The development of load balancer and parallel database management module

R. F. Gibadullin
Department of computer systems
KNRTU-KAI
Kazan, Russia
landwatersun@mail.ru

I. S. Vershinin
Department of computer systems
KNRTU-KAI
Kazan, Russia
vershinin_igor@rambler.ru

R. Sh. Minyazev
Department of computer systems
KNRTU-KAI
Kazan, Russia
txf13@mail.ru

Abstract — In the article it is discussed main development aspects of the load balancer for the cluster. The goal is to increase the speed of queries processing to a database running with PostgreSQL DBMS on Windows operating system. The developed balancing method was chosen by performance and reliability criteria. The software module distributes evenly the load between 4 cluster nodes depending on how much CPU is used on each of the work nodes. The dynamic load balancer processes SQL queries in parallel. The parallel database management module performs encryption of the queries results. The work speed of the developed balancing software module has been studied on a representative set of queries, the obtained results were compared with the work of balancing configuration, consisting of 2 work nodes, and with parallel and sequential processing of the representative set of queries on a single node.

Keywords — database management system, network load balancing, task parallel library, message passing interface

1. INTRODUCTION

The article considers the development of load balancer and parallel database management module.

Relevance of the work.

Load balancing system — it is tool designed to redirect client queries to the least loaded cluster node that contains exact copy of the information resource. The customer has no idea that he appeals to the whole group of servers (cluster): they all appear to him as a single virtual server. Nowadays, there are many strategies and algorithms of load balancing. Lack of balancing results in underutilization computing resources and the potential delay of query results, and the use of balancing mechanisms can significantly increase the speed of query processing. Therefore, the task of development and research of the load balancer and parallel query processing module is relevant [1].

The goals of the work is to increase processing speed of queries for encryption to a database managed by PostgreSQL DBMS.

To achieve these goals the paper investigates and solves the following tasks:

1. Consider the load balancing algorithms.
2. Select the most appropriate method for the load balancing.
3. Development of the load balancing program module by the selected method.

4. Evaluate the developed module by performance and reliability criteria.

Types of query parallelism.

The database software can exploit the three forms of parallelism inherent in dataintensive application workloads. Inter-query parallelism enables the parallel execution of multiple queries generated by concurrent transactions. Intra-query parallelism makes the parallel execution of multiple, independent operations (e.g., select operations) possible within the same query. Both inter-query and intra-query parallelism can be obtained by using data partitioning, which is similar to horizontal fragmentation. Finally, with intra-operation parallelism, the same operation can be executed as many sub-operations using function partitioning in addition to data partitioning [2].

A parallel DBMS can be defined as a DBMS implemented on a multiprocessor computer. This includes many alternatives ranging from the straightforward porting of an existing DBMS, which may require only rewriting the operating system interface routines, to a sophisticated combination of parallel processing and database system functions into a new hardware/software architecture. As always, we have the traditional trade-off between portability (to several platforms) and efficiency. The sophisticated approach is better able to fully exploit the opportunities offered by a multiprocessor at the expense of portability. The solution, therefore, is to use large-scale parallelism to magnify the raw power of individual components by integrating these in a complete system along with the appropriate parallel database software. Using standard hardware components is essential in order to exploit the continuing technological improvements with minimal delay [3].

Types of parallel architectures.

Parallel system architectures range between two extremes, the shared-nothing and the shared-memory architectures. A useful intermediate point is the shared-disk architecture [4]. In the shared-nothing approach, each processor has exclusive access to its main memory and disk unit. Thus, each node can be viewed as a local site (with its own database and software) in a distributed database system. The difference between shared-nothing parallel DBMSs and distributed DBMSs is basically one of implementation platform, therefore most solutions designed for distributed databases may be reused in parallel DBMSs.
Shared-nothing architecture has three main virtues: cost, extensibility, and availability. On the other hand, it suffers from higher complexity and (potential) load balancing problems. Examples of shared-nothing parallel database systems include the Teradata’s DBC and Tandem’s NonStopSQL products as well as a number of prototypes such as BUBBA, EDS, GAMMA, GRACE, PRISMA and ARBRE.

In the shared-memory approach, any processor has access to any memory module or disk unit through a fast interconnect (e.g., a high-speed bus or a cross-bar switch). Several new mainframe designs such as the IBM3090 or Bull’s DPS8, and symmetric multiprocessors such as Sequent and Encore, follow this approach.

Shared-memory has two strong advantages: simplicity and load balancing. These are offset by three problems: cost, limited extensibility, and low availability.

Examples of shared-memory parallel database systems include XPRS, DBS3, and Volcano as well as portings of major RDBMSs on shared-memory multiprocessors. In a sense, the implementation of DB2 on an IBM3090 with 6 processors was the first example. All the shared-memory commercial products (e.g., INGRES and ORACLE) today exploit intra-query parallelism only (i.e., no intra-query parallelism).

In the shared-disk approach, any processor has access to any disk unit through the interconnect, but exclusive (non-shared) access to its main memory. Each processor can then access database pages on the shared disk and copy them into its own cache.

Shared-disk has a number of advantages: cost, extensibility, load balancing, availability, and easy migration from uniprocessor systems. On the other hand, it suffers from higher complexity and potential performance problems.

Examples of shared-disk parallel DBMS include IBM’s IMS/VS Data Sharing product and DEC’s VAX DBMS and Rdb products. The implementation of ORACLE on DEC’s VAXcluster and NCUBE computers also uses the shared-disk approach since it requires minimal extensions of the RDBMS kernel. Note that all these systems exploit inter-query parallelism only.

Load balancing.

The load balancing planning issue should be solved at the early stage of development of any project. At the beginning, the lack of server performance caused by increasing loads can be solved by increasing of the server capacity or by optimization of the program codes, algorithms. But ultimately, there comes a time when these measures are insufficient. We have to resort to clustering, when multiple servers consolidated together in a cluster. The load is distributed between them using a set of special techniques, called balancing. In addition, to resolution of high load, clustering also provides backup of servers between each other. Efficiency of clustering depends on how the load is distributed between nodes of the cluster [5].

Load balancing can be carried out by means of either hardware or software tools.

II. TEST DATABASE. SPATIAL QUERIES

Spatial data (geographic data, location data) – data of spatial objects and sets of them. Objects can be spread randomly or by a law, regularly or irregularly. Spatial data are the basis of information stored in information systems.

There are two types of spatial data geometry and geography. The geometry data type supports planar and Euclidean data (coordinate system for surface of Earth). Geometry corresponds to the specification «Simple Features for SQL» of OGC consortium version 1.1.0 and to SQL MM standard (ISO standard). Geography data type used for storing elliptical data such as GPS coordinates of latitude and longitude.

In this project the geometry type was used.

For the work was selected test database with Russian Federation surface information. The data in the open data set presented in layers, in vector format. The data is in the geographic WGS 84 coordinate system, projection is absent.

The names of the data files are russian, but written by Latin letters. For attribute data encoding format is Win1251. For other formats - UTF-8.

The database consists of 13 layers (tables):
1. Administrative boundaries (administrativnye)
2. Highways (avtodorogi)
3. Drainage system (gidroset)
4. Leisure (dosug)
5. Railways (zheleznye-dorogi)
6. Railway stations (zheleznodorozhnye-stancii)
7. Buildings (zdaniya)
8. Land tenure (zemlepolsovanie)
9. Settlements (naseleynnye-punkty)
10. Vegetation (rastitelnost)
The meaning of the functionality of spatial databases is to perform database queries that otherwise would require desktop GIS.

Spatial query - a structured query to spatial data, the criteria of which are conditions associated with the vector geometry coordinates. For example, the query requesting the presence of crossing points of any line objects.

Non-graphical information can be linked with spatial data, which characterize object additionally in the system. Moreover, any object in the GIS information model can be represented by a collection of features and sets, associated semantic attributes, describing the object in the same way as if it were present in any non-graphical system. For example, if GIS uses DBMS for data store, the semantic part of the objects description - are records in the relational database tables. Quite often structured queries to data under GIS management is a symbiosis of traditional SQL queries to the database and spatial query parameters. SQL is based on a formal query language, relational algebra, it is intuitive, universal and ease in use.

Spatial management systems are special case of expanding DBMS and work with both spatial and non-spatial data, so it is natural to try to find the expansion of the SQL, allows access to spatial data [6].

Generally speaking, the queries can be divided into two categories: those that require a single-pass scan (single-scan queries), and those that require multi-pass scan (multiscan queries). With processing of the queries of first category appeal to the requested record (tuple) of the table (relation) should not be made more than once. Therefore, in the worst case - in terms of time - each record will be removed from the table and checked for compliance with the search criteria.

An example of a query that requires a multi-pass scan, is a join query. To provide a result of the query, the DBMS should generate a sample and combining the two tables of the database. If in query is needed to process more than two tables, they can be processed in pairs. Two tables join by a common attribute. Since the records in the same table may be associated with more than one entry in another table, it is likely that for completion of one join, one apply for records will be insufficient. In the context of spatial database when join attributes are spatial by nature, such a query is called a spatial-join query.

Constructions of object-oriented programming, such as the user types, as well as inheritance of data and functions, have been applied for creation of complex data models. Widespread use of the relational model and SQL language in applications involving simple data types, combined with the functionality of an object-oriented model has led to the birth of a new "hybrid" paradigm of database management systems, referred to as object-relational database management system.

III. ALGORITHMS OF LOAD BALANCING AND PARALLEL MODULE

After reviewing of various load balancing algorithms it becomes clear that the most appropriate algorithm has a dynamic type. In order to make the load balancer more robust, versions of implementation, which would use shared files (for example, local queues of queries on each of the nodes), have been eliminated.

Since each cluster node has 24 cores, to accelerate the query processing it is expedient to run them in parallel.

Technical equipment issued for the project is a cluster consisting of 5 nodes, each node is Intel Xeon E5 2640 2.7GHz, 128Gb ram, mech HDD. (IP are 10.114.22.10, 10.114.22.20, 10.114.22.30, 10.114.22.50, 10.114.22.60). Each node runs on Windows operating system. The nodes are connected by a local network, TCP protocol. On each of the cluster nodes a replica of complete geospatial database is kept. The cluster has shared nothing parallel architecture type.

The cluster has a shared nothing architecture, and each node stores a full replica of the database, the most reasonable is to deal with inter-query parallelism. (when a query entirely, without breaking into sub-queries, sent for execution on one of the nodes). Rejection of intra-query parallelism will minimize the exchange of data over a local network between nodes.

All user queries initially come into a repository, queue, on the main node of the cluster. In my case, the repository is a txt file «Queries», which contains all the queries to be executed. The balancer detects the presence of queries in the repository, and sends it for execution on the least loaded node of the cluster, the load of the node is determined by CPU index, queries run in parallel.

The available cartographic base has no enormous size, has no complicated structure, so there is no need to analyze the queries on join tables parameter. Moreover, the project has character of protected cartographic databases, so it was decided to work with encryption requests. Request of this type are the most difficult for processing. For comparison, the execution of conventional selective query from table containing approximately 10 million rows, took only 32 msec.

The proposed version of load balancing is the most appropriate, and it can be characterized by the classification as: dynamic, non-adaptive, centralized, specialized, not predictive, without taking into account the causes of imbalance, cache-unaware.
Three types of processes are used for load balancing:
At main cluster node:
- 1 process for distribution on the queries between work
  nodes for processing
At work nodes:
- Process sending CPU information to the main node
- Process receiving a query from the main node and
  launching it in parallel module

IV. DEVELOPMENT OF THE LOAD BALANCING METHOD

The load balancer has been implemented by means of C#
language in Visual Studio 2013 environment with using
MPI.NET library.

MPI is a standard on software tools to enable
communication between the branches of a parallel application.
MPI stands for "Message passing interface" [7].

Although MPI-applications show a high level of
performance, the technology has a number of drawbacks:
- low programming levels (programming in MPI is often
  compared to programming in assembler), the need for
detailed management of arrays and distribution between
the processes, as well as the exchange of messages
between processes – all this leads to high complexity of
development programs;
- the need for redundant data type specification in the
  transmitted messages, as well as the presence of severe
restrictions on the types of data transmitted;
- the complexity of writing programs that could be executed
  at arrays of arbitrary size, and with arbitrary number of
processes – makes it virtually impossible to re-use existing
MPI-programs;
- it not supports object-oriented approach. In the
implementation, for the exchange of information between
processes running on different sites I’ve used MPI
functions Send and Receive. But as the MPI standard does
not support object-oriented approach, I could not send
objects of my own type, only the standard types int, string,
etc. However, the objects of its type, can be created and do
with it any operations within a single MPI process.

On each node of the cluster all necessary MPI utilities have
been installed. For balancer uses MPI process manager
smpd.exe, which is a system service (the service application).
Process Manager maintains a list of computer system
components, and runs on these nodes MPI-program, providing
them with the necessary information for processing and
message exchange.

Parallel module takes the data for encryption from DB1
and saves it in an encrypted form in the DB2.

In work of the balancer 9 MPI processes are involved:
- 2 processes at each of the work nodes (there is 4 work
  nodes: 10.114.22.20, 10.114.22.30, 10.114.22.50,
10.114.22.60) it is 8 processes in total:
  - First process sends information about CPU to the main
    node.
  - Second process receives user queries from the main node
    and launch them in parallel by TPL.

1 process on the main node (10.114.22.10), checks whether
there are queries for execution, and then asks 4 processes
about CPU on 4 nodes respectively, identifies the lowest CPU,
sends the query to the node with the lowest CPU index.

CPU of each working unit is taken as follows:

```
var cpuCounter = new PerformanceCounter("Processor", "% Processor Time", "_Total");
usage = (int)cpuCounter.NextValue();
System.Threading.Thread.Sleep(50);
usage = (int)cpuCounter.NextValue();
```

Then the usage value is sent to MPI process on the main
node. Between the values of the usage there is a small sleep
pause, as the CPU value should be compared with the
adjacent, and for getting the new value needs to pass some
time.

Among the difficulties of working with MPI library should
be noted that it is usually assumed that MPI programs perform
computational task of the same kind at each node. In our case,
each process has a specific task, depending on which of the
nodes it was started. But, on which node, process with which
rank (process id) will be launched can not be predicted.
Therefore, a solution was found - just after the start of
balancer, each process sends information about its rank and node at which it was launched, to the process with the rank 0. The process with rank 0, in turn, analyzes the data and performs ordered rank recording in the array, depending on which process has been started on which node. For example, we may know that the ranks of the processes launched on the node 10.114.22.30 are stored in the 3 and 4 elements of the array. After the process with the rank 0 formed the array, it sends the array to the rest MPI processes. So each MPI process is informed about process with which rank at which node was launched. And depending on in which element of the array the rank of the process was recorded, the process gets to know its task. It is necessary because communication between MPI processes is made by taking into account the ranks (IDs of processes). Look the block diagram in the Fig. 7.

V. EVALUATION

Evaluation of performance speed of the described above load balancer was made on a set of 1,000 queries for encryption. Execution of one selective request for 1500 records from «avtodorogi» table takes 463 ms. Withdraw of 1500 records in conjunction with the encryption of the field containing the geospatial data and save of the result in the table «results» takes 4,121 seconds on 1 process. In total the load balancer performs 1,000 operations of such type.

To evaluate the balancer speed, the set of 1000 queries was executed in following configurations [8-15]:

- sequentially on one node with 1 process
- on one node in parallel
- on 3 nodes in parallel (1 main node + 2 work nodes)
- on 5 nodes in parallel (1 main node + 4 work nodes)

Since each task encrypts 1500 entries, and amount of queries is equal to 1000, the amount of records in the result tables must be equal to 1 500 000.

The most appropriate CPU threshold value for launching queries in parallel equals to 70, as in fact, in spite of the CPU threshold equals to 70 the CPU reaches 80. And it should be taken into account, that it is necessary to leave spare CPU for the work of some background programs mutually with the load balancer, so for load balancer at each work node CPU threshold equals to 70 is optimal.

The total amount of records is 1500 000, which indicates that with the 70 CPU threshold all TPL processes have started and completed their work correctly. The load balancer works stable.

The execution time of 1,000 tasks on the configuration of 5 nodes is 90 seconds. About 10 seconds of which it has taken to MPI processes to determine at which node which process was launched, by sending the information process to the process with rank 0, as was described above (Fig. 7).

During experimentation with other parallel configurations (for a single node in parallel; and in 3 nodes configuration), the CPU threshold queries to start was also equal to 70.

TABLE 1. Amount of records in the resulting database tables

<table>
<thead>
<tr>
<th>Amount of records in result table</th>
<th>CPU threshold = 70</th>
</tr>
</thead>
<tbody>
<tr>
<td>345 000</td>
<td></td>
</tr>
<tr>
<td>345 000</td>
<td></td>
</tr>
<tr>
<td>348 000</td>
<td></td>
</tr>
<tr>
<td>462 000</td>
<td>Total: 1500 000</td>
</tr>
</tbody>
</table>

TABLE 2. Speed of 1000 queries processing in different load balancing configurations

<table>
<thead>
<tr>
<th>Speed of the 1000 tasks processing (Intel Xeon E5 2640 2.7GHz, 128Go ram, mech HDD)</th>
<th>Elapsed time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 node, sequential processing</td>
<td>3021</td>
</tr>
<tr>
<td>1 node, parallel processing</td>
<td>341</td>
</tr>
<tr>
<td>Balancing between 2 work nodes (3 node configuration)</td>
<td>169</td>
</tr>
<tr>
<td>Balancing between 4 work nodes (5 node configuration)</td>
<td>90</td>
</tr>
</tbody>
</table>
been excluded from the following bar chart:

For clarity, the result of sequential query processing has been excluded from the following bar chart:

Thus, these results suggest that configuration of load balancer consisting of 5 nodes works faster 1.87 times faster than configuration consisting of three nodes, 3.78 times faster then parallel processing of these requests on a single node, and 55.7 times faster then sequential processing of the data requests on a single cluster node. And it takes 90 seconds to process the representative set of cryptographic queries.

VI. CONCLUSION

The experimental results show that with using MPI and TPL libraries was developed software module for dynamic load balancing on a cluster consisting of 5 nodes (Intel Xeon ES 2640 each). The balancer handles the 1,000 cryptographic tasks in 90 seconds. The configuration of load balancer gives acceleration at 55.7 times compared to the sequential processing of the representative set of queries. Performance of the 5 node configuration load balancer also has been compared with the configuration consisting of 3 nodes, and the parallel launching of the representative set of queries at 1 node of the cluster. Based on the fact, that the CPU graphics of all work nodes are similar, we can conclude that the load is distributed evenly. During the development of software module the problem of how to encrypt data was solved, and all the nuances of working with TPL and MPI libraries were taken into account.

References