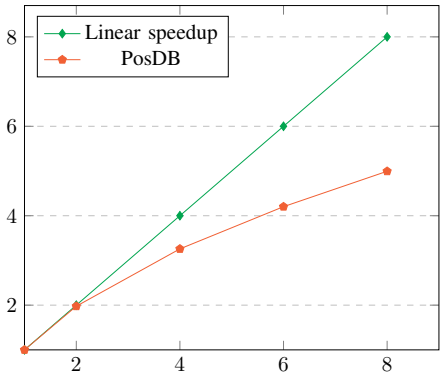

Fig. 5. Speedup from number of servers dependency in PosDB

As you can see, PosDB's performance increases when new nodes are added, although not linearly, which is because our system is yet in its infancy. We believe that such high overhead can be written off on the lack of a proper buffer manager, which means that the same data may be transferred over the network many times.

D. Experiment 3

In this experiment we measured PosDB data scaleup under scale factors $10, 20, 40, 60, 80$ on $1, 2, 4, 6, 8$ nodes. Data and query for each test configuration are distributed similar to the experiment 2. LINEORDER is partitioned between nodes, other tables are fully replicated. Then, parts of query plan that lie below aggregation (or tuple construction) are sent to different nodes, each with a DataSource operator for the corresponding LINEORDER partition. See Figures 2 and 1 for more details.

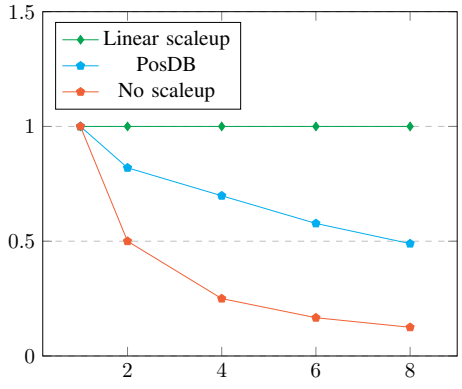

Fig. 6. Data scaleup from number of servers dependency in PosDB

We consider scaleup as $\frac{(server+1\,machine)executiontime}{(server+k\,machines)executiontime}$ relation and present the results in Figure 6. To estimate the PosDB scaleup, we also plotted the "linear scaleup" and "no scaleup" cases. The former is a situation when scaleup is constant (ideal value) during all experiments. In the "no scaleup" case we assume that the amount of data grows linearly, but the computing power remains constant, so scaleup is $1/(number\,of\,machines)$.

We can see that PosDB scaleup is in $[0.5, 0.75]$ boundaries, slowly decreasing with the number of servers growing. Thus, comparing to the case "no scaleup," we can conclude that our system can offer a good scale-up.

VI. Conclusion

In this paper we presented an evaluation of PosDB, our distributed column-store query engine. We used the Star Schema Benchmark — a standard benchmark used for evaluation of OLAP systems. We studied several performance metrics, such as speedup and scaleup. In our experiments we were able to achieve scale factor 200 on a single machine, our system demonstrated sublinear speedup and a good data scaleup. The evaluation also allowed us to discover some anomalies and bottlenecks in our system. They are the subject of our future research.

References

[1] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, P. O'Neil, A. Rasin, N. Tran, and S. Zdonik, "C-store: A column-oriented dbms," in *Proceedings of the 31st International Conference on Very Large Data Bases*, ser. VLDB '05. VLDB Endowment, 2005, pp. 553–564. [Online]. Available: http://dl.acm.org/citation.cfm?id=1083592.1083658

[2] S. Idreos, F. Groffen, N. Nes, S. Manegold, K. S. Mullender, and M. L. Kersten, "Monetdb: Two decades of research in column-oriented database architectures," *IEEE Data Eng. Bull.*, vol. 35, no. 1, pp. 40–45, 2012. [Online]. Available: http://sites.computer.org/debull/A12mar/monetdb.pdf

[3] "Google. supersonic library," https://code.google.com/archive/p/supersonic/, 2017, acessed: 12/02/2017.

[4] J. Arulraj, A. Pavlo, and P. Menon, "Bridging the archipelago between row-stores and column-stores for hybrid workloads," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16, 2016, pp. 583–598. [Online]. Available: http://db.cs.cmu.edu/papers/2016/arulraj-sigmod2016.pdf

[5] Y. Zhang, Y. Xiao, Z. Wang, X. Ji, Y. Huang, and S. Wang, *ScaMMDB: Facing Challenge of Mass Data Processing with MMDB*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–12. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-03996-6_1

[6] A. Lamb, M. Fuller, R. Varadarajan, N. Tran, B. Vandiver, L. Doshi, and C. Bear, "The vertica analytic database: C-store 7 years later," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1790–1801, Aug. 2012. [Online]. Available: http://dx.doi.org/10.14778/2367502.2367518

[7] M. Zukowski and P. Boncz, "From x100 to vectorwise: Opportunities, challenges and things most researchers do not think about," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12. New York, NY, USA: ACM, 2012, pp. 861–862. [Online]. Available: http://doi.acm.org/10.1145/2213836.2213967

[8] D. Abadi, P. Boncz, and S. Harizopoulos, *The Design and Implementation of Modern Column-Oriented Database Systems*. Hanover, MA, USA: Now Publishers Inc., 2013.

[9] D. Abadi, S. Madden, and M. Ferreira, "Integrating compression and execution in column-oriented database systems," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '06. New York, NY, USA: ACM, 2006, pp. 671–682. [Online]. Available: http://doi.acm.org/10.1145/1142473.1142548

[10] V. Raman, G. Attaluri, R. Barber, N. Chainani, D. Kalmuk, V. KulandaiSamy, J. Leenstra, S. Lightstone, S. Liu, G. M. Lohman, T. Malkemus, R. Mueller, I. Pandis, B. Schiefer, D. Sharpe, R. Sidle, A. Storm, and L. Zhang, "Db2 with blu acceleration: So much more than just a column store," *Proc. VLDB Endow.*, vol. 6, no. 11, pp. 1080–1091, Aug. 2013. [Online]. Available: http://dx.doi.org/10.14778/2536222.2536233

[11] D. J. Abadi, D. S. Myers, D. J. DeWitt, and S. Madden, "Materialization strategies in a column-oriented DBMS," in *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, 2007, pp. 466–475. [Online]. Available: http://dx.doi.org/10.1109/ICDE.2007.367892

[12] L. Shrinivas, S. Bodagala, R. Varadarajan, A. Cary, V. Bharathan, and C. Bear, "Materialization strategies in the vertica analytic database: Lessons learned," in *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, April 2013, pp. 1196–1207.

[13] M. G. Ivanova, M. L. Kersten, N. J. Nes, and R. A. Gonçalves, "An architecture for recycling intermediates in a column-store," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '09. New York, NY, USA: ACM, 2009, pp. 309–320. [Online]. Available: http://doi.acm.org/10.1145/1559845.1559879

[14] D. Kossmann, "The state of the art in distributed query processing," *ACM Comput. Surv.*, vol. 32, no. 4, pp. 422–469, Dec. 2000. [Online]. Available: http://doi.acm.org/10.1145/371578.371598

[15] G. Graefe, "Query evaluation techniques for large databases," *ACM Comput. Surv.*, vol. 25, no. 2, pp. 73–169, Jun. 1993. [Online]. Available: http://doi.acm.org/10.1145/152610.152611

[16] M. T. Ozsu, *Principles of Distributed Database Systems*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007.

[17] G. Chernishev, *New Trends in Databases and Information Systems: ADBIS 2015 Short Papers and Workshops, BigDap, DCSA, GID, MEBIS, OAIS, SW4CH, WISARD, Poitiers, France, September 8-11, 2015. Proceedings*. Cham: Springer International Publishing, 2015, ch. Towards Self-management in a Distributed Column-Store System, pp. 97–107. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-23201-0_12

[18] ——, "The design of an adaptive column-store system," *Journal of Big Data*, vol. 4, no. 1, p. 21, 2017. [Online]. Available: http://dx.doi.org/10.1186/s40537-017-0069-4

[19] G. Chernishev, V. Grigorev, V. Galaktionov, E. Klyuchikov, and K. Smirnov, *PosDB: a Distributed Column-Store Engine (paper submitted)*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017.

[20] "P. E. ONeil, E. J. ONeil and X. Chen. The Star Schema Benchmark (SSB)." http://www.cs.umb.edu/~poneil/StarSchemaB.PDF, 2009, acessed: 20/07/2012.

[21] Y. Liu, F. Cao, M. Mortazavi, M. Chen, N. Yan, C. Ku, A. Adnaik, S. Morgan, G. Shi, Y. Wang, and F. Fang, *DCODE: A Distributed Column-Oriented Database Engine for Big Data Analytics*. Cham: Springer International Publishing, 2015, pp. 289–299. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-24315-3_30

[22] "A migration Yandex ClickHouse. A transcript of a talk at Highload++ 2016, http://www.highload.ru/2016/abstracts/2297.html," https://habrahabr.ru/post/322620/, 2017, acessed: 30/04/2017.

[23] "A comparison of in-memory databases." http://www.exasol.com/site/assets/files/3147/a_comparison_of_in-memory_databases.pdf, 2017, acessed: 30/04/2017.

[24] Z. Li and K. A. Ross, "Fast joins using join indices," *The VLDB Journal*, vol. 8, no. 1, pp. 1–24, Apr. 1999. [Online]. Available: http://dx.doi.org/10.1007/s007780050071

[25] D. Tsirogiannis, S. Harizopoulos, M. A. Shah, J. L. Wiener, and G. Graefe, "Query processing techniques for solid state drives," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '09. New York, NY, USA: ACM, 2009, pp. 59–72. [Online]. Available: http://doi.acm.org/10.1145/1559845.1559854

[26] D. Taniar, C. H. C. Leung, W. Rahayu, and S. Goel, *High-Performance Parallel Database Processing and Grid Databases*, A. Zomaya, Ed. Wiley Series on Parallel and Distributed Computing, 2008.