Query Evaluation Techniques for Cluster Database Systems*

Andrey V. Lepikhov and Leonid B. Sokolinsky

South Ural State University, Chelyabinsk, Russia

Abstract. The paper is dedicated to a problem of effective query processing in cluster database systems. An original approach to data allocation and replication at nodes of a cluster system is presented. On the basis of this approach the load balancing method is developed. Also, we propose a new method for parallel query processing on the cluster systems. All described methods have been implemented in "Omega" parallel database management system prototype. Our experiments show that "Omega" system demonstrates nearly linear scalability even in presence of data skew.

1 Introduction

Nowadays, parallel database system applications became more and more widely distributed. Current parallel DBMS solutions are intended to process OLAP queries in petabyte data arrays. For example, DBMS Greenplum based on the MapReduce technology processes six and a half Petabyte of storage for deep analysis on a 96-nodes cluster system in the eBay [1]. DBMS Hadoop handles two and half Petabyte of storage on a cluster system consisting of 610 nodes for popular web-tool facebook. There are several commercial parallel DBMS in the area of parallel OLTP query processing among which the most known are Teradata, Oracle Exadata and DB2 Parallel Edition.

Modern researches in this area follow in a direction of self-tuning DBMS [2], data partitioning and load balancing [3], parallel query optimization [4] and an effective using of modern multicore processors [5].

One of the major objectives in parallel DBMS is load balancing. In the paper [6], it has been shown that a skews appearing in a parallel database management systems with a shared-nothing architecture at processing of a query, can lead to almost complete degradation of a system performance.

In the paper [7], an approach to solve the load balancing problem for parallel database management systems with shared-nothing architecture is proposed. This approach is based on data replication. The given solution allows reducing the network data transfer overhead during load balancing. However this approach is applicable in rather narrow context of spatial databases in a specific segment

* This work was supported by the Russian foundation for basic research (project 09-07-00241-a) and Grant of the President of the Russian Federation for young scientists supporting (project MK-3535.2009.9)

of a range queries. In paper [3], the load balancing problem is solved by partial repartitioning of the database before running a query execution. This approach reduces total amount of data transferred between computing nodes during a query processing, however it demands a very high interprocessor communication bandwidth.

In this paper, we present a new parallel query processing method based on the approach to database partitioning named "partial mirroring". This method solves problem of effective query processing and load balancing in cluster system.

The article is organized as follows. In section 2, the method of parallel query processing in DBMS for cluster system is described. In section 3, a data partitioning strategy for cluster system and algorithm of load balancing are proposed. In section 4, the results of computing experiments are shown and discussed. In conclusion, we summarize the obtained results and discuss the future work directions.

2 Organization of Parallel Query Processing

In relational database management systems with shared-nothing architecture, parallel query processing is based on data fragmentation. In this case, each relation is divided into a number of disjoint horizontal fragments, which are partitioned into different disk units of multiprocessor system. Fragmentation strategy is defined by a fragmentation function ϕ. For each tuple of the relation, this function calculates the cluster node number where this tuple has to be placed. The query is executed on all cluster nodes by *parallel agents* [8]. Each agent processes it's own fragments of the relations and generates partial query result. These partial results are merged in the resulting relation. In general case, it is necessary to perform passing tuples between the cluster nodes in order to obtain the correct result. For managing these communications, we insert the special exchange operator [9] in appropriate points of query execution plan tree.

The exchange operator is defined by a *distribution function* ψ. For each input tuple, this function calculates the number of the cluster node where this tuple has to be processed.

The exchange operator passes tuples between parallel agents by using the communication channels. Each channel is defined by the pair (node number, port number). As the node number we use the parallel agent number and as the port number we use the sequence number of the exchange operator in the query tree.

Let's describe the parallel query processing scheme in a parallel DBMS for cluster systems. We assume that the computing system is a cluster consisting from N of nodes as shown in Figure 1. We suppose the database to be partitioned into all nodes of the cluster system. According to the given scheme, SQL-query processing consists of three stages.

At the first stage, the SQL query is translated into a sequential physical plan. At the second stage, the sequential physical plan is transformed to the parallel plan being a set of parallel agents. It is obtained by inserting the exchange

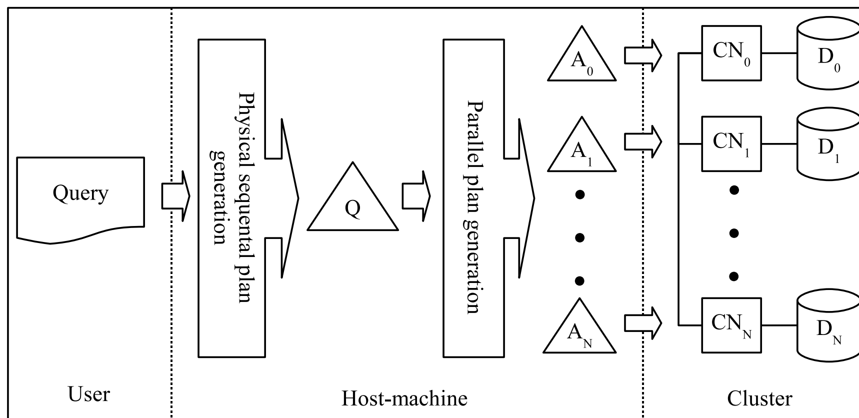

Fig. 1. The scheme of query processing in parallel DBMS for cluster systems. Q – physical query plan, A_i – parallel agent, CN_i – computing node.

operator in appropriate points of the plan tree. All parallel agents have the same structure.

At the third stage, parallel agents are spread among cluster nodes where each of them is interpreted by the query executor. Resulting relations produced by the agents are merged through the root exchange operators on the some dedicated node.

3 Data Placement and Load Balancing

3.1 Data Fragmentation and Segmentation

In the cluster system, we use the following database partitioning [10]. Each relation is divided into disjoint horizontal fragments, which are distributed among cluster nodes. We suppose that tuples in the fragment are ordered in a certain way. This order defines the sequence, in which the tuples are being read by the scan operator. We called this order as *natural order*. In practice, the natural order can be defined by a physical sequence of tuples or by an index.

At the logical level, each fragment is divided into sequence of *segments* with an equal length. The length of the segment is measured in tuples. This is a system parameter. Dividing tuples into segments is performed by the natural order beginning from the first tuple. The last segment of a fragment can be incomplete.

Let's denote the quantity of segments in fragment F as $S(F)$.

$$S(F) = \left\lceil \frac{T(F)}{L} \right\rceil .$$

Here $T(F)$ – number of tuples in fragment F, L – segment length.

3.2 Data Replication

Let fragment F_0 be allocated on disk $d_0 \in \mathfrak{D}$ in the cluster system. We suppose each disk $d_i \in \mathfrak{D}(i > 0)$ contains a *partial replica* F_i, which contains some (may be empty) subset of tuples of fragment F_0.

Segment is the smallest replication unit. The size of replica F_i is determined by the *replication factor*

$$\rho_i \in \mathbb{R}, 0 \leq \rho_i \leq 1,$$

which is a property of replica F_i. It is computed by the formula:

$$T(F_i) = T(F_0) - \lceil (1 - \rho_i) \cdot S(F_0) \rceil \cdot L.$$

The natural order of tuples in replica F_i is determined by the natural order of tuples in fragment F_0. The following formula determines the first tuple number N of replica F_i:

$$N(F_i) = T(F_0) - T(F_i) + 1.$$

If replica F_i is empty, then $N(F_i) = T(F_0)+1$. It corresponds to "end-of-file" position.

3.3 Load Balancing Method

Parallel Agent Work Scheme Let SQL query use n relations. Let $\mathfrak{Q}$ be the parallel plan of the SQL query. Each agent $Q \in \mathfrak{Q}$ has n input streams $s_1, \ldots,$ s_n. Each stream $s_i (i = 1, \ldots, n)$ is determined by the four parameters:

1. f_i – pointer to the fragment;
2. q_i – quantity of segments in part to be processed;
3. b_i – number of the first segment in the part;
4. a_i – load balancing indicator: 1 – load balancing is admitted, 0 – load balancing is not admitted.

An example of the parallel agent with two input streams is presented in Figure 2.

Parallel agent Q can exist in one of the two states: *active* and *passive*. In *active* state, agent Q sequentially scans tuples from the all input streams. Parameters q_i and b_i dynamically changed for all $i = 1, \ldots, n$ during scan process. In *passive* state, agent Q does not perform any actions. At the initial stage of query processing, the agent executes an initialization procedure, which defines all the parameters of the input streams and sets the agent state equal to active.

The agent begins to process fragments associated with its input streams. The agent processes only that part of the fragment, which belongs to the section defined by the parameters of the stream. When all assigned segments in all input streams have been processed, the agent turns into passive state.

Load Balancing Algorithm During query parallel plan processing, some agents became finished while other agents of the plan continue processing segment intervals assigned to them. So we have load *skew*. To prevent the load skew, we proposes the following load balancing algorithm based on partial replication [10].

Let parallel agent $\bar{Q} \in \mathfrak{Q}$ have finished processing the segment intervals in all input streams and be turned into passive state. In the same time, let agent $\tilde{Q} \in \mathfrak{Q}$ still continue processing its segment intervals. So we have to do load balancing. We name the idle agent $\bar{Q}$ as *forward* and the overloaded agent $\tilde{Q}$ – as an *backward*. To prevent load skew we have transfer a part of unprocessed segments transferred from agent $\tilde{Q}$ to agent $\bar{Q}$. The scheme of load balancing algorithm is shown in Figure 3. In the algorithm, we use the *balancing function* Delta. For each stream, the balancing function calculates the quantity of segments, which have been transferred from the backward agent $\tilde{Q}$ to the forward agent $\bar{Q}$.

For effective using the described algorithm, we have to manage the following problems.

1. For each forward agent, we have to choose a backward agent, which will be the subject of balancing. A way of choosing backward agent we will name as *backward choice strategy.*
2. We have to solve, what quantity of unprocessed segments will be transferred from the backward to the forward. We will name the function calculating this quantity as *balancing function.*

Backward Choice Strategy Let cluster system T execute parallel query plan $\mathfrak{Q}$. Let Ψ be the number of nodes of cluster T. Let forward agent $\bar{Q} \in \mathfrak{Q}$ located on the node $\bar{\psi} \in \Psi$ have finished its work. We have to choose a backward agent $\tilde{Q} \in \mathfrak{Q}(\tilde{Q} \neq \bar{Q})$ from the set of agents in parallel plan $\mathfrak{Q}$. Denote by $\tilde{\rho}$ the replication factor which defines a size of the replica $\tilde{f}_i$ for fragment $\bar{f}_i$.

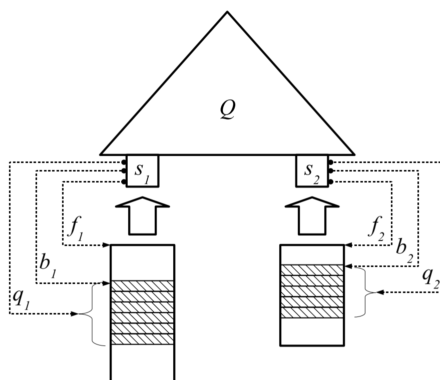

Fig. 2. Parallel agent with two input streams.

```
/* load balancing procedure between agents Q̄ (forward)
and Q̃ (backward). */
ū = Node(Q̄); // pointer to the agent node Q̄
pause Q̃; // turn agent Q̃ into passive state
for (i=1; i≤n; i++) {

  if(Q̃.s[i].a == 1) {
    f̃_i = Q̃.s[i].f; // fragment assigned to agent Q̃
    r̄_i = Re(f̃_i, ū); // replica f̃_i into the node ū
    δ_i = Delta(Q̃.s[i]); // quantity of segments to trans-
fer
    Q̃.s[i].q− = δ_i;
    Q̄.s[i].f = r̄_i;
    Q̄.s[i].b = Q̃.s[i].b + Q̃.s[i].q;
    Q̄.s[i].q = δ_i;
  } else
      print("Load balancing is not permitted.");
};
activate Q̃ // turn agent Q̃ into active state
activate Q̄ // turn agent Q̄ into active state
```

Fig. 3. Load balancing algorithm for two parallel agents.

To choose a backward agent, we will use ratings. In the parallel plan, we assign to each agent a rating, which is a real number. The agent with maximum positive rating has to be selected as the backward. If all the agents have negative ratings, the forward agent $\bar{Q}$ has to be over. If several agents have the maximum positive rating at the same time, we choose from them the agent, which was idle for the longest time.

The optimistic strategy uses the following rating function $\gamma : \mathfrak{Q} \to \mathbb{R}$:

$$\gamma(\tilde{Q}) = \tilde{a}_i \cdot sgn(\max_{1 \le i \le n} \tilde{q}_i - B) \cdot \tilde{\rho} \cdot \vartheta \cdot \lambda.$$

Here λ – some positive weight coefficient; B – the nonnegative integer defining the minimal quantity of segments to be transferred during load balancing; $\tilde{\rho}$ – the replication factor; ϑ – the statistical coefficient accepting one of following values:

- (-1) – the quantity of unprocessed segments for the backward is less than B;
- 0 – the backward did not participate in load balancing;
- *a positive integer* – the quantity of completed load balancing procedures, in which the backward has took a part.

Balancing Function For each stream $\tilde{s}_i$ of backward agent $\tilde{Q}$ the balancing function Δ defines the quantity of segments, which has to be transferred to the

forward agent $\bar{Q}$. In the simplistic case we can define:

$$\Delta(\tilde{s}_i) = \left\lceil \frac{\min\left(\tilde{q}_i, S(\tilde{f}_i) \cdot \tilde{\rho}\right)}{N} \right\rceil,$$

where N is the quantity of the parallel agents participating in query processing.

Function $S(\tilde{f}_i)$, introduced into the section 3.1, calculates the quantity of segments in fragment $\tilde{f}_i$. So, function Δ splits unprocessed segments of fragment $\tilde{f}_i$ into N buckets and transfers one of them from $\tilde{Q}$ to $\bar{Q}$. Such balancing function provides the uniform distribution of the load among the forward agents, which are idle.

4 Experiments

To verify the described load balancing method we performed three series of computing experiments using the "Omega" parallel database management prototype [11]. These experiments had the following aims:

- to obtain an estimations of optimum values for the parameters of the load balancing algorithm;
- to investigate the influence of the load balancing algorithm on the DBMS scalability;
- to perform the efficiency analysis of the load balancing algorithm for different kinds of the skew.

For investigation of the load balancing algorithm we used the in-memory hash-join algorithm. This algorithm is used when one of the input relations can be allocated in main memory.

4.1 Parameters of Computing Experiments

Both relations R and S have five attributes having common domain: nonnegative integers in a range from 0 to 10 million. In our experiments we processed $\theta-$join for relations R and S by the in-memory hash join method.

R was the build relation, and S was the probe relation. We selected the size of build relation R so that any fragment of R fit in main memory of the cluster node. Building hash table did not demand load balancing due to the small size of the build relation R. So we used load balancing only at the second stage of MHJ execution.

Relations R and S were created by the automated generating procedure. We used a probability model to fragment the relations on the cluster nodes. According to this model, the *skew factor* θ, $(0 \leq \theta \leq 1)$ defines the set of weight coefficients $p_i, (i = 1, \ldots, N)$

$$p_i = \frac{1}{i^\theta \cdot H_N^{(\theta)}}, \qquad \sum_{i=1}^{N} p_i = 1,$$

where N - the number of fragments, $H_N^s = 1^{-s} + 2^{-s} + \ldots + N^{-s}$ – N-th harmonic number with the degree s. Each weight coefficient p_i defines the size of $i-$th fragment in tuples.

We used one more skew factor μ to fill join attribute values in different fragments. Skew factor μ determines the percentage of "own" and "alien" tuples in the fragment. The "own" tuple should be processed in the same cluster node where it's being stored. The "alien" tuples should be transferred to another cluster node for its processing.

4.2 Investigation of Load Balancing Parameters

In the first series of experiments we investigated the following parameters: balancing interval, segment size and replication factor. The result of investigation of balancing interval is shown in Figure 4. We can see that the optimal value is 0.1 seconds. Increasing the balancing interval from 0.1 seconds to 6 seconds considerably worsens efficiency of load balancing procedure.

The result of segment size investigation is shown in Figure 5. We can see that the optimal value is 20 000 tuples. This value is approximately equal to 1% of the fragment size.

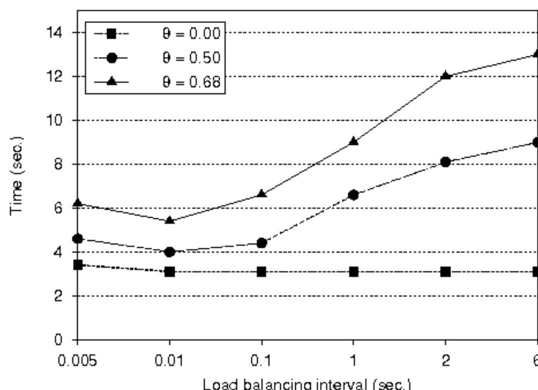

Fig. 4. Impact of balancing interval ($n = 64, \mu = 50\%$).

The impact of replication factor on query processing time is shown in Figure 6. We made tests for four different values of the skew factor θ. In all cases, skew factor μ was equal to 50%. We can see that the optimal value of replication factor is 0.8. In this case, the load balancing almost completely eliminates the negative impact of the skew in sizes of fragments. The full replication ($\rho = 1.0$) demonstrates a worse result due to increasing the overhead of load balancing. Also, we can see that the effectiveness of load balancing grows significantly for big values of skew factor θ. If skew factor θ is equal to 0.2 (rule "25-15"), then

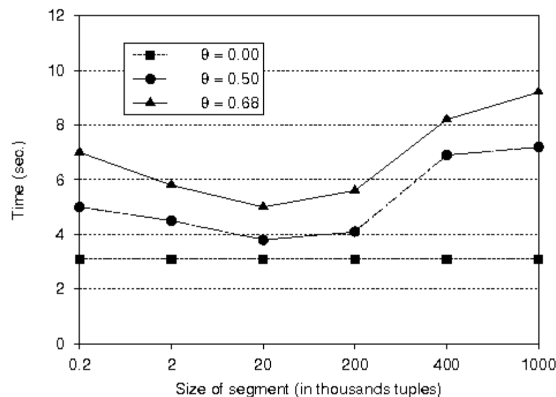

Fig. 5. Impact of segment size ($n = 64, \mu = 50\%$).

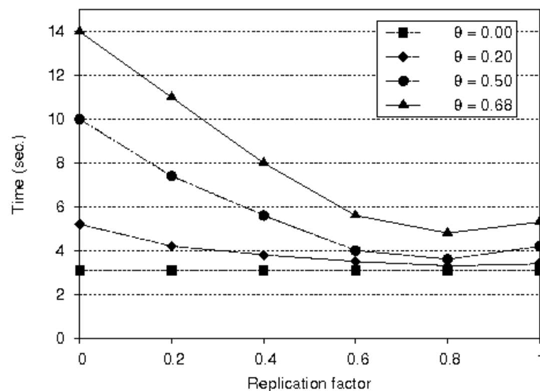

Fig. 6. Impact of replication factor ($n = 64, \mu = 50\%$).

load balancing allows us to reduce the query execution time by 30%. If $\theta = 0.68$ (rule "80-20"), then the load balancing reduces query execution time by 60%. The experiments show that the load balancing method, described in the paper, can be successfully used to eliminate a load disbalance in cluster database systems.

4.3 Scalability of Load Balancing Algorithm

In the last series of experiments, we investigate the scalability of the load balancing algorithm presented in the paper. The results of these experiments are shown on figures 7, 8 and 9.

In Figure 7, we presented the impact of replication factor on speedup. In this experiment, we used the following skew values: $\mu = 50\%, \theta = 0.5$. The result of the experiment demonstrates the significant growth of speedup when load

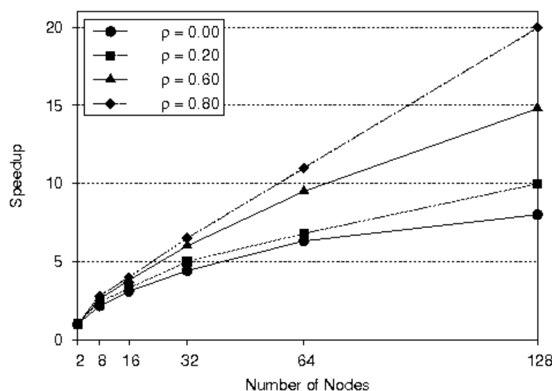

Fig. 7. Speedup versus replication factor ($\theta = 0.5, \mu = 50\%$).

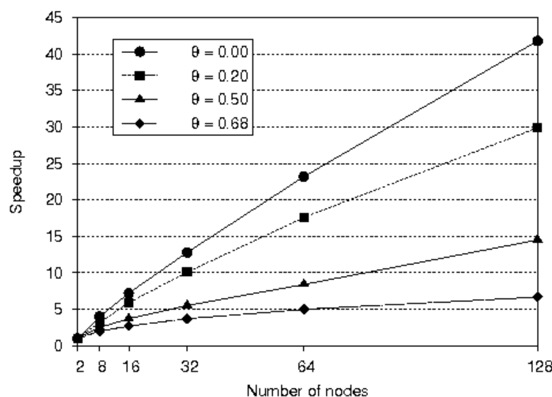

Fig. 8. Speedup versus skew factor θ ($\rho = 0.50,\ \mu = 50\%$).

balancing is being performed. Increasing the replication factor value leads to "lifting" of speedup curve. In case replication factor ρ is equal to 0.8, we have speedup near to linear.

In Figure 8, we presented the speedup curves for various values of skews factor θ. In this experiment we used replication factor $\rho = 0.5$. We can see that load balancing provides a significant speedup even at presence of severe data skew ($\theta = 0.68$ corresponds to rule "80-20").

In Figure 9, we presented the speedup curves for various values of skews factor μ. We can see that, even in the worst case $\mu = 80\%$ (80% of tuples are "alien"), load balancing provides a visible speedup.

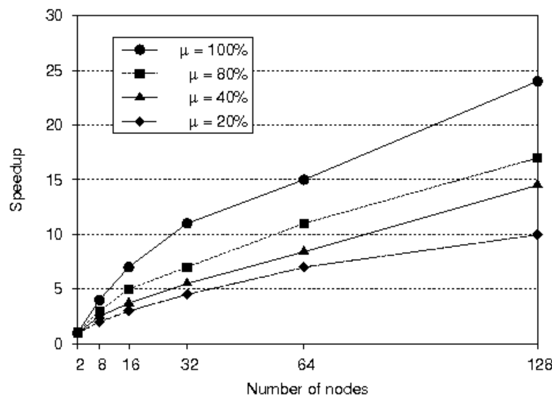

Fig. 9. Speedup versus skew factor μ ($\theta = 0.5,\ \rho = 0.50$).

5 Conclusions

The problem of parallel query execution in cluster systems is considered. A load balancing method and a data placement strategy are presented. Proposed approach is based on logical splitting the relation fragment by segments of equal size. The proposed load balancing algorithm is based on the method of partial replication. A strategy for choosing the backward agent is described. It uses a rating function. Proposed methods and algorithms are implemented in "Omega" DBMS prototype. The experiments at cluster system are performed. The results confirm the efficiency of proposed approach.

There are at least two directions of future work suggested by this research. First, it is useful to incorporate the proposed technique of parallel query execution into open source PostgreSQL DBMS. Second, it is interesting to extend this approach on GRID DBMS for clusters with multicor processors.

References

1. *Dean J., Ghemawat S.* MapReduce: simplified data processing on large clusters // Communications of ACM Vol. 51, No. 1, 2008. P. 107–113.
2. *Chaudhuri S., Narasayya V.* Self-tuning database systems: a decade of progress // In Proceedings of the 33rd international Conference on Very Large Data Bases (Vienna, Austria, September 23 – 27, 2007). 2007. P. 3–14.
3. *Xu Y., Kostamaa P., Zhou X., Chen L.* Handling data skew in parallel joins in shared-nothing systems // ACM SIGMOD international Conference on Management of Data Vancouver, Canada, June 09 – 12, 2008, proceedings. ACM, 2008. P. 1043–1052.
4. *Han W., Ng J., Markl V., Kache H., Kandil M.* Progressive optimization in a shared-nothing parallel database // In Proceedings of the 2007 ACM SIGMOD international conference on Management of data (Beijing, China, June 11–14, 2007). 2007. P. 809–820.

5. *Zhou J., Cieslewicz J., Ross K. A., Shah M.* Improving database performance on simultaneous multithreading processors // In Proceedings of the 31st international Conference on Very Large Data Bases (Trondheim, Norway, August 30 – September 02, 2005). 2005. P. 49–60.
6. *Lakshmi M.S., Yu P.S.* Effect of Skew on Join Performance in Parallel Architectures // Proceedings of the first international symposium on Databases in parallel and distributed systems, Austin, Texas, United States. IEEE Computer Society Press. 1988. P. 107–120.
7. *Ferhatosmanoglu H., Tosun A. S., Canahuate G., Ramachandran A.* Efficient parallel processing of range queries through replicated declustering // Distrib. Parallel Databases. 2006. Vol. 20, No. 2. P. 117–147.
8. *Kostenetskii P. S., Lepikhov A. V., Sokolinskii L. B.* Technologies of parallel database systems for hierarchical multiprocessor environments // Automation and Remote Control. 2007. No. 5. P. 112–125.
9. *Sokolinsky L.B.* Organization of Parallel Query Processing in Multiprocessor Database Machines with Hierarchical Architecture // Programming and Computer Software. 2001. Vol. 27. No. 6. P. 297-308.
10. *Lepikhov A.V., Sokolinsky L.B.* Data Placement Strategy in Hierarchical Symmetrical Multiprocessor Systems // Proceedings of Spring Young Researchers Colloquium in Databases and Information Systems (SYRCoDIS'2006), June 1–2, 2006. Moscow, Russia: Moscow State University. 2006. P. 31–36.
11. Parallel DBMS "Omega" official page. URL: http://omega.susu.ru