ISSN 0146-4116, Automatic Control and Computer Sciences, 2014, Vol. 48, No. 7, pp. 437–444. © Allerton Press, Inc., 2014.
Original Russian Text © E.A. Boytsov, 2013, published in Modelirovanie i Analiz Informatsionnykh Sistem, 2013, No. 4, pp. 136–149.

Designing and Development of an Imitation Model of a Multitenant Database Cluster

E. A. Boytsov
Demidov Yaroslavl State University, ul. Sovetskaya 14, Yaroslavl, 150000 Russia
e-mail: boytsovea@yandex.ru

Received October 10, 2013

Abstract—One of the main trends of recent years in software design is a shift to the Software as a Service (SaaS) paradigm, which brings a number of advantages for both software developers and end users. However, this transition brings new architectural challenges in addition to these benefits. One of them is the implementation of a data storage that would meet the needs of a service-provider while at the same time providing a sufficiently simple application programming interface for software developers. In order to develop effective solutions in this area, the architectural features of cloud-based applications should be taken into account. Among others, such key features are the need for scalability and quick adaptation to changing conditions. This paper provides a brief analysis of the problems concerning arranging cloud data storage systems based on the relational model, and it proposes the concept of database cluster RDBMS designed for applications with multitenant architecture. Moreover, the article describes a simulation model of such a cluster, as well as the main stages of its development and the main principles forming its foundation.

Keywords: databases, SaaS, multitenant architecture, scalability, imitation model

DOI: 10.3103/S0146411614070049

INTRODUCTION

One of the main trends of recent years in software design is a shift to the Software as a Service (SaaS) paradigm. The point of the given method is defined by the following key statements [1]:

—The application is developed in the form of a system of distributed web services interacting with each other.

—The service provider gives calculated powers and infrastructure that are necessary for the application operation.

—The payment for the application is charged on using rather than a flat rate at purchasing a license.

The main advantage of the approach to the development for the end user is that the service provider takes all the expenses for arranging the necessary infrastructure. However, the transition to the “clouds” causes new problems for developers and administrators of such the systems. A key feature of the applications developed according to the SaaS paradigm is the need for simple scaling: since the size of the tenant base is large while the dynamics of its changing is unpredictable, to provide scalability is a vitally important problem for developers.

One of the approaches to provide a high level of scalability is using multi-tenant architecture. Such a method assumes that the basic resources of the application are shared among many custromers, while the program logic of the application is responsible for the isolation of their data. The design of the solution according to such architectural principles allows one to use an almost unlimited number of application instances to serve the tenants at the presence of sufficient calculating powers. However, the mentioned considerations do not expand to DB servers, since the given system component do not scale well. In the traditional client- server systems, the scaling problem is less acute, because a single typical modern DBMS can satisfy the need of a medium and even a large enough company.

Many current RDBMSs provide a wide set of features, among which are the support of transactions, procedure extensions, and different mechanisms of replication and tuning of the server for a certain profile of load. On one hand, such wide functionality makes the system very convenient upon developing the applied software and allows saving a lot of time and money of the development company. On the other hand, it hinders the horizontal scaling, which is one of the key needs of a cloud application. This fact makes the developer company select between two ways:

—To attempt to use the solutions by vertical scaling of RDBMS, which requires a large contribution to the infrastructure, licenses for software, and payment for the labor of highly skilled personnel.

—To start using less functional but more scalable NoSQL solutions, which increases the cost of development due to the necessity of training the personnel in new technologies and implementation of additional functionality by the developers themselves.

As a rule, certain solutions always turn out to be in the middle. Some application parts that probably do not relate to very important information (caches, storage of temporary data, and so on) are converted to the use of NoSQL DBMSs, which are better in serving such problems, while other parts use RDBMSs as before.

In terms of the support of the multitenant architecture on the RDBMS level, the given technology does not have the required instrumental support of the software developers despite there being some design works and patterns in this area [2–4]. The developers should perform the isolation of the tenant data, while the resultant DBs are not easy to administrate owing to their high complexity and large size. The cloud application added to the wide audience requires tens of DBs of such a structure, since it is vitally important to have at least one copy of data of each client due to the need to provide stability against bugs and data integrity, as well as to implement the application productivity and loading distribution. A more significant problem in comparison with the number of databases is the balancing of the load and the optimal use of the calculated powers of the provider company. To maintain the required service level, a cloud application should be able to adapt to a change in the load profile by automatic resource redistribution dynamically. At the present moment, there are not both the program facilities to solve this problem in a complex and certain requirements for such systems and verified algorithms that can be used for their construction. Therefore, the idea of developing a theoretic model of such a system and investigating their properties arises. The mentioned model will be said to be a multi-tenant database cluster. Hereinafter, the main characteristics of the assumed concept will be discusses. Now, we consider the current achievements in the researched area.

1. OVERVIEW OF THE EXISTING SOLUTIONS

Some investigators already considered the problem of developing reliable data storage that is adapted to the needs of cloud applications. There are two working directions:

—The development of radically new NoSQL DBMS and data processing methods.

—The adaptation of RDBMS for the needs of cloud applications with multi-tenant architecture.

The development of the first direction resulted in the appearance of new solutions in the applied software field. Here, such NoSQL DBMSs as Memcached, Redis, Cassandra, and MongoDB, which are used in thousands of applications all over the world, should be mentioned. The given class of systems puts more emphasis on the performance and scalability rather than the functionality. The most simple DBMSs from the mentioned list (Memcached and Redis) theoretically allow one to serve up to 100000 simple queries per second [5, 6]. As the functionality (different methods of data indexation, complex sampling) increases, they are also subjected to degradation of performance and face the same problems as the traditional RDBMSs, which implicitly confirms the verity of the CAP theorem [7]. It states that it is possible to provide not more than two of the following three properties in any implementation of distributed calculations:

—Data consistence; i.e., the data in all the calculated nodes at one time instant do not contradict each other.

—Availability; i.e., any query to a distributed system terminates by the correct response.

—Partition tolerance; i.e., the division of a distributed system into several isolated sections does not lead to the incorrectness of the response of each section.

Moreover, the model of distributed calculations MapReduce and the framework for distributed calculations Apache Hadoop should be noted in the series of technologies simplifying the work of developers of cloud applications.

In traditional RDBMSs, four directions that can be used upon developing cloud distributing applications can be selected:

—Solutions of clustering and vertical scaling of DBMS.

—Architectural concepts for arranging multi-tenant infrastructure within one DBMS.

—Prototypes of solutions for native support of multi-tenant architecture at the DBMS level.

—Prototypes of solutions for controlling distributed clusters of RDBMSs with multi-tenant architecture.

The solutions from the first category are long standing and traditional for RDBMSs. Here, there can be different options: from additional program options to specialized hardware. Note that the solutions of such type were traditionally developed for other purposes, namely, for providing the functioning of software that serves large companies with big arrays of data when the processing of tens of millions of records is required to construct one report. The given scenario does not correspond to the needs of typical cloud applications, whose main users are small and medium companies not generating large calculated loads separately. In the case of cloud applications, the load is created by the number of queries rather than their "heaviness." Correspondingly, though the mentioned solutions can be applied, the developer company has to describe an additional layer of program logic that is responsible for the tenant distribution and query routing. The most relevant investigations [2] were carried out in the area of searching for methods of multi-tenant environment in the existing DBMSs. In the context of the research, two approaches were proposed:

—The use of shared tables for data storage of all the tenants of the provider company. Upon applying such a method, the column storing the tenant identifier, which is the holder of the given record, is added to each table in the DB.

—The dedicated schema for the tenant. Upon using the given approach, the tenant data are isolated from each other by generating their own set of DB objects for each one.

In terms of the given investigation, some developers offered the concepts of expanding RDBMSs, which are meant to provide the native support of multi-tenant architecture on the server level [3]. Such systems allow one to design the DB structure according to the requirements of the multi-tenant architecture while constructing the hierarchy of the tenants in the object-oriented style.

However, one DB server is not enough for the cloud application operation. To serve the tens and hundreds of thousands of queries, a cluster is needed. The problem of arranging such a cluster was considered by some researchers. In particular, the principles of migration of data in clusters with the multi-tenant architecture were discussed, and A protocol of implementing such migration was proposed [8]. Moreover, the minimization of the cost of hosting a cluster [9, 10] in using the in-memory DBs and placing the infrastructure based on the services of IaaS providers, as well as the ways of arranging the tenants for minimization of the expenses to maintain the service level agreements (SLA) [11], were considered.

2. MULTI-TENANT DATABASE CLUSTER ARCHITECTURE

Prior to describing the architecture of a multi-tenant database cluster, let us make several remarks regarding the behavior of the cloud application for which its service is meant. As is known, small and medium companies prefer cloud solutions to reduce the expenses for maintenance of their IT infrastructure. To attract the attention of such enterprises, the provider should present a solution with a relatively low (in comparison with the traditional "box" version) price [12]. Consequently, to pay for the development of an application, the served tenant base should be enough large. Thus, the considered application relates to the flow of queries, which can be divided into N independent and non-intersecting subflows from each tenant. Since the tenants are small and medium enterprises, they do not generate labor-intensive calculation queries, though the total number of queries is large.

The general idea of the offered method for arranging a cluster is in implementing an additional layer of abstraction,with the program interface, which would be maximally close to the interface of the traditional DBMS. The basic advantages of such a solution are the following:

—Simplification of application development related to the possibility of using the existing knowledge by developers without taking the isolation of the tenant data from each other into account.

—Simplification of the system administration.

The given approach can be used since the DB of the application with the architecture is a collection of a large number of small logically isolated DBs for each tenant. The main functions of a multi-tenant database cluster will be as follows [13]:

—Provision of the tenant data isolation.

—Control of tenant data placing on the DB servers and the dynamic redistribution of data depending on the loading of separate servers and the behavior of the client activity in time.

—Provision of fault tolerance in the case of failure of one or several servers.

—Routing of the application servers' queries to the appropriate DB server.

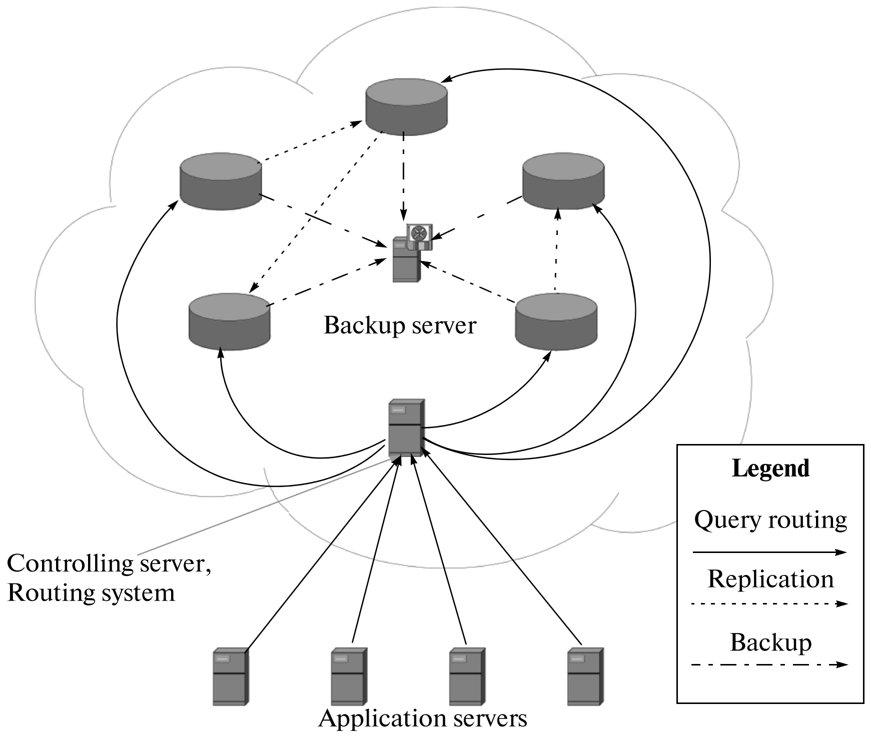

Fig. 1. Multi-tenant database cluster architecture.

—Assessment of the efficiency of using the resources and diagnostics of the system state.

The layout view of the cluster architecture is shown in Fig. 1. Let us consider some of the mentioned functions in more detail.

2.1. Isolation of Tenant Data

In the application of the considered architecture, the data of different tenants should be isolated from each other. Two mentioned methods for providing such isolation currently applied in RDBMSs are as follows: the use of shared tables with the tenant record identification by a special additional column and the isolation of the tenant data at the schema level. In terms of the solution under consideration, only the method of using isolation at the level of separate schemas can be applied, since the alternative approach with shared tables is hard to generalize and hinders the adaptation of the DB schema for the needs of a certain tenant. Though the method with a separate schema for each tenant is taken to be less effective in terms of the combined use of the server resources, it is easily generalized for the needs of the designed system. The scenario of the cluster operation is the following:

(1) Upon registering a new tenant on one of the DB servers as a part of the cluster, there is an empty schema for placing its data.

(2) Upon setting the connection with the cluster, the developer points to the tenant identifier with the data of which he wants to work with; the cluster redirects the queries on the corresponding DB servers and addresses them to the required schema on the basis of this information.

(3) The developer initiates the generated schema by creating the tables, fields, indices, and other DB objects; then, it appears ready for work.

Excluding one link that is the router of the cluster queries, the described technology is not new [14].

2.2. Control of the DB Server Data Placing

One of the basic problems of the designed system is the effective control of the tenant data placing. To solve the given problem, the cluster control system must have the statistics of the intensity and behavior of the flows of the separate tenant queries, as well as a algorithm allowing one to transfer the data from one node of the cluster to another in a maximally rapid manner. The applied algorithms should be able to distribute the tenants over the nodes such that a maximally uniform load is provided.

2.3. Fault Tolerance

One of the most important needs of cloud applications is the constant availability of a solution, which is impossible without providing stability against the faults of the separate cluster nodes. The main method to do this in the data storage subsystem is replication. In terms of the considered concept of the cluster arrangement, the most reasonable version seems to be partial asynchronous master—slave replication. The replication needs to be adjusted so that the whole tenant schema is replicated to other nodes. Upon placing the cues, the distribution of the tenant over the cluster servers and the overloading of each server should be taken into account. Perfectly, the replicas must be placed so that the fault of any server as a part of a cluster does not result in neither the denial of the service for any tenant nor the overloading of the servers to which the load of the left one is distributed. Apart from the provision of the fault tolerance, the replication will allow one to increase the performance of the cluster by the transition of part of the transactions that do not modify the tenant data to the replicas.

2.4. Query Routing

The proposed concept of a multi-tenant cluster assumes that the developed system will become the main point of interaction between the application and the DB servers. At arrival of a query from the tenant that requires sampling of the DB data, the application connects the entry point of the cluster (it is possible that the connection will be a logic operation with the data of the tenant and its operation mode (reading/reading and modification)). Based on these data, the query routing subsystem should define which DB server with a copy of the tenant data will serve the given query. A query modifying the data can be served only by the master-server of the tenant, while a query only requiring reading can be optionally redirected to one of the replicas. Note that the cluster component responsible for the query routing will be one of the most loaded system components; therefore, it cannot use expensive algorithms of load balancing. Thus, the efficiency of the load distribution over the cluster will be defined by the component responsible for placing the tenant data and the replication adjustment almost completely.

3. DEVELOPMENT OF THE CLUSTER IMITATION MODEL

The first stage of investigation of the proposed concept is the development of the imitation model of multi-tenant database clusters [15]. To reflect the specific features of the cloud applications in the model, the small research of one of such applications was implemented.

3.1. Studying the Existing Application

An online system meant for tax accounts reports to the state authority, and the exchange of electronic documents between the enterprises is taken as an example of a cloud application with multi-tenant architecture. At the present moment, the tenant base of the application involves about 43 000 tenants and continues to grow. The application is based on the traditional three-level architecture:

(1) thin client;

(2) application server;

(3) DB server.

The open-source DBMS Postgres SQL is used as the DB server. The main tasks concerning the support and service of the DB cluster are implemented by using a series of specialized applications and servers. There are currently 60 database servers of in the main cluster of the company, and its size continues to increase with growing of the tenant base. To isolate the tenant data, the dedicated schemas are used. The query routing is carried out by means of the system of special services and metadata associated with the tenant session.

To analyze the application operation, a specialized server accumulates the statistics of the main characteristics: the number of tenants, their activity, the data size, and the number of errors. The given service is activated once per 24 hours and analyzes the application log. Due to some features of the given service implementation, the collected statistics are not very exact (the collection of the complete statistics is labor-intensive and unrewarding). Since the service can analyze only the whole tenant RPC query, which generally implies the execution of several SQL queries or vice versa, we cannot rely on the accuracy of its

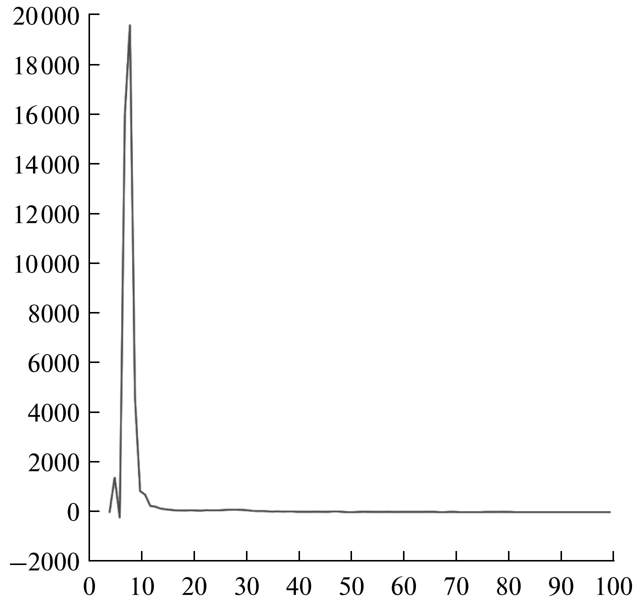

Fig. 2. Tenant schema size distribution.

data, such as the numerical query characteristics relative to the average time of execution. However, the given statistics are appropriate for revealing the basic patterns.

In terms of the development of an imitation model of a multi-tenant cluster, the problem of modeling the tenant base of application and the generated query flow is great interest. At the first stage of the investigation, the basic characteristic of the tenant is the size of its pattern on the DB server. Based on these data of the described service of the statistics collection, the following problems are studied:

—the distribution of the size of the tenant data;

—the interaction between the number of queries and the data size;

—the interaction between the average time of the query processing and the data size.

To answer the first question, the sampling of the type (identifier of the tenant, the data size) is transformed to the sampling (data size, number of tenants with the schema of the given size). The result is shown in Fig. 2. The analysis of the obtained data showed that the size of the average pattern of the tenant is 8 megabytes, while the distribution curve corresponds to the curve of a lognormal distribution.

To study the interaction between the size of the tenant schema and the number of generated queries, the sampling of about 40000 records of type (the size of the tenant pattern, the number of queries during the time interval) was constructed. Based on it, the correlation coefficient between the mentioned values was calculated being equal to 0.7, which proves the strong enough interaction between the considered values.

The last question under consideration was the one concerning the interaction between the size of the tenant pattern and the average duration of the generated queries' execution. To do this, the sampling (the size of the tenant data, the average duration of the query) based on the operation data of application for 24 hours was constructed. It contained about 7500 records. The correlation coefficient of 0.03 between the investigated values was calculated in terms of the sampling. This fact proves the absence of interaction.

Because the investigation of the statistical data showed that the size of the tenant pattern and the average duration of the query do not interact with each other, there was a need to clarify how the average dura-

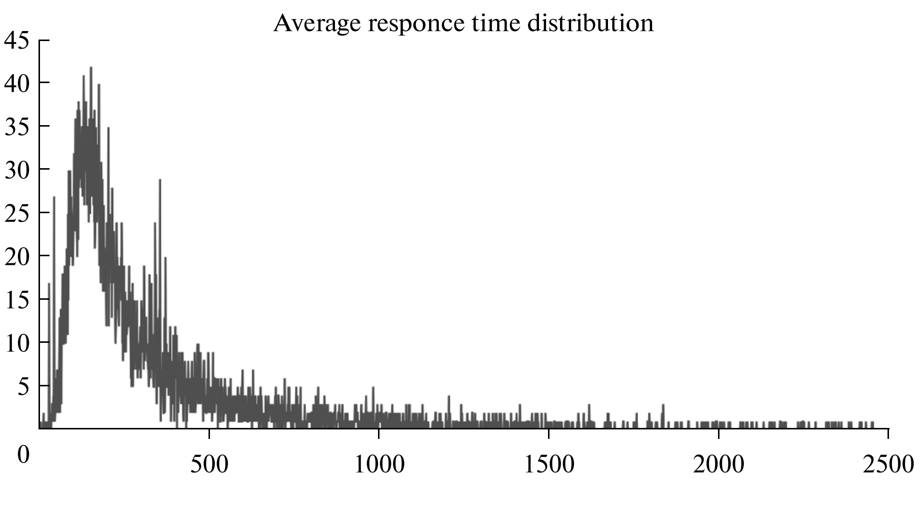

Fig. 3. Average response time duration.

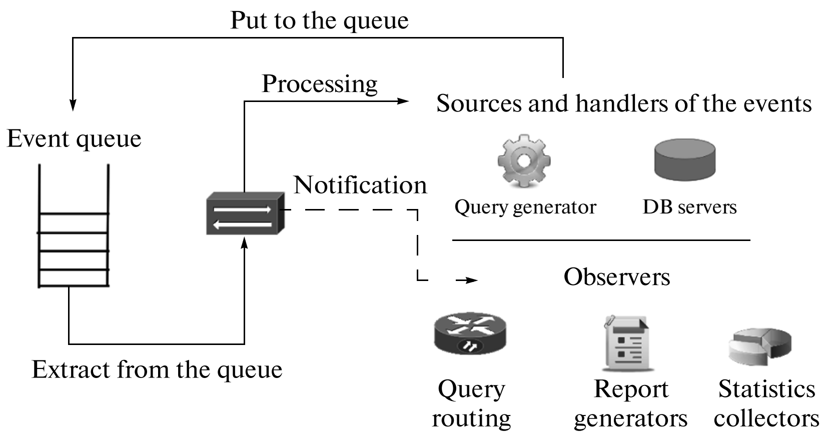

Fig. 4. Imitation model architecture.

tion of the queries is distributed. The result of the given study is shown in Fig. 3. The plot displays the average time of the query execution, which is lognormally distributed.

3.2. Architecture of the Cluster Model

The imitation model of the multi-tenant database cluster was developed on the basis of the collected data. The model is a graphical application for Linux; it was developed using the program language C++ and the Qt framework. The model acts in the discrete time from one event to another.

There are several basic entities for event generation:

—the generator of the queries from tenants and the queries for placing in the cluster of new tenants;

—the servers of the DB as a part of the cluster related to the events of the query processing termination;

—the fault generator related to the temporary failure of the cluster servers.

The model architecture is shown in Fig. 4.

The model consists of four main types of objects: event queues, the event manager, the handlers and sources of events, and observers. The sources of the events generate new events and place them in the queue for processing. Thereafter, the manager extracts the first (nearest) event from the queue and directs it to the corresponding handler. For instance, the event "input query arrival" initially goes to the query router and, then, to the DB server, which is chosen by the router for its processing. The model kernel notifies all the observers, which are responsible for the collection of the statistics of the model operation, reporting the construction, and the functioning of all the stages of processing each event. In particular, one of such observers is the algorithm of the data distribution, which allows it to collect the statistics and analyze the cluster workability to optimize its operation in the future. In some cases, the termination of the processing of one event leads to the appearance of a new one; e.g., the termination of the SQL query modifying the tenant data causes the queries for replication of the given changes on the server replicas.

The query generator models a Poisson flow of events. The intensity of the flow can be specified absolutely (the number of queries per unit of model time, e.g., 100) and relative to the tenant number (e.g., 0.05 query per tenant per unit of model time). Upon operation in the second mode, the flow intensity is equal to the product between the number of tenants and the pointed value of the intensity; i.e., it changes with pacing new tenants dynamically. A special weight coefficient influencing the query execution time of the server is attached to each generated query. According to the obtained statistical data, the mentioned coefficients are distributed by a lognormal law, whose parameters can be indicated at the generator adjustment.

The developed model can generate a large number of graphical reports about its operation (the average time of response for the last 100 queries, the sizes of queues on the separate servers and the cluster as a whole, the distribution of the query flow over the servers, and so on).

4. CONCLUSIONS

The first tests with using the developed model were devoted to the analysis of the efficiency of simple strategies of cluster control. During modeling, it was revealed that the use of the simple control strategies even if the intensity of the input flow is considerably lower than the theoretical one leads to sufficiently long periods of high activity resulting in a dramatic drop of the performance of the served cloud applications in terms of the tenants with data on the mentioned servers in certain time intervals. This means that the mentioned strategies cannot be used in practice. The model is planned to be modified. The key factors influencing the operation efficiency of the proposed control system by clusters will be revealed.

REFERENCES

1. Candan, K.S., Li, W., Phan, T., and Zhou, M., *Frontiers in Information and Software as Services*, International Council for Open and Distance Education, 2009.
2. Chong, F., Carraro, G., and Wolter, R., *Multi-Tenant Data Architecture*, Microsoft Corp. Website, 2006.
3. Schiller, O., Schiller, B., Brodt, A., and Mitschang, B., Native support of multi-tenancy in RDBMS for software as a service, *Proc. 14th Int. Conf. on Extending Database Technology EDBT'11*, 2011.
4. Jacobs, D. and Aulbach, S., Ruminations on multi-tenant databases, *Proc. BTW Conf.*, 2007.
5. Zawodny, J., *Redis: Lightweight Key/Value Store that Goes the Extra Mile*, Linux Magazine, Quarter Power Media, 2009.
6. *Benchmarking Top NoSQL Databases*, DATASTAX Corporation, 2013.
7. Brewer, E.A., Towards robust distributed systems, *Proc. Symp. on Principles of Distributed Computing—PODC*, 2000.
8. Elmore, A.J., Das, S., Agrawal, D., and El Abbadi, A., Zephyr: Live migration in shared nothing databases for elastic cloud platforms, in *Association for Computing Machinery*, 2011.
9. Schaffner, J., Januschowski, T., Kercher, M., Kraska, T., Plattner, H., Franklin, M., and Jacobs, D., RTP: Robust tenant placement for elastic in-memory database clusters, *Proc. SIGMOD Conf., Association for Computing Machinery*, 2013.
10. Yang, F., Shanmugasundaram, J., and Yerneni, R., *A Scalable Data Platform for a Large Number of Small Applications*, Yahoo! Research Tech. Report, 2008.
11. Lang, W., Shankar, S., Patel, J.M., and Kalhan, A., *Towards Multi-tenant Performance SLOs*, Int. Council for Open and Distance Education, 2012.
12. Chong, F. and Carraro, G., *Architecture Strategies for Catching the Long Tail*, Microsoft Corp. Website, 2006.
13. Boytsov, E.A. and Sokolov, V.A., The problem of creating multi-tenant database clusters, *Proc. SYRCoSE 2012*, 2012.
14. Riggs, S. and Krosing, H., *PostgreSQL 9 Administration Cookbook*, Birmingham-Mumbai: Packt Publishing, 2010.
15. Boytsov, E.A. and Sokolov, V.A., The development of an imitation model of a multi-tenant database cluster, *Proc. Int. Symp. on Business Modeling and Software Design, (BMSD-2013)*, 2013.

Translated by A. Evseeva