International Journal of Grid Distribution Computing
Vol.8, No.2 (2015), pp.319-330
http://dx.doi.org/10.14257/ijgdc.2015.8.2.27

Research on Optimization Adjustment Strategy for SaaS Multi-tenant Data Placement

Li Xiaona[1,2], Li Qingzhong[1,*], Zhu Weiyi[3] and Li Hui[1]

[1]School of Computer Science and Technology, Shandong University, Jinan 250101, China
[2]Qingdao University, Qingdao 266071, China
[3]State Grid Shandong Electric Power Company, Jinan 250001, China
lixiaona_sdu@163.com, lqz@sdu.edu.cn, lih@sdu.edu.cn

Abstract

In order to meet the requirements for data access by tenants and management by service providers, the multi-tenant data stored in cloud using replica technology must be reasonably placed. For the outweight nodes and the ultra light nodes, according to characteristics of the multi-tenant data and the load of nodes, through adjusting the number and position of the replicas to maintenance and optimization the strategy so that meet the SLA requirements meanwhile minimize the overall cost. Experimental results through comparison with random placement strategy and greedy placement policy demonstrate the feasibility and effectiveness of the proposed strategy.

Keywords: *SaaS, Multi-tenant database, optimization adjustment, replica placement*

1. Introduction

With the development of cloud computing, SaaS (Software as a service) [1] as a novel business model arouse increasingly attentions. The service provider uses Single Instance Multi-tenancy software delivery to support multiple tenants customizes SaaS application on-demand. Tenants don't care implementation details about applications and data storage, through signing the Service level agreements (SLAs) [2] with service providers to obtain performance guarantee. Tenant's data stored by the service provider in a shared database [3].

In order to meet the characteristics of cloud environment, data management in cloud use replication technology to place the data in different nodes [4]. The Data placement strategies will directly influence the system performance, the writing and reading efficiency and the maintenance of consistency. The placement for multi-tenant data ultimately relates to the cost for the service providers and the service experience for the tenants. Therefore, research on multi-tenant data placement strategy in SaaS model becomes extremely important.

Based on shared database shared schema, through fully considering the characteristics and access load of the nodes, this paper establishes the cost model and proposes data placement strategies that meet the SLA requirements for all tenants meanwhile minimize the cost for service providers, so that realizing the balance between the quality of service and the cost of management.

In the remainder of the paper, is organized as follow. In Section 2 discusses the related work in placement of cloud data. Section 3 formally describes the placement problem of the multi-tenant data in SaaS model. The optimization adjustment strategies and algorithms for multi-tenant data placement are illustrated in Section 4. Section 5 compares the placement strategy in this paper with other strategies, and analyzes the experimental

[*] Corresponding Author

results. The last section concludes the mentality of this paper and proposes the additional work in the future.

2. Related Works

Many scholars [5, 6] have done depth research on the data placement in the cloud. HDFS [7] adopt the rack-aware strategy mainly for analytical data, not suitable for transactional SaaS application workload, then in placement will lead to a large amount of distributed cost.

Carlo Curino's Schism [8] proposed a transaction-driven partition strategy, National University of Singapore proposed ecStore [9] which have three-tier architecture, but all of these partition and placement strategies are not designed for SaaS applications without considering the characteristics of it, then easily affect the load balancing of nodes and increase the cost of tenant data access.

Authors in [10] proved that the issue of replica placement in general network graph is an NP (Non-deterministic Polynomial)-complete problem. Based on heuristic, Literature [11,12] presented several replica placement alglghms, but they are only consider how to place one independent data with p replicas in an optimal way, however, in SaaS model, the optimal placement for one tenant data does not meaning the optimal placement for the global.

In order to ensure the availability of the multi-tenant data for SaaS application in cloud, literature [13] proved the threshold of the replica number. Based on this, in this paper, the number of the replicas is set in a certain range [R_{min}, R_{max}].

The existing replica management techniques in web [14] or in cloud cannot be directly applied to SaaS environment. Therefore, this paper combines some characteristics of shared storage and data partition for multi-tenant data, and further adjusts the number and position of the replicas to optimize the strategy to meet the SLA requirements for all tenants meanwhile to minimize the cost for service providers.

3. Problem Description and Modeling

For convenience, multi-tenant data replica placement discussed in this paper is modeled as the multi-tenant data placement graph, as shown in Figure 1.

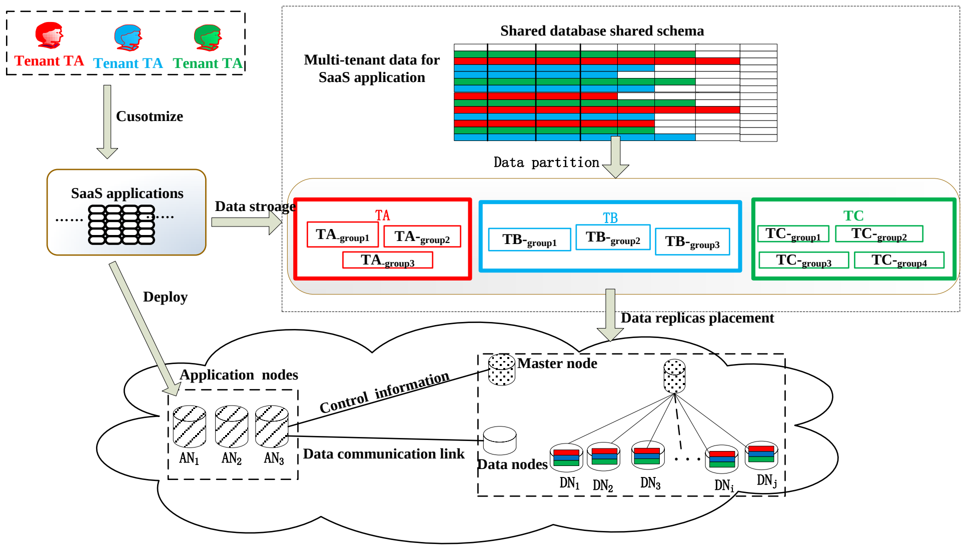

Figure 1. The Placement Graph for SaaS Multi-tenant Data

Previous work [15] proposes gather the multi-tenant data that has been often accessed together in different transactions as the partition granularity group to reduce the distributed transaction cost. This paper will continuously take group as the data placement granularity.

The symbols in figure 1 are explained in Table 1.

Table 1. Symbol Definition

T={TA,TB,TC...}	Tenants set
TX ={$TX_{-group1}$, $TX_{-group2}$,...$TX_{-groupy}$...}	Data set for tenant TX, through data partition
$R_{TX-groupy}$	The replica number of data $TX_{-groupy}$, $R_{min} \le R_{TX-groupy} \le R_{max}$
Master node	Master nodes store and control the metadata information
DN={DN_1,DN_2...DN_i,...}	Data nodes set.
C_{DN_i}, $\bar{S}_{DN_i}$, $\bar{\lambda}_{DN_i}$	The data node computing capacity, the upper limit of storage space, the ideal load in load balance state of DN_i, respectively
AN={AN_1,AN_2,…,AN_i,…}	The application nodes set on which deploy SaaS applications
$R^i_{X,y} = 1$	Indicate the requests about data $TX_{-groupy}$ derived from AN_i
$M^i_{X,y} = 1$	Indicates a replica of $TX_{-groupy}$ is stored on DN_i

Tenants obtain the location list stored in master node about every data replica, via the communication link to visit corresponding data nodes, data transmitted between the application nodes and the data nodes in the reading and writing process.

The tenants SLA requirements and cost in this paper are defined as following.

Definition 1. The tenants SLA requirements. When the loads of all data nodes in load balance state, any request can be completed within service response time specified in the SLA, then referred as meet the tenants SLA requirements in this paper.

Definition 2. Load identification. The load for any one tenant TX is labeled as $\lambda_{TX} = \lambda^w_{TX} + \lambda^r_{TX}$, which consists of reading load and writing load. The load for data $TX_{-groupy}$ and for it's any replica, marked as $\lambda_{TX-groupy} = \lambda^w_{TX-groupy} + \lambda^r_{TX-groupy}$, and $\lambda'_{TX-groupy} = \lambda'^w_{TX-groupy} + \lambda'^r_{TX-groupy}$ respectively.

When the load for all nodes are satisfy $(1-\beta)\bar{\lambda}_{DN_i} \le \lambda_{DN_i} \le (1+\beta)\bar{\lambda}_{DN_i}$, the system is considered as load balance, β is the parameter set by system. This paper supposes $\bar{\lambda}_{DN_i}$ is the ideal load that node DN_i can bear, calculated according to its own computing capability.

Definition 3. Data node maintenance cost $Cost_p$. The maintenance cost for staring a data node per unit time incurred is assumed to be P. When the total numbers of the nodes in system is Z, the maintenance costs per unit time for data nodes recorded as

$Cost_p = Z \times P$ (1)

This paper adopts "Read-One-Write-All" pattern [16] and assumes each data replica share the read load evenly, then $\lambda'^w_{TX-groupy} = \lambda^w_{TX-groupy}$, $\lambda'^r_{TX-groupy} = \frac{\lambda^r_{TX-groupy}}{R_{TX-groupy}}$. This paper defines matrix AD [i,j] to represent the bandwidth between application node AN_i and data node DN_j.

Definition 4. Communication cost in data writes process. The write requests are propagated to all nodes where stored the replicas to synchronize. Communication cost is relative to the write load of the data and the bandwidth value of the matrix AD, written as:

$$\text{Cost}_w = \sum_{TX \in T, TX_{-groupy} \in Data(TX)} [\lambda^{w}_{TX_{-groupy}} \times (\sum_{M^{j}_{X,y}=1, R^{i}_{X,y}=1} AD[\mathrm{i},\mathrm{j}])] \qquad (2)$$

Definition 5. Communication cost in data read process. One replica will response the read request. Communication cost in data read process is relative to the read load of data as well as the bandwidth value of the matrix AD, written as:

$$Cost_r = \sum_{TX \in T, TX_{-groupy} \in Data(TX)} [\lambda^{'r}_{TX_{-groupy}} \times (\sum_{M^{j}_{X,y}=1, R^{i}_{X,y}=1} AD[\mathrm{i},\mathrm{j}])] \qquad (3)$$

Definition 6. The overall cost. It can be expressed as the weighted sum including the maintenance cost of all nodes and the communication cost produced in data accesss, namely $\text{COST}=a\text{Cost}_p+b\text{Cost}_w+c\text{Cost}_r$. The goal is to minimize the value, where $a,b,c\in[0,1]$. This paper based on the principle that priority selecting existing nodes in placement, let $a>b, a>c$.

4. Optimization Adjustment Strategies and Algorithms for SaaS Multi-tenant Data Placement

SaaS multi-tenant data placement is divided into three stages. As shown in Figure 2. This paper is mainly focus on researching the optimization adjustment stage.

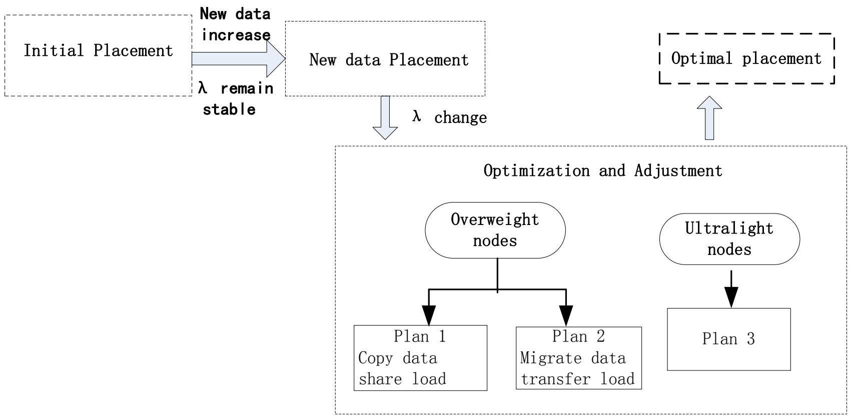

Figure 2. Different Stages of the Multi-Tenant Data Placement

4.1. The Initial Placement Stage

In initial state, the number of tenants is less and the size of data is small. System starts R_{min} data nodes can realize the initial placement, and the replicas are located in different nodes.

4.2. The New Data Placement Stage

The new data denoted as D^{i}_{new}. Through determining whether D^{i}_{new} belongs to existing tenant TX, and further whether belongs to the existing group, will adopt different strategies to place it. The data that belongs to the same tenant and the same group placed together, so the communication cost between replicas kept minimum. Detailed content is described in [17].

4.3. The Optimization Adjustment Stage

The dynamic of the cloud can lead system load changes, affect the load balance of the data nodes, and cause the response time for tenants access requests delay that violate the SLA requirements. So, when the load changes, the optimization adjustment stage through adjusting the number and position of the replicas to ensure the load of all nodes are in normal range and meet the response time specified in SLA, meanwhile making the overall cost minimum.

According to the actual load of data node, the optimization adjustment strategy divides nodes into overweight load nodes and ultralight load nodes (referred to as overweight nodes and ultralight nodes) to deal with respectively. This paper assumes that if the actual load of node DN_i satisfy $\lambda_{DN_i} > (1+\beta)\bar{\lambda}_{DN_i}$, then called overweight node, else, if $\lambda_{DN_i} < (1-\beta)\bar{\lambda}_{DN_i}$, called ultralight node. β is the parameter set by the system. $\lambda^{'old}_{TX-groupy}$ represents the load value of data $TX_{-groupy}$ on each node before the replica number changed, $\lambda^{'new}_{TX-groupy}$ indicates the load value after the replica number adjusted.

4.3.1. Optimization Adjustment Strategy for Overweight Nodes:

Step 1: Sort all data $TX_{-groupy}$ on the overweight node DN_i in descending order according to access load $\lambda'_{TX-groupy}$. Other nodes DN_p are sorted in ascending order based on node load. In figure 3(a), $TB_{-group1}$ (bold line data) has the highest access frequency on overweight node DN_1(bold line node), other nodes arranged as follows: DN_3, DN_4, DN_2.

Step 2: Determine the replica number of the data $TX_{-groupy}$ ordinally, if $R_{min} \le R_{TX-groupy} < R_{max}$, then there are two adjustment plans can adopt which are "copy data share load"(Plan 1) and "migrate data transfer load"(Plan 2). Select the plan with minimum COST value to implement, so that can ensure the overall cost is minimum in optimization adjustment process. If the replica number of data $TX_{-groupy}$ is equal to R_{max}, then directly use plan 2 to adjust, otherwise, the replica number exceed R_{max} can lead to the cost of system increasing.

Figure 3(b) and (c) describe data $TB_{-group1}$ copied or migrated to the existing nodes through the two plans. If the capacity or the load does not meet the conditions of the existing nodes, then new node added as the destination node for $TB_{-group1}$.

Step 3: Repeat step 2 until the load of DN_i in normal range.

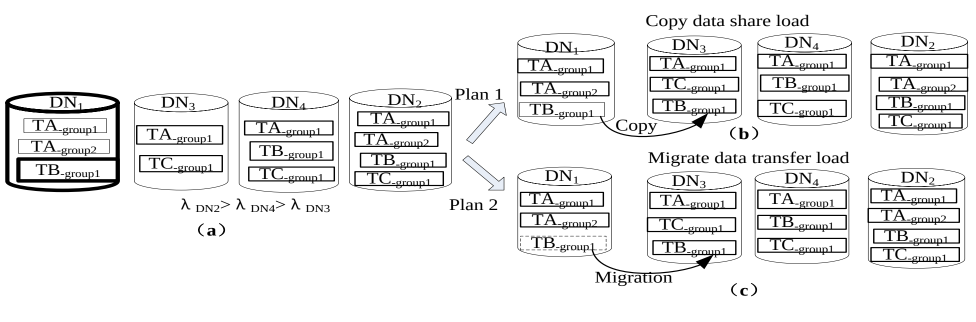

Figure 3. Overweight Node Adjustment Illustration

4.3.2. Optimization Adjustment Algorithm for Overweight Nodes:

For convenience, define the calculation of the overall cost and the minimal communication cost as well as the two plans as operators. As shown below.

Operator 1: Find a node that with nodes in a known set has the minimal communication cost

```
Input: data nodes set F, data $TM_{-groupn}$.
Output: the destination node $DN_g$.
1. MIN=a maximum value set in initial state
2. FOR （each node $DN_p$ in set F）
    2.1 DJ=0;
    2.2 For (each application node $AD_i$ in matrix AD)
            IF（$R^{i}_{M,n} = 1$）THEN  DJ=DJ+AD[i,p];
    2.3 IF $DJ<MIN$ THEN { $MIN=DJ; DN_g=DN_p$ }
3. RETURN ($DN_g$)
```

Operator 2: The value of the overall cost

```
Input: nodes set DN, system load λ.
Output: COST.
1. $COST=aCost_p+bCost_w+cCost_r$
2. RETURN ($COST$)
```

Operator 3: Plan 1 for overweight nodes

```
Input: $TX_{-groupy}$,  $DN_i$,  DN,  λ,  $DN_m$.
Output: COST.
1. $F=\{DN_p|DN_p \in N, DN_p \neq DN_i$, sort $DN_p$ in ascending order according to $\lambda_{DNp}\}$;  $L=\Phi$
2. FOR ( each node $DN_p$ in F)
        IF $(((S_{DN_p} + S_{TX_{-groupy}}) < \bar{S}_{DN_p}) \&\& (M^{p}_{X,y} = 0) \&\& ((\lambda_{DN_p} + \lambda^{'new}_{TX_{-groupy}}) < (1+\beta)\bar{\lambda}_{DN_p}))$ THEN $L=L\cup\{DN_p\}$
3. IF ($L\neq\Phi$)  THEN { $DN_m$=Operator 1($L$,  $TX_{-groupy}$)  let：
            $\lambda_{DN_i} = \lambda_{DN_i} - \lambda^{'old}_{TX-groupy} + \lambda^{'new}_{TX-groupy}, \lambda_{DN_m} = \lambda_{DN_m} + \lambda^{'new}_{TX-groupy}$ }
        ELSE{ add new node as destination node $DN_m$,  let:
            $\lambda_{DN_i} = \lambda_{DN_i} - \lambda^{'old}_{TX-groupy} + \lambda^{'new}_{TX-groupy}$,  $\lambda_{DN_m} = \lambda_{DN_m} + \lambda^{'new}_{TX-groupy}$ }
4. $COST$=Operator 2（$DN$, $\lambda$）
5. RETURN ( $COST$)
```

Operator 4: Plan 2 for overweight nodes

```
Input: $TX_{-groupy}$,  $DN_i$,  $DN$,  $\lambda$,  $DN_e$.
Output: COST.
1. $F=\{DN_p|DN_p \in N, DN_p \neq DN_i$, sort $DN_p$ in ascending order according to $\lambda_{DNp}\}$;  $L=\Phi$
2. FOR ( each node $DN_p$ in F)
        IF $(((S_{DN_p} + S_{TX_{-groupy}}) < \bar{S}_{DN_p}) \&\& (M^{p}_{X,y} = 0) \&\& ((\lambda_{DN_p} + \lambda^{'}_{TX_{-groupy}}) < (1+\beta)\bar{\lambda}_{DN_p}))$ THEN $L=L\cup\{DN_p\}$
3. IF ($L\neq\Phi$) THEN { $DN_m$=Operator 1($L$,  $TX_{-groupy}$), let：
            $\lambda_{DN_i} = \lambda_{DN_i} - \lambda^{'}_{TX-groupy}$,  $\lambda_{DN_e} = \lambda_{DN_e} + \lambda^{'}_{TX-groupy}$ }
        ELSE {add new node as destination node $DN_e$,  let：
            $\lambda_{DN_i} = \lambda_{DN_i} - \lambda^{'}_{TX-groupy}$,  $\lambda_{DN_m} = \lambda_{DN_m} + \lambda^{'}_{TX-groupy}$ }
4. $COST$=Operator 2（$DN$, $\lambda$）
5. RETURN ( $COST$)
```

Algorithm 1: Optimization adjustment algorithm for overweight nodes

```
Input: Nodes set DN which existing overweight node.
Output: Nodes set DN which not existing overweight node.
1. FOR ( each overweight node $DN_i$) {
   1.1   $F=\{TX_{-groupy} | M_{X,y}^{i}=1$,  sort $F$ in a non-increasing order of $\lambda_{TX-groupy}\}$
   1.2   $COST1=0$, $COST2=0$, $DN_m=null$,  $DN_e=null$
   1.3   FOR ( each $TX_{-groupy}$ in F)  DO
         1.3.1  IF ( $R_{min} \le R_{TX-groupy} < R_{max}$ )
                   {COST1=Operator 3 ( $TX_{-groupy}$, $DN_i$, $DN$, $\lambda$, $DN_m$)
                    COST2=Operator 4 ( $TX_{-groupy}$, $DN_i$, $DN$, $\lambda$, $DN_e$)
                    IF ( $COST1 \le COST2$ ) THEN { copy $TX_{-groupy}$ to destination node $DN_m$
,
                    modify $\lambda_{DN_i}=\lambda_{DN_i}-\lambda_{TX-groupy}^{'old}+\lambda_{TX-groupy}^{'new}$ ,  $\lambda_{DN_m}=\lambda_{DN_m}+\lambda_{TX-groupy}^{'new}$ , $M_{X,y}^{m}=1$ }
                   ELSE {copy $TX_{-groupy}$ to destination node $DN_e$, delete from $DN_i$,
                   modify $\lambda_{DN_i}=\lambda_{DN_i}-\lambda_{TX_{-groupy}}^{'}$ ,  $\lambda_{DN_e}=\lambda_{DN_e}+\lambda_{TX_{-groupy}}^{'}$ ,  $M_{X,y}^{e}=1$ ,
              $M_{X,y}^{i}=0$ }
         1.3.2  IF ( $R_{TX-groupy}=R_{max}$ )
                   {Operator 4 ( $TX_{-groupy}$, $DN_i$, $DN$, $\lambda$, $DN_e$ )
                   Copy $TX_{-groupy}$ to destination node $DN_e$,  delete from $DN_i$,
                   modify $\lambda_{DN_i}=\lambda_{DN_i}-\lambda_{TX_{-groupy}}^{'}$ ,  $\lambda_{DN_e}=\lambda_{DN_e}+\lambda_{TX_{-groupy}}^{'}$ ,  $M_{X,y}^{e}=1$ ,
$M_{X,y}^{i}=0$ }
2.  } WHILE  ( $DN_i$ not overweight)
3.  RETURN  ( $DN$ )
```

4.3.3. Optimization Adjustment Strategy for Ultralight Nodes:

If the overall cost that have ultralight node is greater than deleting the ultralight node, then system need to optimization and adjustment. Delete the ultralight node may cause the replica number of some data less than R_{min}, then it is need migrate data to a suitable destination node. Otherwise allow the ultralight nodes exist. Adjustment steps as follows:

Step 1: Classify all data $TX_{-groupy}$ in ultralight node DN_i If $R_{min}<R_{TX-groupy} \le R_{max}$, then the data added into set L1. If the replica number is equal to R_{min}, then the data joined into set L2. Calculation the overall cost of the system, denoted as COST1.

In figure 4(a), the replica number of $TA_{-group1}$ on ultralight node DN_2 (bold line node) is 4 that greater than R_{min} 3, then $TA_{-group1}$ included into set L1. The replica number of $TC_{-group1}$ is 3, then the data joined into set L2.

Step 2: Process the data in L1 and L2, respectively. For each $TX_{-groupy}$ in L1, find the nodes that store the rest of replicas. If delete $TX_{-groupy}$, then each replica shoud share more load due to the decreased replica number, and lead to the load of the node that store the replica increasing. If appear overweight nodes, then should continue to adjust. For each data in L2, migrate to appropriate destination node. Calculate the overall cost after adjustment, denoted as COST2.

In Figure 4(b), if delete the ultralight node (dashed line node), then $TX_{-group1}$ in set L1 can be directly abandon, but $TC_{-group1}$ in set L2 need to migrate the destination node DN_1.

Step 3: If COST2 is the minimum value, then delete the ultralight, Otherwise, the ultralight node remain unchanged.

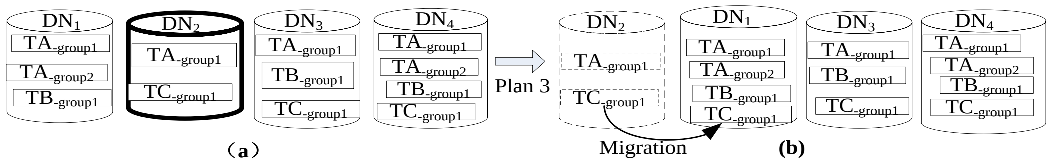

Figure 4. Ultralight Node Adjustment Illustration

4.3.4. Optimization Adjustment Algorithm for Ultralight Nodes:

For convenience, define the step 2 as operators.

Operator 5: The adjustment plan for ultralight nodes

```
Input: $L1'$, $L2'$, $\lambda$, $DN_m$.
Output: COST.
1. FOR each ($TX_{-groupy}$) in $L1'$
     { $DN_p$ that store the replica of the data $TX_{-groupy}$, let $\lambda_{DN_p} = \lambda_{DN_p} - \lambda_{TX-groupy}^{'old} + \lambda_{TX-groupy}^{'new}$ }
2. FOR each ($TX_{-groupy}$) in $L2'$
     {migrate the data $TX_{-groupy}$ to the node $DN_p$ that satisy the following conditions
     $((S_{DN_p} + S_{TX-groupy}) < \bar{S}_{DN_p}) \& \&((\lambda_{DN_p} + \lambda'_{TX-groupy}) < (1+\beta)\bar{\lambda}_{DN_p}) \& \&(M_{X,y}^{p} = 0)$, let $\lambda_{DN_p} = \lambda_{DN_p} + \lambda'_{TX-groupy}$ }
3. IF $\exists DN_i \in DN, \lambda_{DN_i} > (1+\beta)\bar{\lambda}_{DN_i}$ then Algorithm 1 (N)
4. DN=DN-{$DN_m$}; $COST$=Operator 2 ($DN$, $\lambda$)
5. RETURN (COST)
```

Algorithm 2: The optimization adjustment algorithm for ultralight nodes

```
Input: Nodes set DN, The load of the system $\lambda$.
 Outpu: Nodes set DN.
1. FOR ( each ultralight node $DN_i$)
     1.1 { $COST1$= Operator 2 ($DN$, $\lambda$)
     1.2 add all $TX_{-groupy}$ satisfy ($R_{min} < R_{TX-groupy} \leq R_{max}$) into $L1$
     1.3 add all $TX_{-groupy}$ satisfy ($R_{TX-groupy}$=Rmin) into L2
     1.4 $L1'=L1$, $L2'=L2$, $DN_m=DN_i$, $COST2$=Operator 5 ($L1'$, $L2'$, $\lambda$, $DN_m$)
     1.5 IF ($COST2<COST1$) THEN
         {delete the source node $DN_i$, update the load and placement information of the node
}
             ELSE the system maintains the ultralight node }
2. RETURN (DN)
```

4.4 The Analysis of the Algorithm Complexity

N is the number of the nodes. Time complexity of the adjustment algorithm for overweight nodes is O(N^2*D), where D is the number of the data stored on node based on group granularity. Time complexity for ultralight nodes is O(N^3*D). So the time complexity of the optimization adjustment strategies for SaaS multi-tenant data placement is O(N^3*D).

5. Experiments and Results Analysis

This section through simulation experiments to compare and evaluate the performance of the strategy proposed in this paper with greedy strategy and random strategy.

5.1. Experimental Setup

Experimental data collected from research group project “Public foundation support platform for Occupational qualifications” in which every application deployed as SaaS application and rented by many tenants.

Data is stored in shared database shared schema and use the MySQL database which installed on the data nodes. Data nodes and application nodes are all IBM servers, which is configured with Intel (R) Xeon (R) CPU X5620*16 of 2.4GHZ, 8G main memory, and 500G*2 of hard disk.

5.2. Results and Analysis

Depending on the application size and characteristics of the experimental data, we are using the multi-thread technique to simulate application transaction requests in experiments.

(1) Experiment 1: is used to verify the placement strategies impact on the average response time. At first it is assumed that there a SaaS application, there are five data nodes and 20 tenants.

Different strategies are used to place multi-tenant data on five physical nodes respectively. Each tenant sends many kinds of requests. Set a different read rate and write rate for each tenant, after several rounds of repeated tests, we get the average response time results of the tenants. The result of comparative experiment is shown in Figure 5.

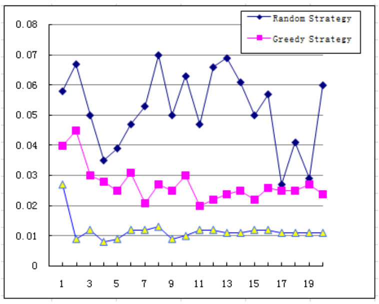

Figure 5. Comparison of Three Strategies about the Average Response Time

As shown in Figure 5, random strategy and greedy strategy will cause the long response time, but the strategy in this paper can greatly reduce the response time for tenants.

(2) Experiment 2: With the data size increasing, system expands into 8 nodes and contains 30 tenants, Rmin is set 3 and Rmax is 5. If the load of system changed then lead to some data node overweight.

Experiment adopts FCFS (first-come-first-service) policy to response the queries from tenants. This experiment set β=10%, let $diff_i = \frac{|\bar{\lambda}_{DN_i} - \lambda_{DN_i}|}{\bar{\lambda}_{DN_i}}$, and the “max difference ratio of the system load” is denoted as $Diff_{max}=max(diff_i)$ which used to measure the strategies impact on the load of all nodes.

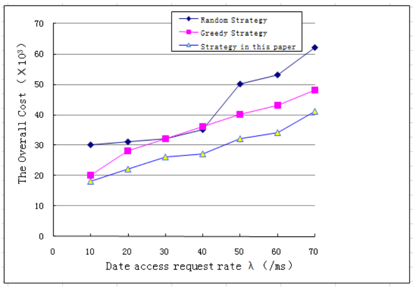

Figure 6. λ Impact on the Overall Cost

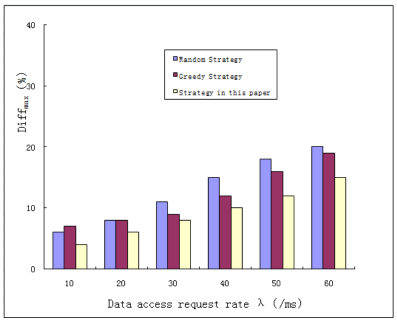

Figure 7. λ Impact on the Value of Diff$_{max}$

As shown in Figure 6 and Figure 7, the corresponding overall cost generated from three placement strategies and the value of Diff$_{max}$ are shown obviously rising trend. But strategy in this paper considers the load of nodes and adopts optimization adjustment, so the value of Diff$_{max}$ is lower than other two.

In conclusion, random placement strategy is simple and without considering the characteristics of multi-tenant data, likely to cause the data that belong to same tenant stored in multiple different nodes, so resulting in a large number of access cost. The greedy strategy mainly from perspective of optimizing the storage space of the nodes to place data, but the load performance of the nodes does not obtain a better effect. The strategy in this paper place the data based on the group granularity, so avoid distribution cost generated by distributed storage, and optimize the placement along with the load of node to realize the overall cost minimum meanwhile meet the tenants SLA requirements.

6. Conclusions and Future Work

Aim at efficiently placing the multi-tenant data on multiple nodes in cloud, this paper proposed optimization adjustment strategy and algorithm. This strategy sufficiently consider the characteristics of the multi-tenant data and the load of the nodes in order to find the optimal placement, where meet the tenants SLA requirements meanwhile reduce the maintenance cost for service providers.

Another feature of the SaaS model is that allow tenants to customize application on-demand when the system is running. If the customized model is changed, how to place the new generated table objects and the corresponding data in a better way is the next issue to be explored in depth.

Acknowledgements

This work is supported by National Natural Science Foundation of China under Grant No.61272241,No.61303085; Natural Science Foundation of Shandong Province of China under Grant No.ZR2013 FQ014; Science and Technology Development Plan Project of Shandong Province No.2012GGX10134; Independent Innovation Foundation of

Shandong University under Grant No.2012TS075, No.2012TS074; Shandong Province Independent Innovation Major Special Project No.2013CXC30201.

References

[1] C. D. Weissman, "The design of the force.com multitenant internet application development platform", Proceedings of the International Conference on Management of Data and 28th Symposium on Principles of Database Systems, **(2009)**, pp. 889-896, Rhode Island, USA.

[2] L. Wu, S. K. Garg and R. Buyya, "SLA-based resource allocation for software as a service provider in cloud computing environments", Proceedings of 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, **(2011)**, pp. 195-204, Newport Beach, United states.

[3] S. Aulbach, D. Jacobs, A. Kemper, *et al.*, "A comparison of Flexible schemas for software as a service", Proceedings of the ACM SIGMOD conference, **(2009)**, pp. 881-888, New York, United states.

[4] W. Lehner and K. –U. Sattler, "Web-scale Data Management for the Cloud", Germany: Springer New York Heidelberg Dordrecht London, **(2013)**.

[5] S. Chemawat and H. Gobioff, "The Google file system", Proceedings of the 19th ACM Symposium on Operating Systems Principles, **(2003)**, pp. 19-22, New York, United states.

[6] G. De Cadia and D. Hastorun, "Dynamo: Amazon's highly available key-value store", Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles, **(2007)**, pp. 14-17, New York, United States.

[7] Shvachko, Konstantin and Kuang, *et al.*, "The Hadoop distributed file system", Proceedings of the 26th symposium on Mass Storage Systems and Technolgoies, **(2010)**, Lake Tahoe, United states.

[8] C. Curino, E. Jones, Y. Zhang and S Madden, "Schism: a workload-driven approach to database replication and partitioning", Proceedings of the VLDB Endowment, **(2010)**, pp. 48-57, New York, United States.

[9] Das, Sudipto, Agrawal and Dlvyakant, *et al.*, "Elas TraS: An elastic, scalable, and self managing transactional database for the cloud", ACM Trans Database Syst., vol. 38, no. 1, **(2013)**, pp. 5.

[10] X. Tang and J. Xu, "Qos-Aware Replica Placement for Content Distribution", IEEE Transactions on Parallel and Distributed Systems, vol. 16, no. 10, **(2005)**, pp. 921-932.

[11] M. Rashedur, B. Ken and A. Reda, "Replica placement in data grid: considering utility and risk", Proceedings of International Conference on Information Technology: Coding and Computing, **(2005)**, pp. 354-359, Las Vegas, United states.

[12] M. Rashedur, B. Ken and A. Reda, "Replica placement strategies in data grid", Journal of Grid Computing, vol. 6, no. 1, **(2008)**, pp. 103-123.

[13] W. Na, S. Zhang and L. Kong, "Dynamic adaptive model of the multi-tenant data replication based on queuing theory", Proceedings of 2nd International Conference on Computer Science and Network Technology, **(2012,)** pp. 1755-1759, Changchun, China.

[14] Y. Dong and Q. Li, "A deep web crawling approach based on query harvest model", Journal of computational information systems, vol. 8, no. 3, **(2012)**, pp. 973-981.

[15] X.–N. Li, Q.–Z. Li and L.–J. Kong, "Research on multi-tenant data partition mechanism for SaaS application based on shared schema', Tongxin Xuebao/Journal on Communications, vol. 33, (SUPPL 1), **(2012)**, pp. 110-120.

[16] C. Amza, A. L. Cox and Z. Waenepoel, "Conflict-aware scheduling for dynamic content application", Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems, **(2003)**, pp. 328-347, Seattle, WA, USA.

[17] X. Li, Q. Li and L. Kong, "Research on multi-tenant data placement strategy for SaaS application based on SLA-COST", Journal of computational information systems, vol. 10, no. 17, **(2014)**.

Authors

Li Xiaona, she born in 1980, she is a Ph.D. candidate at School of computer Science and Technology, Shandong University. Her main research interests include database, cloud computing and multi-tenant data management.

Li Qingzhong, he born in 1965, Ph.D., professor, Ph.D. supervisor. His research interests include large-scale network data management and Web data integration.

Zhu Weiyi, he born in 1971, senior engineer. He mainly engages in power marketing and information technology work.

Li Hui, she born in 1967, Ph.D., associate professor. Her main research interests include software and data engineering.