Realization of Replication Mechanism in PostgreSQL DBMS

Gibadullin R.F., Vershinin I.S., Minyazev R.Sh.
Department of computer systems
KNRTU-KAI
Kazan, Russia
landwatersun@mail.ru, vershinin_igor@rambler.ru, rshminyazev@kai.ru

***Abstract*—Nowadays we have a project of a high-performance parallel encrypted database on a cluster platform that will work with large volumes of data. For the smooth operation of the system, a reliable and high-speed data replication mechanism is required which should satisfy certain criteria of PostgreSQL DBMS and the mentioned project. In this article we consider the very essence of data replication, replication strategy, the basic methods and techniques of replication and also made compliance analyses with the replication method for high-performance parallel DBMS on the cluster platform. We also made an experiment to study the reliability and speed of the method for data replication between nodes in the cluster where the work was done with different amounts of the replica and a different nodes number. The aim was to obtain information on the speed of the Slony-I replication method at different volumes of the replicated data and the different number of nodes in the cluster on which we make a conclusion about the reliability of this method in relation to the project mentioned above.**

Keywords—database management system; replication mechanism; cluster platform

I. Introduction

The replication problem (data synchronization between the nodes of distributed system) arises daily in front of IT companies. In particular, such a problem may arise in the automation process of the company having geographically remote offices. And although the most commonly used approaches to implementing replication algorithms are formed, they have drawbacks, which will be discussed below.

The idea to add data replication between remote nodes of distributed database emerged in the seventies of the last century. Table synchronization task is a special case of data replication, which requires having two tables, it had the same content located at remote sites. Nowadays there are parallel databases, which store large amounts of data. These systems need to have high performance, because they handle large number of queries per minute and the system should give result in acceptable period of time. In this article we consider the replication of large data amounts on the example of a high-performance parallel system, which is based on PostgreSQL database, installed on cluster operating under Windows Server 2012. The problem is that this system in future should work with constantly growing data volumes, which need to be replicated on all nodes of cluster. But still do not have a replication mechanism, which could work with large replicas. And we do not know cluster's behavior under replication workload in gigabytes. The actuality of this work is that we need to analyze and conduct experiments which replication method to use for the smooth performance of the system having replication volumes more than 1GB. The work will be analyzed as well the reliability and speed of the various replication methods for the different replica volumes and different numbers of nodes, whose characteristics are suitable for usage in the cluster system.

II. System replication requirement

Article [1] provides an analysis and description of the main characteristics of the existing data replication systems. The ideal system should satisfy the following set of requirements:

1. Minimizing the impact on existing software systems and databases. Replication system should not require the programmer to access the third-party interfaces and writing of special means of access to the database. It is also appropriate to require changes to existing database structures.
2. The system must detect any changes in the database. They can be made with some third-party software, and using the database management system itself.
3. The system must be efficient, scalable and have a projected cost of the expansion. The system should not duplicate entries are stored, partly or completely. The system should not store long chain of changes in the database. It should be effective even in a very large number of records.
4. The system should be sufficiently autonomous. It is impossible to demand from enterprises need DBA presence on each node of distributed network [2].
5. The system should work well equally on completely different platforms. It should not rely on such specific to each database things like transaction log, etc. But in real life to achieve this requirement is difficult, since in addition to existing the specifics of the implementation of the internal structure there is the specificity of database access language also exists. And, although theoretically the entire SQL compliant database must correspond to a single standard language, in practice it is not so. Note that existing commercial replication systems often do not satisfy all of the guidelines, and some of their characteristics are mutually exclusive, or very difficult in implementation.

In this work it is important to point out that in real life replica of a large (more than 1GB) volume per data transfer session in most cases is dangerous and useless. In case of failure of one system element might cause problems with the availability and/or integrity of the transmitted data. Today, having transfer problems white big data transferring as a replica it is obvious to divide the volume into smaller fragments of data. This gives the following advantages:

1. Work with small data is easier, therefore, chance to successfully deliver data to a remote server is higher.

2. A small piece of data requires less processing resources for transfer it comparing with large volume.

3. Increased control of the transfer process – in case of failure, it is easier to track what and at what stage of implementation something went wrong.

However, in this article, we focus on the large data transfer for investigating the reliability and speed of replication in handling a large amount of data and different numbers of nodes applied to project of high performance encrypted parallel DBMS on cluster platform. Specificity is that the database will handle a small number of requests per minute, but the intensive computation required to perform to get the result database. So, if at least 1 from 10 queries coming for 1 minute on a server is data change request, the replication process is necessary. If the request modifies data more than in a volume of 1 GB, then a delay of data applying on another node exists. This delay negatively affects on relevance of the data, as if during the update request will be sent, the results will already be outdated. One solution to this problem partition replication volume into smaller parts and use them in the correct order. To do this the system must have limiter. But the issue is not the main in this work, so now we move directly to the discussion of our problem.

It is time to define the requirements for replication method applied to the project of high-performance encrypted parallel database on a cluster platform.

So, we have a parallel database system PostgreSQL DBMS running under Windows Server 2012 operation system and we need to find out which methods are available for us.

Let's consider the requirements of the project, what kind of characteristics are necessary for uninterrupted system operation:

Requirements for a project:

1. The project of a high-performance encrypted system is running on parallel computing cluster under Windows Server 2012 operation system. The core of the system is cross-platform GIS system with open source code called QGIS.

2. On each node of the cluster database installed PostgreSQL DBMS. The data on all the nodes are identical.

3. The requests comes from outside to the central node of the. Requests are coming with a frequency of 10 per minute. The request might be *SELECT*, *UPDATE*, *INSERT*, *DELETE* commands. The common satiation is that in this 1 minute we have 9 selection queries and just 1 data changing command like *insert, update, delete*. To make a situation more realistic, we will send data changing commands in random sequence.

4. The physical component of the project is a cluster of 6 servers (nodes); each node has its own database DBMS PostgreSQL 9.5 – the last stable version.

5. Each node has 12 core Intel Xeon processor and 128 GB of RAM. The nodes are connected by local network with a bandwidth of 1GB/s. The disks are typical HDD's with speed rate 7200rpm connected by SATA interface. Each node is independent from others, has it's own memory space, disk storage area and processor. This cluster architecture called share nothing, where nodes are communicating via local network.

To maintain the consistency of data on the cluster nodes we need to have correct data replication mechanism. The main requirement – the maximum speed and reliability of data updates at all nodes. Since the replication process loads processor and local network by additional computational work processes and this consumes computing resources so in the process of replication method selection we should pay attention to generated workload on production systems.

Carefully analyzing the project requirements, the following requirements for replication mechanism have been put forward for this project:

1) Ensuring maximum performance. Since the replication process is followed by additional computational work processes and this consumes computing resources so in the process of replication method selection we should pay attention on generated workload.

2) Support of large amounts of data. Data volumes are growing exponentially, and signs of slowing down this growth are not visible. Therefore, choose a database replication method must quickly process large amounts of data at the same time without overloading the source server.

3) Support for selective replication. Replication tool should enable the selection of individual records and columns or groups, as well as allow you to specify different selection criteria and order for data replication.

4) Provide centralized management. Since modern IT departments have to deal with complex topologies of information, the database administrator should have a functional set of tools to manage them and allows rationally organize the replication process.

5) The replication tool must be easy in implementation and usage. Also it should be flexible and give an opportunity to optimize it to work exactly on this cluster.

So, we have requirements for data replication mechanism and let's analyze:

1. It should be non-synchronous. If we use the eager, the data will be unavailable during the replication session. Especially important that fact, if we are dealing with replication volumes of gigabytes. If during a lazy update come another update request, then it will be queued and problem with command queue do not arise.

2. During the update some nodes may not work, whether the lack of power, or the lack of connection to the node. In this case, there may be situations where some nodes have already updated the data, and some - no. Inconsistency of data occurs, which should not be, especially in large data volume systems. This phenomenon adversely affects the system as a whole, because of spending a lot of time to restore the integrity of the data.

3. Replication mechanism must support not only full, but also partly data replication because this will speed up the process itself

4.Mechanizm must be running under OS Windows Server 2012, because not all PostgreSQL DBMS replication methods are available for Windows. This is due to differences in the Windows and UNIX family of operating systems.

5. Replication mechanism should not be too resource expensive for CPU and network infrastructure

Thus, analyzing the points the requirements of both the project and the proposed replication method, it is possible to list the replication mechanisms that fit the description. On the basis of the analyzed data for the experiments data replication suitable replication technique called Slony-I (fig. 1), since it meets all the criteria, namely:

1. Asynchronous replication
2. Supports full, partial replication
3. Increased fault tolerance by cascading replication
4. Able to work with different hardware components and the various versions of PostgreSQL
5. In case of master failure one of the slaves assigned "temporary" master until the "real" master recovery, which has a positive effect on the data availability [3].

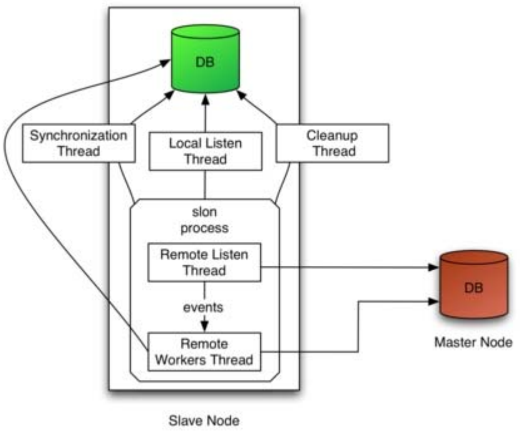

Fig. 1. Slony mechanism

III. Experiment

<u>*Experiment providing*</u>.

For the experiment, we need a large enough data volume on the database. At this point it is important to us to have large amount of data, not quality of data. This is due to the fact that for further research we need to work with the replica volumes whose volumes are measured in gigabytes of data.

For this purposes was created a table with 5 columns:

- Id (primary key)
- full_name
- country
- email
- city

Also we have generated the unique data and placed in the base. For convenience, the unique data entries is equal to 1500, which were added to the table by insert command again and again. The advantage of this approach is that in order to generate a replica of a large volume, which will be sent to other nodes in the cluster, we do not need to write large SQL-queries, but rather a simple insert query type, the delete or update. For example, now the amount of data in the table is 396608000 records, which is equivalent to 107 gigabytes of data. This is more than enough for our purposes. For example, to generate a replica volume of 1GB, we need to change one of 3700000 rows by delete or update command.

For a more detailed research we need to generate also replica volumes of 5 and 10 GB.

Such amount of data also needed to conduct the so-called stress tests, when tested the strength of the replication method itself. For example, one part of our study will be to carry out a stress test on the maximum amount of data to be replicated, in which we measure how many transactions per second the system can handle before failure occurs in the work. Such research process will allows to identify weaknesses in the system and prevent their occurrence in the operation process.

So, we have a database with data volume of 107 GB, and the data on each node in the cluster must be identical. To deploy the database on various nodes we use *pg_dump* and *pg_restore* utilities. The first application work as follows - generate a text file with SQL commands that, when executed on the server, create database in the same state in which it was at the time of the dump. *Pg_dump* is usual for PostgreSQL client application. This means that it is possible to perform the backup procedure from any remote computer that has access to the required database. But remember that *pg_dump* do not have any special privileges. In particular, the utility must have read access to all tables, the backup that you want to do. An important advantage of *pg_dump* over the other backup methods is that *pg_dump* output can be used to restart in a new version of the PostgreSQL, while the backup at the file system level and continuous archiving are strictly dependent on the server version. Also, only *pg_dump* is a method that will work if the database is moved to another machine architecture, for example, when moving from 32-bit to 64-bit version of the server.

Dumps created by *pg_dump* are internally unified, which means that a dump is a snapshot of the database at the time of *pg_dump* launch. *Pg_dump* does not block other operations on the database at the time of their work.

The utility creates a file with the extension «.backup», which contains the entire table, including data, information about the scheme (optional), the information on the table structure, etc.

Pg_restore is a utility for restoring a database from Postgres archive created by *pg_dump* command in any of the non-text formats. It executes the commands needed for recovery of the state of the database in which the database was saved. In the presence of archive files, *pg_restore* can recover data selectively, or even to reorder the items prior to reduction. It should be noted that developed for the archive file format is not tied to the architecture [4].

Pg_restore utility can operate in two modes. If you specify a database name, *pg_restore* connects to that database and restores the contents of the file directly in it. Otherwise, it creates a SQL-script with the commands needed to recreate the database, which is then output to a file or to the standard output device. The generated script will be exactly the same output in the *pg_dump* plain text format. Therefore, some of the parameters that control the output, similar to the ones *pg_dump*.

Obviously, *pg_restore* can restore the information only if it is present in the archive file, and only in its present form. For example, if the archive was created with the indication "dump data as INSERT commands", *pg_restore* will not be able to load the data using COPY statements.

To restore the database on the nodes we need to do the following:

1. Pack the database (in our case, the base called *db1_test*) using *pg_dump* utility.
2. Upload the resulting dump to a network folder, from where it will be available from any node in the cluster
3. Connect to the node on which you want to deploy the dump
4. Use the *pgAdmin* utility to select the location of the file's backup
5. Click restore. Optionally, you can specify restore options.

<u>*Measurements*</u>.

So we have 2 databases, and replication set between the asynchronous master - slave replication Slony. Next, we will study the speed and stability of the method. For doing this it is necessary to conduct a series of experiments, namely:

1. Replication of 1GB data on 1 node
   – 3 nodes
   – 5 nodes
2. Replication of 5GB data on 1 node
   – 3 nodes
   – 5 nodes
3. Replication of 10GB data on 1 node
   – 3 nodes
   – 5 nodes
4. Stress test – a maximum amount of data replication, which can be transmitted at a particular configuration.

The criteria used to measure the reliability and speed of replication module:

1. Changed data transfer rate.
2. The load generated by the server – on storage disks, on network, on processor.
3. The speed of recovery after a failure.

We proceed to the first part of the experiment.

Replicating 1GB of changed data using the update script. This script creates only a single transaction. Therefore, we proceed as follows. To generate changes in the size of 1 GB we will not run the 1 script, but a series of instruction set, which will give a situation close to real. This will require a change 3800000 rows. We divide this operation to 1,000 transactions, and see what happens.

Experimentally it was found that the execution of transactions on the master node takes 27 seconds. This is the actual time of writing data to the hard disk. At the same time we have a 12-core processor, which is loaded with the transfer process only on 1.1% of its maximum capacity. Hard disk read speed is an average of 100MB/s. On LAN transmission it takes another 14s. Data transmission speed over the network depends on the disk read speed. Receiving data over the network and recording on the slave hard disk drive takes about 15 seconds. *Shared_buffer* overflows the buffer, used to store intermediate results in PostgreSQL, does not arise. More operation and *shared_buffer* destination are discussed in chapter 4. Since the Slony-I replication is trigger point technique, modifying any data generate event that starts the replication process. That is, in this case, the process is not waiting for complete formation of the changes list, it starts to work immediately after the first data modification.

As a result, we conclude that the total query execution time for this experience is 27s + 14s + 15s, 56s total, and replication takes about half the time.

Then we continue to experiment and transfer 5GB of data across the network at one slave node. In this case, the process of data relevance recovery takes 405s + 171s + 64s on changes applying on master, transfer over the network and apply the changes on slave node, respectively. As you can see, even the transfer of 5 GB of data takes about replicated 640c. This value is very large for heavy-loaded systems and in the long term can cause malfunction of the system.

However, we go further and try to transfer over the network replica of 10 GB to 1 slave node. In this case, replication is fully takes time 1169s, among them the script on the data change is 830s, directly transfer network covers 97c, the applying of data on the slave node takes 242c. The load on the processor is 8%, the discs are at an average recording speed 68mb/s.

The next step is to replicate the same volume of data from the master node to 3 slave node . Transmission takes 61s 1GB. When the load on the CPU unit and master disks is about the same as in the case of one slave node. But in this case there is an additional load on the local network to which nodes are connected. It may be noted that, since we are dealing with an Ethernet network with a bandwidth of 1 Gbit / s, there is no significant load on the transmission channel.

When transmitting 5 GB of data on a 3 slave node transmission time is 687s.

In the process of transmission of 10Gb on 3 slave nodes host processor spends replication 8.9% of its capacity- that is quite a lot. Such a load on the processor indicates that the server performs serious work and if run an additional application, some difficulties may occur. The average write speed of 71Mb per second, which is close to the theoretical limit for the drive hdd. In this mode, the replica is fully transferred for 1317,1s.

Now we move on to the basic mode of operation, under which the project of high-performance parallel database was designed. We will experience a system to replicate data to 5 slave nodes.

For example, in a case of replica of 1GB can be shown that with this volume the system copes perfectly, showing similar results with a replica of 3 nodes. The main difference – more workload on the disks, the increased load on the network (transmission at 5 nodes). And already the problem begins when a read request to the master node - in the end replication takes 75s.

Just as in the previous case, make 5 GB of data replication. There already exists query processing cost, to establish connections to all databases and sending event notifications. This is a resource-intensive process, so replication takes about 734s.

And finally, the toughest replication mode – transfer 10GB of data at 5 nodes. In this case, time is 1814s. At this stage, the

CPU load of 10%, hard disks almost can not cope with the load (100% load) and are beginning to emerge in the braking system.

TABLE I. TABLE OF REPLICATION SPEED

№ slave nodes/ replica size	*1 GB*	*5 GB*	*10 GB*
1 node	56s	640s	1169s
3 nodes	61s	687s	1317s
5 nodes	75s	734s	1814s

Conducting stress tests.

Experimentally verified that PostgreSQL 9.5 replication can handle a load volume of 20GB with no losses. But this operation is very expensive in terms of resources, as all hard drives are loaded on all servers, then a huge dataflow on network, processor is loaded on 15%, replication is performed at 5 nodes in time of 3422s, together with the implementation of a request to change the data on the master node. The disc is completely loaded by 100%, the data waiting in the buffer, the buffer *shared_buffer* with size of 8GB filled in more than half, that is not good, and could lead to a database crash. All numbers have been taken from the PostgreSQL logs and Windows system analyzer.

IV. THE ANALYSIS OF RESULTS

Transfer 1 GB of data on one node occurs in normal mode, significant load on the processor does not arise. In this mode, replication process takes 1.1% of the CPU power, it should be noted that we are dealing with a 12 core server processor, each core operates at a frequency of 3.3GHz. Trying to run Slony-I replication at a low server could potentially lead to outages. LAN has a bandwidth of 1 Gbit/s, which is enough to transfer this volume. Workload of the communication channel with the replica depends mainly on the speed of reading data on the master node, because we can not pass over more than hard disk drive passes. *Shared_buffer* serves for smoothing excess data when writing data to the disk. However, it should be borne in mind that almost every time while writing operation disks are almost at a maximum writing speed of about 70Mb/s.

The replication of the same volume of 1 GB to 3 and 5 nodes increasing the load on the local network and on the slave processor nodes. You should also take into account the constant read/write process on *sl_log_1* and *sl_log_2* tables on the hard disk on the master takes place. This fact is explanation of increased replication time with data growth. It should be noted that during the replication load on the processors on slave nodes is much smaller than at the master node. For comparison, during replication on the 5 nodes slave's processor load value does not exceed 0.7%.

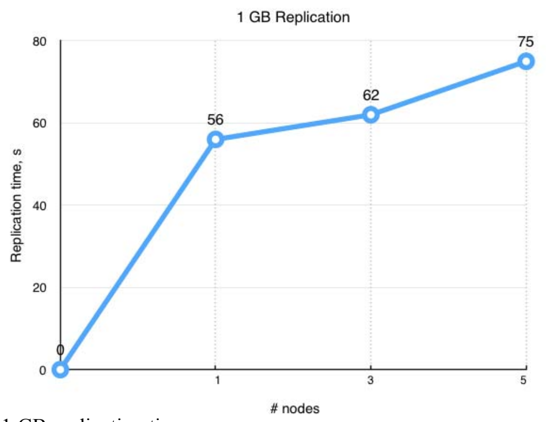

Fig. 2. 1 GB replication time

While replication of 5GB Slony-I behaves itself in the normal mode, handles load correct, but the transmission time increases nonlinearly, due to the increased load on the server as a whole. However, even with such a volume of data transmition critical moments does not appear, but the transfer time becomes totally unacceptable. If you look closely, the number of 640s while transmitting to 1 node means more than 10 minutes server unavailability, while the requests coming to it, will be redirected to the master server. Almost identical situation when working on high-loaded 4 nodes of the cluster, where the data transfer in 3 nodes takes 687s. Additional second explains the costs of communication and the load on the master server, and a request for additional changes on the 2 nodes, compared with a replica on 1 node. Replication on 5 slave nodes takes 734s. Nonlinear increasing of transfer time associated with limited access to the hard drives. The greatest risk of falling occurs on the hard drives, as they have a short time to process large amounts of data.

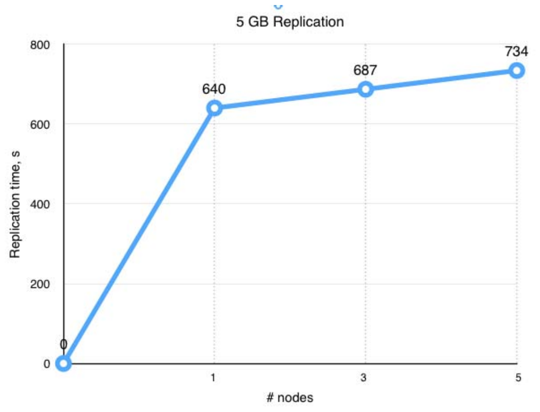

Fig. 3. 5 GB replication time

Finally, the network transmission of 10GB. A rare case in practice, but, nevertheless, consider such a case because data in the near future will grow and may have to work with such volume. Transfer on 1 node takes about 1169s. A more rare case - transfer of 5 GB takes 1317 seconds. And the harsh regime - replicating 10 GB to 5 nodes take 1811 seconds of CPU time. In this mode CPU loads on 10% of its full power of 12 cores. In this mode stable work cannot be recommended, because the failure of one of the components of the system can lead to faulty operation of replication module.

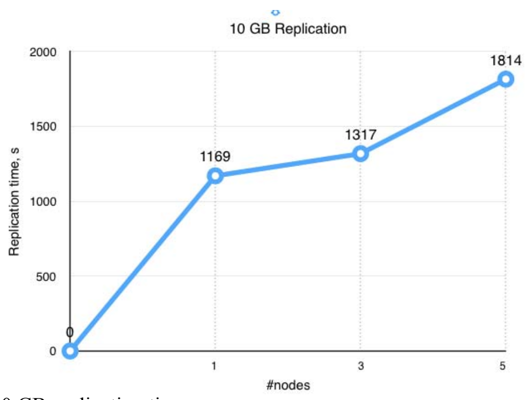

Fig. 4. 10 GB replication time

Check maximum load mode.

For example, try to get a replication with capacity of 20 GB, which is twice more than the maximum projected amount of replication in high-performance database. As can be seen from the results, Slony can cope even with such volume. But when the processor is loading exceeds 15%, the overall system performance is degraded due to insufficient speed of the hard disk and network.

Also we have the results of the experiment with the refusal slave server during replication. So, experimentally proved on a cluster of 1 master and 5 slaves, that during replication session one of the slave servers become unavailable (power failure or unavailability on the local network), there is a situation where some slaves already applied changes and some - no. In this case, the mechanism of Slony goes as follows - it appoints the slave master, then sends EVENT messages to all the other slave and "catch" them with something that is not enough, then send to all the changes, with the result that the entire cluster is synchronized again.

We simulated case of such a refusal and got the result that in the event of failure of one server, the replica of 1GB automatically restored in 95s. Comparing with the usual replica to 1GB, the time difference is due to the time for the restoration of the system, and to read the logs from the tables *sl_log_1* and *sl_log_2* and another system processes.

V. CONCLUSION AND PERSPECTIVE

This article focused on replication of large data volumes in PostgreSQL DBMS within the project of high-performance encrypted cartographic database on a cluster platform [5-15]. Correct replication method had been chosen and implemented into system. Received results obviously present that replication more than 1 GB of data could take lots a time and potentially increase possibility of failures. However, Slony-I can handle gigabytes of data, but in unacceptable time. Finally, Slony-I proved its fault tolerance and could be recommended as a replication method within the project mentioned above.

Further recommendations:

1. The study and comparison of the replication methods of Slony and Bucardo – as 2 equal asynchronous master-slave replication methods on OS Linux. Bucardo was selected as one of the most popular and easiest ways to replicate, available for PostgreSQL.

2. Study of the replication master-master to evenly distribute requests to change data for PostgreSQL database within the project of a high-performance encrypted cartographic database on a cluster platform.

REFERENCES

[1] Overview & Comparison of Data Replication Architectures, Progress Software Whitepaper, 2005.

[2] R. Bhagwan, D. Moore, S. Savage, and G.M. Voelker, “Replication Strategies for Highly Available Peer-to-Peer Storage,” Proceedings of International Workshop on Future Directions in Distributed Computing, Italy, June 2002.

[3] B. Bhargava, A. Helal, and K. Friesen, “Analyzing AvailabilityofReplicated Database Systems,” International Journal of Computer Simulation, 1, pp. 393-418, 1992.

[4] F. Broquedis, O. Aumage, B. Goglin, S. Thibault, P.A. Wacrenier, and R. Namyst, “Structuring the execution of OpenMP applications for multicore architectures,” in Proc. of the IEEE International Symposium on Parallel & Distributed Processing, IPDPS, 2010, pp. 1-10.

[5] V.A. Raikhlin, I.S. Vershinin, R.F. Gibadullin, S.V. Pystogov, “Reliable Recognition of Masked Binary Matrices,” Connection to Information Security in Map Systems, Lobachevskii Journal of Mathematics, vol. 34, no. 4, pp. 319-325, 2013.

[6] C. Mei, G. Zheng, F. Gioachin, and L.V. Kale, “Optimizing a parallel runtime system for multicore clusters: a case study,” in Proc. of the 2010 TeraGrid Conference, TG 2010, New York, 2010.

[7] R.F. Gibadullin, “The development of a uniform formalism protection point, line and area objects of cartography,” Vestnik KSTU named after A.N. Tupolev, vol. 2, pp. 102-107, 2010.

[8] I.S. Vershinin, R.F. Gibadullin, “Changing the recognition results on the set of masked binary matrices by the action of additive noise,” Vestnik KSTU named after A.N. Tupolev, vol. 4, no. 1, pp. 198-206, 2012.

[9] I.S. Vershinin, R.F. Gibadullin, S.V. Pystogov, V.A. Raikhlin, Associative steganography (Appendix to the analysis of scenes), Kazan: KU, 2014.

[10] R.F. Gibadullin, A.A. Novikov, “Parallel execution unit of spatial queries to secure the map database,” in Proc. The search for effective solutions in the process of creating and implementing scientific developments in the Russian aviation and space industry, Kazan, 2014, vol. 2, pp. 421-424.

[11] A.A. Novikov, R.F. Gibadullin, “Parallel execution of spatial queries to secure the map database,” in Proc. High-performance parallel computing on cluster systems, HPC, Perm, 2014, pp. 303-307.

[12] V.P. Gergel, R.G. Strongin, “Parallel computing for globally optimal decision making on cluster systems,” Future Generation Computer Systems, 21(5), pp. 673-678, 2005.

[13] Foster, C. Kesselman, The Grid. Blueprint for a New Computing Infrastructure, Los Altos: Morgan Kaufmann Publishers, 1998.

[14] L. Boloni, D. Turgut, D.C. Marinescu, “Task distribution with a random overlay network,” Future Generation Computer Systems, 22(6), pp. 676-687, 2006.

[15] P. Stelling, I. Foster, C. Kesselman, C. Lee, G. von Laszewski, “A fault detection service for wide area distributed computations,” in Proc. of 7th IEEE Symposium on High Performance Distributed Computing, 1998.