International Journal of Computer Science & Applications
Vol. 4, No. 1, pp. 39 - 52

Selection of Indexing Structures in Grid Data Warehouses with Software Agents

Marcin Gorawski, Michal Gorawski, Slawomir Bankowski
{M.Gorawski, Michal.Gorawski, Slawomir.Bankowski}@polsl.pl
Institute of Computer Science, Silesian Technical University,
44-100 Gliwice, ul. Akademicka 16 , Poland

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 such amount of quickly flowing data. More and more common solutions use distribution of an indexing structure among several system nodes. Indexing structures presented in the following paper are based on a grid solution. Implemented structures assure good performance of every node, and their diversity allows adjusting to any data type. Proposed system offers effective data distribution and index management in grid environment. Additionally Software Agents used in Data Warehouse Grid , are presented.

1 Introduction

Data indexing problem has been already 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 known, popular and effective means, that allows improved record search in a database [1,2]. Indexing structures presented in this paper are adjusted to service spatio-temporal sensor/meter data (readings from telemetric meters (water, gas, heat and electric energy))[3].When taking into cosideration a data warehouse, where the data set size can be over hundred millions records, range queries have long response time [4]. The above mentioned indexes cannot speed up aggregation process and are useless in such a warehouse, 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/meters is described with a certain position in space (denoted as x,y,z coordinates). Every meter reading has its measurement time and measurement type (water, gas, heat and electric energy). In such environment aggregated values are prioritized before 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 (GDWSA(t)) [10] cooperates with the Spatio-Temporal Data Warehouse (STDW) [3,4,5] using the cascaded star data model. The paper has been divided into the following sections: In sections 2, 3 and 4 detailed description of STCAT, MDPAS and MDPAS index in GDWSA(t) environment will be presented followed by the indexes performance tests. The GDWSA(t) system's software agents structure is shown in section 5.

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/meter data are stored in below mentioned indexing structures. A main drawback of the presented solutions is necessity of possession of certain information about serviced data. Along, with a definition of dimension cardinality, 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 prove 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 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.

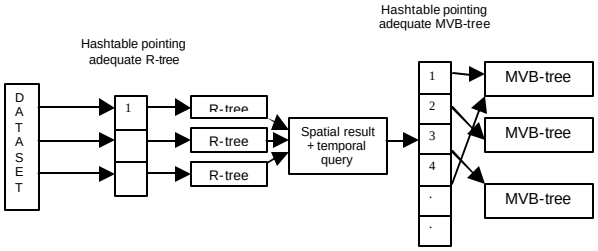

Fig. 1. R-MVB-tree

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, which means that no knowledge is needed prior to index creation, however building time is multiple times longer in comparison with a database (Oracle 10g) or other investigated structures (STCAT, MDPAS).

2.2 STCAT structure

STCAT consists of a 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).

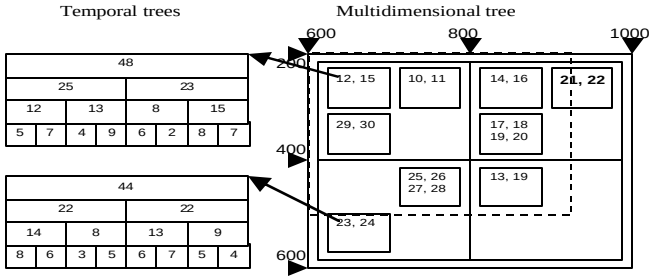

Fig. 2. STCAT tree

2.2.1 Space division's influence on STCAT's performance

STCAT's structure has certain inconveniences. When the structure is created, its basic parameters such as time bucket size and multidimensional leaf size should be determined. Correct choice of above-mentioned parameters enables multiple speed-up comparing to a database (e.g. Oracle 10g). 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 performance, and which of them is default, if a format of data loaded into the tree is unknown.

2.2.2 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 (1):

$$m^w >> k \Rightarrow h = \lceil \log_2 m \rceil; m_h = 2^h \quad (1)$$

$$h_k = \left\lceil \frac{\log_2 k}{w} \right\rceil; h_d = h - h_k = \log_2 m - \frac{\log_2 k}{w}$$

$$f_i = 4 \cdot w \cdot m_i^{w-1}; f_A = \sum_{i=1}^{h} f_i$$

$$f_A \le \frac{2^w \cdot (4 \cdot w \cdot m^{w-1})}{2^{w-1} - 1}; f_B = 4 \cdot w \cdot m^{w-1} \cdot h_d; fc = 4 \log_2 L$$

$$f_D = f_A \cdot (1 + f_C) + f_B$$

where: m – number of parts while dividing every dimension; w – number of dimensions; k – number of added time trees (number of counters); hk – level, on which maximal number of leafs equals k; h – tree height; hd – number of levels below hk; L –total number of multidimensional tree leaves; fi – number of multidimensional tree leaves, which are accessed on level i; fA – number of multidimensional tree leaves, which are accessed above hk level; fB – number of multidimensional tree leaves, which are accessed below hk level; fC –number of access to time tree; fD – average number of accesses during the structure searching.

2.2.3 High-level division

High-level division has to be considered in terms of numbers of meters stored in one leaf. An inadequate range setting causes that all counters are positioned in the root, and 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:

$$m^w >> k \Rightarrow h = \lceil \log_2 m \rceil; m_i = 2^i \quad (2)$$

$$s = \frac{k}{m^w}$$

$$f_i = 4 \cdot w \cdot m_i^{w-1}; f_A = \sum_{i=1}^{h} f_i$$

$$f_A \le \frac{2^w \cdot (4 \cdot w \cdot m^{w-1})}{2^{w-1} - 1}; f_B = 4 \cdot w \cdot m^{w-1} \cdot s; fc = 4 \log_2 L$$

$$f_D = f_A \cdot (1 + f_C) + f_B$$

where: m – actual space division (small value because of high-level division); s – one-way list size which is created from multidimensional tree leaves; fA – number of accessed multidimensional tree leaves; fB – number of accessed multidimensional tree leaves, created from one-way list; fC – number of accesses to the time tree; fD – 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 such as 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):

$$range_i = \max_i - \min_i \tag{3}$$

$$parts_i = 0 \Rightarrow cup_i = 1;\ parts_i \neq 0 \Rightarrow cup_i = \frac{range_i}{parts_i}$$

$$m_i = \prod_{j=1,\ parts_j \neq 0}^{i-1} parts_j$$

$$parts_i = 0 \Rightarrow scale_i = 0;\ parts_i \neq 0 \Rightarrow scale_i = m_i$$

$$hash_j = \sum_{i=0}^{M-1} \frac{(value_{i,j} - \min_i) \cdot scale_i}{cup_i}$$

where: mini – minimal i parameter's value, maxi – maximal i parameter's value, parts – number of column division parts , cupi – range value for i column, scalei – scalability factor for the i-th column.

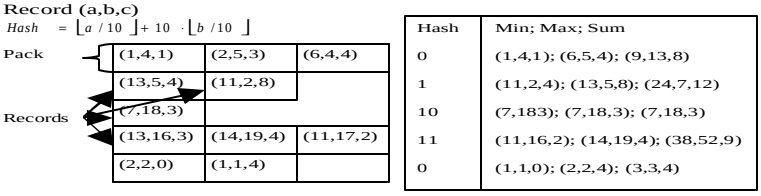

Fig. 3. MDPAS index structure

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). The 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. 4.

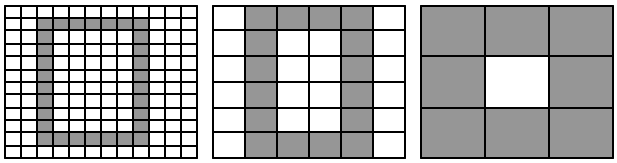

Fig. 4. Division's influence on a query execution

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 for. 4.

$$P_{\min} = m^{w}; h = \frac{L}{P_{\min} \cdot r_{\max}} \tag{4}$$

$$P_{avg} = P_{\min}$$

$$P_{sel} = 2 \cdot w \cdot m^{w-1}$$

$$f_1 = \frac{P_{sel}}{P_{\min}} \cdot L = \frac{2 \cdot w \cdot L}{m}; f_2 = P_{sel} = 2 \cdot w \cdot m^{w-1}$$

$$f = f_1 + 3 \cdot f_2 = \frac{2 \cdot w \cdot L}{m} + 6 \cdot w \cdot m^{w-1} = \frac{2w}{m}\left(L + 3m^{w}\right)$$

where: w – number of dimensions; m – division factor; L – data set size (rows quantity) ; rmax – maximal number of records in a single pack; ? – average pack usage ; Pmin – minimal number of packs; Pavg –average number of packs; Psel – packs selected during range query; f1 –average number of searched rows; f2 – 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).

$$P_{\min} = m^{w}; j = \frac{L}{P_{\min} \cdot r_{\max}} \tag{5}$$

$$P_{avg} = P_{\min} \cdot j$$

$$P_{sel} = 2 \cdot w \cdot m^{w-1} \cdot j$$

$$f_1 = \frac{P_{sel}}{P_{\min} \cdot j} \cdot L = \frac{2 \cdot w \cdot L}{m}; f_2 = P_{sel} = \frac{2 \cdot w \cdot L}{m \cdot r_{\max}}$$

$$f = f_1 + 3 \cdot f_2 = \frac{2 \cdot w \cdot L}{m} \cdot \left(1 + \frac{3}{r_{\max}}\right)$$

where: w — number of dimensions; m – division factor; L - data set size (rows quantity); rmax – maximal number of records in a single pack; ? – average pack usage; Pmin –

minimal number of packs; Pavg –average number of packs; Psel – packs selected when executing an range query; f1 –average number of searched rows; f2 – average number of searched packs; f – average number of accesses when browsing the structure

3 Distributed index structure in GDWSA systems

The user of GDWSA(t) system (which is presented in chapter 5) 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 the application of an adequate indexing technique. Indexes distribution allows obtaining a good scalability. The indexes presented above are designed according to a spatio-temporal interface. This approach enables to utilize the same actions (building, writing on a hard drive, data loading) on various indexes. Data distribution allows division of a data loading process into several parallel tasks. RAM memory and hard drive requirements are divided analogically. GDWSA(t) 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 does not even need to know the indices quantity. The application allows transparent actions concerning data loading and query execution.

3.1 Distributed indexes managing

During the GDWSA(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;gr
oup=indexes;timeout=2000
```

3.2 Data division

There are N nodes in GDWSA(t) system, denoted as S1 to SN. M counters are partitioned and the partitions are marked L1 to LM. For every system node, a set of factors can be calculated: Zi,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. Each node has a profitability factor that is calculated. This factor is crucial while selecting node where a pack should be stored (6).

$$W = \sum_{i=1}^{J} |w_i| \tag{6}$$

$$\Theta_i = \sqrt[W]{\prod_{j=1}^{J} Z_{i,j}^{w_j}}$$

$$\Theta = \sum_{i=1}^{J} \Theta_i$$

$$O_i = \frac{\Theta_i}{\Theta}$$

where: wi – weight of the i-th column; W – weights sum for each column; Zi,j – j factor for i server; i – profitability of the i-th server usage; Oi – 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).

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
T_i	43,5244663	43,41701	51,96572	
O_i	0,31333484	0,312561	0,374104	

Table 1. Factors initial weights

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, whereas servers in the system show different performance. Division depends on the factors weights, and the weight selection has an influence on a certain PC selection.

4 Indexing structures' tests

The tested GDWSA system consists of five Pentium IV machines (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).

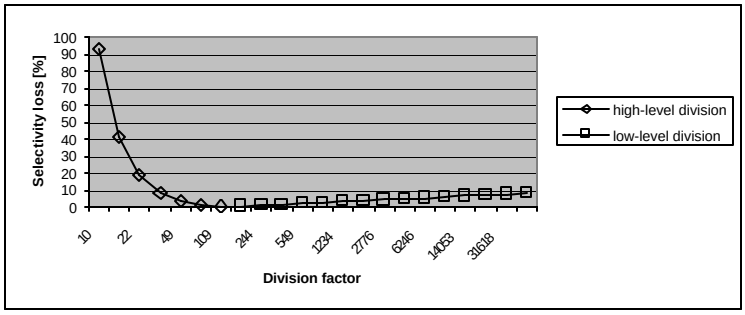

Fig. 5. Division's influence on STCAT selectivity

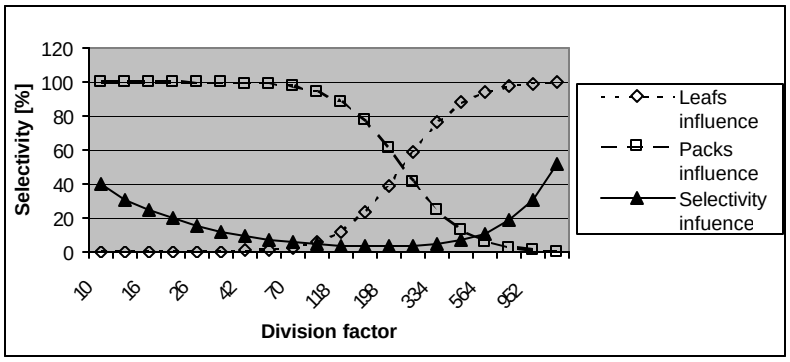

Fig. 6. Division's influence on MDPAS selectivity

Indexes management allows defining the system reaction to actions like: index creating, deleting and response to query. Five GDWSA 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 during application using.

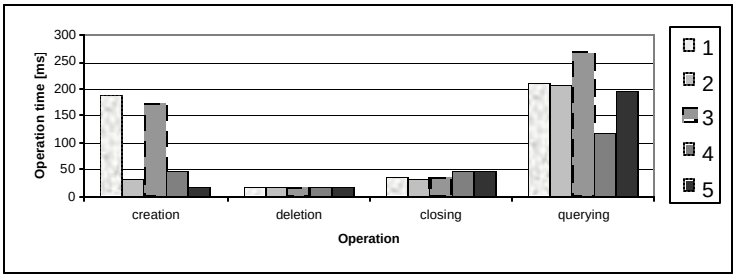

Fig. 7. STCAT index spatial managing

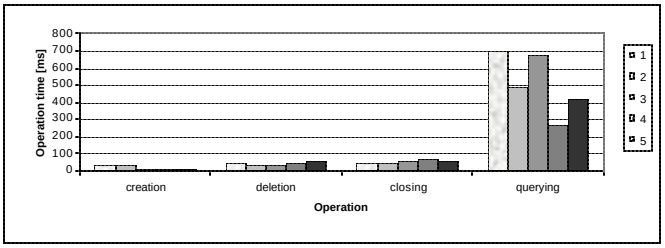

Fig. 8. MDPAS index spatial managing

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

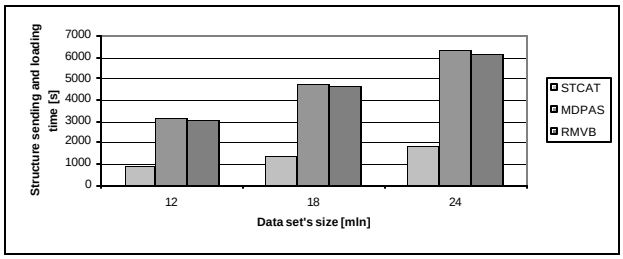

Fig. 9. Data sending with simultaneous data loading for STCAT, MDPAS and RMVB index

5 The GDWSA(t) environment

The Software Agents (SA) realized in a grid technology are considered in [11, 12, 13, 14]. Despite rapid CPU's evolution wide range of their usage, causes increase of demands for larger and faster computer systems. The answer to this problem is multiprocessor and multicore computers, however their price is not competitive, with cheap single processor units. The efficient servers and units dedicated to concrete solutions are very expensive and on the other hand software and data structure is not always compatible with parallel processing. The main problem is work distribution and its equal division. Load balancing can be optimal in an environment witch constant number of nodes and known characteristics [15]. In grid systems unknown number of nodes is available, logging on and off to network at random, so it is extremely difficult to obtain uniform workload on all nodes meanwhile minimizing distribution losses.

5.1 Software Agent and Agents system

The Software Agent is an autonomous unit working in certain environment, capable of communicate, environment observation, decision making, and learning, usage of gathered knowledge, adapting to environment, and error tolerant. All modules are implemented using joint interfaces, what enables uniform tasks and data swap. Agent was implemented in Java 1.5 and along with all components it has over 80 thousands lines of code.

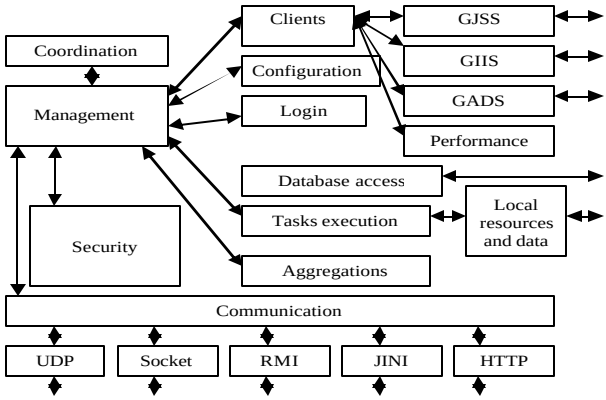

Fig. 10. Agent structure

The Management module is responsible for data distribution inside an Agent. The Security module checks, if incoming tasks comes from a legible source. The Clients module enables access to catalogues and other services via network, and the Configuration module allows setting and storing data important to an Agent. Tasks execution module is crucial for the system efficiency. Tasks are added to Agent space and then serviced by ap-

propriate modules. The Database access module assures management and maintaining of connections to various databases (JDBC and ODBC usage). The Agent system bases on a grid technology. The shared access to all resources assures high scalability.

5.2 Agent's tasks

The task consists of series of parameters with their values. Parameters names are optional and serialization bases on writing task in following manner

```
Parameter1=Value1;Parameter2=Value2;Parameter3=Value3;Parameter4=V
alue4;Parameter5=Value5;Parameter6=Value6;Parameter7=Value7
```

The task example:

```
workerGroup=math;workerFun=do;time=0;groupId=319225031;e0=5000000.
5;version=3.2.0.0;endMode=r(p1);p2=1.8;p1=1.4;giis=false;num=0;gjs
s=true;workerMode=r(p1);b0=1.0;timeout=1116262256207;count=1;timeC
reate=18:50:46;valueSum=3498326.3905740315;endGroup=math;group=wor
kerQueue;id=757738465;endFun=add;fun=set;dimensions=1;splitParts=2
;agentName=Agent11;
```

The Agent receives tasks from a catalogue and sends its tasks or other task's results. To control task execution three queues were created. Every task goes through all of them.

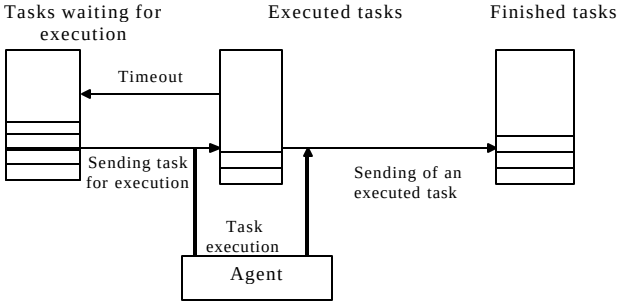

Fig. 11. Queues structure

Every task can be divided into any number of subtasks, however while the fragmentation is to small not all system nodes would be used. In case of a too large fragmentation significant efficiency drop will take place. The task that is to be send is divided into subtasks and loaded into first queue (subtasks waiting to be executed). From this queue subtasks are send to agents and simultaneously their copies are loaded into the second queue (subtasks currently processed). A subtask includes timestamp when it is loaded into queue, to ensure execution time control. The execution time is checked periodically and if it is too long a timeout is generated and the subtask is loaded back to the first queue (subtasks waiting to be executed). After successful execution (execution report or result is obtained) the subtask is loaded into the third queue (finished tasks). If both, the first and the second queues are empty, that means that the whole task was completed successfully and subtasks results can be combined.

5.3 Database Grid

The grid is a form of a distributed computation, that coordinates and shares: calculations, applications, data, memory and network resources. This is obtained through dynamically changing and geographically distributed organization. The grid technology could be the key to solve highly complex computation problems. Computational grids enable wide range of sharing and coordinating of geographically scattered network re-

sources. The sharing can be long or short term and can concern heterogenic resources. This causes continuous creation of dynamic virtual structures. In consequence, users will not have knowledge of resources that are in virtual structure range. Computational grid defines scalability of a large number of network resources. The one of questions is a competition for resources, which determines grid efficiency. It's important to effectively deal with this problem, for example through monitoring and resources schedule. Agents through the grid technology become part of a larger system. The communication and tasks execution is transparent to the user. Connections structure and resources are not important from the system user's point of view

The network transfer speed constantly increases, so it is more and more profitable to send tasks to less loaded node, execute it there, and send back the results. The one of main problems, is when data set size is adequate (considering task execution time), to be send to other node for execution, or even to be distributed among several nodes and executed. Other problem which emerges is a choice of an adequate workstation. The GJSS, GIIS and GADS are information catalogues. They use common data space available to every logged agent. Functionality comes down to collecting and sharing information service which is clear for all agents. As an example every agent can load information about its efficiency, but also ask about most efficient agent in the same catalogue. GJSS, GIIS and GADS catalogues differs mainly in their communication (RMI, JavaSpaces, Socket)

5.4 Communication and distribution

A good communication is crucial for effective operation of distributed system. It has to enable sending various information in an unknown Agent structure. Figure 3 shows various types of Agent connections. Grid structure stores information about agent resources, available indexes, efficiency and plug-ins. Agent can send its information or obtain information of other agents. This information is crucial in determining the choice of an appropriate agent.

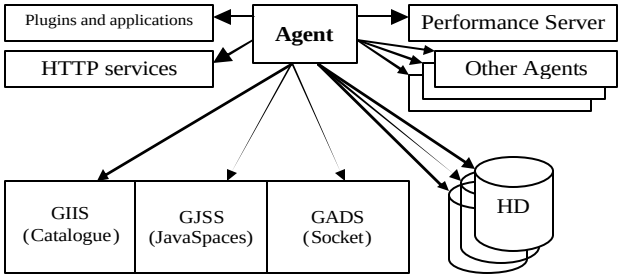

Fig. 12. Agent's communication

5.5 Virtual Organization

Figure 13 shows Agents communication in a hierarchical structure. The VO is a virtual organization, where Agents swap tasks without usage catalogue access. The VO is a set of resources which creates logical entirety.

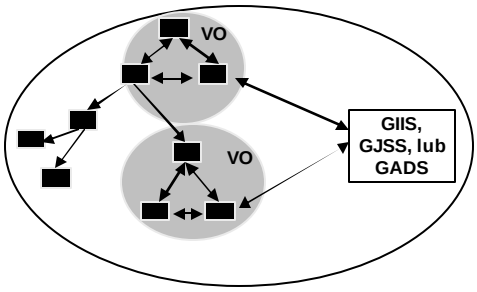

Fig. 13. Virtual organization

When a small number of workstations are accessible, one VO can be created. In such case every Agent will have a list of other Agents' active connections. Communication will be optimal, but balancing will be quite complicated. When a new task is added, it is divided into several parts (usually five or six times more then the number of available Agents), and then distributed to all Agents. When results from a certain Agent are sent, Agent receives a new task. In such case, use of Agents if optimal during a whole computation cycle, except of a short time period (20-80 milliseconds) when Agent awaits for a new task. This time period happens between the action of tasks result sending and the action of a new task sending. During this time period agents can not perform any actions however, due to its length, it has no impact on efficiency. This problem can be easily dealt with, by sending couple tasks to every Agent.

5.6 The GDWSA(t) tests

Tests concerns communication, tasks execution and plug-ins swap in agents system.

The first test proves, that time overhead while communicating using catalogue is in fact quite small.

The second test checks efficiency of tasks distribution, and proves that managing time (task creation, task division, task sending, results receiving) is insignificant.

The third test concerns plug-ins distribution. It shows that presented structure can be easily upgraded by new abilities. Time needed for new plug-ins send and execution is in range of several seconds.

The first test concerns communication's efficiency. Each Task was fragmented and sent through VO, using GIIS, GJSS or GADS. When fragmenting cardinality increased, management time has increased proportionally.

Sent task concerned calculation of of a certain mathematical sequence sum (Fig. 14).

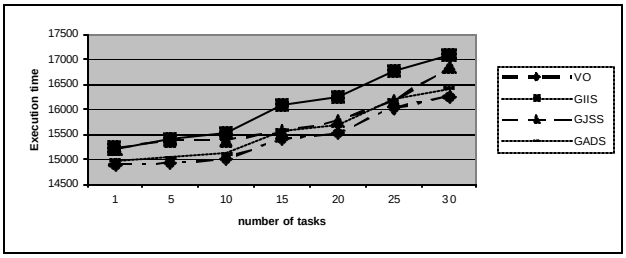

Fig. 14. Task sending

The speed-up of task execution equals number of system nodes. When larger tasks are serviced results are considerably better (management time is the same for constant number of subtasks) then executing the same task on a one station (Fig. 15).

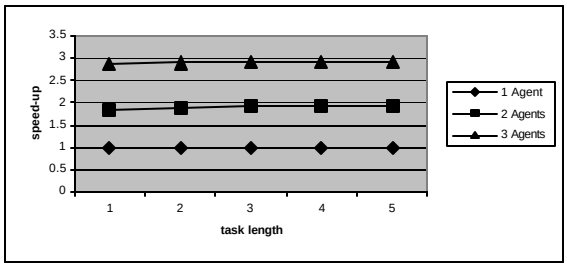

Fig. 15. Dependency of speed-up and task length

In the following test several classes' distribution time was measured (Fig. 16). Plug-in is added to one Agent and time is measured between adding of a plug-in to one Agent and plug-in startup on all nodes.

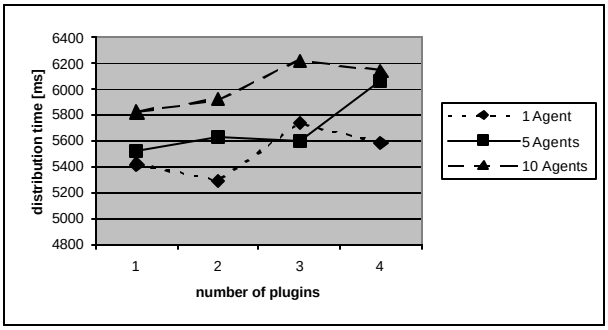

Fig. 16. Plug-ins distribution

5.7 Conclusions

Presented indexes allow efficient range queries evaluation against large data sets. Structures force selection of certain parameters during their creation. Those parameters have 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.

The presented Agent's methodology bases on the grid technology. Single node has the ability to configure, connect to a remote catalogue and directly to other Agents. Tasks are divided and executed parallel on many workstations. This causes considerable speed-up. Built-in functions, that measure efficiency of a local node enable evaluation of a whole system performance. The system is adapted to a multi-user mode, and a good balancing mechanism provides equal use of all system nodes. It is essential for a full usage of a system's capability in an unknown environment. The balancing goal is to omit slower nodes and to send more tasks to fastest nodes.

In case of clustered systems VO usage is recommended. This deals with the Agents' excess problem. The usage of plug-ins increases system flexibility. As presented in Software Agents system tests, tasks creation and parsing is quite fast, and a simple serialization allows data sending using JavaSpaces, RMI, Sockets or datagrams.

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 Tas k Scheduling, 2003.

[10] Gorawski, M., Bankowski, S.: Software Agents in Grid Environment. I National Scientific Conf. -Technology of the Data Processing, Poznan, pp. 118-129, 2005.

[11] Faruk Polat, Reda Alhajj: A multi- agent tuple-space based problem solving framework, Middle East Technical University, 06531 Ankara, Turkey 1998.

[12] H.N. Lim Choi Keung, J.Cao, D.P. Spooner, S.A. Jarvis, G.R. Nudd, Grid Information Services using Software Agents, 2003.

[13] Gorawski, M., Bankowski, S.: Software Agents in Grid Environment. I National Scientific Conf. -Technology of the Data Processing, Poznan, pp. 118-129, 2005.

[14] Jorge R. Bernardino, Pedro S. Furtado, Henrique C. Madera, Approximate Query Answering Using Data Warehouse Stripping, 2003.

[15] 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