An SLA and Operation Cost Aware Performance Re-Tuning Algorithm for Cloud Databases

Liangzhe Li
School of Computer Science, University of Oklahoma
Norman, OK 73019, U.S.A.
lzli@ou.edu

Le Gruenwald
School of Computer Science, University of Oklahoma
Norman, OK 73019, U.S.A.
ggruenwald@ou.edu

Abstract—**Cloud database, also called Database as a Service (DbaaS), can provide subscription-oriented, enterprise-quality services with high availability, reliability and scalability. It may be defined as a pay-per-use model for enabling on-demand access to reliable and configurable services that can be quickly provisioned and released with minimal management. Users/tenants need not set up the infrastructure or buy the software, but pay to the cloud service provider only for the services they use through a performance service level agreement (SLA) that specifies the performance requirements and the pricing associated with the leased services. Due to changes in queries and/or databases, tenants' performance SLA may be violated at some point in time. So a performance tuning technique that can re-guarantee the performance SLA when such a violation occurs, and at the same time, does not increase the service provider's operating cost, is needed. This paper proposes a performance tuning algorithm, called AutoClustC, which estimates the costs of resource provisioning and database re-partitioning and chooses the lower cost approach to tune the system in order to re-guarantee the performance SLA when a performance violation occurs. When database partitioning is chosen, the algorithm implements a novel database attribute partitioning algorithm, which takes both performance SLA and operation cost into consideration, and uses data mining techniques to automatically and dynamically re-partition the databases for a tenant to generate better partitioning solutions.**

Keywords-DbaaS; SLA; attribute partitioning; provisioning

I. INTRODUCTION

DbaaS is becoming a more and more popular model for providing data management functions in a cost-effective way in the Cloud. Compared to traditional database system, DbaaS usually serves more clients or tenants, therefore facing more unpredictable workloads. This makes the SLAs guarantee work more difficult. So current DbaaS providers such as Amazon AWS-RDS [2] and Microsoft SQL Azure [24] offer only reliability SLAs which specify the fraction of availability over a fixed time period for their services that users can expect when they contract to use a service, but do not offer performance SLAs which guarantee the minimum levels of performance such as query response time [26].

Many efforts have been made on how to provide performance SLAs guaranteeing minimum level of performance, such as [18], [26] and [16]. But those works do not address a very important problem, which is what the service provider should do when the pre-defined performance SLAs are violated under some specific circumstance. Some existing solutions including static and dynamic resource provisioning ([34], [41]), queuing and scheduling [13], and admission control [36] could be used to solve such problem. But different approaches have different disadvantages. First, by static and dynamic resource provisioning (provisioning is the process of allocating physical computing resources to virtual machine (VM)), when the system detects that the pre-defined performance SLAs have a higher chance to be violated, more resources such as CPU will be added to improve the situation. A major disadvantage of this method is that the data center operation cost will increase, especially for static provisioning. The consequence of the improper provisioning might be the negative profit to the service provider. Second, by queuing and scheduling, the incoming queries are temporarily held in a queue and then scheduled based on some prioritization criteria, such as worse performance penalty cost. A major disadvantage of this method is that this solution only works for short-term load peaks, and some tenants' performance may be heavily degraded due to their queries' postponed execution. Third, by admission control, new queries are either stalled or rejected when performance SLAs have higher chance to be violated. The major disadvantage of this method is very similar to that of the second solution in that in order to guarantee some tenants' performance SLAs, other tenants' performance might have to be sacrified because their queries were stalled or rejected. As we can see, though a number of methods have been proposed to generate a proper performance based SLA for the service provider, it is really hard to find a way to re-guarantee this performance SLA once a performance violation occurs.

Although the existing DbaaS providers do not provide any performance based SLAs, they still have to deal with the performance degrading problem caused by heavy workloads or query pattern changes. The typical solution is to do resources provisioning [32], [5]. Resources provisioning could eventually guarantee all tenants' performance but it will cause the extra operation cost to the service provider. As the operation cost of the physical computing resources is one of the major costs to the service provider, improper usage of the resources could negatively impact the service provider's profit. In this paper we propose a novel way of re-guaranteeing the performance SLA by implementing a cost-aware attribute (vertical database) partitioning algorithm, called AutoClustC, on cloud databases. By estimating the costs of the two approaches, resource provisioning and attribute partitioning, when a performance SLA violation occurs, this algorithm can choose the more profitable approach for the service provider to take in order to re-guarantee the performance SLA.

The remainder of this paper is organized as follows. Section II discusses the different database partitioning techniques for single computers, cluster computers and clouds. Section III introduces the theoretical aspects of the proposed AutoClustC algorithm. Experimental evaluations are described in Section IV. Finally, Section V presents the conclusions and future work.

II. RELATED WORK

In recent years, new resource provisioning techniques have been developed to address the cost and different user's performance SLA issues (e.g. [5], [7], [32] and [31]). Though these techniques can help cloud service providers improve the quality of services, a common weakness of those techniques is that resource provisioning may significantly reduce the service providers' profit. Though the providers have to handle the performance degrading problem, which may be caused by heavy workloads or query pattern changes, they cannot charge extra fees to the users. Computing resource is a major operation cost to service providers, resource provisioning will definitely increase the data center's operation cost. Due to the above reasons, researchers have been looking for new ways to handle the cloud re-tuning problem. Database partitioning is one of the methods that can help service providers re-guarantee the performance SLAs. Such partitioning techniques should be aware of the performance SLA violation, has the ability to estimate the costs of database repartitioning and resource provisioning to choose the lower cost approach for performance tuning, and generate better database partitioning solutions to replace the old ones if database repartitioning is the chosen approach.

As database partitions change over time due to changes in queries and/or databases, attribute re-partitioning is needed. In order to get a suitable attribute partitioning solution for better database performance, database attribute partitioning approaches have been proposed. In early days, many attribute partitioning techniques for single computers were developed, such as the algorithms published in [1], [27], [40] [28], [29], [17] and [15]. Nowadays, as the volume of data is getting bigger and bigger and the velocity of data is getting faster and faster, distributed databases based on cluster computers or clouds are widely used. The algorithms designed for single computers do not work well for this new environment, which leads to the development of attribute partitioning algorithms for distributed environments. However, many of these algorithms were designed without considering the two major factors of cloud databases, performance SLA and operation cost. For example, the Amossen algorithm [3] is an attribute partitioning algorithm used only for OLTP applications on cluster machines. Generally in an OLTP application, there are many short queries with no many-row aggregates and the queries are processed only from the same site. It means that the queries happening on such a system are usually very simple. In [3] the authors present a cost model and then use simulated annealing to find the close-to-optimal attribute partitioning with respect to the cost model. The algorithm does not address any issue related to performance SLA guarantee or operation cost, and uses a pre-defined query workload to partition. This kind of attribute partitioning approach is called static partitioning; for such an approach, people may have to re-collect queries from a log file if they want to re-partition the database. Overall, this algorithm has no ability to automatically solve the performance SLA violation problem on a cloud database.

Another algorithm called GAC (Genetic Algorithm-Based Clustering) was proposed in [9]. The rows and columns in the attribute usage matrix are permutated to get a suitable attribute partitioning solution. The permutating process is then formulated as a travelling salesman problem (TSP), in which the distance (cost) measures between a pair of rows (attributes) and columns (transactions) are computed. The same as TSP, the best attribute partitioning solution should have the least distance (cost). When the distance is minimal, the corresponding permutating result is the final attribute partitioning solution. Similar to the Amossen algorithm, this GAC algorithm does not discuss any issues related to performance SLA or operation cost. The whole algorithm is built based on an existing workload, i.e. this algorithm is a static attribute partitioning algorithm and can be used only one time. So this algorithm has no ability to detect the performance SLA violation and to choose the most profitable way for the service provider to tune the cloud database.

In [19], another static attribute partitioning algorithm, called AutoClust, was presented. This algorithm uses the query optimizer to generate attribute partitioning solutions. Multiple attribute partitioning solutions are selected from the candidate partitioning solution pool. Every future query will be routed to the computing node containing the partition that gives the best estimated query cost for the query execution. An improved version of the algorithm was presented in [20]. In this version, a query statistics analysis component is added. This component keeps monitoring the most recent queries processed by each computing node and reading the related query information (query id, query context, physical read ratio, logical read ratio, and query count) from the system views until enough physical read queries are collected. Then the query optimizer uses the information collected to evaluate the future average query estimated cost changing trend. If the trend is increasing on the majority of the computing nodes, the re-partitioning process is triggered. Though the algorithm is a dynamic algorithm, the authors do not discuss how this algorithm can guarantee performance SLA and how this algorithm can measure the operation cost. So this algorithm is not suitable for a cloud database as it cannot detect performance SLA violations and estimate the cost for different tuning methods on a cloud database.

In [14], the authors present a cost model which sums the cost of local transaction processing and remote transaction processing. Then the ant clustering algorithm [12] is used and takes this model as the key factor to decide whether an attribute is to be placed in a particular partition or not. The authors point out that the resulted partitions could be distributed to different nodes but provide no details. In addition, the authors do not consider performance SLA or operation cost issues. So this algorithm cannot use on a cloud database either.

From the existing attribute partitioning algorithms discussed above, [1], [27], [40], [28], [29], [17], [15], [3], [9],

[19] and [14], we see that none of them is designed for cloud databases. Though there exist other database performance tuning algorithms designed for cloud databases, none of them is for attribute partitioning. For instance, the work in [35] is for database replication in cloud databases; [32] is for resource provisioning in cloud databases; and [10] is for query scheduling in cloud databases. To fill this gap, in this paper we present AutoClustC, an algorithm to re-partition attributes on a cloud database. This algorithm has the ability to handle the performance SLA violation problem, estimate the cost difference between the database partitioning approach and the resource provisioning approach, and choose the lower cost approach to re-guarantee the performance SLA.

III. AUTOCLUSTC

In this section a performance SLA and cost-aware automatic attribute partitioning technique, called AutoClustC, is presented. This technique can determine whether or not a database re-partitioning process is needed based on an SLA-based profit optimization approach by estimating the costs of resource provisioning and attribute partitioning. The objective of our algorithm is to find the most profitable way and minimize the amount of consumed resources to re-guarantee the DbaaS performance quality (performance SLA) when both resource provisioning and attribute partitioning are available for the service provider. The algorithm analyzes the most recent history resource demand (we use CPU as the example resource in this paper) and predicts the resource demand, which is the resource needed for resource provisioning for the near future if a performance SLA violation is detected for a tenant on a virtual machine (VM). Then the algorithm will compare the resource demand of resource provisioning with the resource demand of attribute partitioning, which is predicted by constructing an Artificial Neural Network using the history partitioning resource demand data, to find the most profitable system tuning approach.

This section is divided into 4 subsections. In this first subsection, different multi-tenancy cloud database structures will be discussed. Since in our algorithm we consider the number of users of the cloud database as an important factor for estimating the cost of partitioning process, and this factor will be used as an input of the algorithm, we need to specify the multi-tenancy structure used in our algorithm first. In the second subsection, the characteristics of CPU utilization, which could be benefited most from dynamic resource provisioning, are briefly described. Since in order to ensure the accuracy of predicting the cost for dynamic provisioning, the CPU utilization has to match some pattern. In this subsection we will figure out what kind of CPU utilization is suitable for our prediction algorithm. In the third subsection, the model of how to estimate the cost of resource provisioning is proposed. In the last subsection, the model of how to estimate the cost of partitioning is proposed.

A. Multi-Tenancy of DbaaS

There are many different ways of deploying DbaaS on a cloud with multi-tenancy [4], [11] and [35]. The four major options of hosting multiple tenants on a cloud database are: (1) all tenants' data are located within the same database and the same tables with an extra identifier such as "TenantID" to differentiate the records from different tenants; (2) all tenants are located within a single database but in separate schemas to differentiate their tables in order to provide better schema level security; (3) each tenant is located in a separate database within the same DBMS instance in order to provide much greater security; and (4) each tenant has a separate VM with its own OS and DBMS in order to provide the greatest security and the most flexible control via VM management.

In this paper we assume the majority users of the cloud database are middle or small size companies since large size companies usually have the economic ability to afford their own data center. So we use the third option as the multi-tenancy structure. This option provides a good trade-off between wasted resources due to extra OS and software license, and the complex management and security issues associated with the first two options [18].

B. CPU Utilization Traces on Cloud Databases

In [5] the authors describe three main categories of CPU utilization behaviors by studying a large number of traces from production servers. The authors conclude that only the utilization behavior shown in Figure 1, which is characterized by strong utilization variability and good autocorrelation (a mathematical representation of the degree of similarity between a given time series and a lagged version of itself over successive time intervals [6]) associated with this periodic behavior, could be benefited greatly by using the dynamic resource provisioning. In practice, this assumption is true for many applications since the workload on the server is often very low during the late evening and early morning, but will increase to a peak during the day time, then go down to the valley status after the day time. We therefore assume this CPU utilization behavior in our research.

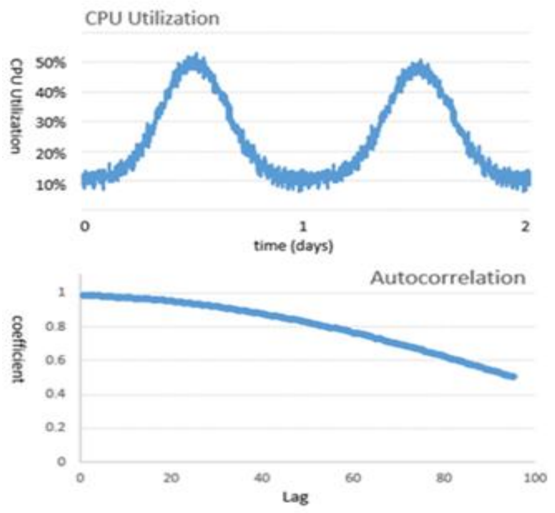

Figure 1. CPU utilization behaviors.

The CPU utilization behavior illustrated in Figure 1 has a specific period of about 1 day. The autocorrelation is always above 0.5 for 100 lags. It has been shown that this kind of time series behavior is very suitable for near future behavior prediction using the most recent historical behavior data [5]. In order to model such utilization behavior, in our algorithm, an Auto Regressive (AR) model will be used. An AR model can describe a certain time-varying process in which the output variables depend linearly on this process's own previous values and on a stochastic term [33]. In the next subsection, the algorithm of how to use an AR model to predict the near future CPU utilization demand is presented.

C. CPU Utilization Forecasting for Resource Provisioning

In order to understand the forecasting process more clearly, consider an example shown in Figure 2, which shows a snapshot of 24 hours CPU utilization demand historical data (U) with the demand probability density function (PDF) u(x). If the current time point is t_0, and the prediction time interval is t, the forecasting process is to compute the average CPU demand, which is denoted by U_{t0+t} in the next prediction time interval. If the prediction error is E_t, the forecasting result will be $U_{t0+t} + E_t$. In [5], the authors use an AR model to compute the gain, which is the ratio of the estimated future CPU demand of dynamic resource provisioning to the estimated future CPU demand of static resource provisioning. In our algorithm we also use an AR model, but the exact estimated future CPU demand is computed so that we can compare the costs of resource provisioning and attribute partitioning.

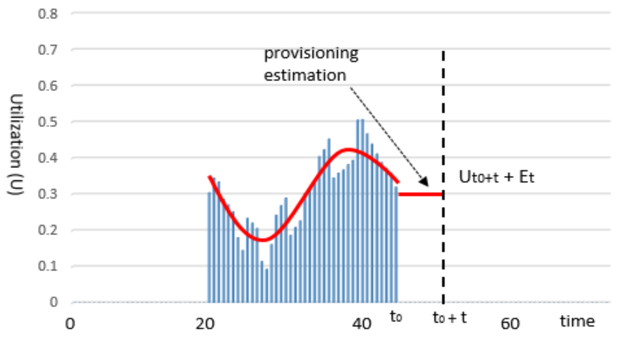

Figure 2. Dynamic provisioning estimation.

If we use $u_t(x)$ to represent the demand probability density function of the predicted time series, the exact average future demand forecast can be represented as

$$U_{t_0+t} + E_t = \int_0^\infty (x + E_t) \times u_t(x)dx \qquad (1)$$

The expression $(x + E_t)$ represents a given CPU resource allocation, which will be weighted with $u_t(x)$, the probability of a particular provisioned CPU amount x.

Equation 1 can be rewritten as

$$U_{t_0+t} + E_t = \int_0^\infty x \times u_t(x)dx + E_t \qquad (2)$$

Equation 2 can be approximated using the following formula:

$$U_{t_0+t} + E_t \approx \int_0^\infty x \times u(x)dx + E_t = E[U] + E_t \qquad (3)$$

where E[U] is the statistical mean of the measured historical CPU demand. Now the only undetermined parameter in Equation 3 is E_t, which will be computed based on [33] using Equations (4), (5) and (6).

Based on the assumption of our algorithm, the CPU demand is characterized by periodic behaviors. So for any time series T, the CPU demand in T can be represented as

$$U_T = D_T + U_T^R \qquad (4)$$

where D_T is the sum of periodic components and U_T^R is the residual component of the CPU demand. Since the period is known, the D_T is deterministic. The only random variable is U_T^R. In [33], U_T^R is modeled using a class of AR processes. An AR model is a simple and effective method in time series modeling. In particular, the authors use the second AR model (AR(2) model) as it has been demonstrated that a second order AR model is sufficient in most of the cases. The AR(2) model can be represented as

$$U_T^R = \alpha_1 U_{T-1}^R + \alpha_2 U_{T-2}^R + \epsilon_T \qquad (5)$$

where α_1 *and* α_2 are two AR(2) parameters that are estimated from the historical data, and ϵ_T is the error term, which is assumed to be an independently and identically distributed Gaussian random variable with a mean of zero and a variance of σ_ϵ^2. In order to study the accuracy of an *n* step prediction, a characteristic function of the AR model is defined as

$$G(j) = \frac{\gamma_1^{j+1} - \gamma_2^{j+1}}{\gamma_1 - \gamma_2} \qquad (6)$$

where γ_1 *and* γ_2 are the roots of the equation $1 - \alpha_1 B - \alpha_2 B^2 = 0$. Then the *n* step prediction error is represented using the Gaussian variable having mean zero and variance $\sigma_e^2(n) = \sum_{j=0}^{n-1} G^2(j)\sigma_\epsilon^2$. Here, σ_ϵ^2 is the error variance of one step prediction.

Once the future CPU demand, *CPU_Demand*, is estimated, it can be used as the argument in the cost function *C(CPU_Demand)* to get the exact operational cost for resource provisioning, where *C* is a function describing the relationship between CPU time and money spent. In the next step, we need to estimate the cost of attribute partitioning. In the next subsection, a novel estimation technique will be proposed to predict the cost of attribute partitioning.

D. CPU Utilization Forecasting for Attribute Partitioning

There are many factors that could impact the CPU time spent on figuring out a suitable attribute partitioning solution for a particular database table. The partitioning method used in our algorithm is based on the Closed Item Sets (CIS) [30] mining. The original version of the algorithm, called AutoClust, was first published in [15]. We can conclude 3 major factors that may heavily impact the CPU utilization spent on the attribute partitioning process. The first factor is the size of the database, *S*. In AutoClust, query optimizer is used to estimate the cost of each partitioning solution, i.e. partitions will be temporarily physically created in order to let the query optimizer compute the cost. If the database size is large the partition creation process will require a high CPU cost to be finished. The second factor is the number of attributes in a database table, *NA* (if the partitioning process covers more than one table, the maximum number of attributes among those tables will be used). When mining the CIS from the query set, possible attribute sets have to be generated. If there are *NA* attributes, the possible number of attribute sets would be 2^{NA}. Then each attribute set has to be compared with the attribute set accessed by each query in order to find out the CIS. There are many algorithms such as [43], [38] to prune the number of possible attribute sets, *NA*; but *NA* is still another factor that will impact the CPU cost on partitioning. Finally, the third factor is the number of query types, *NQ*. From the second factor, we already know each query type will be scanned in order to tell whether an attribute set is CIS. So *NQ* is a factor that has to be considered. Besides the above three factors, one more factor has to be added from the aspect of multi-tenancy, which is the number of users of the DbaaS, *NU*. In DbaaS, common physical resources are shared by multiple tenants. A major consequence of such environment is that the multiple tenants will compete for the resource of the same VM, which will delay the partition creation process. Hence the degree of multi-tenancy becomes an additional factor.

A cloud database is a complex system and the relationship between the partitioning cost and *S*, *NA*, *NQ* and *NU* is highly non-linear. Because of these characteristics, we propose to use an Artificial Neural Network (ANN) [25] to forecast the partitioning process cost since ANN performs well on complex systems that are intrinsically non-linear in nature [8]. In our ANN model, the inputs are *S*, *NA*, *NQ* and *NU*. One hidden layer with 8 neural nodes (twice the size of the input) is used between the input and output layers. The control architecture is a feed forward back propagation network. The activation function used is the sigmoid function, which is a transfer function used to calculate a layer's output from its net input, for all the inner nodes. This function can give the neural network the ability to learn and generate an output for which it is not trained. However, in order to make the ANN work properly, a well-defined training dataset is necessary. The whole ANN works in two phases: (1) the network is trained using the data provided by users; and (2) the new input is fed to the network and the network produces a desired output that is most appropriate for the given input. Since it is important to choose a proper training function and learning function, the TRAINGDX [22] and LEARNGDM [23] functions can be used in the network. The TRAINGDX function is a network training function that updates weight and bias values according to gradient descent momentum and an adaptive learning rate. The LEARNGDM function calculates the weight change for a given neuron from the neuron's input and error, the weight, learning rate, and momentum constant, according to gradient descent with momentum. The structure of the whole network is shown in Figure 3 where *W* represents the weight of a node and *B* represents the bias of a node. There are four different types of input (*S*, *NA*, *NQ* and *NU*) and one type of output (CPU time), so we have 4 nodes in input and 1 node in output. We use twice the size of the input types as the number of nodes in the hidden layer, so we have 8 nodes in the hidden layer.

Once the ANN model is constructed using the training dataset, the CPU time spent on the partitioning process can be predicted by sending the current values of the system parameters which are *S*, *NA*, *NQ* and *NU*, to the ANN model. Then the money spent on performing attribute partitioning tuning and resource provisioning tuning can be calculated by using the *C(CPU_Demand)* function where the CPU time estimated from each of the two forecasting partitioning and resource provisioning processes will be the argument of the *C* function. The method that yields the lowest monetary cost will be selected to improve the system performance. The algorithm is shown in Figure 4.

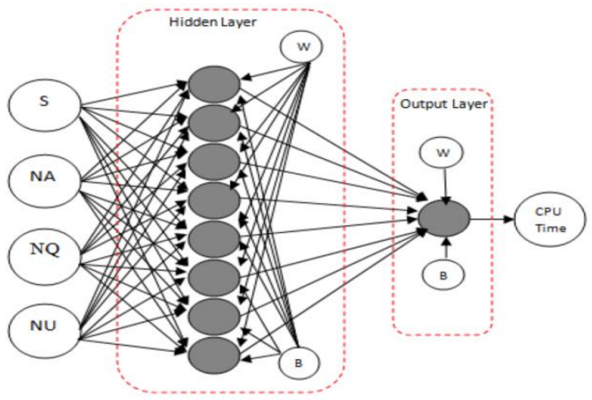

Figure 3. The ANN model used for partitioning cost forecasting.

```
Input:
        1.  Historical CPU utilization data (U)
        2.  Time interval of the future CPU demand forecasting (t)
        3.  Size of the database (S)
        4.  Maximum number of attributes of tables in the database (NA)
        5.  Number of query types (NQ)
        6.  Number of users (NU)
Output:
        1.  Estimated monetary cost of resource provisioning (CPV)
        2.  Estimated monetary cost of attribute partitioning (CPT)
// CPU time demand estimation for provisioning
1.   Compute u(x), the PDF of CPU utilization, using U;
2.   Compute E[U], the statistical mean of CPU utilization, using U and
     u(x);
3.   Estimate the AR(2) parameters, $\alpha_1$ and $\alpha_2$, using U;
4.   Compute the variance of n step prediction error ($E_t$), $\sigma_e^2(n)$, using
     Equation 6 in subsection C of section III;
5.   PDF of $E_t$, $f(x|\mu,\sigma) = \frac{1}{\sigma_e(n)\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma_e^2(n)}}$ ($\mu = 0$);
6.   CPU time of provisioning, CPU_PV = E[U] + $E_t$;
7.   CPV = C(CPU_PV);
// CPU time spent on partitioning
8.   CPU time of partitioning, CPU_PT = ANN(S, NA, NQ, NU);
9.   CPT = C(CPU_PT);
// Select the system tuning method with the lowest monetary cost
10.  If CPV > CPT then
11.      Run the attribute partitioning process described in subsection D
     of section III;
12.  Else
13.      Start the resource provisioning process;
14.  End If
```

Figure 4. The AutoClustC algorithm.

IV. EXPERIMENTAL PERFORMANCE STUDIES

In this section we present our experiment results that evaluate the accuracy of the CPU cost estimations for attribute partitioning and resource provisioning in our algorithm, and the performance of the new databases partition results on the cloud database after an attribute repartitioning takes place. The experiment results are presented in 4 subsections. In subsection A, we first present the performance of the ANN model used for estimating the CPU time for attribute partitioning. The Mean Square Error (MSE) and the regression, which shows the relationship between the outputs of the network and the targets, are used to estimate the accuracy of the ANN model. In subsection B, we present the performance of the AR(2) model used for estimating the CPU time for resource provisioning. The error ratio, which equals to $\frac{prediction\ error}{prediction\ error+prediction\ value}$, is used to estimate the accuracy of the AR(2) model. In subsection C, we present the final monetary cost ratio of the resource provisioning to the attribute partitioning. At last, in subsection D, the performance of the new database partitions measured by query response time is shown by running randomly the selected query sets from the TPC-H benchmark [37]. The experiments are done on AMAZON RDS cloud [2] with the database instance class of db.m1.medium and database engine of SQL Server SE 11.00.5058.0.v1. The experiment program is coded in Java and tcl script.

A. *Performance of the ANN Model for Attribute Partitioning Cost Forecasting*

The training dataset is from the monitor system called Cloud Watch Service provided by Amazon RDS. 20% of the dataset is used as the validation set and 20% of the data set is used as the test set. The performance measured in mean MSE is shown in Figure 4. From this figure we can see that the best performance MSE is about 2.12 happened at 720 epochs, which is a measure of the number of times all of the training vectors are used once to update the weights. If more epochs are performed the network will be over trained since the MSE of validation increases after 720 epochs.

Then the linear regression graph is shown in Figure 5. If the training were perfect, the outputs of the network would be exactly equal to the targets of the network, i.e. the dash line and color line in the graph should be 100% overlapped (R=1). But the relationship is rarely perfect in practice. From Figure 5 we can see that the training, validation and test are all fitting the network targets well.

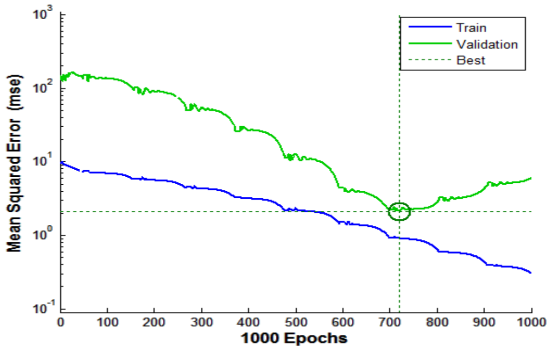

Figure 4. ANN performance on forecasting partitioning CPU time.

B. *Performance of the AR(2) model for Resource Provisioning Cost Forecasting*

In this subsection we use the data shown in Figure 2 as the historical CPU utilization demand. First the two parameters, α_1 and α_2, of the AR(2) model are estimated using the historical data. In this process, we use the classic Yule-Walker method [42] and [39] to estimate α_1 and α_2. Then from the description in subsection C of Section III, the prediction error's variance can be calculated. Figure 6 shows the probability distribution of the prediction error based on the historical data from Figure 2. We can see that 95% of the prediction errors range from -0.015 to 0.015. Comparing with the statistic mean of the historical data, which is 0.31, the error rate of forecasting is about $\frac{0.015}{0.015+0.31} = 4.6\%$.

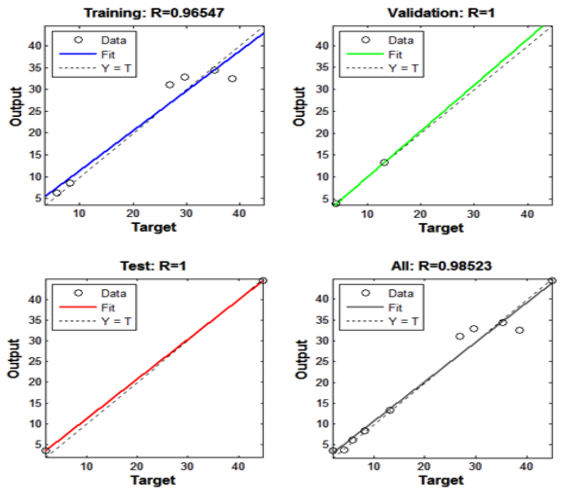

Figure 5. Linear regression of the network on forecasting the partitioning CPU time.

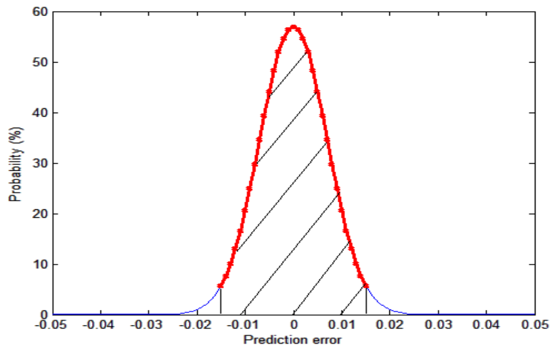

Figure 6. Probability distribution of prediction error in resource provisioning CPU time forecasting.

C. *Monetary Cost Ratio of Resource Provisioning Cost to Attribute Partitioning Cost*

Typically, after resource provisioning, the new assigned resources on the VM may take up to serval minutes for the acquired VM to be ready to use. This time is dependent on the image size (the size of the data mounted from a physical machine to a VM), VM type, data center location, etc. [21]. So the new assigned resources will not be released in minutes. We assume the dynamic provisioned resources will be kept at least for 30 minutes, which is also the time interval of two sample neighbor data points in Figure 2.

From the subsection A of section IV, we use the training data to estimate the CPU time cost of partitioning using the ANN model. The estimation is 42.55 CPU time units. From the subsection B of section IV we estimate the average CPU time provisioned to VM. The estimation is $0.325 \times 30 \times 60 = 585$ CPU time units. If the *C(CPU_Demand)* function is linear, then the final provision monetary cost measured in dollar will be about $\frac{C(585)}{C(42.55)} \approx \frac{585}{42.55} \approx 14$ times as the final monetary cost of partitioning.

D. Performance of the New Database Partitions

In subsection C of section IV we can conclude that the attribute partitioning method costs less money than resource provisioning; so the attribute partitioning method will be used for performance tuning to re-guarantee the performance SLA. In this subsection the performance in terms of percentile query response time of the new database partitions after the partitioning process is performed is presented.

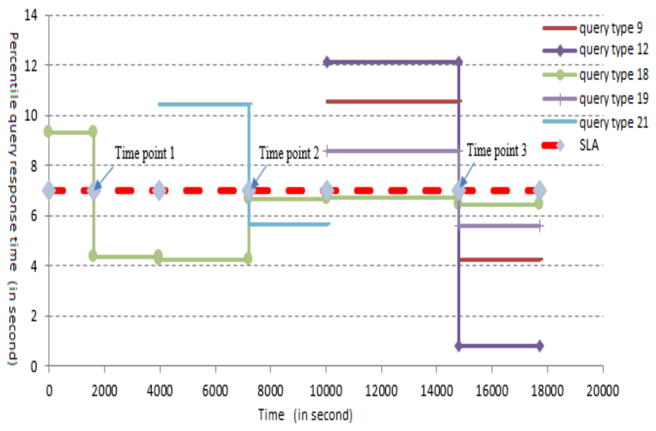

Figure 7. Query response time of 95th percentile of query for different query types before and after attribute partitioning.

We define the performance SLA as follows: at least the 95th percentile query response time of each query type must be within a specific time threshold, *TH*; otherwise the performance SLA is said to be violated. When a performance SLA violation occurs, the partitioning process will be triggered and the new partitions will be generated to replace the old ones. This experiment is conducted as follows:

1. A tenant's behavior is simulated on a cloud database.
2. Half number of the query types are randomly selected from the TPC-H query type benchmark, and each type of query is executed for a random number of times (less than 300).
3. Once all queries are successfully finished, Step 2 is repeated until the experiment time of 5 hours is reached.
4. In Step 3 if a performance SLA violation is detected, the partitioning process is triggered and new partitions are generated before Step 2 is repeated.

In Figure 7, each colored line represents the percentile query response time of the query corresponding to that color. The red dashed line represents the pre-defined performance SLA (*TH*). From Figure 7 we can see that the partitioning process occurs 3 times at the three time points 1, 2, and 3, i.e., performance SLA violations occur at the 3 time points in 5 hours. At time point 1, which is at about 1,800 seconds in the experiment time, a performance SLA violation is detected for query type 18; at time point 2, which is at about 7,100 seconds in the experiment time, a performance SLA violation is detected for query type 21; and at time point 3, which is at about 14,700 seconds in the experiment time, a performance SLA violation is detected for query types 9, 12 and 19. The performance SLA violations are caused by query pattern changes since the query set running on the cloud database is randomly changed in Step 2. If the time threshold *TH* is defined as 7 seconds then from this figure, we can see that the performance SLA is re-guaranteed again after the partitioning process is completed (shown as the colored line falling below the red dashed line again after the partitioning process is completed).

Figure 7 also can give a general idea to the service providers of what performance SLA should be made between them and their customers. If the customers are asking for a better response time, like 5 seconds for example, then from Figure 7, the providers can know that such performance SLA is really hard to guarantee if they still use the current VM configuration. In that case, they can provide better computing resources to the customers by charging a service upgrade fee. So our algorithm can also help the providers make a profitable decision on deriving a correct performance SLA.

V. CONCLUSIONS

In this paper we proposed an algorithm of re-guaranteeing a cloud database performance SLA by using a cost and performance SLA aware attribute partitioning method. The algorithm uses an ANN model and an AR(2) model to estimate the monetary cost spent on each of the two methods, resource provisioning and attribute partitioning, in order to select the most monetary cost saving method to tune the cloud database when a performance SLA violation occurs. The algorithm has the ability to help the service providers make reasonable performance SLAs with their customers. The experiments using the Amazon RDS and the TPH database benchmark show that the attribute partitioning process can provide more profit to the service providers and can re-guarantee the performance SLA caused by query pattern changes.

REFERENCES

[1] E. S. Abuelyaman, "An optimized scheme for vertical partitioning of a distributed database," International Journal of Computer Science and Network Security (IJCSNS), vol.8, no.1, 2008.

[2] Amazon AWS-RDS. Amazon RDS homepage, 2015. https://aws.amazon.com/rds/, Accessed Nov. 2105.

[3] R. Amossen, "Vertical partitioning of relational OLTP databases using integer programming," the 5th International Workshop on Self Managing Database Systems (SMDB), 2010.

[4] S. Aulbach, D. Jacobs, A. Kemper and M. Seibold, "A comparison of flexible schemas for software as a service," Special Interest Group on Management of Data (SIGMOD), 2009.

[5] N. Bobroff, A. Kochut and K. Beaty, "Dynamic placement of virtual machines for managing SLA violations," 10th IFIP/IEEE International Symposium on Integrated Network Management (IM'07), 2007.

[6] G. E. P. Box and G. M. Jenkins, "Time series analysis: forecasting and control," San Francisco, Holden Day, pp. 575. 1976.

[7] R. Buyya, S. K. Garg and R. N. Calheiros, "SLA-oriented resource provisioning for cloud computing: challenges, architecture, and solutions," International Conference on Cloud and Service Computing, 2011.

[8] G. Zhang, B. E. Patuwo and M. Y. Hu, "Forecasting with artificial neural networks: the state of the art," Int. J. Forecast., vol. 14, pp. 35-62, 1998.

[9] C. H. Cheng, W. K. Lee and K. F. Wong, "A genetic algorithm based clustering approach for database partitioning," IEEE Transactions on System, Man and Cybernetics, vol. 32, no. 3, 2002.

[10] Y. Chi, H. J. Moon and H. Hacigumus, "iCBS: incremental cost-based scheduling under piecewise linear SLAs," PVLDB, 2011.

[11] C. Curino, E. P. C. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan and N. Zeldovich, "Relational cloud: a database-as-a-service for the cloud," In CIDR, 2011.

[12] J. L. Deneubourg, S. Aron, S. Goss and J. M. Pasteels, "The self-organizing exploratory pattern of the argentine ant," Journal of Insert Behavior, vol. 3, pp. 159, 1990.

[13] D. Gmach, S. Krompass, A. Scholz, M. Wimmer and A. Kemper, "Adaptive quality of service management for enterprise services," ACM Transactions, Web, 2008.

[14] M. Goli and S. A. Rankoohi, "New vertical fragmentation algorithm based on ant collective behavior in distributed database systems," Knowledge and Information Systems, vol. 30, issue 2, February 2012.

[15] S. Guinepain and L. Gruenwald, "Using cluster computing to support automatic and dynamic database clustering," International Workshop on Automatic Performance Tuning (IWAPT), September 2008.

[16] H. Jayathilaka, C. Krintz and R. Wolski, "Response time service level agreements for cloud-hosted web applications," Proc. of the Sixth ACM Symposium on Cloud Computing (SoCC), pp. 315-328, 2015.

[17] A. Jindal and J. Dittrich, "Relax and let the database do the partitioning online," In Business Intelligence for Real Time Enterprise (BIRTE), September 2011.

[18] W. Lang, S. Shankar, J. Patel and A. Kalhan, "Towards multi-tenant performance SLOs," Proc. of the 28th IEEE Int. Conf. on Data Engineering (ICDE), pp. 702-713, 2012.

[19] L. Li, and L. Gruenwald, "Autonomous database partitioning using data mining on single computers and cluster computers," International Database Engineering & Applications Symposium (IDEAS), August 2012.

[20] L. Li and L. Gruenwald, "SMOPD-C: an autonomous vertical partitioning technique for distributed databases on cluster computers," the 15th IEEE International Conference on Information Reuse and Integration (IRI '14), November 2014.

[21] M. Mao and M. Humphrey, "A performance study on the VM startup time in the cloud," Proc. of IEEE 5th International Conference on Cloud Computing, 2012.

[22] Description of TRAINGDX function, MathWorks Documentation page, 2015. http://www.mathworks.com/help/nnet/ref/traingdx.html, Accessed Nov. 2015.

[23] Description of LEARNGDM function, MathWorks Documentation page, 2015. http://www.mathworks.com/help/nnet/ref/learngdm.html, Accessed Nov. 2015.

[24] Microsoft SQL Azure. Microsoft SQL Azure homepage, 2015. https://azure.microsoft.com/en-us/services/sql-database/, Accessed Nov. 2015.

[25] W. McCulloch and P. Walter, "A logical calculus of ideas immanent in nervous activity," Bulletin of Mathematical Biophysics 5 (4): pp. 115–133, 1943.

[26] V. Narasayya, S. Das, M. Syamala, B. Chandramouli and S. Chaudhuri, "SQLVM: performance isolation in multi-tenant relational database-as-a-service," In 6th Biennial Conference on Innovative Data Systems Research (CIDR), 2013.

[27] S. Navathe, S. Ceri, G. Wierhold and J. Dou, "Vertical partitioning algorithms for database design," ACM Transactions on Database Systems, vol. 9, no. 4, December 1984.

[28] S. Navathe and M. Ra, "Vertical partitioning for database design: a graph algorithm," ACM Special Interest Group on Management of Data (SIGMOD) International Conference on Management of Data, 1989.

[29] S. Papadomanolakis, D. Dash and A. Ailamaki, "Efficient use of the query optimizer for automated physical design," International Conference Very Large Databases (VLDB), September 2007.

[30] N. Pasquier, Y. Bastidem, R. Taouil and L. Lakhal, "Efficient mining of association rules using closed item set lattices," Information Systems, vol. 24, no. 1, 1999.

[31] C. Ryan, C. Kyle, B. Kris and L. Lukasz, "Cost-aware cloud provisioning," IEEE 11th International Conference on e-Science, 2015.

[32] S. Sakr and A. Liu, "SLA-based and consumer-centric dynamic provisioning for cloud databases," Proc. IEEE Cloud, pp. 360-367, 2012.

[33] D. She and J. Hellerstein, "Predictive models for proactive network management: application to a production web server," in Proc. of the IEEE/IFIP Network Operations and Management Symposium, 2000.

[34] P. Shivam, A. Demberel, P. Gunda, D. Irwin, L. Grit, A. Yumerefendi, S. Babu and J. Chase, "Automated and on-demand provisioning of virtual machines for database applications," In Proc. of SIGMOD, 2007.

[35] F. R. C. Sousa and J. C. Machado, "Towards elastic multi-tenant database replication with quality of service," in IEEE 5th International Conference on Utility and Cloud Computing (UCC), 2012.

[36] S. Tozer, T. Brecht and A. Aboulnaga, "Q-cop: avoiding bad query mixes to minimize client timeouts under heavy loads," In Proc. of ICDE, 2010.

[37] Description of TPC-H benchmark, TPC official website, 2016. http://www.tpc.org/tpch/, Accessed Jan. 2106.

[38] T. Uno, M. Kiyomi and H. Arimura, "LCM ver. 2: efficient mining algorithms for frequent/closed/maximal itemsets," Proc. of the IEEE ICDM workshop on frequent itemset mining implementations. Brighton, UK, 2004.

[39] G. Walker, "On periodicity in series of related terms," Proc. of the Royal Society of London, ser. A, vol. 131, pp. 518–532, 1931.

[40] W. C. Wesley and I. Leong, "A transaction-based approach to vertical partitioning for relational database systems," IEEE Transactions on Software Engineering, vol. 19, no. 8, August 1993.

[41] P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu and H. Hacigumus, "Intelligent management of virtualized resources for database management systems in cloud environment," In Proc. of ICDE, 2011.

[42] G. U. Yule, "On a method of investigating periodicities in disturbed series, with special reference to wolfer's sunspot numbers," Philosophical Transactions of the Royal Society of London, ser. A, vol. 226, pp. 267–298, 1927.

[43] M. Zaki and C. J. Hsiao, "Efficient algorithms for mining closed item sets and their lattice structure," IEEE Trans Knowl Data Eng 17(4), pp. 462–478, 2005.