Proceedings of the International Multiconference on
Computer Science and Information Technology, pp. 243 – 252

ISSN 1896-7094

Spatial Telemetric Data Warehouse and Software Agents as Environment to Distributed Execute SQL Queries

Marcin Gorawski[1], Ewa Płuciennik[1]

[1] Silesian Technical University, Institute of Computer Science,
44-100 Gliwice, Poland
{Marcin.Gorawski, Ewa.Pluciennik}@polsl.pl

Abstract. The article presents environment for distributed SQL query execution in a telemetric data warehouse. A parallel spatial data warehouse stores information remotely read from meters grouped in nodes according to their geographical positions. This data warehouse constitutes distributed data structure. Software agents environment enables querying this structure as local one so distribution is transparent to the user. Query evaluation consists of analysis, decomposition, local execution and results merging. There is no need to transfer data between nodes. Authors present operators for executing query modification and sub-results merging and also test outcomes of SQL query realization in different agent environment configurations.

1 Introduction

For many years in database realm there exists tendency to create systems designated to store and process more and more huge data volumes measured even in petabytes. It results from rapidly growing amount of information which human being is unable to analyze. These data are often stored in data warehouses and used for On-line Analytical Processing (OLAP) or Decision Support Systems (DSS). There are several ways to improve efficiency (query response time) of such systems: appropriate indexing, materialized views, data partitioning and parallel processing [1]. The last technique is very interesting because it is natural for organization which consists of a few or more autonomous geographically distributed departments. In this case each department has its own local independent system where data is stored for example in autonomous data warehouses. Still there exists some kind of a central system (for example DSS) which should have access to the data from all subsystems. Moreover, although hardware capabilities in data transfer, storage and analysis are still rising, it does not make a guarantee for effective data processing which can be provided by the system enabling parallel data processing and tasks execution [2]. Parallel data processing can be applied in local multi-processor environment as well as geographically distributed autonomous machines.

Work related to the query execution problem in distributed Database Management Systems (DBMS) concentrates on query evaluation plan (QEP) analysis and optimization [3] and also minimization of total amount of data transferred between nodes [4]. In case of distributed data warehouse very important issue is a way of distribution. One of such methods has been proposed for star schema in [1]. It is data warehouse stripping in which dimension tables are replicated and a fact table is uniformly distributed among all computers in the distributed system. The way of query evaluation in this kind of distributed data warehouse has been presented in [1, 5, 6].

In this paper we present solution which is a combination of parallel, distributed data warehouse and software agents environment. Query evaluation is very similar to the one proposed in publications mentioned above. We

propose formal operators for all operation needed to correctly execute a query in distributed data warehouse (section 3). Section 2.1 presents distribution of cascaded star schema [7] additionally taking into consideration data characteristics which allow to avoid replication of some dimension tables. Aggregates (materialized views) are also distributed. Each node in our system has its own local data warehouse which stores information geographically related to the particular node. Agents whose main task is to support user in data transformation and exploration to gain knowledge [8] constitute environment for effective data access using standard query language whereas the data distribution remains hidden. An agent can be defined as an autonomous unit operating in a given environment, able to communicate with other agents and react to environment changes [9]. In our case an agent is a kind of SQL client adopted to operate in a distributed data environment.

Presented system enables effective distributed data access and querying. Data distribution is transparent to the users. The way data warehouse tables are distributed and accessed eliminates need to transfer raw data between nodes when executing SQL query. Our earlier system tests have shown its advantage with reference to the non-distributed solution. Here we concentrate on examining query execution in various system configurations.

2 Distributed SQL query execution system

Distributed query execution system consists of two main components. The first one is a spatial distributed parallel telemetric data warehouse. The second one is software agents environment which allows, among other things, to query distributed data warehouse as local database.

2.1 Spatial parallel telemetric data warehouse

Spatial telemetric data warehouse stores information about media (electricity, water, natural gas) consumption registered by meters which send data to the appropriate node. This node is a superior unit for meters placed in a particular region [7]. The system can consists of any number of nodes. Each node has its own local data warehouse. Structures of all local data warehouses are identical in each node in terms of the data tables and indices. The only difference is a range of data stored in a particular data warehouse.

Data are distributed among nodes according to the method named complete horizontal fragmentation. Tuple set stored at a given node is limited by the following expression: $\delta_{ci}(R)$, where R denotes relation including all tuples, C_i denotes guard condition qualifying tuples stored at i-th node. In relation R there exists no tuple which for any $i \neq j$ meets simultaneously conditions C_i and C_j [10]. In case of data warehouse this method can be reduced to fact table distribution and copying dimension tables for each node. This kind of approach, in our case can lead to unnecessary data redundancy. While considering the way of data distribution it is necessary to take into account data characteristics. One of the main features of spatial data warehouse is data source geographical position. This feature exemplifies natural guard condition for tuples from fact table and also from dimension tables. Particular node should store only the data related to its geographical position. In consequence we distribute not only fact table but also some dimensional tables which store information about geographical position or about object characterized, among other features, by geographical position. An example of such a dimensional table is the METERS table – each counter has a geographical position and belongs to the particular node so it is unnecessary to store information about it in the other nodes. Data independent from geographical position (time and date) are duplicated in each node. A complete data warehouse structure can be found in [7]. Table 1 presents main data warehouse tables and aggregates distribution according to the tuples count.

MEASURE table is a fact table. Tables METERS and NODES are dimension tables. Tables with prefix MV_ are materialized views (aggregates). Tables representing time dimension (not listed in the above table) are replicated.

Table 1 . Tables division for nodes (tuple count is presented)

Table	Node 1	Node 2	Node 3	Non -distributed
MEASURES	58590919	29442291	53696917	141730127
METERS	998	517	964	2479
NODES	1	1	1	3
MV_LAST_DAILY_MEASURES	364270	188705	351860	904835
MV_DAILY_MEDIA_USE	364270	188705	351860	904835

2.2 Software agents environment

Software agents environment can be described as a set of agents communicating with each other, executing specific task classes committed by a user and allowing to access some resources (database, processors, etc.). Task has a form of parameters sequence. Parameters describe data required to the proper task execution and pass back results and return information. Agents exchange information through Grid Agent Data Service catalogue (GADS catalogue).

One of the task classes realized by agents is SQL query execution in distributed data structure. A user requests query execution to any of the active agents called from now an ordering agent. This one performs query analysis, decomposition and creates a task responsible for query execution. Next, newly created task is cloned as many times as the number of nodes. For all the tasks created this way a parameter describing execution node is set. Then tasks are placed in GADS catalogue. Active agents check GADS periodically for tasks to execute. If there are tasks waiting for execution, the agents execute them at specific nodes. After tasks execution the agents place executed tasks back in GADS. Executed tasks are collected on line by the ordering agent. This agent is responsible for merging SQL node queries sub-results and returns final results to the user. Logical system structure is presented in Figure 1.

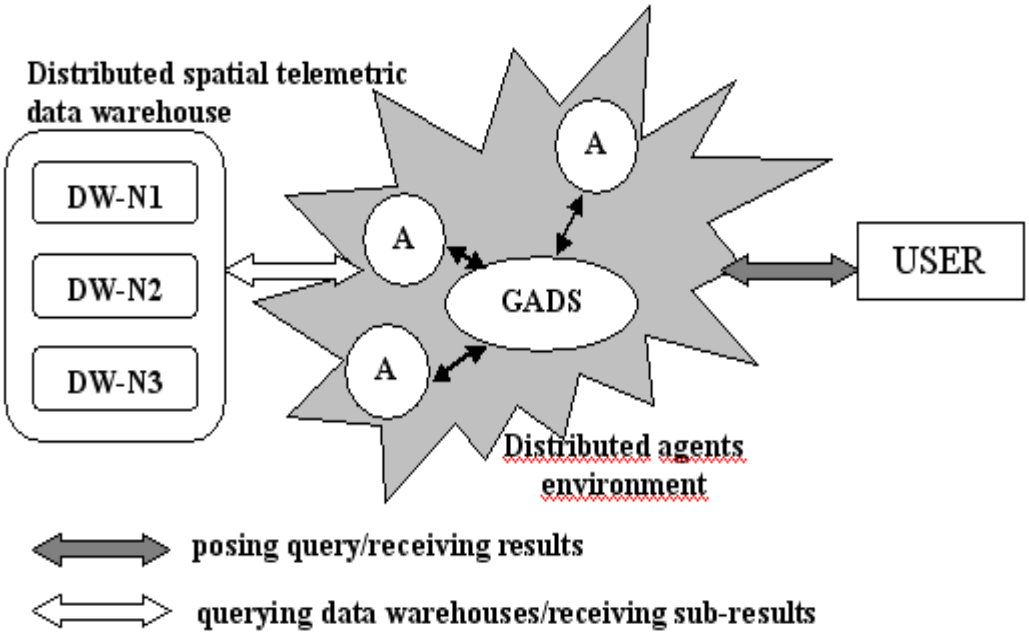

Fig. 1. Logical system structure

It is necessary to point out that such a solution is particularly suitable for distributed systems which are heterogeneous as regards operating system and database management system. Each of the agents can run at any of the computers constituting the system (resource set) and can execute SQL query at any node provided that it has all necessary information (all nodes connections definitions).

3 SQL query execution by software agents

Ordering agent is responsible for query analysis, creating and cloning appropriate query execution task and merging nodes results.

Main problems in case of query distributed data structure are: aggregate functions evaluation, appropriate tuples merging for GROUP BY phrase, tuples ordering (ORDER BY) and respecting HAVING phrase condition.

Our environment can correctly execute SQL queries in a following form:

```
SELECT [[<fields_list>], [<aggregation_function_list>]] || [*]
FROM <tables_list>
[WHERE <condition>]
[GROUP BY [<grouping_fields_list>] || [<grouping_fields_list>,] CUBE
(<grouping_fields_list>)] || [<grouping_fields_list>,] ROLLUP
(<grouping_fields_list>)]]
[HAVING <having phrase condition>]
[ORDER BY <ordering fields list>]
```

Presented above form has some additional restrictions. Aggregate function list can consists of: MIN, MAX, SUM, COUNT and AVG function. Grouping and ordering fields lists can include only fields from SELECT phrase. GROUP BY phrase can be extended by ROLLUP, partial ROLLUP, CUBE and partial CUBE phrases. HAVING phrase condition can consists of any combination of conditions based on tables fields or can take form of comparison of one aggregate function from SELECT phrase with a value, for example: `AVG(value) < 1000`.

Presented syntax does not include more complicated aggregate function evaluation (especially holistic [11) in a distributed environment. This problem will be considered in our future research. Of course a holistic function can be evaluated in distributed environment without need to transfer data between nodes. It requires to replace such function in a query with fields the function is based upon (for example RANK(value) with the value and adding or modifying ORDER BY phrase). Correct function evaluation is possible only if all result tuples are merged (merge phase is presented in section 3.2).

Correct SQL query execution in a distributed environment requires original query form (OQF) analysis according to the aggregate functions, grouping and ordering phrases and condition which eventual groups must meet. Original query form has to be modified so its execution at particular nodes return results which can be merged into correct final query result. By the correct final query result we mean result of executing original query form in non-distributed structure containing all data. The modified original query form which is executed at nodes is called a node query form (NQF) and node query result relations are called node relations and they are denoted by R_n. We also use the following symbols and assumptions:

We have given relations:

B(<attributes list>)

and

$R = {}_{\text{<grouping attributes list>}}\Im_{\text{<aggregate functions list>}}(B)$

$\Im$ - denotes aggregate operator defined in [10].

Aggregate functions list is a sequence of pairs: <function> <attribute>. Notation R_{FUNi} describes relation, for which aggregate functions list consists of (among other functions) i aggregate functions of one type for example: R_{MINi} describes relation with i functions MIN.

$R_{FUNi}=_{<\text{grouping attributes list}>}\Im_{FUNi,,<\text{remaining aggregate functions}>}(B)$

3.1 SQL query analysis and modification

Node query form must not contain aggregate function AVG in SELECT phrase and HAVING phrase with condition based on aggregate function. Node query form must not also contain nested SQL queries – we address this problem in section 3.3.

HAVING phrase is removed from the original query form and its condition is saved as a task parameter. Each AVG function from original query is decomposed into pair of intermediate functions using D_{AVG} operator.

Aggregate decomposition operator D_{AVG} functioning consists in replacing each AVG function in relation R with pair of functions SUM and COUNT:

$R_{SUMi,\ COUNTi} = D_{AVG}(R_{AVGi})$

where:

SUM_i , $COUNT_i \leftarrow AVG_i$, where notation $A_2, A_3 \leftarrow A_1$ means replacing the relation schema attribute A_1 with attribute A_2, and adding new attribute A_3.

3.2 Node results merge

Node relations merging into result relation can be divided into two following steps. First one consists of the activities which can be taken on-line, while receiving following tuple sets. Second one constitute those merging steps, which can be correctly executed only after all node relations are merged.

In the first stage the ordering agent places the first tuple set in the result table. It can be a tuple set received from any node. Subsequent tuples are appended on-line, taking into consideration:

1. Appropriate function MIN, MAX, SUM and COUNT evaluation.
2. Merging tuples belonging to the same group (in case of GROUP BY).
3. Inserting tuples while preserving right order (in case of ORDER BY).

Pipelined append operator with aggregate function merging $*S_{p\ <sc>}$ is responsible for correct tuples group merging with appropriate aggregation functions evaluation. By append we mean both-sided outer natural join.

There are given n node relations with schema R, which does not contain AVG aggregate functions (in advance schema R was exposed to aggregate decomposition operator or it did not contain AVG).

For n = 1:

$R_w = R_n$

For n > 1

$R_w = R_w *S_{p\ <sc>\ (<\text{grouping attributes list}>)} R_n$,

notation $*S_{p\ <sc>\ (<\text{grouping attributes list}>)}$ means append by grouping attributes while simultaneously updating aggregation functions values and <sc> denotes updating method.

<sc> ::= <sc_fun>[,<sc_fun>]

<sc_fun> ::= <sc_MIN>||<sc_MAX>||<sc_SUM>||<sc_COUNT>

$<sc\_MIN> ::= \min(\{\pi_{MIN(attribute)}R_w, \pi_{MIN(attribute)}R_n\})$

$<sc\_MAX> ::= \max(\{\pi_{MAX(attribute)}R_w, \pi_{MAX(attribute)}R_n\})$

$<sc\_SUM> ::= \Sigma\{\pi_{SUM(attribute)}R_w, \pi_{SUM(attribute)}R_n\}$

$<sc\_COUNT> ::= \Sigma\{\pi_{COUNT(attribute)}R_w, \pi_{COUNT(attribute)}R_n\}$

The second stage begins when all node relations are merged. It consists of recounting AVG aggregate functions, removing attributes responsible for storing intermediate values needed to evaluate AVG and applying HAVING phrase condition.

For appropriate AVG recounting we use merging average operator Sc_{AVG} :

$Rw'_{AVGi} = Sc_{AVG}(Rw_{SUMi,\ COUNTi})$

where:

$AVG_i \leftarrow (SUM_i = SUM_i / COUNT_i)$

$\varnothing \leftarrow COUNT_i$,

where notation $A_1 \leftarrow (A_2 = \text{<formula>})$ means changing A_2 attribute name in schema relation to A_1 name and recounting A_1 according to the formula,

and notation $\varnothing \leftarrow A$ means removing attribute A from relation schema.

$Rw_{SUMi,\ COUNTi}$ denotes relation merged from n relations with schema R, schema R was exposed to operator D_{AVG} which means it contains COUNT and SUM functions. Rw'_{AVGi} denotes result relation, for which it is necessary to recount AVG.

Next all merged tuples are checked if they meet HAVING phrase condition. We call this process post-join selection which amounts to apply selection operator with HAVING phrase condition to the result relation:

$R_w' = \delta_{\text{<having phrase condition>}}(R_w)$.

Mentioned above elements of merging process are optional and executed only if needed.

3.3 Nested SQL

SQL query can contain nested SQL queries which can be placed in WHERE and HAVING phrases. Presented above query modification procedure can be easily adjusted to the situation when the user poses nested query in which sub-queries return single value or set of values. It can be done by recursive query analysis and merging sub-results.

When during OQF analysis we find nested SQL we interrupt the main procedure and start nested query analysis. When it is done we obtain nested query NQF. Next we can execute sub-query, merge its node results and replace nested SQL in original query by its result value or set of values.

An example of this kind of query is a list of counters which have daily media usage above average daily media usage for particular media. In WHERE phrase we have condition in form of: <value> > (SELECT AVG(<value>) FROM …). To obtain correct results for such a query first we have to execute nested query to calculate AVG(<value>). Next we can replace nested query in OQF with concrete value and continue OQF analysis until appropriate NQF is obtained.

4 Tests environment

Tests conducted to verify if queries executed in distributed data structure return correct results consist in comparing results of local and distributed query execution. Testing system efficiency encompassed examination how the time needed for information exchange between agents, the time when tasks were waiting for execution in GADS, the time designated for query analysis and modification and results merging affect query execution time understood as the time measured from the moment user poses query to the moment he or she receives results. Influence of active agents number on the parallel tasks execution was also examined.

System operation was tested with queries, which required access to one of the three tables (fact table or aggregates) and eventually joining them with some dimension tables. Table 1 presents number of tuples in tables used in individual queries groups.

Table 2. Number of tuples accessed in queries groups

Queries group	Node 1	Node 2	Node 3	Non-distributed
1	998	517	964	2479
2	364270	188705	351860	904835
3	58590919	29442291	53696917	141730127

Characteristic feature of queries used in tests is the fact that the query result is small but the number of tuples which a query needs to access is relatively large. Queries from the first and the third group return three result tuples, and queries from the second group from 3 up to 36 tuples. Test queries did not come from standard benchmarks like TPC-H. We decided to choose queries adjusted to the data warehouse content. Example queries from particular groups are as follows:

1. Show nodes identifiers with meters number.
2. Show minimal and average particular media usage.
3. Show nodes identifiers with measures number.

Tests were conducted in three system configurations differing from each other by the number of active agents. Tests environment consisted of three nodes which were configured as presented in table 3. Ordering agent was located in node 2. Test configuration with one, two and three active agent were denoted adequately by symbols I, II and III. As the maximal agent number the number of nodes was assumed–activating more agents does not influence execution time – excess agents remain idle.

Table 3. Nodes (PCs) configuration

Node	Processor	RAM	Operating system	DBMS
1	P4 3,2 GHz	1 GB	Windows XP Professional	Oracle 10g
2	P4 2,8 GHz	512 MB	Windows XP Professional	Oracle 10g
3	P4 2,8 GHz	512 MB	Windows XP Professional	Oracle 10g

5 Tests results

During tests queries from each group were executed five times in each configuration I, II and III and also in a local system called configuration L. For each query execution the time from the moment user posed query to the moment of receiving results was measured – this time we denote by WT. WT consists of query analyze and modification time (AMT), results merge time (JT), task execution time at node (EXT), time when task waits in GADS for execution and for ordering agent to take results back – this time we call agent net time (NT). All measurements were taken with one millisecond accuracy and then averaged for each query.

Tests results have shown, that time needed for query analyze and modification combined with merge time is insignificant in comparison to the task execution time (about 0.14%) and can be passed over in further analysis.

5.1 Task execution time and parallelism degree analysis

Average tasks execution times in individual configurations were collected in table 3. It has to be point out that for queries with relatively short execution time (group 1) time used for intra agent communication and net time extend about twice execution time in distributed system in compare to the local configuration. In case of remaining queries situation is reversed – execution time for group 3 in configuration III is almost 14 times shorter than in configuration L.

Table 4. Average tasks execution time (WT) in ms

Configuration	L	I	II	III
Group 1	128	303,2	253,2	246,6
Group 2	2933,4	1872,7	1324,3	865,5
Group 3	1254681	211943,8	136587,6	91172,2

Chart placed on the left side of figure 2 shows how the execution time changes when tests environment is reconfigured from configuration L, trough I and II to III. Right-sided graph presents percentage change of execution time in three agents configuration in comparison with the non-distributed system. Values less than zero represent elongating of execution time. Such a situation takes place in case of group 1 and follows from mentioned above reasons.

Comparing tests results for tasks realization in individual configurations allows to state that, activating an additional agent resulted in shortening execution time by about 30% on average. Above 80% shorter execution time was observed for group three when the system configuration was changed from L to I. For the same group changing configuration from L to III allowed to obtain execution time about 90% shorter. This phenomena appeared in smaller but also satisfying degree for group 2 – in case of changing configuration from L to III it was about 70%. It is worth to notice, that the greatest fall was observed for group 3 which also has the greatest execution times.

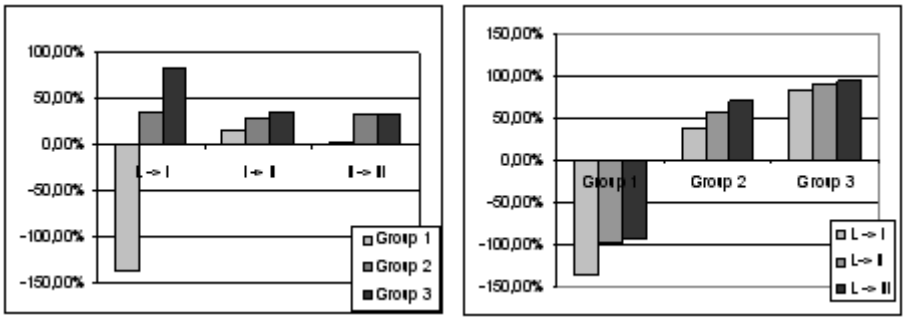

Fig. 2. WT changing in % while changing configurations

Analyzing node query execution (EXT) and net time (NT) required recounting average task execution time at nodes for individual tasks. EXT was measured directly at nodes and NT was calculated according to the following formulas: ST = WT - (AMT + JT)), NT = ST - EXT. Our aim was to specify tasks execution parallelism degree and NT influence on execution time.

For each query we computed EXT and NT percentage of ST. Then results were averaged for each groups and configurations (figure 3). We assumed that greater EXT share in ST means higher parallelism degree – time when tasks wait for execution shortens and this time is the main component of NT.

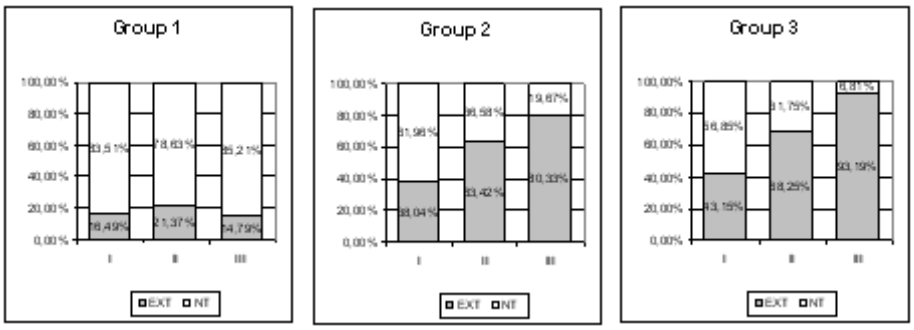

Fig. 3. Parallelism degree

Tests results for the first group confirmed earlier observations, that in case of queries with short execution time or queries which require access to relatively small number of tuples (a few thousands) the time when task stays in agents net is a few times greater than the execution time. For the remaining groups it shows clearly that activating more agents in the system makes better time EXT and NT distribution.

Results for both groups (2 and 3) are similar. In case of the second group the net time percentage decreases from about 52% in configuration I to approximately 19,5% in configuration III. For the third group the net time constitutes almost 57% of execution time in configuration I while in configuration III NT share falls to less than 7%.

Results obtained for groups 2 and 3 allow to state that, when the number of agents increases we gain greater parallelism degree which in consequence shortens tasks execution time. Decreasing the average net time means that tasks wait in GADS catalogue for a shorter time. This situation can take place only if the number of active agents simultaneously executing tasks grows .

It has to be pointed out that the less optimal configuration with respect to parallelism degree is the configuration I. It comes from the fact that there is only one agent which can execute tasks. The best configuration in our case is the configuration III, because the number of agents is equal to the number of nodes – possible parallelism degree is maximal. Increasing the number of agents above the number of nodes can only lead to the situation in which some agents stay idle. Of course one may state that agents number greater about one than nodes number is more appropriate because it can suggest that ordering agent is less loaded. But it has to be remembered that while ordering agent waits for nodes results it can execute another task. This does not decrease system efficiency. Even when the number of agents is greater than the number of nodes it does not guarantee that ordering agent will not take some task to execute.

6 Summary

The system presented above is responsible for querying distributed data structure which stores information in local data warehouses located in autonomous nodes. Query returns results as if it was executed in a non-distributed structure. It is necessary to emphasize that there is no need to exchange any data between nodes. This kind of system is a solution suitable for distributed environments based on autonomous and heterogenic (as regards operating system and database management system) nodes.

Proposed solution can be used for any distributed database system but is especially convenient for spatial and telemetric data because these kind of data are characterized by geographical position. This feature constitutes natural basis for data distribution and allows to distinguish objects belonging to the particular area covered by the particular node. In consequence we can distribute all data characterized by position and avoid unnecessary dimension data replication.

Results of conducted tests allow to state that presented solution enables effective querying in distributed structure. It is enough scalable and flexible so it can be easily adjusted to the current needs by activating another

agents or including additional nodes. This kind of operations require only adding necessary information to the configuration files which describe the system structure. Adding another nodes to the system does not decrease the efficiency provided that principle – number of active agents is equal to the number of nodes – is preserved.

Further research considering presented solution will concentrate on extending SQL query syntax with elements allowing advanced data analysis.

References

1. Bernardino J. and Madeira H.: *Data Warehousing and OLAP: Improving Query Performance Using Distributed Computing,* 12th Conference on Advanced Information Systems Engineering, Stockholm, Sweden, June 2000.
2. Ullman J. D., Widom J.: *A First Course in Database Systems,* Prentice-Hall (1997).
3. Stillger M., Obermaier J. K., Freytag J. C.: *AQuES: An Agent-based Query Evaluation System,* Second IFCIS International Conference on Cooperative Information Systems (CoopIS'97).
4. Suciu D.: *Distributed query evaluation on semistructured data,* ACM Trans. Database Syst. 27(1): 1-62 (2002).
5. Bernardino J., Furtado P. and Madeira H.: *DWS-AQA: A Cost Effective Approach for Very Large Data Warehouses,* International Database Engineering & Applications Symposium, July 2002.
6. Bernardino J., Furtado P. and Madeira, H.: *Approximate Query Answering Using Data Warehouse Striping,* Journal of Intelligent Information Systems − Integrating Artificial Intelligence and Database Technologies, Vol. **19**, # 2, Elsevier Science Publication, September 2002.
7. 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.
8. Chen Z.: *Computational Intelligence for Decision Support,* CRC Press LLC (2000) 108–121.
9. Gorawki M., Bańkowski S.: *Software Agents in Grid Environment,* I National Scientific Conf. -Technology of the Data Processing , Poznań, pp. 118-129, 2005.
10. Elmasri R., Navathe S.,B.: *Fundamentals of Database Systems,* Third Edition, Addison-Wesley (1999).
11. Gray J., Chaudhuri S., Bosworth A., Layman A., Reichart D., Venkatrao M., Pellow F., Pirahesh H.: *Datacube: a relational aggregation operator generalizing group-by, cross-tab and sub-totals,* DataMining and Knowledge Discovery, **1** (1997) 29-53.