Proceedings of the International Multiconference on
Computer Science and Information Technology, pp. 345 – 354

ISSN 1896-7094

Selection of Indexing Structures in Grid Data Warehouses

Marcin Gorawski [1] , Michał Gorawski [1] , Sławomir Bańkowski [1]

[1]Silesian University of Technology, Institute of Computer Science,
Akademicka 16, 44-100 Gliwice, Poland
{Marcin.Gorawski, Michal.Gorawski, Slawomir.Bankowski}@polsl.pl

Abstract. Data warehouse systems service larger and larger sets of data. Effective data indexing is not sufficient, because one system node is unable to store this many quickly flowing data. More and more common solution is distribution of an indexing structure among several system nodes. Indexing structures presented in the following article are based on a grid technology. They assure good performance on every node, and their diversity allows adjusting to any data type. Presented system offers effective data distribution and index management on many system nodes connected via network.

1 Introduction

Data indexing problem was described in many papers, which mainly focused on general use of one and two-dimensional indexes. R-trees family, B-trees, MVB (Multiversional B)-trees, XBR trees, aP-trees and hash tables are popular and effective means of improving record search in a database [1,2]. Indexing structures presented in this paper are adjusted to spatio-temporal sensor/counter data (readings from telemetric counters (water, gas, heat and electric energy))[3]. In a data warehouse where data set size can be over hundred millions range queries have long response time [4]. The above mentioned indexes cannot speed up aggregation process and are useless in this case, while RMVB (a compilation of R-tree and MVB-tree), STCAT (Spatio Temporal Cup Aggregate Tree) and MDPAS (Multidimensional Pack Aggregate Structure) structures provide a considerable speed up of such queries execution. Data from sensors/counter are placed in a certain position in space (described with x,y,z coordinates). Every counter reading has its measurement time and measurement type (water, gas, heat and electric energy). In such environment aggregated values are more important than single readings, hence structures designed to store those data have to support data aggregation and efficient data storing. Indexing structures presented in the following paper are based on a parallel [5, 6, 7] grid technology [8,9]. The Grid Data Warehouse System supported by Software Agents (GDW ^{SA} (t)) [10] cooperates with Spatio-Temporal Data Warehouse (STDW) [3, 4, 5] with cascaded star data model. The GDW ^{SA} (t) system services telemetric data from market of a media recipients and suppliers. Detailed description of STCAT, MDPAS and MDPAS index in GDW^{SA} will be presented followed by performance tests, which proves the structures effectiveness.

2 Spatial indexes

Efficient indexing is a base for every data warehouse system. RMVB, STCAT and MDPAS allow effective data storing and ensure considerable speed-up of spatio-temporal queries. Presented indexes are adapted to a data model called cascade star schema. Sensor/counter data are stored in an adequate structures. A main drawback of

the presented solutions is necessity of possession certain information about the used data. Along, with a definition of dimension quantity, which must be defined in the beginning, space range and temporal and spatial bucket size should be defined. The performance of the presented structures strongly depends on a good definition of above mentioned parameters, and if the division is correct, the structures proves to be very effective. Designed indexes are database-independent (data is stored in a binary file on a hard drive).

2.1 R-MVB structure

R-MVB index (Fig. 1) consist of two indexes – R-tree[6] and MVB-tree[5]. The first index ensures responses to a spatial queries (list of spatial objects in range query). Second index calculates aggregations for a certain spatial object in a certain time span. Apart from a position of added spatial objects, R-tree is optimized for a range-spatial search.

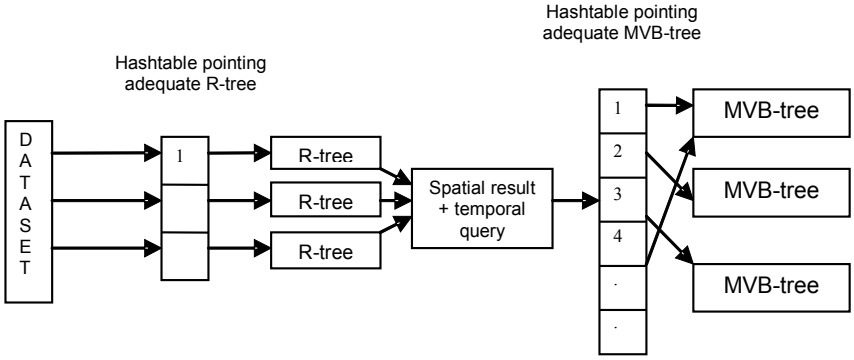

Fig. 1. R-MVB-tree

Such a combination ensures an efficient support for spatio-temporal queries. In this structure the data loading process does not depend on the data characteristic. This means that no knowledge is needed when the index is being build, however building time is multiple times longer in comparison with database or other investigated structures.

2.2 STCAT structure

STCAT consists of quad tree and time-aggregated tree. Data loading is very fast and range queries are supported. Every leaf of a multidimensional tree (quad tree) stores list of time tree roots. Range query selects an adequate time tree for search. Time trees are aggregated trees adapted to allow fast query responses (Fig. 2).

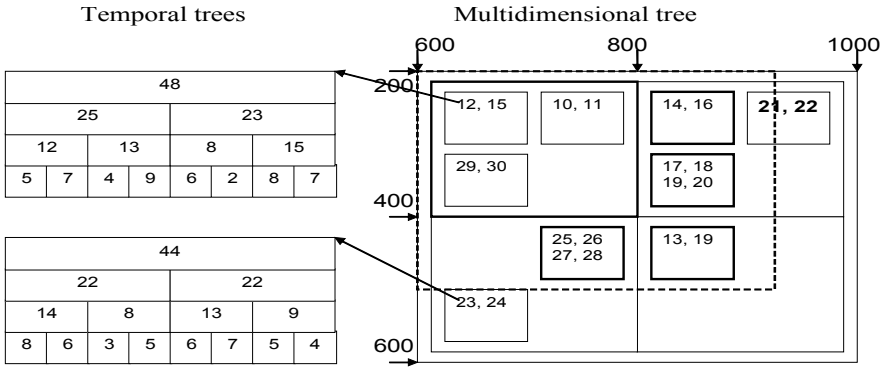

Fig. 2 . STCAT tree

2.2.1 Space division's influence on STCAT's performance

STCAT's structure has certain inconveniences. When a structure is created, basic parameters, like a time bucket size and multidimensional leaf size should be determined. Correct choice of above-mentioned parameters enables multiple speed-up comparing to a database. If parameters are set inappropriately, structure performance could be worse than optimal.

A question can be asked if a low-level or a high-level division decreases the whole structure's performance, and which of them is default, if a format of data loaded into the tree is unknown.

2.1.2 Low-level division

Low-level division means that multidimensional tree height is significantly larger than height that is really needed for a certain data. An average number of accesses during range search can be calculated as follows :

$$
\begin{aligned}
& m^w >> k \Rightarrow h = \lceil \log_2 m \rceil; m_h = 2^h \\
& h_k = \left\lceil \frac{\log_2 k}{w} \right\rceil; h_d = h - h_k = \log_2 m - \frac{\log_2 k}{w} \\
& f_i = 4 \cdot w \cdot m_i^{w-1}; f_A = \sum_{i=1}^{h} f_i \\
& f_A \le \frac{2^w \cdot (4 \cdot w \cdot m^{w-1})}{2^{w-1} - 1}; f_B = 4 \cdot w \cdot m^{w-1} \cdot h_d; fc = 4 \log_2 L \\
& f_D = f_A \cdot (1 + f_C) + f_B
\end{aligned}
\tag{1}
$$

where: m – number of parts while dividing every dimension; w – number of dimensions; k – number of added time trees(number of counters); h_k – level, on which maximal number of leafs equals k; h – tree height; h_d – number of levels below h_k; L –total number of multidimensional tree leaves; f_i – number of multidimensional tree leaves, which are accessed on level i; f_A – number of multidimensional tree leaves, which are accessed above h_k level; f_B – number of multidimensional tree leaves, which are accessed below h_k level; f_C – number of access to time tree; f_D – average number of accesses during the structure searching.

2.1.3 High-level division

High-level division has to be considered in terms of numbers of counters stored in one leaf. If an inadequate range setting causes that all counters are positioned in the root, then the structure becomes a one-way list. In the most pessimistic case the data loading process and data searching process will be more time consuming. Slowdown will equal quotient of a number of time trees and number of roots stored in one leaf (typically 20).

For example: for 10000 counters 35 times slowdown will occur (500 elements list in comparison to tree of height 14). Assuming dividing every dimension into parts, where:

$$
\begin{aligned}
& m^w >> k \Rightarrow h = \lceil \log_2 m \rceil; m_i = 2^i \\
& s = \frac{k}{m^w} \\
& f_i = 4 \cdot w \cdot m_i^{w-1}; f_A = \sum_{i=1}^{h} f_i \\
& f_A \le \frac{2^w \cdot (4 \cdot w \cdot m^{w-1})}{2^{w-1} - 1}; f_B = 4 \cdot w \cdot m^{w-1} \cdot s; fc = 4 \log_2 L \\
& f_D = f_A \cdot (1 + f_C) + f_B
\end{aligned}
\tag{2}
$$

where: m – actual space division (small value because of high-level division); s – one-way list size which is created from multidimensional tree leaves; f_A – number of accessed multidimensional tree leaves; f_B – number of accessed multidimensional tree leaves, created from one-way list; f_C – number of accesses to the time tree; f_D – average number of accesses during the structure searching.

2.3 MDPAS structure

MDPAS tree (Multi Dimensional Pack Aggregate Structure) is a hierarchical structure dividing data in a certain, chosen manner. Parameters like number of columns, their types and division can be set optionally to match certain data. During data loading, for every row a hash number is calculated (3):

$$
\begin{aligned}
&range_i = \max_i - \min_i \\
&parts_i = 0 \Rightarrow cup_i = 1; parts_i \neq 0 \Rightarrow cup_i = \frac{range_i}{parts_i} \\
&m_i = \prod_{j=1, parts_j \neq 0}^{i-1} parts_j \\
&parts_i = 0 \Rightarrow scale_i = 0; parts_i \neq 0 \Rightarrow scale_i = m_i \\
&hash_j = \sum_{i=0}^{M-1} \frac{(value_{i,j} - \min_i) \cdot scale_i}{cup_i}
\end{aligned}
\tag{3}
$$

where: $\min_i$ – minimal i parameter's value, $\max_i$ – maximal i parameter's value, parts – number of column division parts, cup_i – range value for i column, $scale_i$ – scalability factor for the i-th column.

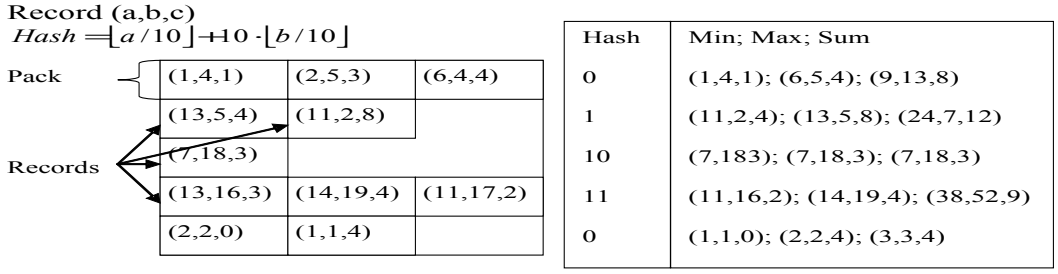

Fig. 3 . MDPAS index structure

Rows which are close in Euclidian space have the same hash number and they are stored in the same pack, or in the next pack identified by the same hash number (Fig 3).

While query: Select min(c), max(c), sum(c)
where: $a \in (10,20)$ and $b \in (10,20)$

The searching process operates with two range creating records. Every pack is checked if it includes or crosses the searched values. Three scenarios may occur:

- pack is out of range – the whole pack is omitted,
- pack includes range – the whole pack is included (sum record of the pack is used),
- pack partially crosses with record – all pack's records must be loaded from a file and checked. This is the only case when the hard drive access is inevitable

In the presented example (Fig. 3.) a query concerns only one pack (Hash value =11), Min, Max and Sum records are used: (11,16,2), (14,19,4), (38,52,9). Last numbers in brackets are query results (2,4,9). Records quantity in a pack is optional, values form 100 to 10000 were tested. Packs partial search instead of search in full records set

allows considerable response speed up, because major part of the records can be omitted or pack sum record can be included into the query result.

2.3.1 Space division's influence on MDPAS's performance

To create an MDPAS structure, every column maximal and minimal range has to be selected, along with bucket size. Appropriate selection of those parameters is critical for a structure performance. Proper choices are very important. Two-dimensional space division is shown in Fig. 3.

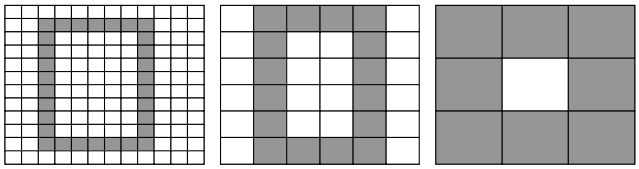

Fig. 4 . Division's influence on a query execution.

2.3.2 Low-level division

Low-level division means space fragmentation into too small parts (Fig .4a). Number of packs increases and their usage decreases. This causes the increase of a data loading time, along with increased hard drive usage. In the case of small bucket size, queries response time reduction can be observed, however tiny division results in increasing the number of accesses, as shown in (4).

$$
\begin{aligned}
&P_{\min} = m^{w}; \eta = \frac{L}{P_{\min} \cdot r_{\max}} \\
&P_{avg} = P_{\min} \\
&P_{sel} = 2 \cdot w \cdot m^{w-1} \\
&f_1 = \frac{P_{sel}}{P_{\min}} \cdot L = \frac{2 \cdot w \cdot L}{m}; f_2 = P_{sel} = 2 \cdot w \cdot m^{w-1} \\
&f = f_1 + 3 \cdot f_2 = \frac{2 \cdot w \cdot L}{m} + 6 \cdot w \cdot m^{w-1} = \frac{2w}{m}\left(L + 3m^{w}\right)
\end{aligned}
\tag{4}
$$

where: w – number of dimensions; m – division factor; L – data set size (rows quantity) ; $r_{\max}$ – maximal number of records in a single pack; η – average pack usage ; $P_{\min}$ – minimal number of packs; P_{avg} –average number of packs; P_{sel} – packs selected during range query; f_1 –average number of searched rows; f_2 – an average number of searched packs; f – average access quantity during structure searching/browsing.

2.3.3 High-level division

High-level division means space fragmentation into too big parts (division factor m is too small). Large set of data is stored in a single pack. This causes pack duplication for the same hash code (5).

$$
\begin{aligned}
&P_{\min} = m^{w}; \varphi = \frac{L}{P_{\min} \cdot r_{\max}} \\
&P_{avg} = P_{\min} \cdot \varphi \\
&P_{sel} = 2 \cdot w \cdot m^{w-1} \cdot \varphi \\
&f_1 = \frac{P_{sel}}{P_{\min} \cdot \varphi} \cdot L = \frac{2 \cdot w \cdot L}{m}; f_2 = P_{sel} = \frac{2 \cdot w \cdot L}{m \cdot r_{\max}} \\
&f = f_1 + 3 \cdot f_2 = \frac{2 \cdot w \cdot L}{m} \cdot \left(1 + \frac{3}{r_{\max}} \right)
\end{aligned}
\tag{5}
$$

where: w — number of dimensions; m – division factor; L - data set size (rows quantity); $r_{\max}$ – maximal number of records in a single pack; η – average pack usage; $P_{\min}$ – minimal number of packs; P_{avg} –average number of packs; P_{sel} – packs selected when executing an range query; f_1 –average number of searched rows; f_2 – average number of searched packs; f – average number of accesses when browsing the structure

3 Distributed index structure in GDW ^{SA} systems

In GDW^{SA} systems a user is interested in the aggregated telemetric data. To optimize query response time, which has to be executed in such a system, the best solution is application of an adequate indexing technique. Indexesdistribution allows obtaining a good scalability. Presented indexes are designed according to spatio-temporal interface. Such approach allows performing the same actions (building, writing on a hard drive, data loading) on different indexes. Data distribution allows division of a data loading process into several parallel tasks. RAM memory and hard drive requirements are divided analogically. GDW^{SA} system administers distributed indexes, divides them into packs, creates and assembles queries. User does not need to know which index is on a certain system node and even does not need to know the indices quantity. The application allows transparent actions concerning data loading and query execution.

3.1 Distributed indexes managing

During the GDW^{SA} (t) application implementation a language for distributed resources managing was created. By sending parameterized commands a user can create, save, load or close an index. Single record or whole data pack can be loaded.

Command example:
f=query;type1=1;type2=1;time1=0;time2=999999999999;d01=0.0;d02=1000.0;d11=0.0;d12=1000.0;d21=0.0;d22=1000.0;from=c239847561;group=indexes;timeout=2000

3.2 Data division

There are N nodes in GDW ^{SA}*(t)* system, denoted as S_1 to S_N. M counters are partitioned and the partitions are marked L_1 to L_M. For every system node, a set of factors can be calculated: $Z_{i,j}$ – j factor for i server, $j=1..J$.

Factors: (a) CPU speed (cpuFreq), (b) average CPU usage (cpuAvg), (c) available RAM memory (memAvail), (d) average available RAM memory (memAvg), (e) TCP/IP delay when sending data – ping (pingTime), (f) average transfer (connSpeed), (g) number of counters assigned to a certain system node (pointsCount), (h) sever data set size (dataCount), (i) average server usage (servUsed).

Each of those factors has certain weight, for each node profitability factor is calculated, which factor is crucial while selecting node for pack storing (6).

$$W = \sum_{i=1}^{J} |w_i|$$
$$\Theta_i = \sqrt[W]{\prod_{j=1}^{J} Z_{i,j}^{\ w_j}} \qquad (6)$$
$$\Theta = \sum_{i=1}^{J} \Theta_i$$
$$O_i = \frac{\Theta_i}{\Theta}$$

where: w_i – weight of the i-th column; W – weights sum for each column; $Z_{i,j}$ – j factor for i server; Θ i – profitability of the i-th server usage; O_i – profitability of i-th server usage

For certain factors initial weights can be calculated and basing on those weights a counter position can be obtained (Tab.1).

Table 1 . Factors initial weights

Factor	S1	S2	S3	Weight (wi)
CpuFreq	1700	2000	3200	1
CpuAvg	0,89	0,97	0,82	-1
MemAvail	450	450	850	1,5
MemAvg	150	200	200	1
PingTime	31	45	121	-0,5
PointsCount	3000	5000	7000	2
DataCount	45	78	113	0,2
ServUsed	14	34	41	-1,5
Θ_i	43,5244663	43,41701	51,96572	
O_i	0,31333484	0,312561	0,374104	

The larger O_i server factor the more profitable is index storing on such a server. Data division allows load balancing in a distributed structure, while when servers in the system show different performance. Division depends on the factor's weight, and the weight selection has an influence on a certain PC selection.

4 Tests

The tested GDW^{SA} system consists of five Pentium IV (2,8 or 3,2GHZ, 512 or 1024MB RAM memory).

The results presented below prove that for STCAT query response time rapidly increases for high-level division, and it slowly increases for low-level division. The MDPAS operates in different way for low-level division, when number of packs needed to be accessed increases. In Fig. 5, 6 influence of searched records on selectivity is presented. For STCAT and MDPAS wide range can be selected in which division is quite good - STCAT: (50;1000), MDPAS: (90;350).

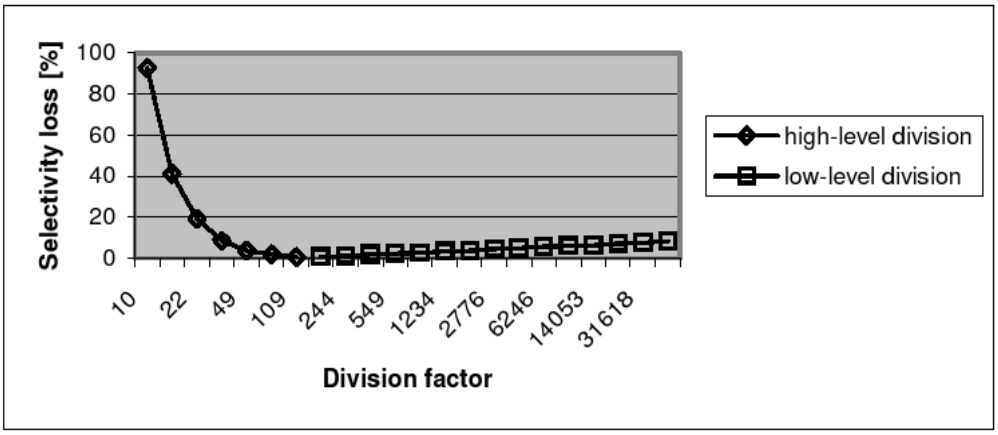

Fig. 5. Division's influence on STCAT selectivity

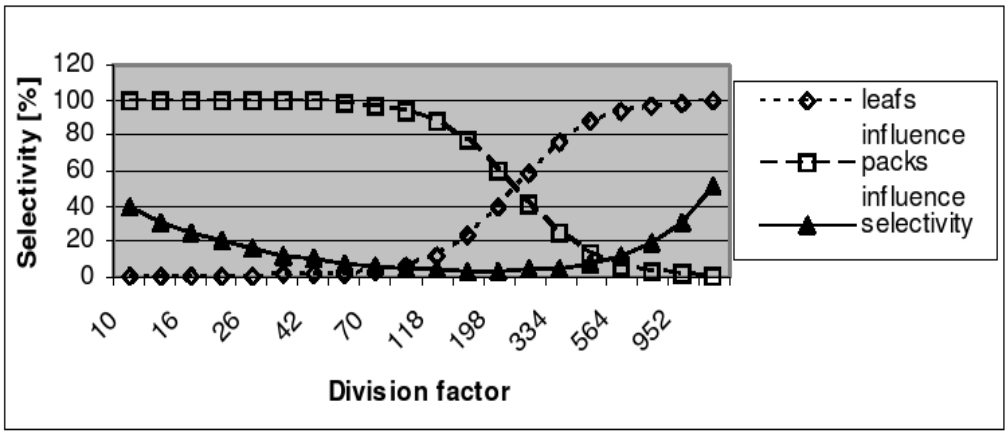

Fig. 6. Division's influence on MDPAS selectivity

Indexes management allows defining the system's reaction to actions like: index creating, deleting and response to query. Five GDW ^{SA} system nodes were tested with client application running on each of them. On one of the nodes index manager application has been started. Creating and deleting tasks lasted shortly and were imperceptible while application using.

Fig. 7. STCAT index spatial managing

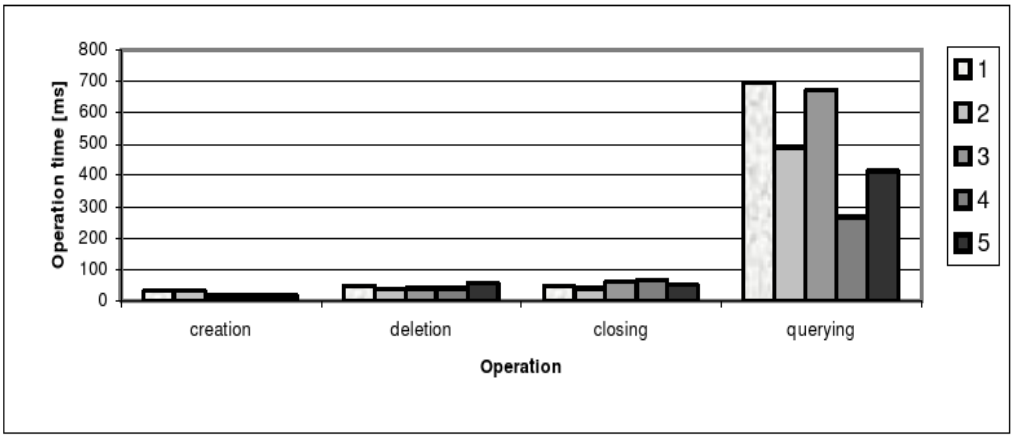

Fig. 8. MDPAS index spatial managing

Data sending with simultaneous data loading proved to be efficient manner of index distribution.

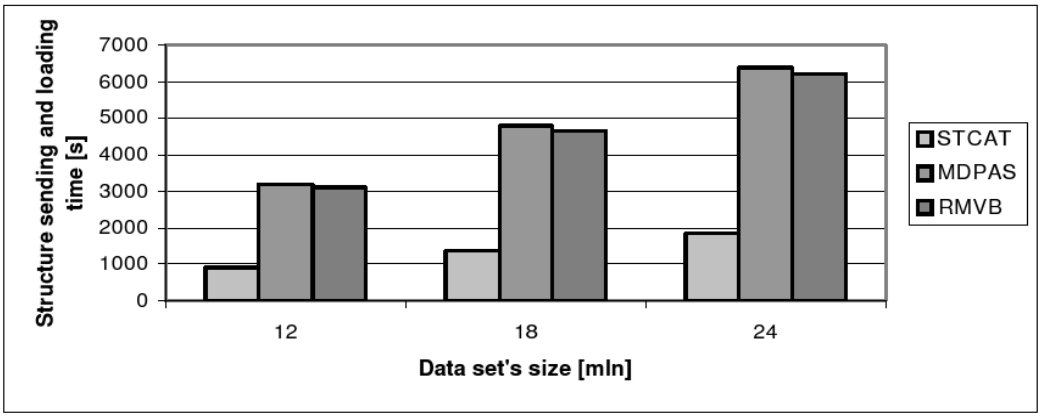

Fig. 9. Data sending with simultaneous data loading for STCAT, MDPAS and RMVB index

5 Conclusions

Presented indexes allow efficient range queries evaluation against large data sets. Structures force selection of certain parameters during their creation. Those parameters have an influence on indexes performance, but as shown in the test results their role is not crucial.

Distribution allows division of data set (consisting of even several million records) between several system nodes. Division is transparent to the user.

In the future we plan to test a new index and upgrade existing ones. In addition, we will develop a data division system based on a genetic algorithm. We also want to continue investigation of the data redundancy problem.

References

1. Manolopoulos Y, Nanopoulos A., Papadopoulos A., Theodoridis Y.: *R-Trees have grown everywhere,* unpublished technical report, 2003.
2. Tao Y., Papadias D.: *Range Aggregate Processing in Spatial Databases,* IEEE Trans. Knowl. Data Eng. **16** (12) pp: 1555-1570, 2004.
3. Gorawski M., Malczok R.: *Distributed Spatial Data Warehouse Indexed with Virtual Memory Aggregation Tree,* 2th Workshop on Spatial-Temporal Database Management, pp. 25–32, Canada 2004.
4. Gorawski M., Malczok R.: *On Efficient Storing and Processing of Long Aggregate Lists,* Data Warehousing and Knowledge Discovery, 7th International Conference, DaWaK, Copenhagen, Denmark, August 22-26, 2005, LNCS ***3589***, pp. 190-199, 2005.
5. Gorawski M.: *Architecture of Parallel Spatial Data Warehouse: Balancing Algorithm and Resumption of Data Extractions,* Software Engineering: Evolution and Emerging Technologies, FAIA series IOS PRESS, vol. **130**, pp. 49-59, 2005.
6. Holger Märtens Erhard Rahm Thomas Stöhr, *Dynamic Query Scheduling in Parallel Data Warehouse,* University of Leipzig, Germany, 2002.
7. Dehne F., Eavis T., Rau-Chaplin A. *Parallel Multi-Dimensional ROLAP Indexing,* Proc. 3rd IEEE/ACM, CCGrid (2003), Japan, 2003.
8. H. N. Lim Choi Keung, J. Cao, D. P. Spooner, S. A. Jarvis, G. R. Nudd: *Grid Information Services using Software Agents,* 2003.
9. Junwei Cao, Daniel P. Spooner, Stephen A. Jarvis: *Subhash Saini and Graham R. Nudd, Agent-Based Grid Load Balancing Using Performance-Driven Task Scheduling,* 2003.
10. Gorawski M., Bańkowski S.: *Software Agents in Grid Environment,* I National Scientific Conf. - Technology of the Data Processing, Poznań, pp. 118-129, 2005.