2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing

A Crisis Mitigation Policy for SLA Profit Maximization in Multitenant Database

Zhenkun Wang
School of Computer Science and Technology
Shandong University
Jinan, China
wzk@mail.sdu.edu.cn

Yongqing Zheng
School of Computer Science and Technology
Shandong University
Jinan, China
yqz@sdu.edu.cn

Lanju Kong
School of Computer Science and Technology
Shandong University
Jinan, China
klj@sdu.edu.cn

Zhiyong Chen
School of Computer Science and Technology
Shandong University
Jinan, China
chzy@sdu.edu.cn

***Abstract*—SLAs (Service Level Agreement) are agreements negotiated by the multitenant database service providers and their tenants in order to measure the quality of database service. They rule the fee paid to the service providers if the SLAs are satisfied, and the penalty by the service providers if their database service violated the SLA constraint. In fact, resources are shared among different tenants in a cloud multitenant environment, and tenants usually have changeable workload pressure over time. To maximize the profit, it is important for database service providers to intelligently managing the usage of resources. This paper focuses on managing tenants' workloads for fixed resources of a distributed database. The system uses historical operation data of every tenant to learn a model that describes the resource consumption for each tenant. Based on the learned model the crises detecting module periodically collects the free resources state of every node and judge if it has crises. If crises are detected, the crises mitigating model plays its role by leveraging a revised sliding window algorithm to find a rational target node, the following workload of the crisis tenant will be dispatched to the target. Extensive experimental results demonstrate the effectiveness and efficiency of our crises mitigation policy.**

***Keywords*—multitenant; SLA; crises mitigation**

I. Introduction

SaaS (Software as a Service) is the third form of the Cloud Computing, particularly, database service provident is critical to efficient work of SaaS application. to achieve economies of scale, an important practice is to use each physical server to host multiple tenants. With the rapidly increasing scale of the data, the service providers risk of more dynamic workload pressure from queries and transactions which are sent by application servers. Usually, to measure the SLA constrain, the providers and tenants use SLA - an agreement signed by tenant and service provider - to describe the effectiveness of multitenant database service [1]. It is essential to ensure the rational usage of cluster's resources (e.g., CPU, Mem, and I/O, we don't take I/O bandwidth into consideration for the reason of connection pooling in LAN). In fact, the problem requires both resources monitoring and workload managing. (1) Resources monitoring. To make sure all tenants run well, the database server nodes should monitoring their resources usage, once one's free resources is not enough to meet the need of running any tenant on it, the node should report the crisis and take right action in time. (2) Workload management. The aim of managing workload is to scheduling requests for a fixed amount of resources. Different tenants have different workload pressure, once a node detects crises, we should find a suitable target to dispatch the following workload. Moreover, the schedule policy not only can mitigate crises of source node, but also can not bring new crises to the target.

Taking the problems and challenges into consideration, a research presented Delphi to mitigate crises using hill-climbing algorithm [3]; while another study designed Smart-SLA, addressing the issue of how to intelligently manage the resources among different tenants to maximize its own profits [4]. However, we think their polices have several weak points. First, they focused on system level metrics such as database throughput, average query response time, and the granularity they used is too rough to leverage the resources efficiently. Second, they neglected the significance of anticipating future crises of target node. The cloud model has to operate in a fast changing environment to provide services to unpredictably diverse tenants. we should develop a robust policy that takes into free resources state and resources consumption rate.

In order to maximize the profit, this paper uses a revised sliding window algorithm to dispatch analysis query workload. Every node of the distributed database learns the resources usage of each tenant of it and periodically monitors its free resource state to judge if any crises occurred. To make good use of the whole resources, we use a revised sliding window algorithm to dispatch the following workload if crises are detected. Compared with other methods we just migrate the following requests to nodes who have the same tenant data. The major contributions are as following:

- A self-managing peer-to-peer multitenant DBMS architecture. We design and implement a distributed database architecture, which eliminates the bottleneck of master node. Requests from a application server

978-1-5090-0154-5/15 $31.00 © 2015 IEEE
DOI 10.1109/CIT/IUCC/DASC/PICOM.2015.121

will be sent to nearest database server node, and every database server node of the system can execute requests or dispatch requests to other nodes without directions of a master.

- Crisis migration mechanism. This paper uses sliding window algorithm to help find targets when nodes detect crises. The principle not only considers the status of free resources of target nodes, but also thinks about their resource consumption rate in case of causing crises in target nodes.

The rest of the paper are organized as follows. Section 2 surveys the related work. In Section 3, we provide background on database multitenancy and overview the database architecture. Sections 4 presents the detailed implementation of our schedule policy and explains how it works. We present the experiment results in Section 5 and section 6 concludes the paper.

II. Related Work

In this section we survey prior works in maximizing SLA profit. This includes database multitenancy, consolidation, resource orchestration, performance modeling, and crises mitigation. We mainly talk about workload management.

Workload management aim at eliminating the problem of wasting resources caused by heterogeneous requests, traditional methods open opportunities of underprovisioning and overprovisioning [6]. Overprovisioning typically occurs if resources are allocated for a peak load and results in a waste of resources and money. Underprovisioning means that not enough resources are provided to satisfy the performance requirements of the tenants [1]. Recent efforts tried to allocate resources by leveraging machine learning techniques. For example, one way to tackle the problem was using regression analysis, [4] learned a regression model to predict performance for different CPU, replicas and query rate situation, the performance is expressed in average SLA penalty costs for a simply step function. Ganapathier proposed another approach based on correlation analysis [5], it firstly used KCCA (Kernel Canonical Correlation Analysis) to project the query plan features and machine performance features onto dimensions of maximal correlation across the data sets, then the performance of a new query will be reversed by mapping its query features to the correlation space. To minimize the costs and penalties possible approaches range from rather simple rule-based strategies to complex decision models, likes Xeround[1] and Scalebase[2], they allowed to allocate resources automatically. Data is organized in virtual partitions which are replicated to different data nodes, if predefined thresholds for data volume or throughout are exceeded the system automatically scales out by allocating data nodes [1]. However, these basic methods allow to allocate resources automatically, they do not support directly the provider's goal of meeting SLAs, this requires to optimize the resources allocations among all tenants. In [4] SmartSLA used a grid-based search method for the optimal configuration, it defined a total SLA cost function using system performance model and the SLA penalty cost function, and learned each factor's weight in representing different customer classes. However, they focused on the resources usage predicting, and resources re-allocating, it is slow and costly in a realtime distributed database. Different tenants have different workloads, it is effectively to dispatching requests of heavy nodes to those free nodes other than re-allocating resources.

[1]http://xeround.com
[2]http://www.scalebase.com

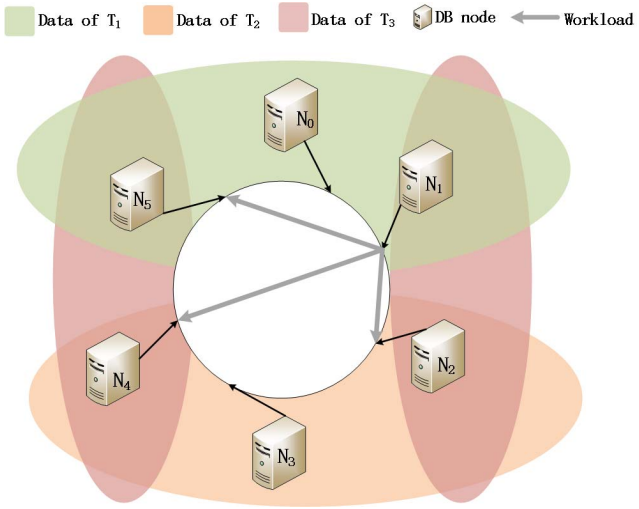


Fig. 1. Overview of database architecture

Another topic related to the SLA profit maximization is making requests execution planning. The goal of making execution sequence for requests is to finish requests as many as possible. Peha et al. [2] proposed CBS, a cost-based scheduling that schedules queries according to their expected cost. Chun et al. Besides, to achieve the goal of making execution sequence to finish more requests several scheduling techniques such as static priorization and goal-oriented approaches are available. Static priorization approaches were used by some commercial DBMS, e.g., IBM DB2 with Query Patroller or Oracle's Resource Manager, these components admitted requests and determined if they should be blocked, queued or run immediately based on user-given thresholds. Popovici and Wilkes [8] used simulation to develop schedule policies in the uncertain resource environment.

III. Database Multitenancy and Overview of Database Architecture

In this section, we first introduce the background of database multitenancy, describing how our data is organized in database. Then we provide an overview of our database architecture, including several models used when making the schedule.

A. Database Multitenancy

In SaaS model, different service quality is corresponded with the level of tenants and the fees paid by tenants. Data of one tenant is not so much compared with the overall tenants, so it is of big advantage to manage data in tenant fashion. In order to implement multitenancy, most hosted services have studied lots in designing multitenant database. For instance, Frederick Chong identified three distinct approaches for creating data architectures, such as separating

database, sharing database but separating schema and sharing database and schema architecture [9, 10]. Stefan Aulbach compared the three approaches, and proved that numerous tables can intensify the competition in node resources [11], which increased the risk of degrading of the performance of the node or even crashing it. Therefore, nowadays sharing table attracts more and more attention when maintaining data for several tenants. Every request submitted by application server were marked with tenant property. In response to the solution, Force.com's optimized metadata-driven architecture for multi-tenant applications, however metadata node had a latent visit bottleneck. We eliminated these limitations by developing the system with peer-to-peer (P2P) architecture, then we focus on maximizing the whole SLA profit by mitigating crises.

B. Overview of the Models

We provide an outlook of the database architecture in Figure 1. The prototype implementation consists of some DBMS nodes, each runs a DBMS engine and maintains one or more tenants' data, to that they can execute requests from application servers. The nodes can also dispatch requests to other nodes if crises are detected. Let N be the set of nodes, $\{T|T_j \in T\}$ be the set of tenants, and $\{Q|q_k \in Q\}$ be a collection of successive requests, we then introduce some models in explaining the crisis mitigation policy.

1) Node Model.: We introduce node model $\{N|N_i \in N\}$ to describe behaviors of every node. In practice, it has three functions: execute requests, detect crises and mitigate crises. Every node maintains data of one or more tenants, so they are responsible for providing database service to the relevant tenants by executing their requests. Requests from different have different resource consumption, we introduce the resource vector to express the resource state of a node and the resource consumption of executing a tenant's requests.

Definition 1 (Resource Vector): The resource vector $V = [CPU, Mem, I/O]$ is the resource state of CPU, Mem and I/O. Respectively, V_i expresses the free resource state of N_i and $V_{i,j}$ expresses the resources N_i needs to executing requests workloads about tenant T_j.

So, once the free resource of a node can not meet the need of executing its relevant workload, we say a crisis was detected.

Definition 2 (Crisis): For each dimension $D \in \{CPU, Mem, I/O\}$, a crisis is detected to N_i, if $V_{i,j}[D] >= V_i[D] \times \eta$, ($\eta$ is a coefficient of relaxation generated by experiments).

Suppose one of the replicas of T_j is placed on N_i, so N_i can handle requests workloads from T_j. A crisis will be detected if the resources demand for executing requests workloads of T_j exceeds the supply ability of N_i, then we should look for mitigate the workloads.

2) Tenant Model.: One tenant's data is replicated to several nodes, so we use tenant model $T_j = \{N_k|\forall N_k \rightarrow T_j\}$ to express a set of nodes who have the same data replica of T_j, which means all of them can execute requests about T_j, and if they detect crises, they can dispatch workload to others in the same tenant model.

Definition 3 (Collaborative Node): Node models in a tenant model maintain the same tenant data replica and can provide the same database service, we call them collaborative node for each other.

Review the example of Figure 1, tenant models $T_1 = \{N_0, N_1, N_5\}$. N_0, N_1 and N_5 are collaborative nodes in T_1. tenant model main has two functions. Firstly, it evaluate the economic results of executing requests. Requests from a tenant will be executed by nodes in the tenant model, they share the same SLA profit model, after executing the requests, the tenant model assigns the profit to the node, e.g. T_1 is responsible for assess the profit of executing requests from T_1. For another, tenant model records how the tenant data distributed in the distributed database system, so if a node detects performance crises, the node model can present dispatch candidates which we define as collaborative nodes.

TABLE I. PROFILE IN N_1

	N_0	N_1	N_2	N_3	N_4	N_5
T_1	1	1	0	0	0	1
T_3	0	1	1	0	0	0

Every node model likes N_1 is configured with a profile as show in Table 1 in order to distinguish it collaborative nodes in different node model, value 1 means collaborative relation and value 0 doesn't, e.g., $profile[T_1, N_0] = 1$ means if N_1 detects crises when executing requests of T_1, N_1 can dispatch the coming requests of T_1 to N_0. The tenant model is the collection of all the collaborative nodes of a tenant, when dispatch requests among a tenant, we adopt a sliding window algorithm, which we will describe later.

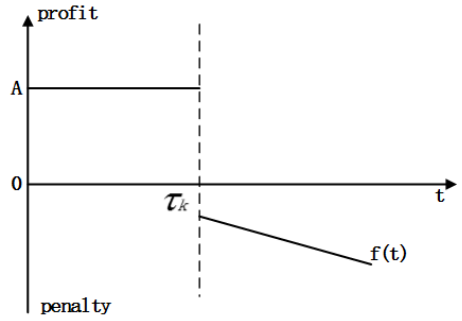


Fig. 2. SLA Profit Model

3) SLA Profit Model.: In multitenant database, we assume there is an associated SLA profit model for each tenant, requests of one tenant enjoy the same profit model whichever node model they are executed. Different studies have explored many profit modes in various shapes, we believe that a segmented function as shown in Figure 2 is a more commonly choice used in the real-world. The figure denotes that if the response time t of request q_k is shorter than τ_j, the service provider obtains a revenue A_j, otherwise, pays a penalty back to the tenant according to the final response time. In addition, what has not illustrated is if a request rejected up-front, the service provider has to pay a constant penalty according to its tenant. The profit formulation is defined as:

$$profit(q_k) = \begin{cases} A_j & \text{if } t < \tau_j \\ f(t) & \text{if } t >= \tau_i \end{cases} \quad (1)$$

Once a request finished at a node model, the tenant model will report its economic result to the service provider by using Equation (1).

IV. The Crises Mitigation Policy

Our P2P distributed database architecture designed to have no single point of failure, nodes of it have equal role of providing database service. Every node model can execute requests or dispatch them to other node models, this paper focuses on leveraging cluster's resources by determining which node model to execute requests. Firstly, requests from application server will be submit to the nearest node which has its tenant data, then we use crises detection models of every database server node to periodically check if its free resources are enough to supply tenants on it. Once crises are detected, target nodes will be chosen to help allay its load pressure.

A. Training phrase.

A node model can execute requests of different tenants, before providing the database service, we executed request of every tenant T_j on idle node model N_i to learn its service ability. We send requests to the node simulate their normal frequency, and record a vector $V_{i,j}$ of the CPU, MEM and I/O consumption when some requests violated their SLA constrain, we regard it as the resources consumption boundary, we also record the average rate of resources consumption of this process as a empirical threshold θ.

B. Crises Detection.

This process responsible for checking if it has abundant resource to provide database service for every tenant, otherwise, we say crises occurred. At first, application server submits analysis queries to database, and they will be routed to the nearest node model which maintains their tenants' data. Then, the node model periodically statistics its free resource status using the OS tools, and we compared it with every tenant resources consumption boundary we learned in training phrase. If the current resource state is sufficient to execute every tenant's request workload, requests will be sent to the queue of node model. Otherwise, the current resource state can not meet the need of executing some tenant's requests workload, therefore, this node model should dispatch the following requests of this tenant to other node models who have its tenant data. For example, After the database servers running, N_i check its free resources state V_i periodically, once some dimension of V_i is less than $V_{i,j}$, we conclude a crisis occurred, the crises mitigation process will play its role. To make the process more flexible, we usually add a slack parameter η here, its value is calculated according to the tenant's load pressure.

C. Crises Mitigation.

When crises are detected to a node model, it uses sliding window here rather than using the last resource status, providing a more confident view about shifts in behavior. We maintain a per-node sliding window of last two resource vectors V for every candidate (collaborative node), one records the latest history resource status, another shows the current resource status, each V is marked with timestamp. We use

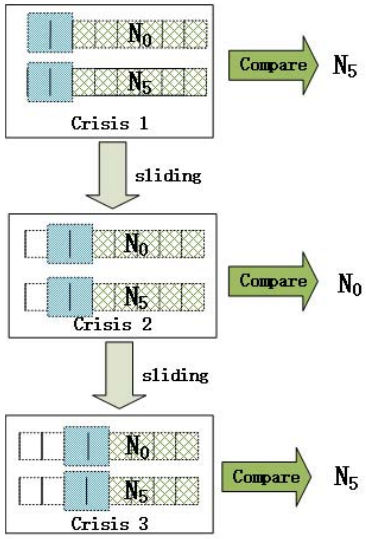
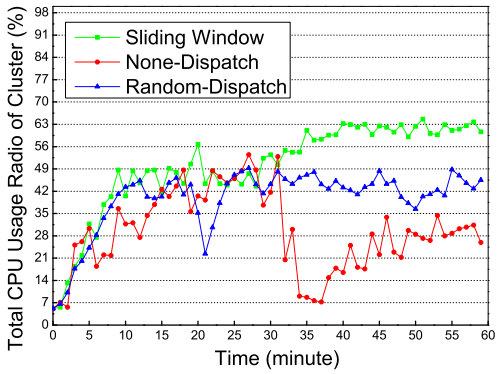


Fig. 3. Overview of Sliding Window Algrithm

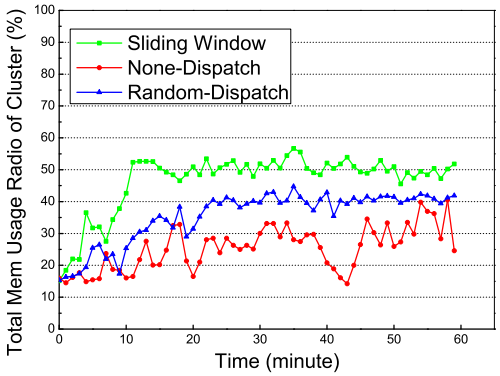
the two resource vector and respective timestamp to calculate resource consumption rate of candidates, and compare the resources consumption rate with relevant threshold θ we learn in training phrase and exclude those who have the high rate. Then we calculate positive distance of V_i and $V_{i,j}$ as the candidate node's resource sufficient degree and choose one who have the most abundant resource as the target node. After choosing, each candidate's window slides forwards. Take Figure 3 as example, assume N_1 detects crises when executing requests of T_1. We find the following requests can be dispatched to N_0 or N_5 by scanning the profile table of N_1. N_1 also records resource status V_i, and then fetches new resource status vector V_i' to fill the second window. In short, We formulate the dispatching process as:

$$
\begin{aligned}
& target = \arg\max_{i=2,3}(\parallel V_i' - V_{i,j} \parallel) \\
s.t. \quad & \frac{V_i' - V_i}{T - T'} < \bar{\theta}, \quad (\bar{\theta}\ is\ an\ empirical\ vector) \\
& V_{i,j}[D] < V_i'[D] \times \eta,\ D \in \{CPU, MEM, I/O\}
\end{aligned}
\tag{2}
$$

The goal of Equation 2 is to find a target node who has the most sufficient resources to dispatch requests of T_j among all the collaborative nodes of N_1. As described above, the resources sufficient degree measurement is defined as the distance of resource vector V_i' and $V_{i,j}$, it is obviously that the larger this distance is, the more resources will left after dispatching requests about T_j to this candidate. Besides, Equation 2 shows we have two constraint conditions. The first one restrains our scheduling policy will not cause crises to candidates. V_i' represents the current free resources state of candidate N_i, while V_i represents the latest resources state recorded by N_1, they are respectively marked with timestamp S' and S, we calculate the gradient of this two vector, which means resources consumption tendency of the candidate. We compare it with the threshold value θ, which is an empirical vector gotten in training phrase, it evaluate normal resources consumption rate of the candidate. The second constrain con-



(a) CPU Consuption



(b) Mem Consuption

Fig. 4. The Average Usage of Resource

dition indicates that the CPU, Mem and I/O resources are all enough to handle requests dispatched by N_1. we also use the same slack parameter η here to make our schedule policy more flexible. After choosing process, V_i' takes place of V_i, windows of every candidate slide forward to wait for next crisis.

V. Experiment

In this section, we deployed and evaluated our system on a cluster of 5 nodes dedicated to database processed. For the test bed, the database server run MySQL v5.0 with InnoDB storage engine on CentOS generated by openstack, each is configured with an Intel Xeon E312xx processor, 4G memory, and 20GB Disk. The following section describes data set and benchmark we used for workload generation, and a detailed evaluation of the schedule policy.

A. Benchmark Description

Existing benchmarks provide little support for evaluating the effects of multiple tenants database, such as the TPC suite, focusing on testing the performance and limits of a single high performance DBMS dedicated either for transaction processing (TPC-C) or for data analysis (TPC-H) [3]. To adapt to our multiple tenant environment, we changed TPC-C, systematically generating workloads and data set for 20 tenants. Tenant's data size varies from 100MB to 2GB.

Our benchmark is centered around the principal activities (transactions) of an order-entry environment. The workloads generator maintains a configuration file describing how tenant data distributed in the system, it generated tenant workloads with different rate to simulate heterogeneous tenant pressure. In the training phase, we executed requests in an idle node with maximum possible resources, then we calculated the average response time as the SLA standard, besides, we experimented the resource consumption of every execute unit. During the working phase of the system, the arrival rate of requests followed a Poisson distribution with the rate set in each test, for the reason that it is widely used to model independent arrival requests to a website [4]. Every node model gathered CPU and Mem status using Linux tool *top* and gathered the I/O usage with the tool *iostat* per minute. If crises were detected, node model sent requests to others by TCP pipelines, otherwise, the node model used relative execute unit to execute requests. After finishing a request, the node model reported its profit. We measured the total resource consumption and the final profit under 5 random parameter workload setting.

B. Experiment Results

We compared our schedule policy's performance with the previous works, to view the results, we plot the distribution of the outcoming with average statistical result.

We firstly proved our system can make good use of the resource by mitigating crises. Take the example of CPU and Mem consumption, Figure 4 shows their utilization ratio of the whole cluster, the data is summarize with several experiments. In each subfigure, the x-axis represents the system running time, during which we kept sending requests to nodes with different rate, and the y-axis represents the resource utilization ratio, it indicates the load-balancing capacity of the system. From the plot of Figure 4 (a) and (b) we can see clear distinguish of resource usage ratio between sliding window, random-dispatch and none-dispatch schedule that doesn't use any schedule policy. With the workload increasing, the resource consumption of none-dispatch fashion is substantially worse than sliding window, the experiment result shows sliding window improves 30% to 35% in CPU usage, and 15% to 20% in Mem usage compared with none-control method. It mainly due to the reason that heterogeneous workloads will cause wasting of resource, but sliding window balances workloads to free nodes if some nodes detect crises. What's more, sliding window improves 10% to 15% in CPU usage, and 5% to 10% in Mem usage compared with none-dispatch method, it is because when dispatch requests sliding window method take the future crises risk of target into consideration.

We then compared the final profit gained by different methods in out experiment. Figure 5 demonstrates the profit gained by sliding window, random-dispatch and a none-control schedule. As can be seen, (1) At first, the workload isn't heavy, all the three methods gained the similar profit because

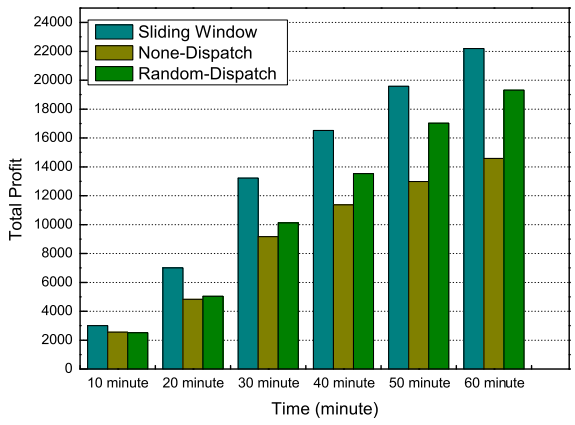


Fig. 5. The Total SLA Profit

these systems are underloaded. (2) With the workload pressure increase, some nodes have performance crises, the outcoming is different. The profit increasing rate of sliding window is the highest, while none-control fashion becomes slow. The final statistic shows sliding window achieved 35% to 40 % more profit than none-control method and achieved 10% to 15% more profit compared with random-dispatch method, for the reason we have proved above, sliding window used the total resources effectively when facing the same workloads. . The result shows our schedule policy makes more reasonable schedule policy. (3) We carefully analyze this phenomenon and conclude that our schedule policy does well in maximizing the profit. In summary, the experiment results in this section demonstrate that our crises mitigation policy, using a sliding window algorithm, can make good use of resources of the cluster, and the final profit of service provider is improved.

C. Conclusion

Our crises mitigation policy mainly focusing on maximize profit in multi-tenant database. To deal with unpredictable and dynamic workloads, the system faces multiple challenges such as efficient resource utilization and suitable execution plan. This paper periodically monitor the resource consumption, avoiding over- or under- provisioning. Once a node is deficient in executing requests, we choose vacant target to mitigate is load pressure in time. Our experiments, using a variety of tenant types and workloads, demonstrated that our method is effective and efficient. Further studies will pay more attention on complicated SLA profit models and request execution making.

Acknowledgments.: This work is funded by National Natural Science Foundation of China under Grant No.61272241,No.61303085; Natural Science Foundation of Shandong Province of China under Grant Grant No.ZR2013 FQ014; Science and Technology Development Plan Project of Shandong Provience No.2014 GGX101047, No.2014GGX101019; the Fundamental Reasearch Funds of Shangdong University No.2014JC05.

REFERENCES

[1] Lehner W, Sattler K U, Sattler K U, Web-scale data management for the cloud, 2013, pp.137-145.

[2] Peha J M, Tobagi F A. Cost-based scheduling and dropping algorithms to support integrated services[J]. Communications IEEE Transactions on, 1996, 44(2):192 - 202.

[3] Elmore A J, Das S, Pucher A, et al. Characterizing tenant behavior for placement and crisis mitigation in multitenant dbmss[C]//Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM, 2013: 517-528.

[4] Xiong P, Chi Y, Zhu S, et al. Intelligent management of virtualized resources for database systems in cloud environment[C]//Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE, 2011: 87-98.

[5] Ganapathi A, Kuno H, Dayal U, et al. Predicting multiple metrics for queries: Better decisions enabled by machine learning[C]//Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on. IEEE, 2009: 592-603.

[6] Armbrust M, Fox O, Griffith R, et al. M.: Above the clouds: A Berkeley view of cloud computing[J]. Eecs Department University of California Berkeley, 2009, 53(4):50-58.

[7] Srikantaiah S, Kansal A, Zhao F. Energy aware consolidation for cloud computing[C]//Proceedings of the 2008 conference on Power aware computing and systems. 2008, 10.

[8] Popovici F I, Wilkes J. Profitable services in an uncertain world[C]//Proceedings of the 2005 ACM/IEEE conference on Supercomputing. IEEE Computer Society, 2005: 36.

[9] Azeez A, Perera S, Gamage D, et al. Multi-tenant SOA middleware for cloud computing[C]//Cloud computing (cloud), 2010 ieee 3rd international conference on. IEEE, 2010: 458-465.

[10] Jacobs D, Aulbach S. Ruminations on Multi-Tenant Databases[C]//BTW. 2007, 103: 514-521.

[11] Aulbach S, Jacobs D, Kemper A, et al. A comparison of flexible schemas for software as a service[C]//Proceedings of the 2009 ACM SIGMOD International Conference on Management of data. ACM, 2009: 881-888.

[12] Elnikety S, Nahum E, Tracey J, et al. A method for transparent admission control and request scheduling in e-commerce web sites[C]//Proceedings of the 13th international conference on World Wide Web. ACM, 2004: 276-286.

[13] Tozer S, Brecht T, Aboulnaga A. Q-Cop: Avoiding bad query mixes to minimize client timeouts under heavy loads[C]//Data Engineering (ICDE), 2010 IEEE 26th International Conference on. IEEE, 2010: 397-408.

[14] Chase J S, Anderson D C, Thakar P N, et al. Managing energy and server resources in hosting centers[C]//ACM SIGOPS Operating Systems Review. ACM, 2001, 35(5): 103-116.

[15] Ganapathi A, Kuno H, Dayal U, et al. Predicting multiple metrics for queries: Better decisions enabled by machine learning[C]//Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on. IEEE, 2009: 592-603.

[16] Hall M, Frank E, Holmes G, et al. The WEKA data mining software: an update[J]. Department of Computer Science Government Degree College Macherla He, 2009, 11(1):10-18.