Comparison of Data Management Strategies for Multi-Tenant Database Cluster

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

Keywords: Databases, SaaS, Multi-tenancy, Data Management Strategies.

Abstract: This paper discusses the problem of tenant data distribution in a multi-tenant database cluster - the concept of reliable and easy to use data storage for high load cloud applications with thousands of customers, based on ordinary relational database servers. The formal statements of the problem for cases with and without data replication are given and a metric for evaluating the quality of data distribution is proposed. The proposed metric is compared with ad-hoc data management strategies using an experiment at the imitation model of the multi-tenant database cluster and the result of the experiment is provided and summarized.

1 INTRODUCTION

One of recent main trends in the software development industry is the propagation of cloud technologies and corresponding change of the main architectural paradigm in an enterprise segment of the market. This tendency leads to the increase of the software complexity, since a typical cloud application consists of tens and even hundreds distributed web-services interacting with each other. One of the most significant aspects of software design is a data-storage subsystem. This subsystem should provide high performance, fault-tolerance and reliable tenants data isolation from each other. Modern software development techniques tend to solve these tasks by designing an additional layer of application logic at the level of application servers. Such approaches are discussed in many specialized papers for application developers and other IT-specialists (Chong and G., 2006; Candan et al., 2009). 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.

One of the main challenges when implementing such a system is to choose the most efficient data management strategy which will provide the best distribution of the query flow among database servers within the cluster. In this context, the word "best" implies a number of questions that can be answered in different ways. An optimization can be done by various criteria and we need to use some consumer characteristics to evaluate the observed quality of service. The average cluster response time, the total amount of required resources within the given service level agreements (SLA's) or something else can be used as such characteristics. Often These characteristics are often difficult to evaluate and sometimes they conflict with each other. Besides, many of the above characteristics can be evaluated only when the distribution of clients has been already done. So far, an additional metric is required which has a direct correlation with the above consumer characteristics and can be used to find the optimal tenant distribution. This paper discusses one approach to choosing such a metric and compares its results with ad-hoc data management strategies.

2 BACKGROUND

The problem of providing a reliable and scalable data storage for cloud applications was discussed in several works. Usually, NO-SQL databases are used as cluster nodes. In particular, the problem of tenant migration in a multi-tenant environment was studied and the protocol to implement such a migration was proposed in (Elmore et al., 2011). Other researches were devoted to minimizing an owning cluster consisting of NO-SQL in-memory databases in IaaS environment (Schaffner et al., 2013; Yang et al., 2008). The algorithm of tenant distribution for minimizing expenses with respect to SLA's was proposed in (Lang et al., 2012).

A multi-tenant database cluster (Boytsov, 2013)

Boytsov E. and Sokolov V.
Comparison of Data Management Strategies for Multi-Tenant Database Cluster.
DOI: 10.5220/0005426302170222
In *Proceedings of the Fourth International Symposium on Business Modeling and Software Design (BMSD 2014)*, pages 217-222
ISBN: 978-989-758-032-1

discussed in this paper is a concept of a data storage subsystem for cloud applications. It 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 customers data, load-balancing, routing the queries among servers and fault-tolerance. The main idea is to provide an application interface which has most in common with the interfaces of the traditional RDBMS (relational database management system).

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 servers. 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.

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.

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 $\lambda_i, i \in 1,..,N$:

$$\Lambda = \sum_{i=1}^{N} \lambda_i \qquad (1)$$

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 intensity of client query flow. The analysis of the statistics also shows that the above tendency is not comprehensive and there are clients within the cluster having the intensity of the query flow that does not match the size of the stored data. The client query flow can be divided into two sub-flows: read-only queries and data-modifying queries.

$$\lambda_i = \lambda_{i_{read}} + \lambda_{i_{write}} \qquad (2)$$

Another obvious characteristics of the query flow is an average duration μ of a query at the server. This value has a significant impact on the quality of load-balancing, since it affects the formation of the cluster total load. As we know from the queueing theory, if $\Lambda\mu > B$, where B is a bandwidth of the cluster, the cluster will fail to serve the incoming flow of requests. It is also known that intensities of incoming query flows change during the lifetime of the application, that is $\lambda_i = \lambda(t), i \in 1,..,N$.

3 THE LOAD-BALANCING PROBLEM WITH CONSTANT FLOW OF QUERIES

In the work, we discuss the load-balancing of the cluster in a 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.

3.1 Clusters without Replication

We start our discussion with clusters without data replication (that is, such clusters do not provide 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 the multi-tenant database cluster that consists of database servers $(S_1,..,S_M)$, for each of which we know the following values:

1. $\bar{\lambda}_i, i \in 1,..,M$ - the bandwidth of the $i-th$ database server;
2. $\bar{v}_i, i \in 1,..,M$ - the capacity of the $i-th$ 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 the j-th client query flow;
2. $v_j, j \in 1,..,N$ - the data size of the j-th client.

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

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

4. $\forall i \in 1,..,M\ \sum_{j=1}^{N} d_{i,j}\lambda_j \leq \bar{\lambda}_i$ - total query flow intensity at each server is less than or equal to the server bandwidth.

We call the matrix $\tilde{D}$ the optimal matrix of distribution of clients set T at the cluster C, if for a function $f(C,T,D)$ the following condition is met:

$$f(C,T,\tilde{D}) = min\{f(C,T,D) : D - distribution\,matrix\} \tag{3}$$

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 the optimal distribution matrix for a given cluster C, a set of clients T and a measure of efficiency f.

3.2 Clusters with Replication

The usage of a master-slave replication allows to provide fault-tolerance and gives a chance to achieve a better query flow distribution. When discussing clusters with replication, we deal with multiple data instances of the same tenant. In this case, we have to take into account division of the tenants query flow into read-only and data-modifying parts. Only the server which hosts tenants master data-instance can serve data-modifying queries.

To precise this situation, we need to add several new features into our model. First of all, we need to introduce the notion of a replication matrix. We call a $M \times N$ matrix R a matrix of replication (of tenants data instances at the cluster C) for the given matrix of distribution D, if the following conditions are met:

1. $R_{i,j} = 1$, if a replica of data of the j-th tenant is stored at the i-th server, and $R_{i,j} = 0$, otherwise
2. $\forall i \in 1,..,M\, and\, j \in 1,..,N : D_{i,j} = 1 \implies R_{i,j} = 0$ - if i-th server has a master copy of the tenant data, it can not host a tenant data replica.

Obviously, clusters with replication have the same service level requirements as its counterparts without replication. The disk capacity restriction is transformed into:

$$\forall i \in 1,..,M : \sum_{j=1}^{N} d_{i,j}v_j + \sum_{j=1}^{N} r_{i,j}v_j \leq \bar{v}_i \tag{4}$$

It is much difficult to formulate the second restriction on incoming flow intensities, since we don't know exactly the policy of query flow distribution among tenant data instances. All we can say is that all data-modifying queries are served at the master server. Read-only queries can be served either by the master server, or by slave servers, and the cluster control system is free to choose any conformant strategy. It can forward all read-only queries to the master server, using replicas just to provide fault-tolerance, it can route all such queries to replicas, somehow dividing the flow among them, or it can use an intermediate approach. These considerations lead us to the need to define an additional function:

$$shr : (C,T,D,R) \rightarrow S, \tag{5}$$

where S is a $M \times N$ matrix and $S_{i,j} \in [0,1]$. This function takes the set of servers C, the set of clients T, and the distribution of tenants data instances among servers within the cluster, which is described by matrices D and R and maps it to the matrix of the read-load share S. The read-load share matrix S has the following requirements:

1. $\forall j \in 1,..,N\ \sum_{i=1}^{M} s_{i,j} = 1$ - the read-only flow is completely distributed among tenant data instances
2. $\forall i \in 1,..,M,\ j \in 1,..,N : D_{i,j} = 0 \wedge R_{i,j} = 0 \implies S_{i,j} = 0$ - if the i-th server doesn't host data instance of the j-th tenant its load-share is equal to 0.

Having the matrix S introduced, we can formulate the flow-intensity constraint as the following:

$$\sum_{j=1}^{N} (d_{i,j}\lambda_{j_{write}} + d_{i,j}\lambda_{j_{read}}s_{i,j} + r_{i,j}\lambda_{j_{read}}s_{i,j}) \leq \bar{\lambda}_i, \quad \forall i \in 1,..,M \tag{6}$$

If we introduce the shorthand $load(i,j)$ as $load(i,j) = d_{i,j}\lambda_{j_{write}} + d_{i,j}\lambda_{j_{read}}s_{i,j} + r_{i,j}\lambda_{j_{read}}s_{i,j}$ then we can rewrite (6) as

$$\sum_{j=1}^{N} load(i,j) \leq \bar{\lambda}_i, \forall i \in 1,..,M \tag{7}$$

We call the combination of a distribution matrix D and a replication matrix R sustainable to the fault of k servers, if $\forall i_1,..,i_k, i_l \in 1,..,M$ the fault of servers $i_1,..,i_k$ and redistribution of the query flow among servers left will produce tenant distribution $(\hat{D},\hat{R})$, where $\hat{D}$ still conforms to the definition of the distribution matrix, and the combination $(C,T,\hat{D},\hat{R})$ still conforms to (7). In this paper, we omit the discussion on the term "redistribution of the query flow", since in general case it implies the definition of another function, which is responsible for election of a new master data instance, when the existing master data instance is placed at a failed server.

So we can finally formulate the load-balancing problem for clusters with replication and the requirement of k-faults sustainability as finding a combination of matrices $(\tilde{D},\tilde{R})$, which, together with the given structure of the cluster C, the set of tenants T and the read-load share function *shr* satisfies the following conditions:

1. $(\tilde{D},\tilde{R})$ corresponds to k-server faults sustainable distribution of tenants data instances
2. $f(C,T,shr,\tilde{D},\tilde{R}) = min\{f(C,T,shr,D,R)\}$ for some metric f

This problem reduces to the problem of cluster load-balancing without replication when $R = \Theta$. In this case, the function *shr* can be removed from the problem, since there is no alternative for $S = D$, which gives $load(i,j) = d_{i,j}\lambda_j$ as in (3).

4 SELECTION OF THE EFFICIENCY MEASURE

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. 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. So, the function f can be written as follows:

$$f = \sum_{i=1}^{M} \left(\frac{\sum_{j=1}^{N} load(i,j)}{\sum_{j=1}^{N} \lambda_j} - \frac{\bar{\lambda}_i}{\sum_{i=1}^{M} \bar{\lambda}_i} \right)^2 \qquad (8)$$

With the measure of efficiency (8), the load-balancing problem becomes 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(Beckman and Koopmans, 1957) to model the problem of allocating a set of n facilities to a set of n locations while minimizing the quadratic objective 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(Lee and Ma, 2004) 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 objective is to minimize the total installation and interaction transportation cost.

The QAP is well known to be NP-hard(Sahni and Gonzalez, 1976) and, in practice, problems of moderate sizes are still considered very hard. For surveys on QAP, see the articles Burkard(Burkard, 1990), and Rendl, Pardalos, Wolkowicz (Rendl et al., 1994). An annotated bibliography is given by Burkard and Cela(Burkard and Cela, 1997). The QAP is a classic problem that still defies all approaches for its solution and where problems of dimension $n = 16$ can be considered large scale. Since GQAP is a generalization of QAP, it is also NP-hard and even more difficult to solve.

The discussed multi-tenant database cluster load-balancing problem deals with tens and hundreds of database servers and tens and hundreds of thousands of tenants. Due to NP-hardness of the GQAP, it is obvious that such a problem can not be solved exactly or approximately with high degree of exactness by existing algorithm. So, we can conclude that to solve the above 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.

5 MODELLING OF DATA-MANAGEMENT STRATEGIES

The above measure of efficiency of cluster load-balancing is a heuristics which can be used to search for an efficient tenant distribution. But does it correlate with consumer characteristics of the cluster and lead to the better results than ad-hoc solutions, that can be written by any programmer? To answer these questions and to test the target function (8), several experiments were conducted at the simulation model of the cluster. The structure of the cluster with M database servers of different bandwidth (M is a parameter of the experiment) was generated using the modelling environment. At the initial moment, the cluster had no clients. Each experiment within the series consisted of 30 iterations with a selected combination of simulation parameters.

5.1 The Description of the Experiment

The experiment was conducted for clusters with and without replication. The model of the query flow was configured in a way which provided progressive registration of new clients at the cluster and therefore the corresponding increase of query flow intensity. 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 of time. It is also obvious that if one load-balancing strategy allows to place more clients than another one within similar external conditions with the similar requirements to cluster fault-tolerance, this load-balancing strategy is more effective and should be preferred in real systems.

5.2 Clusters without Replication

In this series of experiments the ratio between read-only and data-modifying queries is not important, since data replication is not used. Three load-balancing algorithms were used during the experiment.

The first algorithm tries to balance the load of the cluster by balancing the amount of clients at each server according to its bandwidth ratio. When deciding on where to host a new client, 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 a cluster and selects the one with the minimal ration (if there are several such servers, it randomly selects one of them). The algorithm takes into account only those servers that have enough free space to host a new client. This algorithm will be referred to as Algorithm wr1.

The second algorithm tries to balance the load of the cluster by balancing the size of data that are stored at each server according to its bandwidth ratio. When deciding on where to host a new client, 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 a cluster and selects the one with the minimal ration (if there are several such servers, it randomly selects one of them). Like the previous algorithm, this algorithm also takes into account only those servers that have enough free space to host a new client. This algorithm will be referred to as Algorithm wr2.

The third algorithm is based on the minimization of the target function (8). 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 can not be implemented and values of query flow intensities should be obtained by some statistical procedures, but this approach is applicable for experimental purposes and testing the theoretical model. 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 (8). Finally, the client is hosted at the server which gave the minimal value. This algorithm will be referred to as Algorithm wr3.

All three algorithms were tested in the same environment, that is, with the same mean of query cost and tenants activity coefficients distribution. The experiment results are given in Table 1. The first two columns show the parameters of the model and the algorithm that were used in the particular experiment. The third column shows the average amount of clients which was hosted at the cluster when the model met the experiment stop condition (one of the servers had the queue with more than 100 pending requests). The algorithm wr3 has shown better results than others for all three models.

Table 1: The results of the first experiment series for clusters without replication.

Algorithm	N. of servers	Avg. N. of tenants
wr1	7	385
wr2	7	278
wr3	7	387
wr1	9	520
wr2	9	373
wr3	9	523
wr1	15	834
wr2	15	578
wr3	15	844

5.3 Clusters with Replication

The same experiment setup was used for the case with the replication. Since the previous series of experiments showed the same results for clusters of different sizes, in this series the size of the cluster was constant and equal to 16. The ratio of query types was the main parameter of the experiment instead of the cluster size. Three load-balancing algorithms were used during the experiment. Each algorithm was configured to create two replicas of every data instance.

The first algorithm tries to balance the load of the cluster by balancing the amount of clients at each server according to the servers bandwidth ratio. This algorithm is a generalization of the Algorithm wr1 from the first experiment series. When deciding on where to host a new client and its replicas, this algorithm calculates the ratio of the number of clients that are hosted at the server to the bandwidth of the server for all servers in a cluster, and selects the one with minimal ration (if there are several such servers, it randomly selects one of them). The same procedure is applied for replicas (two in this case). The algorithm takes into account only those servers that have enough free space to host a new client or its replica. This algorithm will be referred to as Algorithm r1.

The second algorithm divides the cluster into groups of n servers, where n=Number of Required Replicas + 1 (three in this experiment series). The server with the largest bandwidth within the group is selected to be the "master", other $n-1$ servers become "replicas". When deciding on where to host a new client, this algorithm calculates the ratio of the usage for each group, and selects the group with minimal ration (if there are several such groups, it randomly selects one of them). The algorithm takes into account only those groups that have enough free space to host a new client or its replica at every server within the group. This algorithm will be referred to as Algorithm r2.

The third algorithm is a generalization of the Algorithm wr3. For every incoming request, it finds the best placement of master data instance and its replicas in terms of minimization of the function (8). The kind of branch and bounds algorithm is used to find the best solution for a current tenant. This algorithm will be referred to as Algorithm r3.

The experiment results are given in Table 2. The first two columns show the parameters of the model (ratio between read-only and data-modifying queries) and the algorithm was used in the particular experiment. The third column shows the average amount of clients which was hosted at the cluster when the model met the experiment stop condition, which was the same as in the first experiment series. The algorithm r3 has shown better results than others for all three ratios of query types.

Table 2: The results of the first experiment series for clusters with replication.

Algorithm	RO/W	Avg. N. of tenants
r1	70/30	724
r1	50/50	723
r1	30/70	666
r2	70/30	682
r2	50/50	610
r2	30/70	448
r3	70/30	564
r3	50/50	530
r3	30/70	494

6 CONCLUSION

The experiment has shown that the load-balancing strategy based on the analysis of incoming query flows intensities is more effective than ad-hoc strategies. This fact leads to the conclusion that the above 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 more complicated environment. Especially interesting questions to study are:

- how to determine the incoming query flow intensity of the 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 clients assignment problem equally valuable when intensities of incoming query flows are not constant;
- what strategy should be used to relocate client data when the load balancing subsystem decides to do so.

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

REFERENCES

Beckman, M. and Koopmans, T. (1957). Assignment problems and the location of economic activities. *Econometrica*, 25:53–76.

Boytsov, E. (2013). Designing and development of the imitation model of a multi-tenant database cluster. *Modeling and analysis of information systems*, 20.

Burkard, R. (1990). Locations with spatial interactions: The quadratic assignment problem. *Discrete location theory*, pages 387–437.

Burkard, R. and Cela, E. (1997). Quadratic and three-dimensional assignment problems. pages 373–392.

Candan, K., Li, W., Phan, T., and Zhou, M. (2009). Frontiers in information and software as services. In *Proceedings of ICDE*, pages 1761–1768.

Chong, F. and G., C. (2006). Architecture strategies for catching the long tail.

Elmore, A., Das, S., Agrawal, D., and El Abbadi, A. (2011). Zephyr: Live migration in shared nothing databases for elastic cloud platforms. In *SIGMOD Conference*.

Lang, W., Shankar, S., Patel, J., and Kalhan, A. (2012). Towards multi-tenant performance slos. In *ICDE*.

Lee, C.-G. and Ma, Z. (2004). The generalized quadratic assignment problem. Technical report, University of Toronto, Department of Mechanical and Industrial Engineering, Toronto, Canada.

Rendl, F., Pardalos, P., and Wolkowicz, H. (1994). The quadratic assignment problem: A survey and recent developments. In *Proceedings of the DIMACS Workshop on Quadratic Assignment Problems*, volume 16, pages 1–42. American Mathematical Society.

Sahni, S. and Gonzalez, T. (1976). P-complete approximation problems. *Journal of ACM*, 23(3):555–565.

Schaffner, J., Januschowski, T., Kercher, M., Kraska, T., Plattner, H., Franklin, M., and Jacobs, D. (2013). Rtp: Robust tenant placement for elastic in-memory database clusters. In *SIGMOD Conference*.

Yang, F., Shanmugasundaram, J., and Yerneni, R. (2008). A scalable data platform for a large number of small applications. Technical report, Yahoo! Research.