The Formal Statement of the Load-Balancing Problem for a Multi-Tenant Database Cluster with a Constant Flow of Queries

Evgeny A. Boytsov
Valery A. Sokolov
Computer Science Department
Yaroslavl State University
Yaroslavl, Russia
{boytsovea, valery-sokolov}@yandex.ru

***Abstract* — The concept of a multi-tenant database cluster offers new approaches in implementing a data storage for cloud applications. One of the most important questions to solve is finding a load-balancing algorithm to be used by the cluster, which is able to effectively use all available resources. This paper discusses theoretical foundations for such an algorithm in the simplest case when the flow of incoming queries is constant, that is, every tenant has a predefined intensity of the query flow and there are no changes in the state of the tenant's data.**

Keywords — database; cluster; multi-tenancy; load-balancing; quadratic assignment problem; imitation modeling;

I. INTRODUCTION

When a company designs a high load cloud application, its developers sooner or later face the problem of organizing the storage of data in the cloud with the requirement of high performance, fault-tolerance and reliable tenants' data isolation from each other. At the moment these tasks are usually solved at the level of application servers by designing an additional layer of an application logic. Such a technique is discussed in many specialized papers for application developers and other IT-specialists [1, 2, 3]. There are also some projects of providing native multi-tenancy support at the level of a single database server [4]. This paper is devoted to an alternative concept of a multi-tenant database cluster which proposes the solution of the above problems at the level of a data storage subsystem and discusses theoretical foundations of this concept in a particular case.

II. THE ARCHITECTURE OF THE MULTI-TENANT DATABASE CLUSTER

A multi-tenant database cluster [5,6] is an additional layer of abstraction over ordinary relational database servers with a single entry point which is used to provide the isolation of cloud application customer data, load-balancing, routing of queries among servers and fault-tolerance. The main idea is to provide an application interface which has most in common with the interfaces of traditional RDBMS (relational database management system). At the moment the typical scenario of interaction with the cluster from the developer's point of view is seen as the following:

```
Connect( TenantId, ReadWrite / ReadOnly );
SQL-commands
Disconnect();
```

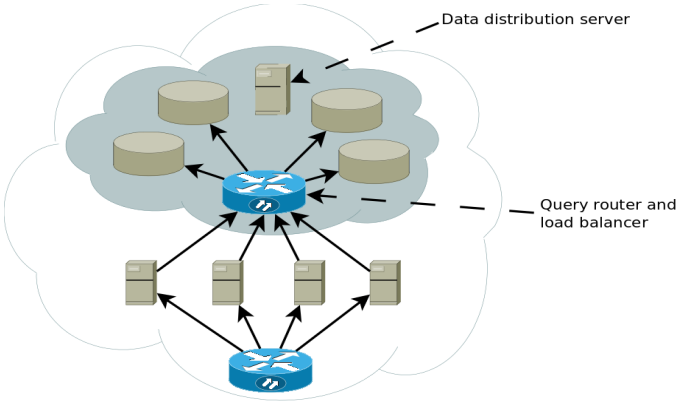

Fig. 1. Multi-tenant database cluster architecture

A multi-tenant cluster consists of a set of ordinary database servers and specific control and query routing servers.

The query routing server is a new element in a chain of interaction between application servers and database ones. This is the component application developers will deal with. In fact, this component of the system is just a kind of a proxy server which hides the details of the cluster structure, and whose main purpose is to find an executor for a query and route the query to him as fast as possible. It makes a decision based on the map of the cluster.

It is important to note that a query routing server has a small choice of executors for each query. If a query implies data modification, there is no alternative than to route it to the master database of a tenant, because only there data modification is permitted. If the query is read-only, it also can be routed to a slave server, but in the general case there would be just one or two slaves for a given master, so even in this case the choice is very limited.

The data distribution and load balancing server is the most important and complicated component of the system. Its main functions are:

- initial distribution of tenants data among servers of a cluster during the system deployment or addition of new servers or tenants;

- management of tenant data distribution, based on the collected statistics, including the creation of additional data copies and moving data to another server;
- diagnosis of the system for the need of adding new computing nodes and storage devices;
- managing the replication.

This component of the system has the highest value since the performance of an application depends on the success of its work.

III. Main characteristics of the query flow

When modeling and analyzing the multi-tenant database cluster the most important things to study are characteristics and properties of the incoming query flow. The quality of implementation of this model component significantly affects the adequacy and applicability of results obtained during modeling.

The flow of incoming queries of the multi-tenant database cluster can be divided into N non-intersecting and independent sub-flows for each tenant $\lambda_i, i \in 1, N$:

$$\Lambda = \sum_{1}^{N} \lambda_i$$

The study of statistics on existing multi-tenant cloud applications shows that there is a significant dependency between the size of data that the client stores in the cloud and the intensity of a client's query flow (the more data has the client, the larger organization it represents and, therefore, the more often these data are accessed by individual users of that client). The analysis of the statistics also shows that the above tendency is not comprehensive and there are clients within the cluster which have the intensity of the query flow that does not match the size of the stored data (it can be both more or less intensive than it is expected). The client's query flow can be divided into two sub-flows: read-only queries and data-modifying ones.

$$\lambda_i = \lambda_{i_{read}} + \lambda_{i_{write}}$$

Such a division only has sense when the data replication is used within the cluster or when the solution is tuned to specifics of the particular database server or operating system. The higher-level analysis can omit this division.

Another obvious characteristics of the query flow is an average duration μ of a query at the server. The duration of different operations is not equal and this consideration should be taken into account during the modeling. This value has a significant impact on the quality of load-balancing, since it affects the formation of the total cluster load. As we know from the queuing theory, if $\Lambda \mu > N_{queries}$, where $N_{queries}$ is the maximum amount of queries that can run in parallel in the cluster, then the cluster will fail to serve the incoming flow of requests. It is also known that intensities of incoming query flows are changed during the lifetime of the application, that is, $\lambda_i = \lambda(t), i \in 1, N$. Some clients begin to use the application more intensively, the activity of others decreases, new clients appear and existing clients disappear. Besides, some applications may have season peaks of load.

IV. The load-balancing problem with a constant flow of queries

A. General problem

The present paper is devoted to the study of load-balancing the cluster in the case when flows of incoming queries have a constant intensity, i.e. $\lambda_i = const, i \in 1, N$. The solution of this problem can be considered as a solution of the general problem «at the point». Clusters without data replication will be studied (that is, without providing fault-tolerance). For simplicity we assume that $\mu = 1$ (or, equivalently, the bandwidth of each server in the cluster is divided by μ).

Let *C* be a multi-tenant database cluster that consists of *M* database servers $(S_1, \ldots, S_M)$, for each of which we know the following values:

1. $\bar{\lambda}_i, i \in 1, .., M$ - the bandwidth of the database server;
2. $\bar{v}_i, i \in 1, .., M$ - the capacity of the database server.

There are also *N* clients, comprising the set *T*, for each of which we also know two values:

1. $\lambda_j, j \in 1, .., N$ - the intensity of *j*-th client query flow;
2. $v_j, j \in 1, .., N$ - the data size of *j*-th client.

We will call the $M \times N$ matrix *X* a distribution matrix (of clients in the cluster), if *X* satisfies the following constraints and conditions:

1. $x_{i,j} = 1$ when data of the *j*-th client are placed at the *i*-th server and $x_{i,j} = 0$ otherwise;
2. $\forall j \in 1, .., N \; \exists ! i \in 1, .., M : x_{i,j} = 1$ - the data of each client are placed at the single server;
3. $\forall i \in 1, .., M \; \sum_{j=1}^{N} x_{i,j} v_j \leqslant \bar{v}_i$ - the total data size at each server is less than or equal to the server capacity;
4. $\forall i \in 1, .., M \; \sum_{j=1}^{N} x_{i,j} \lambda_j \leqslant \bar{\lambda}_i$ - the total query flow intensity at each server is less than or equal to the server bandwidth.

We will call the matrix $\tilde{X}$ the optimal matrix of distribution of clients set *T* in the cluster *C* if for some function *f(C, T, X)* the following condition is met:

$$f(C, T, \tilde{X}) = min\{ f(C, T, X): \\ X - \text{distribution matrix} \}$$

The function *f* in this definition is the measure of load-balancing efficiency among the servers of the cluster. The problem of effective cluster load-balancing in this formulation reduces to finding an optimal distribution matrix $\tilde{X}$ for a given cluster *C*, a set of clients T and the measure of efficiency *f*.

B. The measure of efficiency

What is the best way to measure the efficiency of load-balancing among servers? Uniformity of the load is a good criteria here, therefore the target function which will measure this characteristics should be searched. At the first approximation, it may seem that the sum of squares of differences between the load of the i-th server and the average load of servers in the cluster can be used as the above measure. This can be written as:

$$\sum_{i=1}^{M}\left(\frac{\sum_{j=1}^{N} x_{i,j}\lambda_j}{\bar{\lambda}_j} - Z\right)^2$$, where Z — is the average load of the cluster servers, that is

$$Z = \frac{\sum_{i=1}^{M}\frac{\sum_{j=1}^{N} x_{i,j}\lambda_j}{\bar{\lambda}_i}}{M}$$

However, on closer examination this measure is consistent only if the cluster C consists of servers with a uniform performance, otherwise it leads to an intuitively wrong result. This consideration can be illustrated by the following example. Let the cluster C consist of twelve servers and two of them are 45 times more powerful than other ten (that is $\bar{\lambda}_{1,2} = 45\bar{\lambda}_k, k \in 3,..,12$). In this case these two servers constitute 90 percent of total cluster computational power and therefore they play a crucial role in solving the problem of effective load-balancing. The operation mode of other ten servers is not important in such a configuration (for example, they can not serve any queries at all). But the above formula assumes that all terms are equivalent and therefore these ten servers will bring a decisive contribution to the measure. This example shows the need to somehow normalize the terms in accordance with the powers of the cluster components. So the desired situation can be formulated in the following way: the share of a total query flow at each server should be as close as possible to the share of this server in the total computational power of the entire cluster. Using this formulation, the function f can be written as follows:

$$f(C,T,X) = \sum_{i=1}^{M}\left(\frac{\sum_{j=1}^{N} x_{i,j}\lambda_j}{\sum_{j=1}^{N}\lambda_j} - \frac{\bar{\lambda}_i}{\sum_{i=1}^{M}\bar{\lambda}_i}\right)^2 \quad (1)$$

C. Additional considerations

Since a set of distribution matrices X is discrete and finite, then, if it is not empty (that is there are some feasible cluster configurations), there is a non-empty subset X_{min} , whose elements are the points of minimum of the target measure function *f,* that is, the problem of optimal cluster load-balancing always has a solution.

V. THE LOAD-BALANCING PROBLEM AS THE QUADRATIC ASSIGNMENT PROBLEM

The above problem is a special case of the generalized quadratic assignment problem (GQAP) which, in turn, is a generalization of the quadratic assignment problem (QAP), initially stated in 1957 by Koopmans and Beckmann[7] to model the problem of allocating a set of n facilities to a set of n locations while minimizing the quadratic objective function arising from the distance between the locations in combination with the flow between the facilities. The GQAP is a generalized problem of the QAP in which there is no restriction that one location can accommodate only a single equipment. Lee and Ma[8] proposed the first formulation of the GQAP. Their study involves a facility location problem in manufacturing where facilities must be located among fixed locations, with a space constraint at each possible location. The aim is to minimize the total installation and interaction transportation cost. The formulation of the GQAP is:

$$min \sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{k=1}^{M}\sum_{n=1}^{N} f_{ik} d_{jn} x_{ij} x_{kn} + \sum_{i=1}^{M}\sum_{j=1}^{N} b_{ij} x_{ij}$$

subject to:

$$\sum_{i=1}^{M} s_{ij} \le S_j, j \in 1, N \quad ,$$

$$\sum_{i=1}^{N} x_{ij} = 1, i \in 1, M \quad ,$$

$$x_{ij} = \{0,1\}, i \in 1, M, j \in 1, N \quad ,$$

where:

M is the number of facilities,

N is the number of locations,

f_{ik} is the commodity flow from a facility i to a facility k,

d_{jn} is the distance from a location j to a location n,

b_{ij} is the cost of installing a facility i at a location j,

s_{ij} is the space requirement if a facility i is installed at a location j,

S_j is the space available at a location j,

x_{ij} is a binary variable which is equal to 1 if a facility i is installed at a location j.

The objective function sums the costs of installation and quadratic interactivity. The knapsack constraints impose space limitations at each location, and the multiple choice constraints ensure that each facility is to be installed at exactly one location.

The QAP is well known to be NP-hard [9] and, in practice, problems of moderate sizes, such as n=16, are still being considered very hard. For recent surveys on QAP, see the articles by Burkard [10], and Rendl, Pardalos, Wolkowicz [11]. An annotated bibliography is given by Burkard and Cela [12]. The QAP is a classic problem that defies all approaches for its solution and where the problems of dimension n=16 can be considered as of large scale. Since GQAP is a generalization of QAP, it is also NP-hard and even more difficult to solve.

The above load-balancing problem for a multi-tenant database cluster deals with hundreds of database servers and hundreds of thousands of clients. Due to NP-hardness of the

GQAP, it is obvious that such a problem can not be solved exactly or approximately with a high degree of exactness by an existing algorithm. So we can conclude that to solve the load-balancing problem we need to suggest some heuristics that can provide acceptable performance and measure its efficiency and positive effect in comparison with other load-balancing strategies.

VI. Load-balancing algorithm heuristics and its experimental verification

To test the above-mentioned target function used to evaluate the efficiency of multi-tenant database cluster load-balancing strategy, an experiment was conducted on the simulation model of the cluster. The structure of the cluster with *N* database servers of different bandwidth (*N* is a parameter of the experiment) was created by using the modeling environment. At the initial moment the cluster had no clients. The model of the query flow was configured so that it provided progressive registration of new clients at the cluster and due to it the corresponding increase of query flow intensity. Subsystems of the model which provide data size refreshing and recalculation of tenants activity coefficients were disabled. Since the above subsystems are responsible for the dynamism of the model, the resulting configuration corresponded to a cluster with constant intensities of incoming query flows. Since the computational power of the cluster is limited and the total intensity of incoming query flow constantly increases, it is obvious that the cluster will stop serving queries at some point in time. It is also obvious that if one load-balancing strategy allows to place more clients than another within similar external conditions, then this load-balancing strategy is more effective and should be preferred in real systems. The experiment was meant to compare the simple algorithm, that is based on the analysis of incoming query flows intensities and the minimization of the above target function, with other simple load-balancing strategies which do not take intensities into account at all. It should be mentioned, that the ratio between read-only and data-modifying queries is not important in this experiment, since data replication is not used.

Three load-balancing algorithms are used in the experiment. The first algorithm tries to balance the load of the cluster by balancing the size of data that are stored at each server according to the server bandwidth ratio. When deciding to host a new client on the server, this algorithm calculates the ratio of the total data size of clients that are hosted on the server to the bandwidth of the server for all servers in the cluster (the amount of stored data per the processor core), and selects a server with the minimal ratio (if there are several such servers, it randomly selects one of them). The algorithm takes into account only the servers that have enough free space to host a new client. In a pseudo-code this algorithm can be written as the following:

```
min_servers = {};
min_ratio = max_double();
for each s in S
  if datasize( s ) + sz > capacity( s )
    continue;
  ratio = num_clients( s ) / bandwidth( s );
  if ratio < min_ratio
    min_ratio = ratio;
    min_servers.clear();
  if ratio == min_ratio
    min_servers.add( s );
return random( min_servers );
```

Here *S* denotes a set of database servers within the cluster , *min_servers* is a set of servers with minimum amount of clients taking into account server bandwidth, *sz* is a data size of a new client . This algorithm will be referred to as “Algorithm 1”.

The second algorithm tries to balance the load of the cluster by balancing the amount of clients at each server according to the server bandwidth ratio. When deciding to host a new client on the server, this algorithm calculates the ratio of the number of clients that are hosted on the server to the bandwidth of the server for all servers in the cluster (the number of clients per the processor core), and selects the server with a minimal ratio (if there are several such servers, it randomly selects one of them). As the previous algorithm, this algorithm also takes into account only the servers that have enough free space to host a new client. In a pseudo-code this algorithm can be written as the following:

```
min_servers = {};
min_ratio = max_double();
for each s in S
  if datasize( s ) + sz > capacity( s )
    continue;
  ratio = datasize( s ) / bandwidth( s );
  if ratio < min_ratio
    min_ratio = ratio;
    min_servers.clear();
  if ratio == min_ratio
    min_servers.add( s );
return random( min_servers );
```

The meaning of variables here is the same as in the previous example. This algorithm will be referred to as “Algorithm 2”.

The third algorithm is based on the minimization of the target function (1). For the sake of simplicity this algorithm was connected to the query generator information subsystem of the model to get exact values of incoming query flow intensities for each client. In reality such an approach cannot be implemented and values of query flow intensities should be obtained by some statistical procedures, but for experimental purposes and testing theoretical model this approach is applicable. The main principle of the algorithm is simple: it alternately tries to host a new client at each server and computes the resulting value of the target function (1). Finally, the client is hosted at the server which gave the minimal value of all the above. In a pseudo-code this algorithm can be written as the following:

```
min_server = null;
min_ratio = max_double();
for each s in S
  if datasize( s ) + sz > capacity( s )
    continue;
  ratio = F( S | new client hosted at s );
  if ratio < min_ratio
    min_ratio = ratio;
    min_server = s
return min_server;
```

In this example *F* denotes the target function (1). This algorithm will be referred to as “Algorithm 3”.

All three algorithms were tested in the same environment, that is, with the same mean of query cost and tenants activity coefficient distribution. The results of experiments are given in the table 1. The first three columns show the parameters of the model and the algorithm used in the particular experiment. The fourth column shows the average amount of clients hosted at the cluster when the model met the experiment stop condition (one of the servers had the queue with more than 200 pending requests). Algorithm number 3 has shown significantly better results than others for all three models.

TABLE I. RESULTS OF EXPERIMENTS

Number of servers	Algorithm	Number of experiments	Average number of hosted clients
5	1	30	701,95
5	2	30	1197,63
5	3	30	1353,45
9	1	30	1090,6
9	2	30	1851,7
9	3	30	2155,45
15	1	30	1766,5
15	2	30	3235,35
15	3	30	3835,2

VII. CONCLUSION

The experiment has shown that the load-balancing strategy based on the analysis of incoming query flow intensities is more effective than others. This fact leads to the conclusion that the above-mentioned theoretical concepts are correct and can be applied to construct more complicated load-balancing strategies which take into account more factors and can be used in a more complicated environment. Some interesting questions to study are:

- How to determine the incoming query flow intensity of a client in a real environment;
- What algorithms can be used to find a better solution for the clients assignment problem;
- Are all solutions of the client assignment problem equally valuable when the intensities of incoming query flows are not constant;
- How to deal with the data replication and how the intensity of a client query flow should be divided among servers which have copies of the client data;
- What strategy should be used to relocate the clients data when the load balancing subsystem decides to do so.

All these questions are crucial in implementing an efficient load-balancing strategy for the cluster.

[1] F. Chong, G. Carraro, “Architecture Strategies for Catching the Long Tail“, Microsoft Corp. Website, 2006.

[2] F. Chong, G. Carraro, R Wolter, “Multi-Tenant Data Architecture“, Microsoft Corp. Website, 2006.

[3] K.S. Candan, W. Li, T. Phan, M. Zhou, "Frontiers in Information and Software as Services", proceedings of ICDE, 2009, pages 1761-1768.

[4] Oliver Schiller, Benjamin Schiller, Andreas Brodt, Bernhard Mitschang, “Native Support of Multi-tenancy in RDBMS for Software as a Service“, proceedings of the 14-th International Conference on Extending Database Technology, 2011.

[5] E.A. Boytsov, V.A. Sokolov, “The Problem of Creating Multi-Tenant Database Clusters”, proceedings of SYRCoSE 2012, Perm, 2012 , pages 172-177.

[6] E.A. Boytsov, V.A. Sokolov, “Multi-tenant Database Clusters for SaaS”, proceedings of BMSD 2012, Geneva, 2012 , page 144.

[7] Koopmans, T.C. and M.J. Beckmann, “Assignment problems and the location of economic activities”, Econometrica 25, 1957, pages 53-76.

[8] Lee C.-G. and Z. Ma, “The generalized quadratic assignment problem”, Research Report, Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Canada, 2004.

[9] S. Sahni and T. Gonzales, “P-complete approximation problems”, Journal of ACM, 1976.

[10] R.E. Burkard, “Locations with spatial interactions: the quadratic assignment problem”, Discrete Location Theory. John Wiley, 1991.

[11] P. Pardalos, F. Rendl, and H. Wolkowitz. “The quadratic assignment problem: A survey and recent developments”, proceedings of the DIMACS Workshop on Quadratic Assignment Problems, volume 16 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science, pages 1-41, 1994.

[12] R.E. Burkard and E. Cela, “Quadratic and three-dimensional assignment problems”, Technical Report SFB Report 63, Institute of Mathematics, University of Technology Graz, 1996.