Data Science and Engineering (2020) 5:80–93
https://doi.org/10.1007/s41019-020-00116-2

Clusterix-Like BigData DBMS

Vadim A. Raikhlin[1] · **Roman K. Klassen**[1]

Received: 21 June 2019 / Revised: 2 February 2020 / Accepted: 10 February 2020 / Published online: 20 February 2020

Abstract
Commercial OLAP systems are economically unavailable for organizations with limited financial capabilities. Analytical processing of large amounts of data in these organizations can be accomplished using open-source software systems on a cost-effective cluster platform. Previously created Clusterix-like DBMS using a regular query processing plan is not efficient enough. Therefore, research on such systems was developed with a focus on a full load of processor cores and using the GPU acceleration (systems Clusterix-N, N—from new) up to the development of a system comparable in efficiency to the open-source system Spark, which is currently considered the most promising. The development methodology was based on the constructive system modeling methodology.

Keywords Clusterix-like DBMS · Databases of significant volumes · BigData · Regular query processing plan · Full load of processor cores · GPU acceleration · Constructive modeling of systems

1 Introduction and Solvable Problem

Earlier, we already presented the results of our research on the topic of this article [21] as part of the report at the conference. The report included only the results of the work without a detailed review. Now, we give the necessary explanations, because this is one of the most important functions of science. In addition, this article presents new results of iteration 5 (Sect. 5.6).

Database volumes of hundreds GB or more are not uncommon for relatively small businesses with limited financial capabilities. Acquisition of cost-effective computing clusters and specialized software of conservative (with an occasional update of data) by such organizations makes it possible for them to timely process the accumulated data. For conservative DBMSs, OLAP load [6, 29] is typical, and it is characterized by a high weight of complex queries such as "selection–projection–connection," which operates with a set of tables with numerous connection operations. Developments in this direction are under way. Commercial DBMSs have high performance and reliability, but are too expensive. For example, MS SQL Server 2016 DBMS [3, 17] on the one Lenovo x3950 X6 server [14] has a total system cost of $2 634 342 ($1.5 million for server + $1 million for software). Oracle Database [18] with an extension for OLAP and a license for 384 cores will cost $9 million. Plus the cost of hardware (Exadata) is $1.5 million.

A good alternative to expensive parallel DBMS in the field of BigData is freely distributed open-source systems Hadoop [8, 30] and Spark [15, 31, 32]. Both systems have high performance and are well scaled, and their hardware platform requirements are quite modest. This makes Hadoop and Spark very promising systems for analytical processing of large data sets with MapReduce technique [27].

The typical relational DBMS architecture by Stonebraker and Helerstein [11] includes five main components (Fig. 1):

1. Client communication manager, including information exchange protocols for local and remote clients.
2. A process control manager that performs the functions of a dispatcher and a processing scheduler.
3. Transaction manager—access, block, log and data buffer management.
4. Common components and utilities: batch utilities, replication and load services, administration and monitoring utilities, directory and memory managers.
5. Relational query processor

✉ Roman K. Klassen
klassen.rk@gmail.com

Vadim A. Raikhlin
no-form@evm.kstu-kai.ru

[1] Kazan National Research Technical University named after A. N. Tupolev - KAI, Kazan, Russian Federation

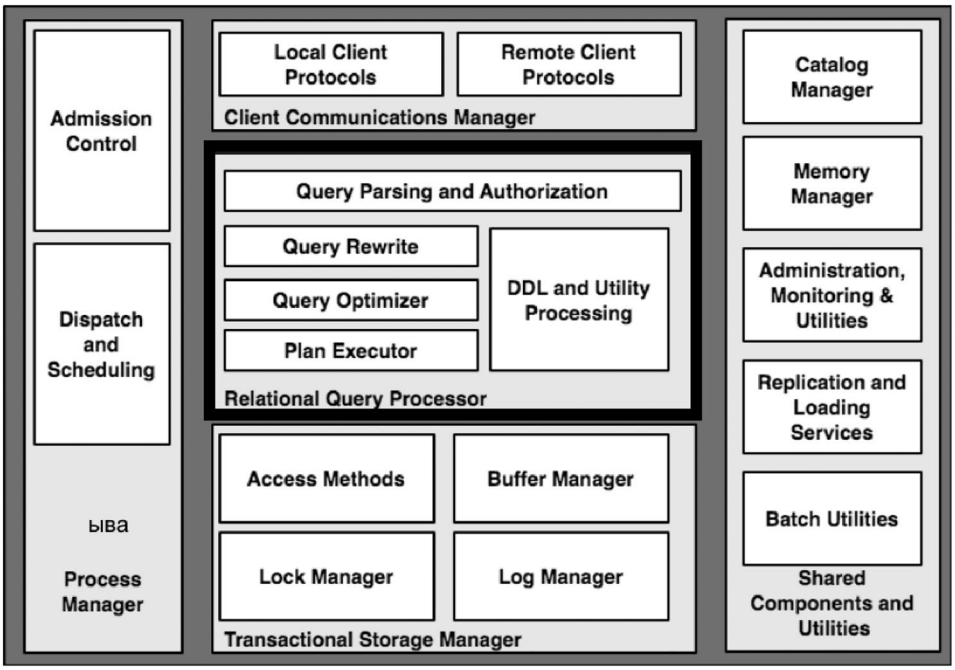

Fig. 1 The typical relational DBMS architecture

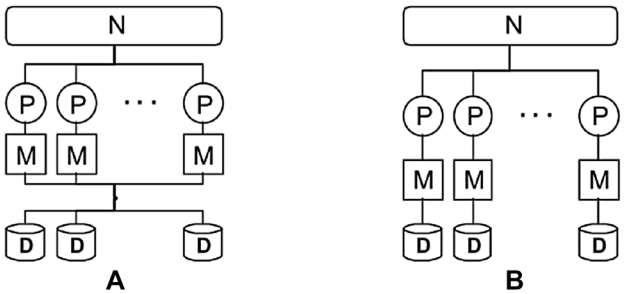

Fig. 2 Stonebraker classification (**a** SD, **b** SN)

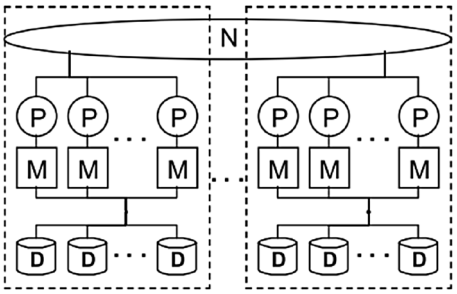

Fig. 3 Stonebraker classification extension

This architecture is suitable for single-node sequential DBMS and is actively used by them. For example, this architecture is applied to each node on developing Clusterix-N system, where the highlighted black is an instrumental DBMS MySQL. The architecture of parallel DBMSs is significantly different from single-node/sequential: Network communication between modules is added, process management is performed on multiple nodes, and utilities are used by various modules as needed, but in general the components are the same, presented in a slightly different (extended) interpretation. Below are shown the parallel architectures *SD*, *SN* and *CD* used in the work.

In [28], Stonebraker proposed a classification of parallel DBMSs by the distribution of data across disks, memory and processors. A schematic image of the classification is presented in Fig. 2. In the figure, P—processor, M—memory, D—disk, and N—data transmission network.

In accordance with this classification, parallel DBMS is divided into the following base classes depending on the division of hardware resources:

- *SD (Shared-Disks)*—shared disk architecture.
- *SN (Shared-Nothing)*—architecture without sharing resources.

Copeland and Keller [7] proposed an extension of the Stonebraker classification by introducing additional classes of architectures of parallel database machines (Fig. 3):

- *CD (Clustered-Disk)*—architecture with SD clusters, united by the principle of SN. The boundary of the SD clusters in Fig. 3 is extended to the common (global) connecting network, as they may have their own (local) connecting network.

For conservative DBMS, the most important is the case of processing the flow of queries translated to scheme

$$\text{SELECT}(\sigma) - \text{PROJECT}(\pi) - \text{JOIN}(\sigma_\theta(R \, x \, S)).$$

Here, $\langle x \rangle$ is the Cartesian product. Selection in a join operation is performed according to θ-matching the tuples of the R and S relations. Development of a parallel DBMS is desirable to accomplish from the condition of implementing a stream-pipelined method for query processing. It is not easy to fulfill such a condition, because it implies an ideal balance of all parts of the pipeline. But if we assume that acceptable balancing is achievable, then the choice of a regular plan (tree) (Fig. 4) for processing queries [22] is valid.

An algebraic expression representing a query to a relational database, written in terms of "x," "σ," "π," is always reducible to this tree. During the SQL queries pretranslation to a regular plan, subqueries *select-project*, *join* and *sort* (perform aggregation operations (SUM(), AVG(), MAX(), MIN(), etc.) and sort the result) are formed.

When using the strategy "many cluster nodes—for one query," the database is distributed across nodes. Obtaining any intermediate R_i' and any temporary R_{Tj} relations occurs in parallel on the IO and JOIN processors. At the same time, it is theoretically possible to combine both processes if during the preprocessing (selection and projection) of the initial relation R_i, the relation $R_{T(i-2)}$ is formed, which is the basis for the implementation of a balanced pipeline with an acceptable duration of its stages. These are the motives of our hypothesis:

Regular query processing plan is preferred for conservative parallel DBMSs and large database volumes.

But the early created research versions of the Clusterix-like systems were ineffective. It was necessary to look for ways to improve their efficiency. The objective of this work is analyzing possibilities of developing economical conservative high-volume DBMSs comparable in efficiency (by performance/cost criterion) with the Spark system while processing a query flow to a database with data amounts of hundreds and more GB on relatively inexpensive cluster platforms using a regular query processing plan, and also using MySQL and GPU accelerators at the executive level. MySQL allows you to use different "engines" and has an extension system [19]. These features simplify and speed up system development compared to using PostgreSQL.

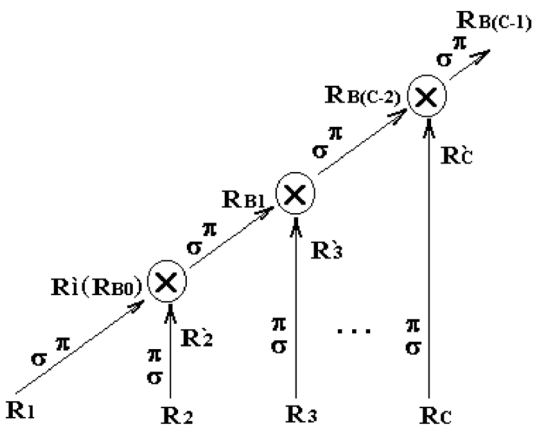

Fig. 4 Regular plan

2 Accepted Limitations

They are dictated by the requirement of the economy:

1. The hardware platform of the studied DBMSs is computing clusters assembled from supplied components by firms.
2. Cluster SMP nodes—two processor nodes, equipped with MySQL instrumental DBMS and Linux/Windows operating system.
3. Processors in the nodes—serial with the number of processor cores not more than 8.
4. It is allowed to connect to nodes via PCI-e bus GPU accelerators with the number of cores not more than 512.
5. Communication network between nodes—GigabitEthernet (10 GigabitEthernet/Infiniband—if possible).
6. Disk subsystem—SATA (SAS—if possible).
7. The amount of RAM in the node—no more than 512 GB.
8. The hashed database is fully hosted in the aggregate RAM of all cluster nodes.
9. The considered DBMS is multi-user systems with batch query processing.

Accordingly, all the research experiments were carried out on the GPU cluster platform consisting of seven nodes. Node parameters: 2 six-core E5-2640 CPU/2.5 GHz/DDR3 128 GB; 2 448-core Tesla GPU C2075/1.15 GHz/GDDR5 6 GB (no GPU MGM). Node disk subsystem—RAID 10 of 4 WD1000DHTZ/1 TB total volume (minus RAID "mirror") 2 TB. The operating system is Windows Server 2012 R2. Interconnect between nodes—GigabitEthernet with 24-port switch SSE G24-TG4. The volume of DB—120 GB. The representative test (RT) is the concatenation of six permutations of the TPC-H throughput test without write operations.

In the experiment with Spark, the database was presented in the form of structured text files and was evenly distributed over six execution nodes. Data access was implemented using Hadoop (HDFS). Load balancing was performed by the YARN module, also part of Hadoop. The query processing was performed by Spark in the configuration "worker per core" (total $6 \times 12 = 72$ workers per cluster). Queries were launched without any changes and optimizations. The

Spark spark-sql extension was responsible for working with SQL queries, which performed parsing and optimization of the original query.

3 The Methodology Used for Solving the Problem

The basis of the research was adopted methodology CSM—constructive system modeling [23]. Cardinal questions of synthesis under *incomplete information* conditions are:

- WHERE (in which area of some space) to find the right solution?
- HOW (by what methods) to organize such a search?
- WHY exactly there and so?

The methodological basis of the CSM consists of the following provisions:

1. It is assumed that the object being synthesized models the behavior of a certain hypothetical system—something of a single whole, infinitely knowable and explicable, given by its purpose operator. Modeling of this system is treated as ***S***-modeling synthesis process (***S***—from synthesis). Under the process in cybernetics, we mean the sequential change of states of some object. Therefore, the model being developed is not a static formation, but a dynamically developing (*evolving*) system, each state of which corresponds to a certain quality of modeling. Development is stopped by obtaining the required quality. As a result, we obtain the desired *constructive method*. This is the rationale for the adopted name—*constructive system modeling*.
2. The properties that a device must possess in order for ***S***-modeling to be sufficiently effective can be revealed in the dynamics of ***S***-modeling in the form of *postulates that state sufficiently proved ideas*. The ***S***-modeling process is considered as a multi-step iterative process, in which both explanatory and informative premises (postulates as elements of the theory) and the constructive method itself (realization of an acceptable **S**-model iteration) are complementary. The system of postulates must be open for corrections. The initiation of postulates is advisable, only if the development of a constructive method based on them shows its prospects for its time, and the method itself does not fit into the framework of the existing theory.
3. The ultimate goal of ***S***-modeling is to develop a theoretically justified constructive method, i.e., synthesis procedure. Formally, the S-modeling process includes two stages—*external modeling* (postulating a mathematical ***S***-model as a relevant description—frame, logical, algebraic or others—of an oriented sequence of a complete set of solutions areas—answers to questions: WHERE? And WHY?) and *internal* (an iterative study of the found ***S***-model in order to develop a constructive method—the answer to the question: HOW?).
4. There are ***S***-models: *unitary* (***US***-models) and *hierarchical* (***IS***-models). The ***US***-model is a single abstract image (a single search area), for example a local area of a certain metric space. The preference of such a description of the systems is undoubted. ***IS***-model is a set of representations of the hierarchical system. It is built when a single abstract image of the system cannot be found. Systemic balancing of model index values at all levels of the hierarchy is achieved in the process of internal modeling.

Larger systems are typically described as hierarchical ***IS***-models. The process of ***IS***-modeling should not take too much time, as is typical of natural evolution. Therefore, in such a process, a mathematical (external) model should be found as the minimum set of states (areas) in the space of all possible states of the ***IS***-model, the transitions between which form the *shortest path* to obtain the desired result. Algorithmic and software development of each state is the subject of internal modeling.

Among the ***IS***-models is also DBMS with hierarchy levels: *select-project*, *join*, *sort*, dynamic segmentation of relations, their indexing, network, etc. In this case, the assignment operator of a hypothetical system is set by the condition of obtaining high-efficiency query processing with the minimum system cost determined by the previously formulated restrictions. The state of the ***IS***-model is the architecture of the software system as a set of interacting software modules. Its name will be associated with some characteristic feature, and the full software development will be called the full state.

IS-modeling is never carried out on the "empty place." From the space of complete states, we can go to the parameter space. Under the parameter, we will understand the average processing time of a single request of the RT at a particular level. For a given platform, there is a unique mapping of the space of complete states into the space of parameters (we leave the question of mutual uniqueness open), in which we will carry out the consideration. By analogy with that adopted in synergetics [10], for each complete state we will single out the so-called rank parameter, minimizing the effect of which on system performance will determine transitions between iteration states.

The parameter having the maximum value for a given full state is taken as the "rank parameter." But the functioning of all levels in a large system is interconnected (system unity principle). Therefore, reducing the influence of "rank parameter" on system performance inevitably leads to a

change in the influence of other levels as well. The number of iterations is usually relatively small if the quality criterion of the final solution is acceptable (in this case—obtaining efficiency comparable to the efficiency of the Spark system).

4 Accepted Postulates

In the process of ***IS***-modeling, a system of postulates was formulated as a declaration of expedient directions for the development of the desired models.

POSTULATE 1. The solution of the problem should ensure the evolution of Clusterix-like DBMS from the initial implementation of the principles of hybrid technology (see below the accepted initial state of the ***IS***-model).

POSTULATE 2. The search for the next states (iterations) of the Clusterix-like DBMS ***IS***-model should be carried out in the way of replacing the "core for one relation" strategy adopted for its initial state with the strategy "node group (cores) for one relation." This is necessary to ensure the reliable operation of an effective system with significant volumes of databases and requires dynamic segmentation of relations, which can be both concentrated and distributed.

POSTULATE 3. Internal ***IS***-modeling of Clusterix-like systems should be carried out in the directions determined by the external (mathematical) synthesis process frame model shown in Fig. 5. In the figure:

CONSTR—constraint frame.
BA—frame of the adopted block architecture consisting of five program blocks: IO (DB data access module, executes *select-project* subqueries), JOIN (*join* subquery processing module), MGM (control module), SORT (query final processing module) and HASH (implements dynamic segmentation relations; it is not in the initial state).
Clusterix-N (N—from new)—a development frame for Clusterix-like systems.
HT w/o DS—frame of the initial state of the ***IS***-model (transition to the hybrid technology of Clusterix-like systems without dynamic segmentation of relations).
w/DS—Clusterix-N frame with dynamic relations segmentation.
CON DS HT—frame (w/DS) systems with concentrated dynamic segmentation of relations in the framework of the hybrid technology (the first iteration of the ***IS***-modeling).
DIS DS—frame (w/DS) systems with distributed dynamic segmentation.
DIS DS HT—frame (DIS DS) systems implemented using hybrid technology (the second iteration of ***IS***-modeling).
DIS DS CS—frame (DIS DS) systems in the "combined symmetry" configuration (the third iteration of the ***IS***-modeling).
DIS DS CC—frame (DIS DS) systems in the "combined core" configuration.
DIS DS CC DSM—frame (DIS DS CC) systems in the "database server for module" configuration (the fourth iteration of the ***IS***-simulation).
DIS DS CC MDS—frame (DIS DS CC) systems in the "many database servers" configuration (the fifth iteration of the ***IS***-simulation).

5 *IS*-Modeling

5.1 Characteristic of the Initial State

In the first Clusterix-like systems, dynamic segmentation of intermediate and temporal relations was used to speed up the join operations as the individual records R'_i and R_{Tj} were formed. It took a lot of time and, with an increase in the number of nodes, could lead to system malfunctions. In [24], it was shown that the performance can be improved by moving to the Clusterix-N architecture by abandoning the

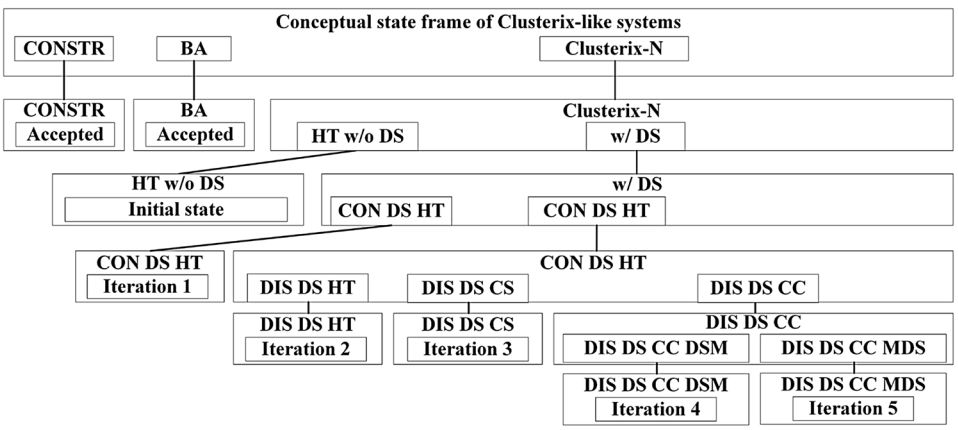

Fig. 5 External frame synthesis process model of Clusterix-like systems

principle of "homogeneity" (typical for the optimal Clusterix configuration "combined symmetry" [25]) in favor of "hybridity," which implies a cluster into two different parts—IO and JOIN blocks—with independent variation of the nodes number in each block. It was the reason for the choice of the initial state.

The database was hashed at the IO node level. They implemented the "core for one relation" strategy. At the level of JOIN nodes, the "query to the core" strategy was used (the feasibility of such a strategy using MySQL was shown in [13]), which made it possible to exclude dynamic segmentation of intermediate and temporal relations and perform the join as a single procedure

$$R1' \text{ join } (\text{join } R2' \ (\text{join } R3' \ (\ldots))) \ \ldots)$$

for each query. This is much faster than its consistent implementation and leads to a significant increase in efficiency compared to Clusterix.

Detailed program development of the initial state allowed us to create peculiar "billet modules," which were further modified for each new state. The presence of such "blanks" greatly facilitated the carry out of subsequent iterations, where they were modified accordingly. These are subsystems of statistics collection, visualization and journaling; network interaction module; DBMS driver; MGM module as the core of the system; IO, JOIN and SORT modules; the way to pretranslating query to the regular plan; configuring MySQL for maximum load on all node processor cores.

The effectiveness of the initial state was significantly lower than that of Spark. In addition, even with database volumes < 100 GB, a large total volume of intermediate relations for some queries of the TPC-H test led to an overload of JOIN nodes RAM and, as a result, to loss of the DBMS performance. *Unreliability* is a "rank parameter" for the initial state.

5.2 The First Iteration of *IS*-Modeling

Reliable work with a database of hundreds GB and more requires switching to the strategy "set of cores in each block for relation," which requires dynamic segmentation. Therefore, it is restored in the first iteration of Clusterix-N, but (unlike Clusterix) with the segments transfer as a whole. Distributing data across all processor cores of the JOIN level is implemented by the HASH module on a dedicated node with GPU accelerators. Hashing is performed using the division algorithm [16]. The result of the hash is placed in the send buffer for the cores. (HASH module forms a buffer in its memory for each core in the JOIN nodes.) Sending data occurs when the hash operation is ready.

As a result of the changes made, the Clusterix-N program now consists of five modules: MGM, IO, JOIN, HASH and SORT. As before, the database is distributed over the IO nodes. The IO and JOIN modules implement the "node group for relation" strategy. Cluster configuration for the first iteration experiment: two IO nodes, three JOIN nodes, one HASH node and one MGM node, what combining MGM and SORT modules.

The principal feature of this iteration (and further) is the pipeline-cyclic execution of *select-project* and *join* operations for each request (the presence of internal pipelining plus the external). But now Clusterix-N remains uncompetitive. The results of the experiments for the processing time RT are as follows: Clusterix-N—19.7 h; Spark—4.5 h. They are clearly not in favor of Clusterix-N.

The rank parameter revealed as a result of the first iteration is determined by the histogram in Fig. 6 (ordinate—time in seconds). It is time contribution of the *network layer*.

5.3 The Second Iteration of *IS*-Modeling

The main idea underlying the second iteration, and hoping for success, is to implement dynamic segmentation of intermediate/temporal relations (hashing) in the IO and JOIN modules with transferring the hashed data directly between the execution nodes (bypassing the MGM). The rejection of the dedicated HASH hashing node and the transfer of its functionality to the execution nodes (using GPU accelerators to hash data) should speed up the data transfer process (due to a decrease in the amount of data transferred). The hash implementation developed for the HASH module has been adapted and transferred to the IO and JOIN software modules. The organization of the system operation mode with direct data transfer between execution nodes required changes in the MGM module.

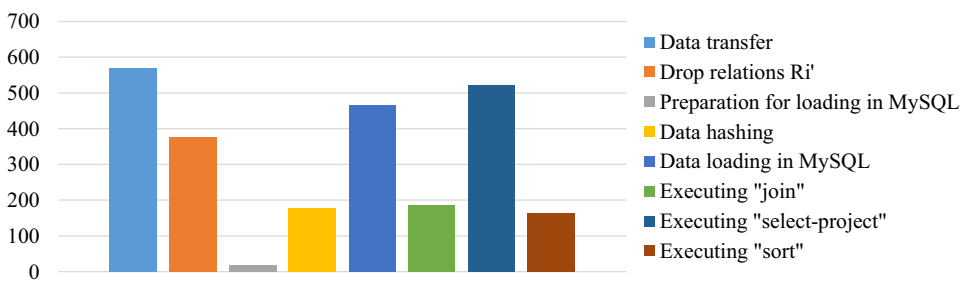

Fig. 6 The average processing time for a single query of the RT by levels (iteration 1)

The IO module performs a *select-project* operation for one relation in parallel on the set of available processor cores to produce a set of result blocks. These blocks are subjected to GPU-accelerated hashing and are transferred immediately to certain JOIN nodes. Block-by-block selection allows us to combine three operations in time: After one block formation, the next select operation is started, the result is transferred to the hash queue, and the hashed blocks are sent to the transmission queue.

JOIN module completely repeats the algorithm of its work in the first iteration. The only difference is in the processing of the *join* result. Now, it is hashed on the GPU and passed to the JOIN nodes (to perform the next to *join* operation) or SORT with overlapping operations, similar to IO. The SORT module uses the "core for query" strategy and transmits the result to MGM. The only change in his work is getting data from JOIN nodes, not from BUF MGM.

An experimental study of the software-implemented new version of Clusterix-N was made in the configuration of a GPU cluster: two IO nodes, four JOIN nodes and one MGM node, what combining MGM and SORT modules. The database is distributed over the IO nodes. Analysis of the experimental results showed that the transmission time over the network decreased ~ 3 times, and the join operations accelerated ~ 1.5 times (due to adding one more node and reducing the amount of data in each node). And yet, the time of the RT is 14.5 h, i.e., the total processing time of the RT decreased by only ~ 26% compared with the previous modification of Clusterix-N. This is clearly not enough to talk about possible competition with Spark.

The rank parameter for iteration 2 is determined by the histograms in Fig. 7. In this case, this is the execution time of operations at the *select-project* level.

5.4 The Third Iteration of *IS*-Modeling

The easiest way to speed up operations at a specified level is to reduce the amount of data processed in one node, which requires an increasing number of nodes in each block. With an unchanged total number of cluster nodes, the desired effect can be achieved by returning (on the new turn of "spiral") to the "combined symmetry" configuration [25], which involves placing two modules on one node at once: IO and JOIN.

An experiment in this configuration was carried out with the following distribution of nodes of a GPU cluster: six nodes with IO and JOIN modules, one MGM node combining MGM and SORT modules. The database is distributed over six nodes. For each module are allocated one CPU (six cores) and one GPU accelerator. On all nodes (except MGM), two MySQL DBMSs are functioning at once: one for IO and second for JOIN. The use of two MySQL DBMS due to the SMP node architecture and different DBMS configuration is: for IO, the configuration is aimed at optimizing the work with *select-project* queries, and for JOIN—at optimizing the work with the MEMORY engine and *join* operations.

The obtained histograms are shown in Fig. 8.

The results of the experiment in comparison with Spark are presented in Table 1, where T—execution time of the entire RT, M—average waiting time for a response to a query, and σ—standard deviation.

The fact that Clusterix-N is significantly inferior to the Spark system in values of M and σ is important to the user. A comparison of the times of individual operations execution for Clusterix-N averaged over a set of RT queries is presented in Table 2. As follows from Table 2 and from the

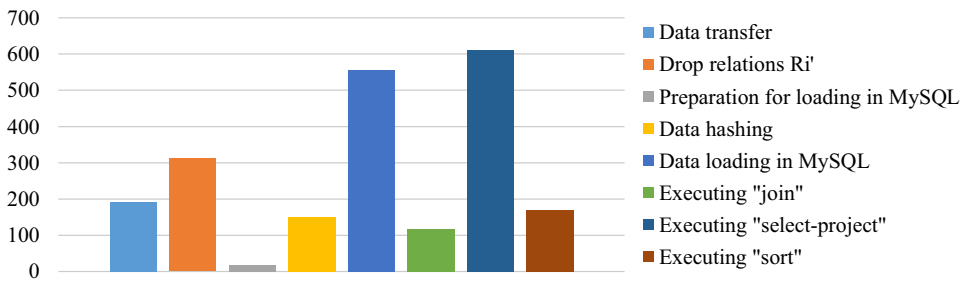

Fig. 7 The average processing time for a single query of the RT by levels (iteration 2)

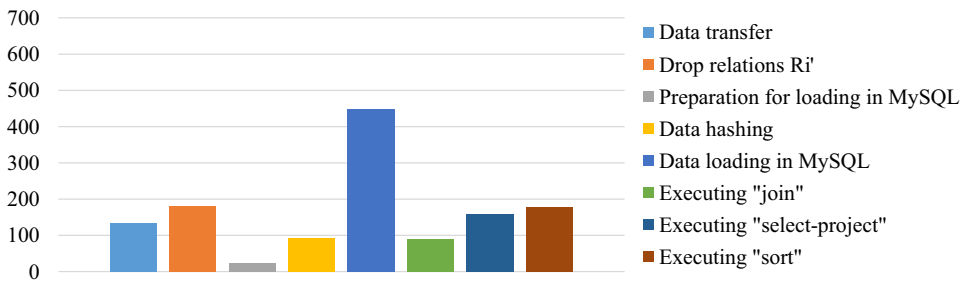

Fig. 8 The average processing time for a single query of the RT by levels (iteration 3)

Table 1 Results for the "combined symmetry" configuration versus spark

	Clusterix-N	Spark	Clusterix-N/spark ratio
T (min)	455.8	260.6	1.75
M (min)	14.3	3.1	4.61
σ (min)	15.7	0.9	17.44

histogram in Fig. 8, in the "combined symmetry" configuration, the longest operation was *loading data in MySQL*. This is the notorious "rank parameter" for the third iteration.

The dynamics of the processes in this iteration are illustrated in Fig. 9.

5.5 The Fourth Iteration of *IS*-Modeling

Data loading in MySQL for JOIN modules can be accelerated by increasing the number of cores on which these modules are work. It can be noticed (see Fig. 9) that IO modules are far from always busy at processing queries, because one of the CPUs at each node is idle for a long time. Would not it be better to work consistently with *select-project* and *join* operations on the same core and load all cores of the cluster with these operations? In this case, the IO modules will work without downtime, and the load operations in MySQL will be significantly accelerated, which should reduce the values of M and σ. But would such a violation of external pipelining reduce efficiency?

To obtain an answer to this question, a preliminary study was carried out with minimal system modifications. Each node uses one GPU for hashing the results of the IO and JOIN modules. The operations of the SORT module are accelerated by changing the MySQL engine from MyISAM to MEMORY.

The experimentally obtained histograms are shown in Fig. 10. They meet the data in Tables 3 and 4. They confirm the predictions and concerns made. Despite the serious acceleration of the load operation in MySQL, with a decrease in M and σ, the processing time of the RT increased by $\sim 23\%$ compared with iteration 3.

This is the fee for a partial violation of pipelining. Further acceleration of the select-project and a number of other operations can slightly improve the situation. With an unchanged platform, the efficiency of iteration 4 can be improved, firstly, by switching to IO from database hashing by nodes to hashing by cores; this should speed up the *select-project*, and secondly the transition to a more advanced version of MySQL 8.0. It can be expected that its use will speed up the execution of a number of other operations. The software implementation of such transitions was associated with modifications of database driver and IO module.

The results of the experiment illustrate the histogram in Fig. 11 and the data in Tables 5 and 6. Now, the estimates for Clusterix-N and Spark are comparable to a greater degree.

5.6 The Fifth Iteration of *IS*-Modeling

In the article [21], we found that performance can be improved by developing specialized engines for MySQL. This is necessary to speed up relation drop and loading operations. The main reason for the slowness of these operations is table locking. But in [2], it is written that the MEMORY engine performs table-level locking. That is, when performing data modification operations (INSERT, UPDATE, DELETE, ALTER and others), the table is denied access until the named operations are completed. In Clusterix-N, these operations are performed in the "query for table" mode, i.e., locks should not occur. But the experimental data of four iterations suggest the opposite.

A detailed analysis of the MySQL source codes helped to establish that the MEMORY engine storage is presented as heap and is shared to the entire MySQL process. Therefore, locking a single table causes blocking of the entire memory of this engine in a single process. There are two ways around this limitation. First, make such changes to the MEMORY engine that would allow you to lock within the table or remove it altogether. Secondly, run multiple instances of MySQL in the number of CPU cores.

Table 2 The average execution time of individual operations in Clusterix-N

	Iteration 1 α (s)	Iteration 2 β (s)	Iteration 3 γ (s)	$\frac{\alpha}{\beta}$	$\frac{\beta}{\gamma}$	$\frac{\alpha}{\gamma}$
Data transfer	569.77	190.82	133.62	2.99	1.43	4.26
Drop relations R'_i, R_{Tj}	375.27	311.87	180.03	1.20	1.73	2.08
Preparation for loading in MySQL	18.90	19.01	22.40	0.99	0.85	0.84
Data hashing	177.03	151.05	93.61	1.17	1.61	1.89
Data loading to MySQL	466.64	555.30	447.28	0.84	1.24	1.04
Executing "join"	186.09	116.06	89.09	1.60	1.30	2.09
Executing "select-project"	522.73	611.32	159.86	0.86	3.82	3.27
Executing "sort"	162.61	168.97	178.42	0.96	0.95	0.91

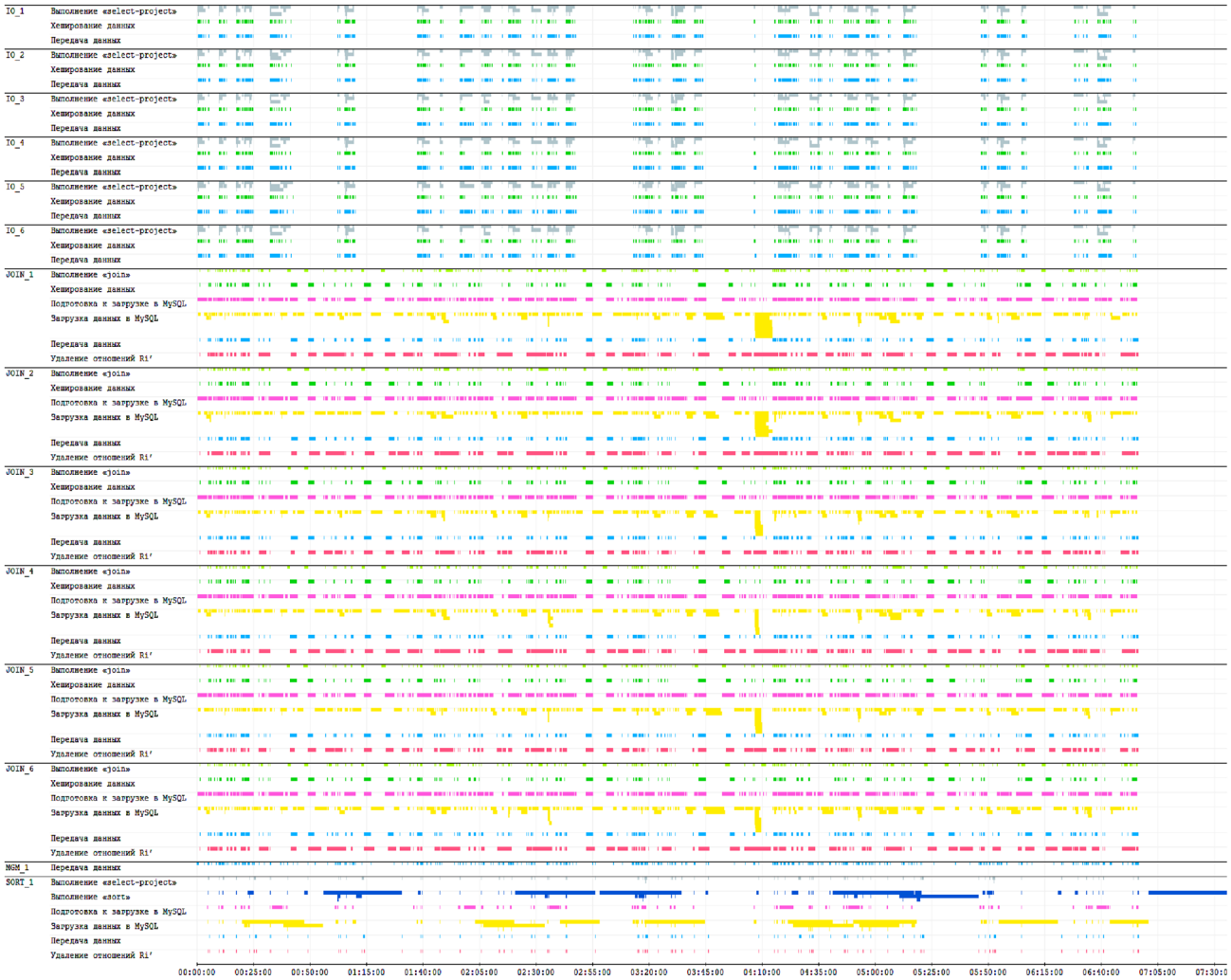

Fig. 9 Visualization of RT execution (iteration 3)

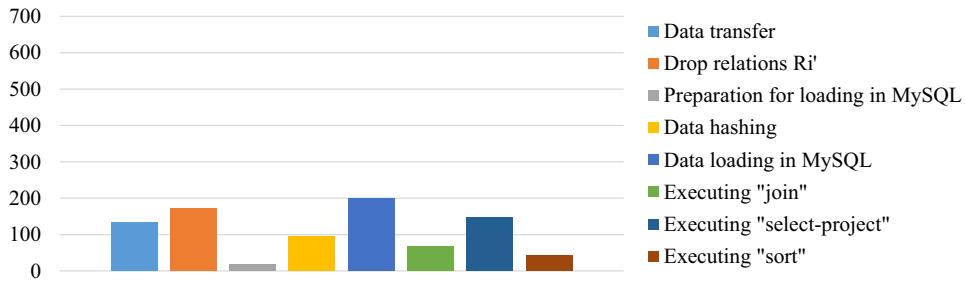

Fig. 10 Histograms for the start of iteration 4

The first option is extremely time-consuming and can damage the stability of the system. The second option is simple and can be implemented without modifying the instrumental DBMS (MySQL), so this option is preferable for us. To implement it, we need to make a number of changes to the operation of Clusterix-N.

Working with multiple MySQL will require establishing many Clusterix-N ↔ MySQL connections, controlling the distribution of hashed result data, and replicating queries in multiple connected MySQL. Some of these changes were implemented in fourth iteration for IO modules during the transition to hashing by cores at the IO level. Transferring these developments to other modules (JOIN and SORT) with a slight improvement made it possible to perform hashing in several instrumental DBMSs. This allows Clusterix-N to work with multiple MySQL on a single host.

Table 3 Comparative evaluation of accelerations for iteration 3 and start of 4

	Iteration 3, γ (s)	Start of iteration 4 δ (s)	$\frac{\gamma}{\delta}$
Data transfer	133.62	133.36	1.00
Drop relations R'_i, R_{Tj}	180.03	170.47	1.06
Preparation for loading in MySQL	22.40	17.80	1.26
Data hashing	93.61	94.62	0.99
Data loading in MySQL	447.28	198.75	2.25
Executing "join"	89.09	66.83	1.33
Executing "select-project"	159.86	148.17	1.08
Executing "sort"	178.42	43.03	4.15

Table 4 Comparative estimates for T, M and σ for iteration 3 and start of 4

	Iteration 3, γ (min)	Start of iteration 4 δ (min)	$\frac{\gamma}{\delta}$
T	455.80	560.60	0.81
M	14.30	8.20	1.74
σ	15.70	6.40	2.45

The experiment for this iteration was carried out as follows. On each node launched 13 MySQL servers: one for IO and 12 for JOIN. MySQL for IO uses the InnoDB engine, which does not have the problems described at the beginning of this section, but it is not applicable for loading data in real time, since this operation takes a long time. The experimental design is shown in Fig. 12.

The results of the experiment are presented in Tables 7 and 8. From Table 7, it is seen that the operations of "Drop relations" and "Data loading in MySQL" were significantly accelerated: 3.4 and 1.9 times, respectively. The average execution time for other operations has not changed significantly. Table 8 shows the parity of Clusterix-N and Spark on test processing time. But Clusterix-N is still significantly inferior to Spark in parameters M and σ.

Table 5 The estimation of accelerations at the transition from the start of iteration 4 to its end

	Start of iteration 4 δ (s)	Iteration 4 θ (s)	$\frac{\delta}{\theta}$
Data transfer	133.36	106.12	1.26
Drop relations R'_i, R_{Tj}	170.47	140.91	1.21
Preparation for loading in MySQL	17.80	24.15	0.74
Data hashing	94.62	56.43	1.68
Data loading in MySQL	198.75	164.01	1.21
Executing "join"	66.83	54.77	1.22
Executing "select-project"	148.17	55.40	2.67
Executing "sort"	43.03	42.90	1.00

Table 6 The final comparison on T, M and σ for iteration 4 and Spark

	Iteration 4 γ (min)	Spark δ (min)	$\frac{\gamma}{\delta}$
T	403.8	260.6	1.55
M	6.3	3.1	2.03
σ	5.6	0.9	6.22

The histograms from the experimental results (Fig. 13) show that the "rank parameter" for the fifth iteration is data transmission.

6 Conclusion

The paper shows that using a regular query processing plan with appropriate architectural and software-algorithmic development of cost-effective, conservative, high-performance BigData class DBMS shows results comparable with the best open systems. The cost of the acquisition and commissioning of a GPU cluster similar to that used in the comparative experiments will be no more than $85 000. All versions of the Clusterix-N software system are placed in open access [12] and can be used by interested organizations.

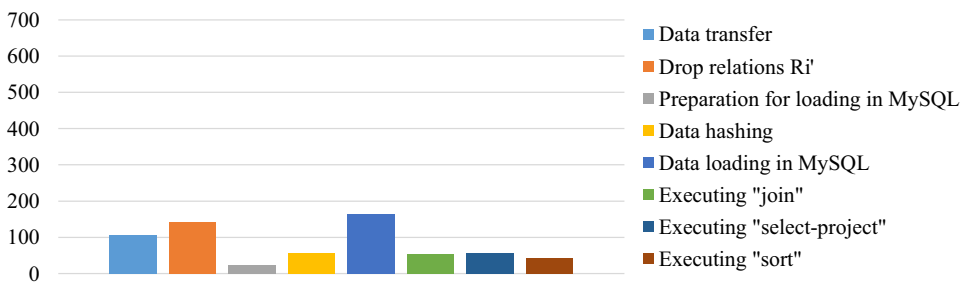

Fig. 11 Histograms for the final version of iteration 4

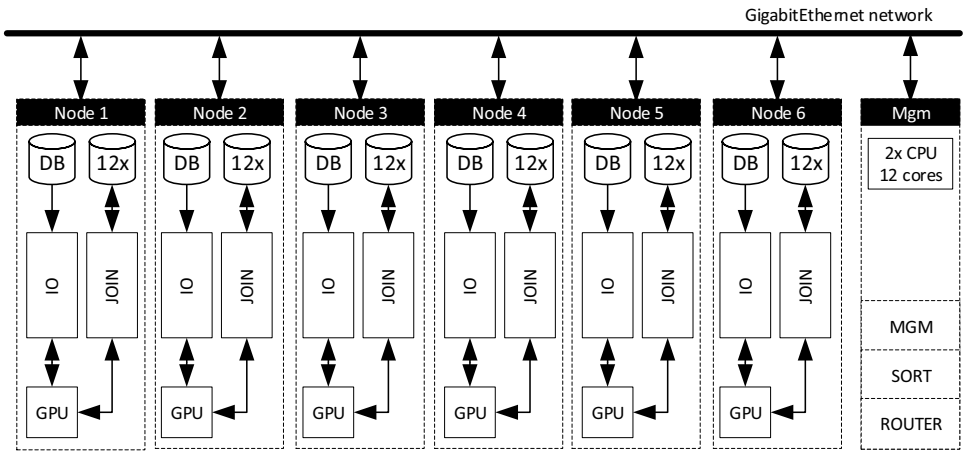

Fig. 12 Experiment design of iteration 5

Table 7 The estimation of accelerations for iterations 4 and 5

	Iteration 4 θ (s)	Iteration 5 ε (s)	$\frac{\theta}{\varepsilon}$
Data transfer	106.12	116.17	0.91
Drop relations R'_i. R_{Tj}	140.91	41.06	3.43
Preparation for loading in MySQL	24.15	25.29	0.95
Data hashing	56.43	56.14	1.01
Data loading in MySQL	164.01	85.18	1.93
Executing "join"	54.77	26.76	2.05
Executing "select-project"	55.40	59.93	0.92
Executing "sort"	42.90	43.41	0.99

Table 8 The comparison on T, M and σ for iterations 4 and 5 and Spark

	Iteration 4 γ (min)	Iteration 5 ε (min)	Spark δ (min)	$\frac{\gamma}{\varepsilon}$	$\frac{\varepsilon}{\delta}$
T	403.8	276.13	260.6	1.46	1.06
M	6.3	4.8	3.1	1.33	1.54
σ	5.6	4.5	0.9	1.26	4.78

Nevertheless, the Clusterix-N DBMS is significantly inferior to the Spark system in terms of M and σ. As follows from the histograms in Fig. 13, operations remain rather slow (1) data transfer, (2) data loading into MySQL, (3) data hashing, (4) "select-project." Operations (1, 4) can be accelerated by working with compressed databases (see "Appendix"). Operation (2)—development of a specialized MySQL engine. The issue of accelerating operation (4) remains open so far. All this is the subject of special studies.

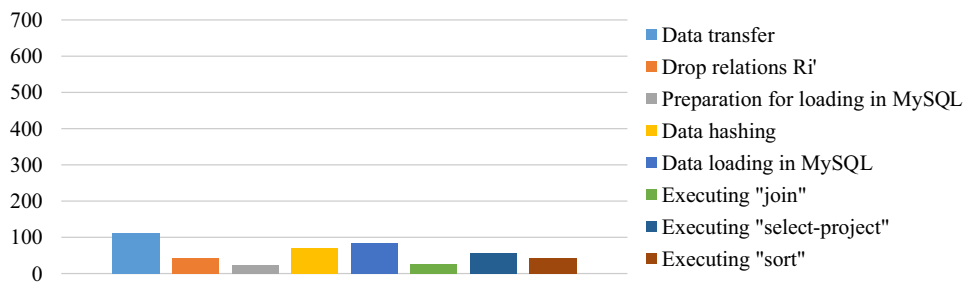

Fig. 13 Experiment design of iteration 5

Appendix: Note on Working with Compressed Databases

With a limited number of nodes, a database of significant volumes may not fit in the RAM of the cluster. Therefore, it is useful to consider the possible organization of work with compressed databases. We show that when storing a compressed database in the memory of the nodes, using the GPU acceleration of the *select* operation can have a serious impact. In this case, we slightly shift the emphasis of [20].

As is known, the use of a GPU allows us to repeatedly reduce the time to perform individual operations [1, 4]. But with significant volumes of DB, the need to exchange CPU ↔ GPU data significantly reduces the performance of whole system. The data transfer rate on the PCI-e bus is significantly lower than the exchange rate with the RAM. So, the read/write speed for three-channel DDR3-1600 RAM is 38.4 GB/s, while for the PCI-e 2.0×16 bus—6.4 GB/s. It is for this reason that the increase in the performance of the database server from using the GPU in processing fairly simple single requests achieved in [26] did not exceed 40%.

Distinguish between DBMS-oriented data storage in rows and columns. The second type DBMS has the best values of compression. This question is studied in [9]. The compressed database is divided into blocks. Farther, the data block means the part of a compressed column (or a set of "short" columns) with decompressed data volume, equal to the decompressed data buffer volume on the GPU. The *select* operations and *dynamic segmentation of relations* are performed by graphics accelerators, and the CPUs at the IO stage are engaged in projecting, promotion of the subquery queue, forming and transmitting compressed blocks to the GPU and obtaining results from them.

The data preparation for compression is next:

1. Find the longest field (length *RS*) in treated ratio *R*.
2. Find in the appropriate column(s) number of records RC, which guaranteed fit in reserved memory, $RC = [BS/RS]$, where *BS*—the memory volume for the decompressed data in a graphics accelerator.
3. Issue from the ratio R data into columns for compression with increments of RC.

The tool has been selected DBMS MySQL 5.6. The experiment was performed on the basis of the computing node with the following characteristics: Quad-core Intel Core i5-4670K CPU/2.5 GHz/24 GB RAM (DDR3-1600 in two-channel mode), 64-bit OS Windows 8, node disk subsystem—SSD SV300S37A/120 GB with a bandwidth of 450 MB/s. The experiment was aimed at comparing times what spent on simple copy, with the sum of times, necessary to copy data, compressed by algorithm RLE (run-length encoding) [5], and their expansion in GPU. The highest efficiency of data transmission was achieved for compression ratio $K = 4 - 5$. Higher compression increases the data compression time, which may exceed the regular copy time.

Table 9 The data transmission efficiency change

Size (MB)	Copy (ms)	Compressed data copying and decompression (ms)		The data transmission efficiency increasing (%)	
		$K = 4$	$K = 5$	$K = 4$	$K = 5$
0.38	1	4	4	300	300
4.20	3	6	8	167	100
8.01	5	9	9	80	80
11.83	7	12	12	71	71
15.64	21	15	15	29	29
19.45	26	20	19	27	23
23.27	31	23	24	23	26
27.08	35	26	27	23	26
30.90	41	30	29	29	27
34.71	45	35	32	29	22
38.53	50	35	35	30	30
42.34	55	38	38	31	31
46.16	59	41	42	29	31
49.97	64	44	44	31	31
53.79	68	48	48	29	29
57.60	75	50	51	32	33
61.42	80	53	53	34	34

In Table 9, it is shown the increasing efficiency percentage of the transmission of the compressed data (K = 4 and 5) with following expansion in comparison with simple copying. As shown in table, during transferring small data volume, simple copying comes out faster. When volumes > 12 MB, precompression provides acceleration in about 30% (approximately 1.5 times).

A qualitative assessment of the effectiveness of the transition to compressed databases with GPU-accelerated *select* operations was obtained as follows. According to the regular processing plan for 14 selected TPC-H test, queries have been generated load modules "*select-project*." Thus, all "project" operations have been "pulled down." The volume of the test database was 1 GB. It is preloaded into main memory. The experiment platform remained the same.

When calculating the amount of data for any of the considered query, defined length of each field of each line of the tables involved in the query processing, and all received lengths summed. Counting was performed by

modification requests with tools of MySQL. Example of query number 3 modification:

```
-- O'
SELECT  O_ORDERDATE,
        O_SHIPPRIORITY,
        O_ORDERKEY,
        O_CUSTKEY
FROM ORDERS
WHERE O_ORDERDATE < DATE '1995-03-31';
-- O' LENGTH
SELECT SUM(LENGTH(O_ORDERDATE),
           LENGTH(O_SHIPPRIORITY),
           LENGTH(O_ORDERKEY),
           LENGTH(O_CUSTKEY))
FROM ORDERS
WHERE O_ORDERDATE < DATE '1995-03-31';
```

`Here, O'`—one of the subqueries "select-project," which obtained after this request pretranslation; `O' LENGTH`—query to calculate the amount of data returned.

Measurement of information amount, which is necessary for processing requests, is given in Table 10. In this table: V_{overall}—the full scope of the relationship needed to execute the query; $(\sigma, \pi)_{\Sigma}$—sum of the volumes of the same relationship after reducing the amount of data using the sampling conditions of the request (section 'where' of sql query). Have, on average, approximately sevenfold reduction of amount of data

Table 10 The data volumes of RT before and after "*select-project*"

Query #	The volume of information for processing		Coefficient of reducing the
	Before sampling	After sampling	
	V_{overall}	$(\sigma, \pi)_{\Sigma}$	$V_{\text{overall}}/(\sigma, \pi)_{\Sigma}$
1	675 846 277	134 255 929	5.03
2	137 651 393	13 526 703	10.18
3	855 794 582	76 991 966	11.12
4	832 798 438	428 049 230	1.95
5	857 126 234	139 324 211	6.15
6	675 846 277	1 339 256	504.64
7	857 125 865	78 759 670	10.88
8	879 261 359	179 338 247	4.90
9	970 449 462	234 598 226	4.14
10	855 796 681	49 964 804	17.13
11	342 555 947	27 528 586	12.44
12	832 798 438	23 140 549	35.99
13	179 948 305	18 706 950	9.62
14	697 981 402	6 548 108	106.59
Total	96 950 980 660	1 412 072 435	6.87

Thus, the total volumes of CPU → GPU transmissions at the IO stage for compressed and uncompressed databases are correlated as 7:5 (40% excess). But the time of selection on the CPU in tens or more times is longer than the same time on the GPU.

When working with compressed databases, the additional use of GPU to speed up the select operation should lead to an increase in the efficiency of the DBMS in comparison with the case of working with the database without compression.

References

1. PGStrom (2016) https://wiki.postgresql.org/index.php?title=PGStrom&oldid=25517. Accessed: 09 May 2018
2. The MEMORY Storage Engine—MySQL 8.0 Reference Manual (2016) https://dev.mysql.com/doc/refman/8.0/en/memory-storage-engine.html. Accessed 03 Dec 2019
3. TPC-H Result Highlights (2016) Lenovo system x3950 X6. http://www.tpc.org/3321. Accessed 09 Aug 2018
4. CoGaDB—Column-oriented GPU-accelerated DBMS (2018) http://cogadb.cs.tudortmund.de/wordpress/ (2018). Accessed 09 May 2018
5. Breß S (2015) Efficient query processing in co-processor-accelerated database. Ph.D. Dissertation, University of Magdeburg
6. Codd E, Codd S, Salley C (1993) Providing olap (on-line analytical processing) to user-analysts: an IT mandate. Codd & Associates. https://books.google.ru/books?id=pt0lGwAACAAJ
7. Copeland G, Keller T (1989) A comparison of high-availability media recovery techniques. ACM SIGMOD Rec 18(2):98–109. https://doi.org/10.1145/66926.66936
8. EMC Education Services (2015) Data science and big data analytics: discovering, analyzing, visualizing and presenting data. Wiley. https://books.google.ru/books?id=J94WBgAAQBA
9. Fang W, He B, Luo Q (2010) Database compression on graphics processors. Proc VLDB Endow 3(1–2):670–680. https://doi.org/10.14778/1920841.1920927
10. Haken H (2004) Synergetics: introduction and advanced topics. Physics and astronomy online library. Springer. https://books.google.ru/books?id=0bc6cLK0w7YC. https://doi.org/10.1007/978-3-662-10184-1
11. Hellerstein JM, Stonebraker M, Hamilton J (2007) Architecture of a database system. Found Trends Databases 1(2):141–259. https://doi.org/10.1561/1900000002
12. Klassen RK (2018) Clusterix-N. https://bitbucket.org/rozh/clusterixn/. Accessed 09 Mar 2019
13. Klassen RK (2018) PerformSys. https://github.com/rozh1/PerformSys/. Accessed 09 Dec 2018
14. Lenovo (2017) System x3950 X6 Rack Server. https://www3.lenovo.com/ru/ru/data-center/servers/mission-critical/System-x3950-X6/p/WMD00000002. Accessed 15 July 2018
15. Li X, Zhou W (2015) Performance comparison of hive, impala and spark sql. In: Proceedings of the 2015 7th international conference on intelligent human-machine systems and cybernetics, volume 01, IHMSC '15. IEEE Computer Society, pp 418–423. https://doi.org/10.1109/IHMSC.2015.95
16. Martin J (1977) Computer database organization, 2nd edn. Prentice Hall PTR, Upper Saddle River
17. Microsoft (2018) Parallel query processing. https://technet.microsoft.com/en-us/library/ms178065(v=sql.105).aspx. Accessed 05 Apr 2018

18. Oracle (2018) Oracle exadata database machine X7. https://www.oracle.com/ru/engineered-systems/exadata/database-machine-x7/index.html. Accessed 10 Aug 2018
19. Oracle (2018) The MySQL plugin API. https://dev.mysql.com/doc/refman/5.7/en/plugin-api.html. Accessed 09 Apr 2018
20. Raikhlin VA, Klassen RK (2017) Can GPU-accelerator significantly increase the effectiveness of conservative DBMS considerable volumes on cluster platforms? In: 2017 international siberian conference on control and communications (SIBCON), pp 1–5. https://doi.org/10.1109/SIBCON.2017.7998474
21. Raikhlin VA, Klassen RK (2019) Constructive modeling of conservative DBMS. In: 2019 international Russian automation conference (RusAutoCon), pp 1–5. https://doi.org/10.1109/RUSAUTOCON.2019.8867678
22. Raikhlin VA (1996) Simulation of distributed database machines. Program Comput Softw 22(2):68–74
23. Raikhlin VA (2005) Methodology of constructive system modeling [Metodologija konstruktivnogo modelirovanija sistem]. Artif Intell News 1:5–17
24. Raikhlin VA, Klassen RK (2018) Relatively inexpensive hybrid technology of large volumes conservative dbms. Inf Technol Comput Syst 68(1):46–59
25. Raikhlin VA, Minjazev RSh (2011) Multiclusterization of distributed dbms of conservative type. Nonlinear World 9(85):473–481
26. Rauhe H (2014) Finding the right processor for the job co-processors in a DBMS. Ph.D. Dissertation, University of Magdeburg
27. Stonebraker M, Abadi D, DeWitt DJ, Madden S, Paulson E, Pavlo A, Rasin A (2010) Mapreduce and parallel DBMSs: friends or foes? Commun ACM 53(1):64–71
28. University MS, Stonebraker M (1986) The case for shared nothing. Database Eng 9:4–9
29. Wikipedia Contributors (2018) Online analytical processing. https://en.wikipedia.org/w/index.php?oldid=850545800. Accessed 08 Oct 2018
30. Wikipedia Contributors (2019) Apache Hadoop. https://en.wikipedia.org/w/index.php?title=Apache_Hadoop&oldid=887023781. Accessed 15 Mar 2019
31. Wikipedia Contributors (2019) Apache Spark. https://en.wikipedia.org/w/index.php?title=Apache_Spark&oldid=887875725. Accessed 15 Mar 2019
32. Xin RS, Rosen J, Zaharia M, Franklin MJ, Shenker S, Stoica I Shark (2013) SQL and rich analytics at scale. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data, SIGMOD '13. ACM, New York, pp 13–24. https://doi.org/10.1145/2463676.2465288