Technologies of parallel database systems for hierarchical multiprocessor environments

Article *in* Automation and Remote Control · January 2007

DOI: 10.1134/S0005117907050116

CITATIONS
10

READS
113

3 authors:

Pavel S. Kostenetskiy
National Research University Higher School of Economics
16 PUBLICATIONS **254** CITATIONS
SEE PROFILE

Andrei Lepikhov
Postgres Professional
18 PUBLICATIONS **11** CITATIONS
SEE PROFILE

Leonid Sokolinsky
South Ural State University
64 PUBLICATIONS **437** CITATIONS
SEE PROFILE

ISSN 0005-1179, Automation and Remote Control, 2007, Vol. 68, No. 5, pp. 847–859. © Pleiades Publishing, Ltd., 2007.
Original Russian Text © P.S. Kostenetskii, A.V. Lepikhov, L.V. Sokolinskii, 2007, published in Avtomatika i Telemekhanika, 2007, No. 5, pp. 112–125.

TOPICAL ISSUE

Technologies of Parallel Database Systems for Hierarchical Multiprocessor Environments[1]

P. S. Kostenetskii, A. V. Lepikhov, and L. V. Sokolinskii

South-Ural State University, Chelyabinsk, Russia

Received December 14, 2006

Abstract—For the multiprocessor systems of the hierarchical-architecture relational databases, a new approach to data layout and load balancing was proposed. Described was a database multiprocessor model enabling simulation and examination of arbitrary multiprocessor hierarchical configurations in the context of the on-line transaction processing applications. An important subclass of the symmetrical multiprocessor hierarchies was considered, and a new data layout strategy based on the method of partial mirroring was proposed for them. The disk space used to replicate the data was evaluated analytically. For the symmetrical hierarchies having certain regularity, theorems estimating the laboriousness of replica formation were proved. An efficient method of load balancing on the basis of the partial mirroring technique was proposed. The methods described are oriented to the clusters and Grid-systems.

PACS number: 89.20.Ff

DOI: 10.1134/S0005117907050116

1. INTRODUCTION

The hierarchical multiprocessor architectures are becoming increasingly popular. In the hierarchical-architecture multiprocessor system, the processors, memory, disks, and so on are interconnected according to a certain hierarchy. The processor kernels nested on one chip lie at the first hierarchical level. The second level contains the multikernel processors combined into the shared-memory multiprocessors (SMP). At the third level, the SMP modules are clustered by a high-speed network. The fourth level is represented by the *corporate* Grid-systems consisting of more than one cluster. The corporate Grid-systems may be united into the Internet-based *cooperative* Grid-unions, and so on. The parallel database systems capable to store and process petabytes of data represent one of the most important applications of the multiprocessor systems [1]. A good deal of publications is devoted to the parallel database systems with hierarchical multiprocessor architecture. Their review can be found in [2], yet the problems of such systems are still little-studied.

The present paper considers organization of the parallel databases oriented to effective use of the multiprocessor hierarchies. The following issues are brought into focus: modeling of the hierarchical architectures, data placement, and load balancing. The hierarchical architectures give rise to many diverse classes of configurations. A classification of such architectures can be found in [3]. Studies of such architectures are hindered by the fact that practical design of the multiprocessors requires large financing for acquisition and reconfiguration of the costly equipment.

As the result, the problem of developing models of the multiprocessor database systems which enable one to study various multiprocessor configurations without implementing them in hardware becomes topical. A. Bhide and M. Stonebraker modeled the one-level architectures for fast

[1] This work was supported by the Russian Foundation for Basic Research, project no. 06-07-89148, and the South-Ural State University, project no. 2006-112.

transaction processing [4, 5]. Some classes of two-level configurations were modeled in [6, 7], but the general form of the multiprocessor hierarchical configurations was not considered yet. In this connection, we face the problem of choosing the optimal class of configurations for a certain class of database applications. The present paper describes a database multiprocessor model (DMM) enabling simulation and study of arbitrary multiprocessor hierarchical configurations in the context of the applications of the on-line transaction processing (OLTP) class [8].

The problem of data placement and the related problem of load balancing in the parallel database systems without resource sharing were used in a number of works (see, for example, [9–12]), but these studies were oriented mostly to the one-level multiprocessor systems. The present paper proposes a strategy of data layout for the hierarchical computer systems and an algorithm of load balancing based on the method of partial mirroring. The algorithm is used to process the requests in the prototype of the *Omega* parallel database system [13].

Section 2 describes the DMM-model. Models of the hardware platform and operational environment, cost, and transactions are presented. Section 3 discusses the strategies of data layout and the algorithm to balance the load of the hierarchical computer system. A formal definition of the symmetrical multiprocessor hierarchy is introduced; the segment-based mechanism of data replication is described, and the theorems estimating the total size of replicas and laboriousness of their formation in the absence of noise are proved. The replication mechanism-based algorithm of load balancing is presented. Conclusion sums up the results obtained and discusses the lines of further studies.

2. MODELING OF THE HIERARCHICAL ARCHITECTURES

Proposed is a new DMM enabling simulation and investigation of arbitrary multiprocessor hierarchical configurations in the context of the class of OLTP applications. The DMM includes the hardware and software models, as well as the cost model.

2.1. Model of the Hardware Platform

The hardware of the parallel database system is represented as the DM-tree which is an oriented tree [14] whose nodes belong to one of the following three classes:

(1) processor modules;

(2) disk modules;

(3) network concentrator modules.

The arcs of the tree correspond to the data flows. The *processor modules* are abstract representations of the physical processor devices. The *disk modules* are storages based on the hard magnetic disks. The *network concentrator modules* are used to represent an arbitrary *interconnect* of various processor and disk facilities. Both individual network units such as switches, concentrators, and

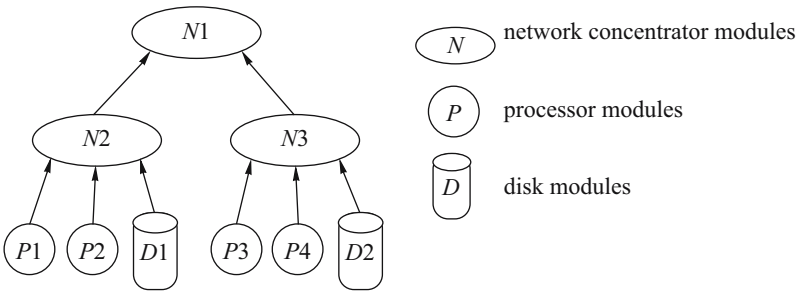

Fig. 1. Example of the DM-tree.

so on, as well as the system bus connecting the processor to the periphery, may be used as the interconnects. Since in the OLTP problems the time of interaction between the processors and main memory is incomparably smaller than that of data exchange with the external units, the main-memory modules are not represented in the DMM.

The following constraints are imposed on the structure of the DM-tree:

(1) only the network concentrator module can be the root of the DM-tree;

(2) the disk and processor modules cannot have daughter nodes, that is, they always are the leaves of the DM-tree.

The two-level DM-tree is exemplified in Fig. 1.

2.2. Model of the Operational Environment

Within the DMM framework, the least indivisible unit of data processing is represented by the *packet*. All packets are assumed to be of the same size. The packet has a heading which includes the addresses of the sender and the receiver, as well as auxiliary information. Transmission of a packet may correspond to the transmission of one or more tuples in the physical database system.

Since consideration is given only to the OLTP applications, the overhead of the exchange between two processors through the shared main memory and the cost of data processing within the processors may be disregarded. Any DMM processor module can exchange data with any disk module. In the DMM, a *queue* of the transmitted packets is associated with each disk and network concentrator module.

The DMM admits asynchronous exchange of packets in the sense that the processor module can initialize a new exchange without waiting for the completion of the current exchange. However, we assume that at each time instant the processor module can have at most s_r outstanding reading and s_w outstanding writing operations.

The system operation time in the DMM is divided into discrete time intervals called the *clocks*. The clock is defined as a fixed sequence of steps whose semantics is defined in what follows.

Let $\mathfrak{P}$ be the set of all processor modules in the DM-tree, $\mathfrak{D}$ be the set of all disk modules, $\mathfrak{N}$ be the set of all network concentrator module, and $\mathfrak{M} = \mathfrak{P} \cup \mathfrak{D} \cup \mathfrak{N}$ be the set of all nodes of the DM-tree. For an arbitrary $M \in \mathfrak{M}$, we denote by $F(M)$ the parent module of the node M and by $T(M)$ the subtree with the root at the vertex M.

Processor module $P \in \mathfrak{P}$ can initiate packet reading and writing.

Reading operation. Let the processor module P have to read the packet E from the disk $D \in \mathfrak{D}$. If previously P initiated s_r outstanding reading operations, it is driven to the waiting state. If the number of outstanding operations is less than s_r, then the packet E with the receiver address $\alpha(E) = P$ and the sender address $\beta(E) = D$ is queued on the disk D. This algorithm has the following symbolic code:

```
if  r(P) < s_r  then
        Place E
            with address P
            on queue D;
        r(P) + +;
else
        wait;
end if,
```

where $r(P)$ is the number of the outstanding reading operations of the processor P and s_r is the maximum permissible number of the outstanding reading operations.

Writing operation. Let the processor module P have to write the packet E on the disk $D \in \mathfrak{D}$. The symbolic code of the algorithm that initiates writing is as follows:

```
if  w(P) < s_w  then
      Place packet E
        with address D on queue of
        parent network
        concentrator;
      w(P) + +;
else
      wait;
end if,
```

where $w(P)$ is the number of the outstanding writing operations of the processor P and s_w is the maximum permissible number of the outstanding writing operations.

Module of network concentrator $N \in \mathfrak{N}$ permanently transmits the packets through the connectional network by executing the following algorithm:

```
Extract packet E from queue N;
if α(E) ∉ T(N) then
      Place E on queue F(N);
else
      Determine the maximal subtree U
            of tree T(H), containing α(E);
      if T(α(E)) = U then
            if α(E) ∈ 𝔓 then
                  r(α(E)) − −;
            else
                  Place E on queue α(E);
            end if
      else
            Place E on queue R(U);
      end if
end if,
```

where E is the packet, $\alpha(E)$ is the addressee of E, $T(N)$ is the subtree with the root N, $F(N)$ is the parent module of the node N, $\mathfrak{P}$ is the set of the processor modules, $r(P)$ is the number of the outstanding reading operations of the processor P, and $R(U)$ is the root of the subtree U.

Disk module $D \in \mathfrak{D}$ permanently reads and writes packets according to the algorithm

```
Extract packet E from queue D;
if α(E) ∈ 𝔇 then
      w(β,(E)) − −;
else
      Place E on queue of parent node;
end if,
```

where $\beta(E)$ is the packet sender.

In the DMM, the data are processed as a cycle performing a standard sequence of steps which is called the *clock* and defined as follows:

(1) each module of the network concentrator processes all waiting packets;

(2) each active processor module executes one read/write operation;

(3) each disk module processes one packet from its queue.

Obviously, in this case more than $|\mathfrak{P}| + |\mathfrak{D}|$ packets may be queued by any concentrator, the queue of any disk simultaneously can have at most $s_r s_w |\mathfrak{P}|$ packets.

2.3. Cost Model

Each module $M \in \mathfrak{M}$ has an associated *laboriousness coefficient* $h_M \in \mathbb{R}$, $1 \leqslant h_M < +\infty$. Since for the OLTP applications the time of processing one packet is smaller by the factor of 10^5–10^6 than the time of exchange with the disk or transmission through the network, we assume that

$$h_p = 1, \ \forall P \in \mathfrak{P}.$$

Since in one clock the network concentrator module can transmit more than one packet, the noise function

$$f_N(m_i^N) = e^{\frac{m_i^N}{\delta_N}},$$

where m_i^N is the number of packets passing through N at the ith clock and $\delta_N > 1$ is the scaling factor, is introduced for each network concentrator module $N \in \mathfrak{N}$. Therefore, the time required for the network concentrator module N to execute the ith clock obeys the following formula:

$$t_i^N = h_N f_N(m_i^N), \qquad \forall N \in \mathfrak{N}.$$

The total system time for processing a transaction mix (see Section 1.4) in k clocks follows

$$t = \sum_{i=1}^{k} \max\left(\max_{N\in\mathfrak{N}}(t_i^N), \max_{D\in\mathfrak{D}}(h_D)\right).$$

2.4. Model of Transactions

The transaction Z is modeled by defining two groups of processes ρ and ω: $Z = \{\rho, \omega\}$. The group ρ includes the *reading* processes; the group ω, the *writing* processes. The processes abstract the read/write operations executed in the course of transaction processing.

The number of reading processes is defined by the number of disks from which the transaction Z reads data, a process per disk. Similarly, the number of the writing processes is defined by the number of disks to which write-in is made at executing the transaction Z. At that, if the same disk is used both for reading and writing, then two individual processes—one reading and the other writing—are assigned to it.

The DMM admits execution of a parallel transaction mix by one processor. At that, each transaction Z_i $(i = 1, \ldots, k)$ is represented by its pair $Z_i = \{\rho_i, \omega_i\}$ of groups of the read and write processes. The entire set of processes modeling execution of a transaction mix is defined as

$$\Phi = \bigcup_{i=1}^{k} (\rho_i \cup \omega_i).$$

For each process $\phi \in \Phi$, the operation probability p_ϕ, that is, the probability of accessing the disk associated with this process, is defined. The *activity function* $\mathbf{g}(p_\phi)$ is defined in accordance with this probability. The activity function $\mathbf{g}(p_\phi) = G$ is the function of a discrete random variable G whose distribution law is defined by the following table.

Table

Value of the random variable G	Probability
1	p_ϕ
0	$1 - p_\phi$

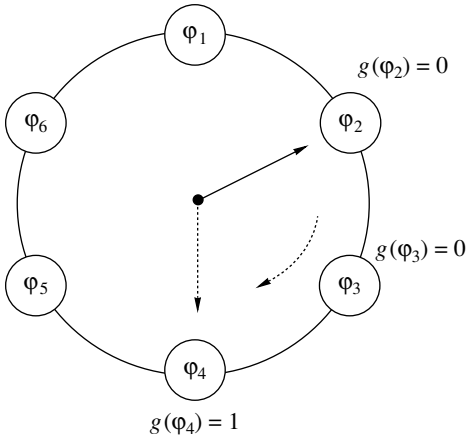

Fig. 2. Choice of the active process.

At each clock, all active processor modules must execute read/write operations (see Section 2.2). In this connection, the active processor module must select some process $\phi \in \Phi$ and perform reading from or writing on the disk associated with ϕ. We refer to this process as *active*. The activity function has the following semantics of values: 1—the process is active at the given clock, 0—the process is inactive.

The active process is chosen as follows. All processes from the set Φ are organized in a cyclic list, and an indicator of the current list element (its initial position may be arbitrary) is introduced. When choosing the active process, the list is scanned cyclically beginning from the current element. Scanning is stopped as soon as the activity function of any element of the list takes value 1 (see Fig. 2).

3. DATA LAYOUT AND LOAD BALANCING

This section is devoted to the strategy of data layout and the algorithm of load balancing in the hierarchical multiprocessor system.

3.1. Symmetrical Hierarchies

The multiprocessor system architecture is often the decisive factor in the development of the data layout strategy. In this section we construct a model of the symmetrical hierarchical multiprocessor database system based on the notion of the DM-tree introduced in Section 2.1. We define the isomorphism of two DM-trees. Let $\mathfrak{E}_T$ be the set of arcs of the DM-tree T and h_M be the laboriousness coefficient of the node $M \in \mathfrak{M}_T$ on the DM-tree T. The DM-trees A and B are called the *isomorphic* trees if there exists a one-to-one map f of the set $\mathfrak{M}_A$ onto the set $\mathfrak{M}_B$ and one-to-one map g of the set $\mathfrak{E}_A$ onto the set $\mathfrak{E}_B$ such that:

(1) ν is the end node of the arc e on the tree A if and only if $f(\nu)$ is the end node of the arc $g(e)$ on the tree B;

(2) w is the initial node of the arc e on the tree A if and only if $f(w)$ is the initial node of the arc $g(e)$ on the tree B;

(3) $P \in \mathfrak{P}_A \Longleftrightarrow f(P) \in \mathfrak{P}_B$;

(4) $D \in \mathfrak{D}_A \Longleftrightarrow f(D) \in \mathfrak{D}_B$;

(5) $N \in \mathfrak{N}_A \Longleftrightarrow f(N) \in \mathfrak{N}_B$;

(6) $h_M = h_{f(M)}$.

The ordered pair of maps $q = (f, g)$ will be called the *isomorphism* of the DM-tree A onto the DM-tree B.

We define recursively the *node level* with respect to the tree T as follows [15]. The level of the root of T is zero, and the level of any other node is one greater than the level of the minimal subtree of the tree T containing this node.

By the *level* $l(M)$ of the subtree M of the tree T is meant the level of the root of the subtree M on the tree T.

Two subtrees of the same level are referred to as *adjacent* if there exists a node on the tree which is the common parent to the root nodes of these subtrees.

We define the *height* of the oriented tree T as the maximal level of the subtree on this tree [14].

The DM-tree T of the height H is referred to as *symmetrical* if the following conditions are satisfied:

(1) any two adjacent subtrees of the level $l < H$ are isomorphic, and

(2) any subtree of the level $H - 1$ contains exactly one disk and one process.

Condition 2 in the definition of symmetricity of the DM-tree is an abstract model of the SMP system in the sense that in the context of the multiprocessor hierarchies all processors of the SMP system can be regarded as one "megaprocessor," and all disks, as one "megadisk." At that, since the SMP system has a shared memory and all disks are equally accessible to all processors (see [16, 17]), at the level of the SMP system the load, obviously, must be balanced using basically different methods. We defined the *node degree* as the number of arcs entering this node. On the symmetrical tree, all nodes of the same level have identical degree called the *degree of the given level*.

3.2. Data Fragmentation and Segmentation

Placement of the database in the nodes of the multiprocessor hierarchical system is defined as follows. Each relation is decomposed into disjoint horizontal fragments [12] located in different disk modules. At that, we assume that the fragment tuples are ordered somehow and this order is fixed for each request and defines the sequence of tuple reading at fragment scanning. We refer to this order as *natural*. In practice, this natural order may be defined by the physical order of tuples or by an index.

At the logical level, each fragment is decomposed into a sequence of fixed-length *segments*. The segment length is measured in tuples and is an attribute of the fragment. Segmentation is carried out in compliance with the natural order and always begins with the first tuple, Accordingly, the last segment of a fragment may be incomplete.

The number of segments of the fragment F is denoted by $\mathrm{S}(F)$ and follows the formula

$$\mathrm{S}(F) = \left\lceil \frac{\mathrm{T}(F)}{\mathrm{L}(F)} \right\rceil, \tag{1}$$

where $\mathrm{T}(F)$ is the number of tuples in the fragment F and $\mathrm{L}(F)$ is the length of the segment for the fragment F.

3.3. Data Replication

Let the fragment F_0 be located in the disk module $D_0 \in \mathfrak{D}_T$ of the multiprocessor hierarchical system T. We assume that in each disk module $D_i \in \mathfrak{D}_T$ there is a *partial replica* F_i comprising some, possibly empty, subset of the tuples of the fragment F.

Segment is the least unit of data replication. The length of the replica segment always coincides with the length of the segment of the replicated fragment:

$$\mathrm{L}(F_i) = \mathrm{L}(F_0), \quad \forall D_i \in \mathfrak{D}_T.$$

The size of the replica F_i is defined by the *replication coefficient*

$$\mu_i \in \mathbb{R}, \qquad 0 \leqslant \mu_i \leqslant 1$$

which is the attribute of the replica F_i and obeys the formula

$$\mathrm{T}(F_i) = \mathrm{T}(F_0) - \lceil (1-\mu_i)\mathrm{S}(F_0) \rceil \, \mathrm{L}(F_0). \tag{2}$$

The *natural order of the tuples of the replica* F_i is defined by the natural order of the tuples of the fragment F_0. At that, the *number* N *of the first tuple* of the replica F_i follows the formula

$$\mathrm{N}(F_i) = \mathrm{T}(F) - \mathrm{T}(F_i) + 1.$$

For an empty replica F_i, we get $\mathrm{N}(F_i) = \mathrm{T}(F_0) + 1$, which corresponds to the "file end" tag.

The above replication mechanism allows one to employ in the multiprocessor hierarchies a simple and effective method of load balancing described in Section 3.6.

3.4. Method of Partial Mirroring

Let the symmetrical DM-tree T of height $H > 1$ be given. We define the *replication function* $r(l)$ which assigns the replication coefficient $\mu = r(l)$ to each level $l < H$ of the tree T and assume that always $r(H-1) = 1$. This fact is explained by that the level of the hierarchy $H-1$ includes the subtrees of height 1 to which the SMP systems correspond. In the SMP system, all disks are equally accessible to any processor. Therefore, there is no need for physical data replication. At the logical level, the load is balanced by segmentation of the initial fragment, that is, the fragment itself plays the role of its replica.

Let the fragment F_0 be on the disk $D_0 \in \mathfrak{D}_T$. We use the following *method of partial mirroring* to construct the replica F_i on the disk $D_i \in \mathfrak{D}_T$ $(i > 0)$. We construct the sequence of the subtrees of the tree T

$$\{M_0, M_1, \ldots, M_{H-2}\}, \tag{3}$$

having the following characteristics:

$$\begin{cases} l(M_j) = j \\ D_0 \in \mathfrak{D}(M_j) \end{cases}$$

for all $0 \leqslant j \leqslant H-2$, where $l(M_j)$ stands for the level of the subtree M_j.

Obviously, for any symmetrical tree T there exists only one such sequence. Let us determine the greatest index $j \geqslant 1$ such that

$$\{D_0, D_i\} \subset \mathfrak{D}(M_j).$$

We assume that

$$\mu_i = r(j). \tag{4}$$

We use the algorithm of Section 3.3 with the replication coefficient obeying (4) to generate the replica F_i on the disk D_i.

The size of replica is estimated by the following theorem.

Theorem 1. *Let T be a symmetrical DM-tree of height $H > 1$, the fragment F_0 be located on the disk $D_0 \in \mathfrak{D}_T$, M be a subtree of the tree T such that $0 < l(M) < H$ and $D_0 \in \mathfrak{D}_M$, and M' be an arbitrary subtree of the tree T which is adjacent to M. Then, the following estimate of the size of replica F_i of the fragment F_0 located on the disk D_i is true for any $D_i \in \mathfrak{D}_{M'}$:*

$$\mathrm{T}(F_i) = r(l(M) - 1)\mathrm{T}(F_0) + O(\mathrm{L}(F_0)),$$

where $\mathrm{L}(F_0)$ is the length of the segment for the fragment F_0.

Proof can be found in [18].

We note that the size of the segment $\mathrm{L}(F_0)$ is a replication parameter which is not related to database fragmentation. Therefore, $\mathrm{L}(F_0)$ can be regarded as a constant whose value is negligibly small as compared with the total size of the database.

The following theorem estimates the total size of all fragment replicas.

Theorem 2. *Let T be a symmetrical DM-tree of height $H > 1$, and the fragment F_0 be located on the disk $D_0 \in \mathfrak{D}_T$. We denote by δ_l the degree of the level l of tree T and by $R(F_0) = \sum\limits_{i=1}^{|\mathfrak{D}_T|} \mathrm{T}(F_i)$ the total number of tuples in all replicas of the fragment F_0. Then,*

$$R(F_0) = \mathrm{T}(F_0) \sum_{j=0}^{H-2} r(j)(\delta_j - 1) \prod_{k=j+1}^{H-2} \delta_k + O(\mathrm{L}(F_0)). \tag{5}$$

Proof can be found in [18].

3.5. Choice of the Replication Function

It is advisable to take into consideration the laboriousness coefficients of the nodes of DM-tree at choosing the replication function $r(l)$. On the symmetrical DM-tree, obviously, all vertices of the level l have identical laboriousness $h(l)$ which is called the *laboriousness of level l*.

We refer to the symmetrical DM-tree T as *regular* if

$$l < l' \Rightarrow h(l) \geqslant h(l') \tag{6}$$

is true for any two levels l and l' of the tree T, that is, the higher the hierarchical level, the higher its laboriousness. The following theorem allows one to estimate the laboriousness of the tuplewise replica formation on the regular DM-tree.

Theorem 3. *Let T be a regular DM-tree of height $H > 1$, the fragment F_0 lie on the disk $D_0 \in \mathfrak{D}_T$, M be a subtree of the tree T such that $0 < l(M) < H - 1$ and $D_0 \in \mathfrak{D}_M$, M' be an arbitrary subtree of the tree T adjacent to M, and F_i be a replica of the fragment F_0 located on the disk $D_i \in \mathfrak{D}_{M'}$. We denote by $\tau(F_i)$ the laboriousness of the tuplewise formation of the replica F_i in the absence of noise. Then,*

$$\tau(F_i) = h\,(l(M) - 1)\, r\,(l(M) - 1)\, \mathrm{E}\,(F_0) + O\,(h_0)\,,$$

where $h_0 = h(0)$ is the laboriousness coefficient of the root of tree T.

Laboriousness of the tuplewise formation of all fragment replicas disregarding noise can be estimated by the following theorem.

Theorem 4. *Let T be a regular DM-tree of the height $H > 1$ and the fragment F_0 lie on the disk $D_0 \in \mathfrak{D}_T$. We denote by δ_l the degree of the level l of the tree T and by $\tau(F_0) = \sum_{i=1}^{|\mathfrak{D}_T|} t(F_i)$ the total laboriousness of the tuplewise formation of all replicas of the fragment F_0 disregarding noise. Then,*

$$\tau(F_0) = \mathrm{E}(F_0) \sum_{j=0}^{H-2} h(j)\, r(j)\, (\delta_j - 1) \prod_{k=j+1}^{H-2} \delta_k + O(h_0). \tag{7}$$

Proof can be found in [18].

We define recursively the *normal* replication function $r(l)$ as follows:

(1) for $l = H-2$: $\quad r(H-2) = \dfrac{1}{h(H-2)(\delta_{H-2}-1)}$;

(2) for $0 \leqslant l \leqslant H-2$: $\quad r(l) = \dfrac{r(l+1)h(l+1)(\delta_{l+1}-1)}{h(l)(\delta_l - 1)\delta_{l+1}}$.

The following theorem is valid.

Theorem 5. *Let T be a regular DM-tree of the height $H > 1$, $\mathbb{F}$ be the set of fragments making up the database, $\mathbb{R}$ be the set of all replicas of all fragments from the set $\mathbb{F}$ constructed using the normal replication function, $\mathrm{T}(\mathbb{F})$ be the size of the database in the tuples (it is assumed that all tuples are of identical length in bytes), and $\tau(\mathbb{R})$ be the total laboriousness of the tuplewise formation of all replicas disregarding noise. Then,*

$$\tau(\mathbb{R}) \approx k\mathrm{T}(\mathbb{F}),$$

where k is some $\mathbb{F}$-independent constant.

Proof can be found in [18].

This theorem demonstrates that with the use of the normal replication function the laboriousness of replica updating in the regular multiprocessor hierarchical system is proportional to the size of the updated part of the database, provided that the connectional network has a sufficient throughput.

3.6. Load Balancing

Let a request $\mathbb{Q}$ having n input relations be given, and let $\mathfrak{Q}$ be the parallel plan [19] of $\mathbb{Q}$. Each agent $Q \in \mathfrak{Q}$ has n input flows $s_1, \ldots, s_n$, and each flow $s_i (i = 1, \ldots, n)$ is defined by the four parameters:

(1) f_i—the indicator of the relation fragment;

(2) q_i—the number of segments over the interval to be processed;

(3) b_i—the number of the first segment in the processed interval;

(4) a_i—the balancing indicator: 1—balancing permitted; 0—balancing is forbidden.

The parallel agent Q may be either in the *active* or *passive* state. In the active state, Q successively reads and processes the tuples of all input flows. At that, in the course of processing the values of the parameters q_i and b_i are dynamically modified for all $i = 1, \ldots, n$. In the passive state, Q sleeps. At the initial stage of the request processing, the agent is initialized, and as the result, the parameters of all input flows are determined. Then, the agent is driven into the active state and starts processing of the fragments associated with its input flows. In each fragment, only those segments are processed that lie within the interval defined by the parameters of the flow associated with the given fragment. When all assigned segments in all input flows are processed, the agent passes to the passive state.

At execution of a parallel plan of request, some agents may complete their work and stay passive, whereas the other agents continue processing of their assigned intervals, which gives rise to the skew effects [20]. The following data replication-based *algorithm of load balancing* is proposed.

Let the parallel agent $\overline{Q} \in \mathfrak{Q}$ complete processing of its segments in all input flows and pass into the passive state, whereas the agent $\widetilde{Q} \in \mathfrak{Q}$ goes on with processing its portion of data, and let it be required to balance the loads. We refer to the idling agent $\overline{Q}$ as the leader and the overloaded agent $\widetilde{Q}$ as the outsider. In this situation, we perform the procedure of load balancing between the leader $\overline{Q}$ and outsider $\widetilde{Q}$ which lies in transferring part of the outstanding segments from $\widetilde{Q}$ to $\overline{Q}$. The balancing algorithm in terms of a C-like symbolic code is as follows:

```
/* Procedure of load balancing between parallel agents Q̄
(leader) and Q (outsider). */
ū = Node(Q̄); // indicator of node of agent Q̄.
pause Q; // drive Q into passive state.
for (i = 1; i <= n; i++) {
  if (Q.s[i].a == 1) {
    f_i = Q.s[i].f; // fragment processed by agent Q.
    r̄_i = Re (f_i, ū); // replica of fragment f_i on node ū.
    Δ_i = Delta (Q.s[i]); // number of transferred segments.
    Q.s[i].q− = Δ_i;
    Q̄.s[i].f = r̄_i;
    Q̄.s[i].b = Q.s[i].b + Q.s[i].q;
    Q̄.s[i].q = Δ_i;
  }else
    print ("Balancing forbidden");
};
activate Q; // drive Q to the active state
activate Q̄; // drive Q̄ to the active state.
```

At load balancing, the external balancing function *Delta* is used to calculate the number of segments of the corresponding input flow that are transferred from the outsider $\widetilde{Q}$ to the leader $\overline{Q}$.

To use efficiently the above load balancing algorithm, the following problems must be settled.

(1) In the presence of idling leader agents, select an outsider agent which will be the object of balancing. The method of selecting the outsider agent will be called the *outsider choice strategy*.

(2) Make decision about the number of outstanding data segments to be transferred to the leader from the outsider. The function to calculate this number will be called the *balancing function*.

Strategy of outsider selection. We define an outsider selection strategy which we will refer to as *optimistic*. At that, we rely on the partial mirroring method of Section 3.4. We consider a hierarchical multiprocessor system structured as a symmetrical DM-tree T. Let $\mathfrak{Q}$ be the parallel plan of the request $\mathbb{Q}$ and Ψ be the set of nodes of the DM-tree where the parallel plan $\mathfrak{Q}$ is executed. Let at some time instant during the request processing the leader agent $\overline{Q} \in \mathfrak{Q}$ located at the node $\overline{\psi} \in \Psi$ complete operation and pass to the passive state. It is required to select in the set of agents of the parallel plan $\mathfrak{Q}$ an outsider agent $\widetilde{Q} \in \mathfrak{Q}$ ($\widetilde{Q} \neq \overline{Q}$) to be aided by the leader agent $\overline{Q}$. We assume that the agent $\widetilde{Q}$ is located at the node $\widetilde{\psi} \in \Psi$ and $\widetilde{\psi} \neq \overline{\psi}$. We denote by $\widetilde{\overline{M}}$ the minimal subtree of the tree T containing the nodes $\overline{\psi}$ and $\widetilde{\psi}$.

We make use of the rating mechanism to select the outsider agent and in the course of balancing assign a rating defined by a real number to each agent of the parallel plan. The agent with the maximal positive rating is always selected as the outsider. If there are no such agents, then the released agent $\overline{Q}$ just completes its operation. If simultaneously more than one agent has the

maximal positive rating, then that is taken to which the load balancing procedure was not applied for a longer time.

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

$$\gamma(\widetilde{Q}) = \widetilde{a}_i \operatorname{sgn}\left(\max_{1 \leqslant i \leqslant n}(\widetilde{q}_i) - B\right) \lambda r(l(\widetilde{\overline{M}})) \sum_{i=1}^{n} \widetilde{q}_i,$$

where λ is a positive scaling factor regulating the impact of the replication coefficient $r(l)$ on the value of rating and B is a nonnegative integer defining the lower boundary of the number of segments that can be transferred at load balancing. We recall that $l(M)$ means the level of the subtree M on the tree T (see Section 3.1).

Balancing function Δ determines for each flow $\widetilde{s}_i$ of the outsider agent $\widetilde{Q}$ the number of segments transferred for processing to the leader agent $\overline{Q}$. In the simplest case, we may assume that

$$\Delta(\widetilde{s}_i) = \min\left(\lceil \widetilde{q}_i/2 \rceil, r(l(\widetilde{\overline{M}}))\mathrm{S}(\widetilde{f}_i)\right).$$

The function $\mathrm{S}(\widetilde{f}_i)$ introduced in Section 3.2 calculates the number of segments of the fragment $\widetilde{f}_i$. Therefore, the function Δ transfers half of the outstanding segments from $\widetilde{Q}$ to $\overline{Q}$ if only the replica of the fragment $\widetilde{f}_i$ in the node of the agent $\widetilde{Q}$ is not smaller than this value. Otherwise, only as many segments are transferred as there are segments in the corresponding replica of the fragment $\widetilde{f}_i$.

4. CONCLUSIONS

A model of the symmetrical multiprocessor hierarchical system was introduced. It describes a rather wide class of physical systems and gives a good mathematical grounds for determination of the data layout strategy in the multiprocessor hierarchies. An algorithm to generate a replica on the basis of logical decomposition of a relation fragment into equal segments was proposed for the symmetrical hierarchy. This algorithm underlies the partial mirroring method involving definition of the replication function which maps the hierarchy level into the replication coefficient defining the size of replica with respect to the replicated fragment. Theorems enabling estimation of the replica size and laboriousness of its generation disregarding noise were proved. A variant of the replication function was proposed where the laboriousness of replica updating in the multiprocessor hierarchical system is proportional to the size of the updated part of the database, provided that the connectional network has a sufficiently high throughput. A method of load balancing on the basis of the proposed method of partial mirroring was described.

The following lines of future research deserve mentioning. First, analytical estimation of the replica updating laboriousness with regard for noise. Second, it is planned to develop a DMM-based program simulating the hierarchical multiprocessor database system and use it to study experimentally the problems of data layout and load balancing. Third, for the clusters and Grid systems, it is planned to realize the above methods and algorithms in a prototype of the *Omega* parallel database control system [13]. Fourth, it is planned to study effectiveness of using different rating functions at load balancing. The following statistical information about the history of load balancing of the agent playing the role of assistant is planned to be used:

(1) what agents were aided by the given assistant in the course of request processing and

(2) how many times this assistant helped the given agent and what number of segments it received.

REFERENCES

1. Gray, J., Liu, D., DeWitt, D.J., and Heber, G., Scientific Data Management in the Coming Decade, *SIGMOD Record*, 2005, vol. 34, no. 4, pp. 34–41.
2. Graefe, G., Query Evaluation Techniques for Large Databases, *ACM Comput. Surveys*, 1993, vol. 25, no. 2, pp. 73–169.
3. Sokolinskii, L.B., Review of the Architectures of Parallel Database Systems, *Programmirovanie*, 2004, no. 6, pp. 49–63.
4. Bhide, A. and Stonebraker, M., A Performance Comparison of Two Architectures for Fast Transaction Processing, in *Proc. 4th Int. Conf. Data Engin.*, Los Angeles, 1988, pp. 536–545.
5. Bhide, A., An Analysis of Three Transaction Processing Architectures, in *Proc. 4th Int. Conf. Very Large Data Bases*, Los Angeles, 1988, pp. 339–350.
6. Bouganim, L., Florescu, D., and Valduriez, P., Dynamic Load Balancing in Hierarchical Parallel Database Systems, in *Proc. 22th Int. Conf. Very Large Data Bases*, Mumbai, 1996, pp. 436–447.
7. Xu, Y. and Dandamudi, S.P., Performance Evaluation of a Two-Level Hierarchical Parallel Database System, in *Proc. Int. Conf. Comput. Their Appl.*, Tempe, 1997, pp. 242–247.
8. DeWitt, D.J. and Gray, J., Parallel Database Systems: Future of the High-performance Database Systems, *SUBD*, 1995, no. 2, pp. 8–31.
9. Bitton, D. and Gray, J., Disk Shadowing, in *Proc. 4th Int. Conf. Very Large Data Bases*, Los Angeles, 1988, pp. 331–338.
10. Chen, S. and Towsley, D.F., Performance of a Mirrored Disk in a Real-Time Transaction System, in *Proc. 1991 ACM SIGMETRICS Conf. Measurement and Modeling Comput. Syst.*, San Diego, 1991; *Performance Evaluat. Rev.*, vol. 19, no. 1, pp. 198–207.
11. Mehta, M. and DeWitt, D.J., Placement in Shared-nothing Parallel Database Systems, *The VLDB J.*, 1997, vol. 6, no. 1, pp. 53–72.
12. Williams, M.H. and Zhou, S., Data Placement in Parallel Database Systems, in *Parallel Database Techniques*, IEEE Comput. Soc., 1998, pp. 203–218.
13. *Prototype of the Parallel Database Control System "Omega,"* manuscript on the site `http://omega.susu.ru/prototype/`.
14. Knuth, D.E., *The Art of Computer Programming.* vol. 3: *Sorting and Searching*, Reading: Addison-Wesley, 1969. Translated under the title *Iskusstvo programmirovaniya dlya EVM. T. 3: Sortirovka i poisk*, Moscow: Mir, 1978.
15. Knuth, D.E., *The Art of Computer Programming. Vol. 1, Fundamental Algorithms*, Reading, Massachusetts: Addison-Wesley, 1968. Translated under the title *Osnovnye algoritmy*, Moscow: Vil'yams, 2000.
16. Lu, H. and Tan, K.L., Dynamic and Load-balanced Task-oriented Database Query Processing in Parallel Systems, in *Proc 3rd Int. Conf. Extending Database Technology*, Vienna, 1992, pp. 357–372.
17. Omiecinski, E., Performance Analysis of a Load Balancing Hash-Join Algorithm for a Shared Memory Multiprocessor, in *Proc. 17th Int. Conf. Very Large Data Bases*, 1991, pp. 375–385.
18. Lepikhov, A.V. and Sokolinskii, L.B., Data Layout Strategy in the Multiprocessor Systems with Symmetrical Hierarchical Architecture, *Technical Report OMEGA12 of Yuurgu*, 2006, manuscript on the site: `http://omega.susu.ru/reports/TR12-06-07-89148-Y1N1.pdf`.
19. Kostenetskii, L.B., Lepikhov, A.V., and Sokolinskii, L.B., Some Organizational Aspects of Parallel Database Systems for the with Hierarchical Multiprocessor Architecture, in *Algoritmy i programmnye sredstva parallel'nykh vychislenii. Sb. nauch. tr.* (Algorithms and Software for Paralle Computations. Collected Papers), 2006, no, 9, pp. 42–84.
20. Maertens, H., A Classification of Skew Effects in Parallel Database Systems, in *Proc. 7th Int. Euro-Par Conf.*, 2001, pp. 291–300.

This paper was recommended for publication by V.M. Vishnevskii, a member of the Editorial Board