

# Analysis of Hierarchical Multiprocessor Database Systems<sup>\*</sup>

Pavel S. Kostenetskiy  
Southern Ural State University  
kostenetskiy@ya.ru

Leonid B. Sokolinsky  
Southern Ural State University  
sokolinsky@acm.org

## Abstract<sup>\*</sup>

*In this paper we propose a new approach to data distribution and load balancing in multiprocessor database systems with hierarchical architecture. A model of hierarchical database multiprocessor architecture is described. This model is called DMM. It allows us to simulate and analyze various multiprocessors configurations. An important subclass of multiprocessor hierarchies is considered. We call it symmetric hierarchies. For the symmetric hierarchies, a new strategy of data distribution is proposed. This strategy is based on the partial mirror technique. The analytical estimations for disk space overhead due to data replication are obtained. For the regular symmetric hierarchies, the theorems giving estimations of replica building overhead are proven. An efficient method for load balancing is proposed. This method exploits the partial mirror technique. Presented methods are designed for cluster and Grid systems.*

## 1. Introduction

Hierarchical multiprocessor architectures become more and more popular nowadays. In the multiprocessor architecture, processor devices, memory modules and disks are tied together by a hierarchical topology. The first level of hierarchy consists of processor cores disposed on one chip. The second level is build by multicore processors coupled in multiprocessor modules with shared memory (SMP). On the third level, SMP modules are combined into a cluster by high-speed interconnect. The fourth level is presented by Intra-Grids, each of which includes several clusters connected by LAN (Local Area Network). Several Intra-Grids can be combined into an Extra-Grid by WAN (Wide Area Network) and so on.

One of the most important applications for the multiprocessor systems is a parallel database system, which is able to maintain Terabytes of data [1]. Parallel database systems are discussed in many papers (see a survey [2]), however the issues concerning hierarchical database multiprocessor architectures have not been explored up to the present.

In this paper, we present an analysis of hierarchical multiprocessor database systems. The following subjects are emphasized: a simulation of the hierarchical multi-

processor architectures, data distribution and load balancing.

The hierarchical architectures generate a wide variety of configurations. Some classification of these architectures is given in [3]. Investigation of such architectures is difficult, since the process of constructing multiprocessor systems demands large financial expenditures connected with acquisition and reconfiguration of expensive equipment. Therefore it has become necessary to develop an accurate model of multiprocessor database system, which allows us to investigate various multiprocessor configurations without their hardware implementation. The investigation of one-level architectures for on-line transaction processing has been done by Stonebraker and Bhide [4, 5]. The simulation of some classes of two-level architectures is presented in [6, 7]. However, in general case the multilevel hierarchical architectures have not been explored.

This paper describes a new *Database Multiprocessor Model (DMM)*, which can be used for simulating and investigating the hierarchical architectures of parallel database systems with an arbitrary number of levels under OLTP (On-Line Transaction Processing) workload.

The problems of data distribution and load balancing for parallel database systems have being investigated in many papers (see e.g. [8, 9, 10]). But all these investigations were mainly aimed to one-level multiprocessor systems. In this paper we propose a new data distribution strategy for hierarchical systems and a load balancing algorithm based on partial mirroring technique.

The remainder of this paper has the following outline. Section 2 describes the DMM model, which includes hardware model, software model, cost model and transaction model. In Section 3 we propose a data distribution strategy and an algorithm for load balancing in hierarchical multiprocessor system. The formal definition of the symmetric multiprocessor hierarchy is introduced. We also describe a technique for data replication based on segments. The theorem for estimation of total replica size and the theorem for estimation of replica creation overhead are presented. An original algorithm for load balancing is described. This algorithm is based on the replication technique described in the paper. Finally, in Section 4 we summarize the major results of the paper and identify some directions for further research.

---

<sup>\*</sup> This work was supported by the Russian foundation for basic research (project 06-07-89148).

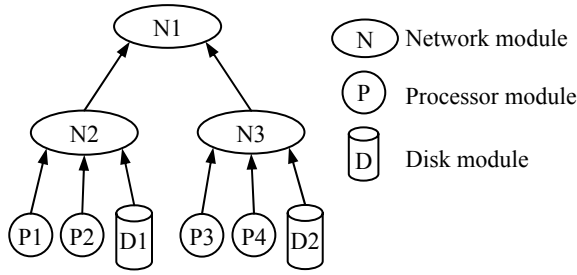


Figure 1. DM-tree example

## 2. Simulating hierarchical architectures

The DMM model includes the following sub-models: hardware model, software model, cost model and transaction model.

### 2.1. Hardware model

The hardware platform of parallel database system is represented by *DM-tree*. *DM-tree* is a directed tree whose nodes have one of the following types:

- 1) processor modules;
- 2) disk modules;
- 3) hub modules.

The edges of the tree represent the dataflow. The *processor module* is an abstracted representation of a physical processor. The *disk module* represents a physical hard disk device. The *hub modules* are used to represent a generic interconnect coupling different disk and processor devices. The DMM model does not provide a representation of memory modules, since in OLTP workload, the disk access time is much more than memory access time.

The structure of the *DM-tree* is limited by the following rules:

- 1) The root of *DM-tree* must be a hub module.
- 2) Disk and processor modules may not have any descendants, and thus are always the leaf nodes of *DM-tree*.

An example of *DM-tree* is shown in Figure 1.

### 2.2. Software model

The smallest unit of measuring data in the DMM is defined as a “*packet*”. We will assume that all packets have the same size. The header of packet includes sender address, the recipient address and some other information. Transfer of a packet corresponds to transfer of one or several tuples in the real databases system. Any processor module can exchange data with any disk module. Disk modules and hub modules have queue buffers, which are used for packet transfer.

```

Remove packet  $E$  from queue of  $N$ ;
if  $\alpha(E) \notin T(N)$  then
    Put  $E$  in queue  $F(N)$ ;
else
    Find maximum subtree  $U$  of  $T(N) : \alpha(E) \in U$ ;
    if  $T(\alpha(E)) = U$  then
        if  $\alpha(E) \in P$  then
             $r(\alpha(E))--$ ;
        else
            Put  $E$  in queue  $\alpha(E)$ ;
        end if
    else
        Put  $E$  in queue  $R(U)$ ;
    end if
end if

```

Figure 2. Hub module algorithm

The DMM model assumes the asynchronous transfer mode. It means that the processor module may start a new exchange without waiting the completion of previous one. However, we suppose that processor module may have not more than  $s_r$  uncompleted read operations and not more than  $s_w$  uncompleted write operations.

In the DMM model, data processing is divided into discrete time intervals called *ticks*. The tick can be defined as a predetermined sequence of steps, which will be described below.

Let  $\mathbf{P}$  denotes a set of all processor modules in *DM-tree*,  $\mathbf{D}$  – a set of all disk modules,  $\mathbf{N}$  – a set of all hub modules and  $\mathbf{M} = \mathbf{P} \cup \mathbf{D} \cup \mathbf{N}$  – a set of all nodes in *DM-tree*.

Let  $M \in \mathbf{M}$ . We will use the following denotations:  $F(M)$  – parent module of node  $M$ ,  $T(M)$  – subtree, which has  $M$  as a root.

**Processor module**  $P \in \mathbf{P}$  may activate read/write operations. Let's define their semantic as follows.

**Read Operation.** Let processor module  $P$  has to read packet  $E$  from disk module  $D \in \mathbf{D}$ . If  $P$  has already activated  $s_r$  uncompleted read operations then  $P$  has to be suspended. Otherwise, the packet  $E$  has to be put in the queue buffer of disk  $D$ . At that case, packet  $E$  has  $\alpha(E) = P$  as a recipient address and  $\beta(E) = D$  as a sender address.

**Write Operation.** Let processor module  $P$  has to write packet  $E$  to the disk module  $D \in \mathbf{D}$ . If  $P$  has already activated  $s_w$  uncompleted write operations then  $P$  has to be suspended. Otherwise, the packet  $E$  has to be put in the queue buffer of parent hub module. At that case, packet  $E$  has  $\alpha(E) = D$  as a recipient address and  $\beta(E) = P$  as a sender address.

**Hub module**  $N \in \mathbf{N}$  permanently transfers packets through interconnect. It performs an algorithm, which is shown in Figure 2.

```

Remove packet  $E$  from queue of module  $D$ ;
if  $\alpha(E) \in D$  then
     $w(\beta(E))--$  ;
else
    Put  $E$  in the parent queue;
end if
    
```

**Figure 3. Disk module algorithm**

There,  $E$  is a packet,  $\alpha(E)$  is the recipient address of packet  $E$ ,  $T(N)$  is a subtree, which has  $N$  as a root,  $F(N)$  is the parent module for  $N$ ,  $\mathbf{P}$  is a set of all processor modules,  $r(P)$  is the number of uncompleted read operations for processor module  $P$ ,  $R(U)$  is the root of a subtree  $U$ .

**Disk module**  $D \in \mathbf{D}$  permanently executes read/write operations. It performs an algorithm, which is shown in Figure 3. There,  $\beta(E)$  is a sender address.

In the *DMM* model, data processing is divided into discrete time intervals called *ticks*. The tick can be defined as a predetermined sequence of the following steps:

- 1) each hub module handles all packets, which are waiting for transfer;
- 2) each active processor module performs one read or write operation;
- 3) each disk module handles one packet from its queue.

It is obvious that in this case, the queue of any hub module may contain not more than  $|\mathbf{P}|+|\mathbf{D}|$  packets, and the queue of any disk module may contain not more than  $s_{r,w}|\mathbf{P}|$  packets.

### 2.3. Cost model

Each module  $M \in \mathbf{M}$  has a *cost coefficient*  $h_M \in \square$ ,  $1 \leq h_M < +\infty$ .

The time needed for a processor to process one packet for OLTP applications is roughly  $10^5$ - $10^6$  times smaller, than the time it takes to transfer data to or from a hard disk, or from the network. Therefore, for all processor modules:  $h_p = 1$ ,  $\forall P \in \mathbf{P}$ .

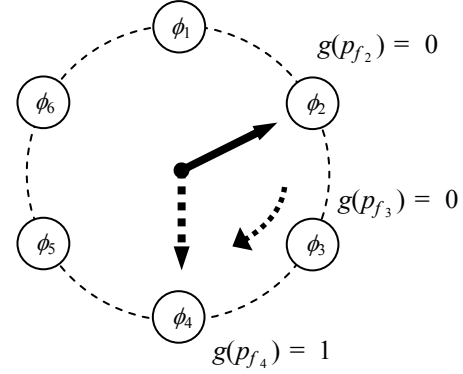
Hub module  $N$  can send more than one packet in each tick. Therefore the following interference function is associated with each hub module  $N \in \mathbf{N}$ :

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

There,  $m_i^N$  – is the number of packets transferred by  $N$  in the  $i$ -th tick;  $\delta_N > 1$  – scale coefficient. Therefore, the time it takes for hub module  $N$  to process  $i$ -th tick can be calculated by the formula:

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

The total time spent by the system to process a mix of transactions over  $k$  ticks can be calculated by the formula:


**Figure 4. Active Process Choosing**

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

### 2.4. Transaction model

*DMM* model admits processing the mix of  $k$  parallel transactions on one processor. Each transaction  $Z_i$  ( $i=1, \dots, k$ ) is represented by definition of two process groups  $\rho$  and  $\omega$ :  $Z_i = \{\rho_i, \omega_i\}$ . Group  $\rho_i$  includes *read* processes. Group  $\omega_i$  includes *write* processes. These processes are an abstract representation of read/write operations performed during transaction processing. The mix of transactions is defined by the formula:

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

Each process  $\phi \in \Phi$  has a probability  $p_\phi$  of disk access. This probability is calculated by the discrete random variable function  $g(p_\phi)$ , which defined by the following distribution law:

$g$	1	0
$p$	$p_\phi$	$1-p_\phi$

On every tick, each active processor module has to process I/O operation. In accordance to this, the active processor module has to choose some process  $\phi \in \Phi$  and perform write or read operation over the disk associated with  $\phi$ . Let us call such process active. Process  $\phi$  becomes active if function  $g(p_\phi)$  returns 1.

The choice of active process performs as follows. All processes from  $\Phi$  are organized as a circular list. The current element pointer is maintained. To choose a process as active, we should move the pointer to the next element  $\phi_i$  and calculate function  $g(p_{\phi_i})$ . If  $g(p_{\phi_i}) = 1$ , we choose  $\phi_i$ , otherwise we move the pointer to the next element  $\phi_{i+1}$  and so on (see Figure 4).

### 3. Data distribution and load balancing

In this section, we present a data distribution strategy and an algorithm for load balancing in hierarchical multiprocessor system.

#### 3.1. Symmetric hierarchies

The multiprocessor system architecture is a key factor for developing the data distribution strategy. In this Section, we will build the *symmetric model* of hierarchical multiprocessor database systems. The symmetric model is based on the notion of *DM-tree* introduced in Section 2.1.

Let us give a definition of *DM-trees isomorphism*. Let  $E_T$  be a set of edges of *DM-tree*  $T$ ;  $h_M$  – the cost coefficient of node  $M \in M_T$  of *DM-tree*  $T$ . *DM-tree*  $A$  is *isomorphic* to *DM-tree*  $B$  if there exist bijective mappings  $f: M_A \rightarrow M_B$  and  $g: E_A \rightarrow E_B$  such that:

- 1) node  $v$  is an endpoint of edge  $e$  in tree  $A$  if and only if node  $f(v)$  is an endpoint of edge  $g(e)$  in tree  $B$ ;
- 2) node  $w$  is an initial point of edge  $e$  in tree  $A$  if and only if node  $f(w)$  is an initial point of edge  $g(e)$  in tree  $B$ ;
- 3)  $P \in P_A \Leftrightarrow f(P) \in P_B$ ;
- 4)  $D \in D_A \Leftrightarrow f(D) \in D_B$ ;
- 5)  $N \in N_A \Leftrightarrow f(N) \in N_B$ ;
- 6)  $h_M = h_{f(M)}$ .

Let us define *node level* in a recursive manner. The level of the root of *DM-tree*  $T$  is equal to zero. The level of any other node is one unit larger than the level of the root of minimal subtree of tree  $T$ , which contains this node.

Let us define the *level*  $l(M)$  of a subtree  $M$  of a tree  $T$  as the level of the root of the subtree  $M$  in the tree  $T$ .

Two subtrees of the same level are called *adjacent* if their roots are brothers.

Let us define the *height* of directed tree  $T$  as the maximum path length in that tree.

We call the *DM-tree*  $T$  with height  $H$  as *symmetric* if the following conditions hold:

- 1) any two adjacent subtrees with level  $l < H$  are isomorphic trees;
- 2) any subtree with level  $H-1$  contain exactly one disk and one processor.

The second condition represents an abstract model of SMP system in the sense that, in the multiprocessor hierarchies, all the processors of an SMP system can be considered as one *mega processor* and all the disks as one *mega disk*. Obviously, load balancing for SMP system must be solved by essentially different techniques, since the SMP system has shared memory and each disk can be accessed by any processor. We define *node degree* as the

number of edges, which end into this node. In symmetric tree, the nodes of the same level have the equal degree called the *level degree*.

#### 3.2. Data fragmentation and segmentation

The database placement in nodes of a hierarchical multiprocessor system is specifying in the following way.

Each relation is divided into disjoint horizontal fragments, which are partitioned into different disk modules. We will suppose that fragment's tuples are ordered in some way called *natural*.

At the logical level, each fragment is divided into a sequence of fixed-length *segments*. Segment length is measured in tuples and is a fragment attribute. Partitioning of segments is performed according to natural order and always begins with the first tuple. In accordance with this, the last segment may be incomplete. The number of segments in fragment  $F$  is denoted by  $S(F)$  and can be computed by the formula:

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

where  $T(F)$  is the number of tuples in fragment  $F$ ,  $L(F)$  – the segment length assigned to fragment  $F$ .

#### 3.3. Data replication

Let us allocate fragment  $F_0$  into disk module  $D_0 \in D_T$  of multiprocessor hierarchical system  $T$ . We suppose that in each disk module  $D_i \in D_T$  ( $i > 0$ ) a partial replica  $F_i$  (can be empty) of fragment  $F$  is allocated.

A segment is the smallest unit of data fragmentation. The segment length of replica is always equal to the segment length of replicated fragment:

$$L(F_i) = L(F_0), \quad \forall D_i \in D_T.$$

The size of replica  $F_i$  is specified by the *replication factor*

$$\mu_i \in [0, 1], \quad 0 \leq \mu_i \leq 1,$$

which is an attribute of replica  $F_i$  and is computed by the formula:

$$T(F_i) = T(F_0) - \lceil (1 - \mu_i) \cdot S(F_0) \rceil \cdot L(F_0). \quad (2)$$

The *natural order of tuples in replica*  $F_i$  is defined by the natural order of tuples in fragment  $F_0$ . Here, the following formula determines the *first tuple number*  $N$  of replica  $F_i$ :

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

If  $F_i$  is empty we obtain  $N(F_i) = T(F_0) + 1$  that corresponds to the "end-of-file" indicator.

Described data replication technique permits the use of a simple and effective algorithm of load balancing in mul-

tiprocessor hierarchies. Such algorithm is described in Section 3.6.

### 3.4. Partial mirroring technique

Let  $T$  be a symmetric  $DM$ -tree with height  $H > 1$ . We introduce a *replication function*  $r(l)$ , which maps level  $l < H$  to the replication factor  $\mu_l = r(l)$ . Let us assume, that  $r(H-1) = 1$ . This is motivated by the following. Hierarchy level  $H-1$  contains subtrees with height equal to 1, which correspond to SMP systems. In the SMP system, all the disks can be equally accessed by each processor. Therefore, physical replication is not needed in the SMP system. On the logical level, load balancing in SMP system can be managed by segmenting the source fragment, i.e. the fragment acts as its own replica.

Let us allocate fragment  $F_0$  to disk module  $D_0 \in \mathbf{D}_T$ . In order to form replica  $F_i$  in disk  $D_i \in \mathbf{D}_T$  ( $i > 0$ ), the following technique (called a *partial mirroring technique*) can be used. We build a sequence of subtrees for tree  $T$ :

$$\{M_0, M_1, \dots, M_{H-2}\}, \quad (3)$$

which satisfies the following properties:

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

for each  $0 \leq j \leq H-2$ . In this formula,  $l(M_j)$  is the level of subtree  $M_j$ . Obviously, such a sequence exists for any symmetric tree  $T$ . Let us find the largest index  $j \geq 1$  such that

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

We assume

$$\mu(F_i) = r(j). \quad (4)$$

To build replica  $F_i$  in disk  $D_i$  we use the algorithm described in Section 3.3 with the replication factor defined by formula (4). The following theorem gives estimation for the total size of all replicas of the fragment.

**Theorem 1.** *Let  $T$  be a symmetric  $DM$ -tree having height  $H > 1$ . Let fragment  $F_0$  be allocated to disk module  $D_0 \in \mathbf{D}_T$ . Let  $\delta_l$  denote the degree of level  $l$  of tree  $T$ . Let us denote the total count of tuples in all replicas of*

*fragment  $F_0$  as  $R(F_0) = \sum_{i=1}^{|\mathbf{D}_T|} T(F_i)$ . Then*

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

The theorem proof can be found in [11].

### 3.5. Choosing the replication function

When choosing the replication function  $r(l)$ , it is reasonable to take into account the overhead coefficients of the  $DM$ -tree nodes. Obviously, that all the nodes in a symmetric  $DM$ -tree have the same overhead coefficient  $h(l)$ , which we call the *overhead of level  $l$* .

We call symmetric  $DM$ -tree  $T$  regular if for any two levels  $l$  and  $l'$  of tree  $T$  the following statement is true:

$$l < l' \quad \Rightarrow \quad h(l) \geq h(l'), \quad (5)$$

i.e., the overhead coefficient increases with increasing the hierarchy level.

The estimation for the overhead of tuple-building all the replicas of the fragment without interference can be obtained by the following theorem.

**Theorem 2.** *Let  $T$  be a symmetric  $DM$ -tree having height  $H > 1$ . Let fragment  $F_0$  be allocated to disk module  $D_0 \in \mathbf{D}_T$ . Let  $\delta_l$  denote the degree of level  $l$  of tree*

*$T$ . Let  $\tau(F_0) = \sum_{i=1}^{|\mathbf{D}_T|} t(F_i)$ . Then*

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

The theorem proof can be found in [11].

Let us define *normal replication function*  $r(l)$  in a recursive manner:

- 1) if  $l = H-2$  then  $r(H-2) = \frac{1}{h(H-2)(\delta_{H-2} - 1)}$ ;
- 2) if  $0 \leq l < H-2$  then  $r(l) = \frac{r(l+1)h(l+1)(\delta_{l+1} - 1)}{h(l)(\delta_l - 1)\delta_{l+1}}$ .

The following theorem is valid.

**Theorem 3.** *Let  $T$  be a regular  $DM$ -tree having height  $H > 1$ . Let  $\mathbf{F}$  be the set of fragments composing the database. Let  $\mathbf{R}$  be the set of all the replicas of all the fragments of  $\mathbf{F}$  built by using the normal replication function. Let  $T(\mathbf{F})$  be the size of the database in tuples (here, we assume that all tuples are of same size in bytes). Let  $\tau(\mathbf{R})$  be the total overhead of tuple-building of all the replicas without interference. Then*

$$\tau(\mathbf{R}) \approx k T(\mathbf{F}).$$

Here,  $k$  is a constant and is independent of  $\mathbf{F}$ .

The theorem proof can be found in [11].

Theorem 3 shows that, by using a normal replication function, the overhead of replica update in a regular hierarchical multiprocessor system is proportional to the size of updating part of the database under the condition, that the interconnect has sufficient bandwidth.

```

/* The procedure of load balancing for parallel
agents  $\bar{Q}$  (leader) and  $\tilde{Q}$  (outsider). */
 $\bar{u}$  = Node( $\bar{Q}$ ); // the processor node of  $\bar{Q}$ 
pause  $\tilde{Q}$ ; // set  $\tilde{Q}$  to the passive state
for (i=1; i<=n; i++) {
  if ( $\tilde{Q}.s[i].a == 1$ ) {
     $f_i = \tilde{Q}.s[i].f$ ; // the fragment processed by  $\tilde{Q}$ 
     $\bar{r}_i = \text{Re}(f_i, \bar{u})$ ; // Replica of  $f_i$  on node  $\bar{u}$ .
     $\Delta_i = \text{Delta}(\tilde{Q}.s[i])$ ; // Count of reassigned
                                // segments

     $\tilde{Q}.s[i].q- = \Delta_i$ ;
     $\bar{Q}.s[i].f = \bar{r}_i$ ;
     $\bar{Q}.s[i].b = \bar{Q}.s[i].b + \tilde{Q}.s[i].q$ ;
     $\bar{Q}.s[i].q = \Delta_i$ ;
  } else
    print("Balancing is not allowed");
};
activate  $\tilde{Q}$ ; // turn  $\tilde{Q}$  to the active state

```

Figure 5. Load balancing algorithm.

### 3.6. Load balancing

Let  $Q$  be a query having  $n$  input relations. Let  $\mathbf{Q}$  be a parallel plan of query  $Q$ . Each agent  $Q \in \mathbf{Q}$  has  $n$  input streams  $s_1, \dots, s_n$ . Each stream  $s_i$  ( $i=1, \dots, n$ ) is defined by the following four parameters:

- 1)  $f_i$  is a pointer to the fragment of relation;
- 2)  $q_i$  is a count of segments to be processed;
- 3)  $b_i$  is a number of the first segment;
- 4)  $a_i$  is an indicator of balancing: 1 – balancing is allowed, 0 – balancing is not allowed.

Parallel agent  $Q$  may be in one of the following two states: *active* or *passive*. In the active state, agent  $Q$  reads sequentially and processes tuples from all the input streams. The values of parameters  $q_i$  and  $b_i$  ( $i=1, \dots, n$ ) are being dynamically changed during the process. In the passive state, agent  $Q$  does nothing. On the first stage of query processing, the agent has to be initialized to assign initial values to the parameters of all input streams. Then the agent state is set to active and the agent begins to process the fragments associated with its input streams. In each fragment, those segments will only be processed, which belongs to the interval determined by the stream parameters associated with given fragment. If all assigned segments are processed, the agent becomes passive.

During parallel plan processing, some agents can finish their work and be in the passive state. At the same time, other agents are processing the intervals assigned to them. So we get the *processor load imbalance*. We propose the following *load balancing algorithm* exploiting data replication.

Let parallel agent  $\bar{Q} \in \mathbf{Q}$  have finished processing of assigned segments in all input streams and has been set into passive state, while agent  $\tilde{Q} \in \mathbf{Q}$  is still processing its data interval. So we have a situation when load balancing is needed. Let us call the idle agent  $\bar{Q}$  as a *leader* and the overloaded agent  $\tilde{Q}$  as an *outsider*. In order to achieve load balancing, we perform a procedure shown in Figure 5. This procedure passes a part of the not processed segments from agent  $\tilde{Q}$  to agent  $\bar{Q}$ . There, function *Delta* calculates a count of segments to be passed from one agent to another.

To use described above algorithm effectively, we have to solve the following two tasks.

1. We need some strategy for outsider choosing.
2. We must define a way function *Delta* calculates a count of segments to be passed.

**Strategy of outsider choosing.** Let us define an *optimistic strategy* of outsider choosing. Let  $T$  be a symmetric *DM-tree* representing the hierarchical multiprocessor system. Let  $\mathbf{Q}$  be the parallel plan of query  $Q$ ,  $\Psi$  – the set of *DM-tree* nodes processing the parallel plan  $\mathbf{Q}$ . Let leader-agent  $\bar{Q} \in \mathbf{Q}$  located on the node  $\bar{\psi} \in \Psi$  have finished its work and been set into passive state. We have to choose some outsider agent  $\tilde{Q} \in \mathbf{Q}$  ( $\tilde{Q} \neq \bar{Q}$ ) needed a help by leader agent  $\bar{Q}$ . Let agent  $\tilde{Q}$  be located on node  $\tilde{\psi} \in \Psi$  and  $\tilde{\psi} \neq \bar{\psi}$ . Let  $\tilde{M}$  be the minimal subtree of tree  $T$ , which includes nodes  $\bar{\psi}$  and  $\tilde{\psi}$ .

For choosing agent outsider, we use the rating technique. During the balancing procedure, every parallel agent is assigned a rating specified by a real number. As an outsider, the agent having the maximal positive rating has to be chosen. If there are no such agents,  $\bar{Q}$  has to be terminated. If several agents have a positive rating being equal to the maximal one, the least recently helped agent should be chosen as outsider.

In case of using the optimistic strategy, we use *rating function*  $g : \mathbf{Q} \rightarrow \mathbb{R}$ , defined as follows:

$$g(\tilde{Q}) = \tilde{a}_i \operatorname{sgn}\left(\max_{1 \leq i \leq n} (\tilde{q}_i) - B\right) \lambda r(l(\tilde{M})) \sum_{i=1}^n \tilde{q}_i.$$

Here,  $\lambda$  is a positive scale factor, which regulate influence of replication factor  $r(l)$  on the rating value;  $B$  is a positive integer determining the lowest value of segment count, which can be passed during load balancing. Let us remind that  $l(M)$  is the level of subtree  $M$  in tree  $T$ .

**Balancing function.** For every stream  $\tilde{s}_i$ , balancing function  $\Delta$  determines the count of segments having to be passed from outsider agent  $\tilde{Q}$  to leader agent  $\bar{Q}$ . In the simplest case, we can define

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

Function  $S(\tilde{f}_i)$ , introduced in Section 3.2, calculates the total segment count of fragment  $\tilde{f}_i$ .

#### 4. Conclusion

In this paper, we introduced the symmetric model of hierarchical multiprocessor system. It represents a wide class of real multiprocessor configurations and can be used as a mathematical base for defining the data distribution strategy in multiprocessor hierarchies. An algorithm for replica building in the symmetric hierarchy is proposed. This algorithm is based on logical fragment stripping by the equal size segments. Using this algorithm, the partial mirroring technique is developed. This technique assumes a definition of replication function. The replication function maps the hierarchy level to the replication factor, which defines the replica size of the fragment. The theorems allowing to estimate the replica sizes are proposed. Also, we presented the theorems allowing to estimate replica building overhead without interference. Further, the load balancing algorithm based on proposed partial mirroring technique was described.

The main directions of future work are the following. First, it is interesting to obtain an analytical estimation of replica update overhead including interference. Second, we plan to develop a simulator based on *DM*-model to evaluate various configurations of hierarchical multiprocessor database systems. Third, we suppose to implement proposed techniques and algorithms in Omega DBMS prototype, which is designed for cluster and Grid systems.

#### 5. References

- [1] Gray J., Liu D., DeWitt D. J., Heber G. Scientific Data Management in the Coming Decade // SIGMOD Record. 2005. V. 34. No. 4. P. 34-41.
- [2] Graefe G. Query evaluation techniques for large databases // ACM Comput. Surveys. 1993. V. 25. No. 2. P. 73-169.
- [3] Sokolinsky L.B. Survey of Architectures of Parallel Database Systems // Programming and Computer Software. -2004. - Vol. 30, No. 6. -P. 337-346.
- [4] Bhide A., Stonebraker M. A Performance Comparison of Two Architectures for Fast Transaction Processing // Proceedings of the Fourth International Conference on Data Engineering, February 1-5, 1988, Los Angeles, California, USA. IEEE Computer Society. -1988. -P. 536-545.
- [5] Bhide A. An Analysis of Three Transaction Processing Architectures // Fourteenth International Conference on Very Large Data Bases (VLDB'88), August 29 - September 1, 1988, Los Angeles, California, USA, Proceedings. Morgan Kaufmann. -1988. -P. 339-350.
- [6] Bouganim L., Florescu D., Valduriez P. Dynamic Load Balancing in Hierarchical Parallel Database Systems // VLDB'96, Proceedings of 22th International Conference on Very Large

Data Bases, September 3-6, 1996, Mumbai (Bombay), India. Morgan Kaufmann. -1996. -P. 436-447.

[7] Xu Y., Dandamudi S.P. Performance Evaluation of a Two-Level Hierarchical Parallel Database System // Proceedings of the Int. Conf. Computers and Their Applications, Tempe, Arizona. -1997. -P. 242-247.

[8] Bitton D., Gray J. Disk Shadowing // Fourteenth International Conference on Very Large Data Bases, August 29 - September 1, 1988, Los Angeles, California, USA, Proceedings. Morgan Kaufmann. -1988. -P. 331-338.

[9] Chen S., Towsley D.F. Performance of a Mirrored Disk in a Real-Time Transaction System // 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, San Diego, California, USA, May 21-24, 1991, Proceedings. Performance Evaluation Review. -May 1991. -Vol. 19, No. 1. -P. 198-207.

[10] Mehta M., DeWitt D.J. Data Placement in Shared-Nothing Parallel Database Systems // The VLDB Journal. -January 1997. -Vol. 6, No. 1. -P. 53-72.

[11] Sokolinsky L.B. Data Placement Strategy in Hierarchical Symmetric Multiprocessor Systems. Technical report OMEGA12. -Chelyabinsk: Southern Ural State University, 2006.