An Efficient Resource Sharing Technique for Multi-Tenant Databases

Pallavi G B  
BMSCE  
pallavi.cse@bmsce.ac.in

Dr. P Jayarekha,  
BMSCE  
jayarekha.ise@bmsce.ac.in

Abstract— In recent times, multitenant database systems in Software as a Service (SaaS) paradigm have gained a lot of attention in academics, research and business arena. The reason being, multitenant database systems are advantageous for both SaaS providers and customers (organizations/companies referred as tenants) who use SaaS applications. By using a single database system to store data of different tenants the providers gain operational and economic benefits. The tenants in turn get access to shared software and database at an affordable cost. However, this scenario leads to several challenges. One of them being, providing tenants dynamic access to computing resources for accessing the database, nonetheless maintaining isolation and service level agreement requirements among different tenants. The presented work proposes a dynamic resource sharing technique for accessing multitenant databases addressing the aforementioned challenge.

Keywords— SLA, TPP-C, Transaction, Worker, Tenant

I. INTRODUCTION

Cloud computing is the latest technology being adopted by various organizations/companies around the globe in order to run their businesses. The kinds of services offered by this technology falls under three categories viz., Infrastructure as a service (IAAS), Platform as a service (PAAS) and Software as a service (SAAS). While IAAS providers offer various hardware computational needs, PAAS providers offer frameworks and programming languages required to develop software/applications and SAAS providers offer a full fledged ready to use application as a service. SAAS is an attractive offer for software companies as they can use various applications without the need to purchase and maintain them on their own infrastructure. Also, service provider achieves full economy of scale by hosting such SaaS application using a multitenant model where tenant refer to an organization/company.

Multitenancy is one of the key concerns in SaaS. It refers to a principle in software architecture, which is the ability to enable SaaS application to serve multiple tenants using a single service instance. Multitenancy invariably occurs at the database layer of the SaaS application[6] i.e., multiple tenants are consolidated on to the data tier resource while at the same time isolating them among one another as if they were running on physically segregated resources. A multi-tenant DBMS may isolate tenants in a shared database system - by dedicated databases (shared machine approach), - by shared databases and separate tables or schemas (shared process approach) or - by an association of each dataset in a shared table with the appropriate tenant (shared table approach) [12]. Identification of records for a particular tenant is done based on a unique tenant id [5].

However, one of the major challenges posed by multitenant applications are effective utilization of resources [13]. Each tenant is statically assigned an equal amount of resource. This may lead to inefficient utilization of resources when there are fewer or more loads of queries on databases than expected and is therefore undesirable in a multitenant system. Moreover, service providers must also meet the criteria of Service Level Agreement (SLA) [1] of tenants. In a traditional single tenant database system, the two aspects of performance analysis are server hardware for operating the database and workload. However with multi-tenancy, since different tenant access the same database at different rates, workloads and complexities, vendors need to keep a check on performance attainment of each tenant. As a result, optimal resource utilization becomes a key requirement for the service providers.

In this paper we have tried to attain a solution for the problem of inefficient utilization of resources by proposing a dynamic resource sharing technique for multitenant databases.

The paper is organized as follows: In section II, a study of related work is been carried out. A simple multitenant database architecture and related algorithms and flowcharts are discussed in section III. Experimental set up and results are discussed in section IV. Finally section V concludes the paper.

II. RELATED WORK

In [3] S. Walraven et al has presented an adaptive middleware design that efficiently enforces different and competing performance constraints in multi-tenant SaaS applications. The middleware can manage a combination of performance constraints in terms of latency, throughput and deadlines at a fine-grained level, and enables rapid response on changing circumstances, while preserving the resource usage efficiency of application-level multi-tenancy. In [2] Sudipto Das et al highlighted the problem of traditional CPU sharing approach for Database as a service (DAAS) scenario and have proposed an effective and efficient CPU sharing technique. They have focused on fine-grained reservation of CPU without static allocation. The work also supports on demand resource availability. However sharing of CPU reduces the system cost but at same time it reduces the system performance as well.

In [6], Stefan Aulbach and team discussed about various storage models used to implement multitenant databases of shared table. They in turn proposed a new technique called a chunk folding schema mapping technique. In this technique,
the logical source tables are vertically partitioned into chunks that are folded together into different physical multi-tenant tables and joined as needed. In [7], Vivek Narasayya et al. have proposed a reservation technique called SQLVM of key resources in a database system such as CPU, I/O, and memory. The authors claim that unlike a traditional VM, a SQLVM is much more lightweight as its only goal is to provide resource isolation across tenants. In [8], Ying Hua Zhou et al. have introduced a DB2MMT (massive multi-tenant database platform) high level architecture. The authors have addressed key technical challenges, including resource, tenant and offering management, monitor, scalability and security. They have compared the economics of DB2MMT and traditional solution with precise data showing acceptable performance. In [9], X. Zhou et al., have analyzed the migration method for business component and the essential problems that the business component migration faced with. To solve this problem they have introduced memory database as the implementation mechanism of distinguishing table instances for multi-tenant data isolation, a Data Relationship Engine to realize the relationship between the business components and multi-tenant database, and have presented a multi-tenant database development framework for business component migration in SaaS.

III. ARCHITECTURE OF MULTITENANT DATABASE SYSTEM

A. Modelling of Multitenant System

In a multitenant scenario, active tenants compete for resources in order to access the database. If one tenant blocks up the resources, the performance of all the other tenants may be restricted and a fair sharing of the resources may be compromised. The performance of tenants must not be affected by resource-intensive activities and volatile workloads of other tenants [11]. Moreover, the prime goal of providers is to accomplish low cost of operation, satisfying specific schemas/SLAs of each tenant. Consequently, there is a need to design and develop effective and dynamic resource sharing algorithms which can handle above mentioned issues.

An overall architecture of Multi-tenant database system is presented in Figure 1. SLA based requirement is considered for designing a multitenant system and maintaining the system QoS. Tenants request for the task execution or data base accessing. Tenant Manager collects the information of load, SLA requirement from tenant and schedules the tenant as per availability of the workers. Workers/Threads execute the task based on tenant workload which is composed of SLA requirement of tenant. Workload manager assigns a separate worker to each tenant where the workload may be composed of strict SLA requirement. DB-Connector is used for establishing the connection between database server and Tenant Manager. The type of database sharing approach used is the schema based multi-tenancy approach. A dynamic resource scheduler for assigning the job is introduced in the next section.

IV PROPOSED MODEL

A. System Model

Utilizing CPU resource efficiently is the most desired feature of any scheduling approach. However overloading CPU may induce latency for tenant and affects the SLA of tenant requirement. Failure in designing an efficient scheduler for task execution may affect the overall performance of

![Figure 1. Architecture of Database Multi-tenancy](image-url)
manager divides the available workers based on CPU, I/O and memory based processing.

A queuing model is presented for dynamic scheduler. The queuing model aid in balancing the load of worker and improve utilizing the idle instance. Dynamic scheduler works between tenant manager and global tenant manager. Scheduler takes the request task queue from tenant manager and information from the global tenant manager and schedules the task based on best compatible value for both global tenant manager and tenant task manager.

B. System Parameters

Let’s assume there are T tenants and N workers and M number of requested query. N workers are represented as \( w_1, w_2, \ldots, w_N \). M queries are represented as \( q_1, q_2, q_3, \ldots, q_M \). Each thread (worker) has its CPU utilization, I/O and memory utilization for processing a query. Each query contain the following parameters.

C. Scheduling Approach

First tenant manager accepts the query from tenant and divides its category based on requested query. Scheduler accepts submitted task list from tenant manager and start working with accessing the data of global tenant manager. Scheduler assigns each task to independent worker using dynamic scheduling approach. This approach is based on the workers sorting and type of query requested by the tenant. For each query weights are assigned, based on which high or low CPU utilization threads are allocated. If a query has less weight, then it is assigned a thread with less processing power and for higher weight queries a thread with high processing power is assigned. To maintain the load, the work can be assigned to idle threads as well which aid in reducing overload.

The global tenant manager collects the workers information (Memory usage, I/O waiting period and CPU utilization) from local worker manger and updates to the dynamic scheduler. Dynamic scheduler divides the queries into sub queues as shown in flow chart in Figure 3. It will assign each query from each queue to a different available worker. So each query from each queue is executed simultaneously. This aids in reducing the load and enhance system efficiency.

If the number of workers are more than the requested number of queries then based on requirement the scheduler will assign the query to the worker maintaining the load. However if requested number of queries are more than the available workers, then queries needs to be assigned in group. It makes one batch of queries from sub queries, queues it as \( q = \frac{M}{G} \), where \( G \) represents the number of queues created.

Remaining \( M - q \) queries will be considered in next group. If \( M - q > N \) then the process of grouping the queries is continued otherwise workers are assigned to queries on a regular basis. This process is repeated until execution of last query. In this approach each worker is assigned with one task and usages of CPU, memory and IO are all maintained. Tenant query execution is also faster yielding to high system performance and throughput. The proposed model aid in reducing latency for tenant’s workload execution by utilizing the idle instance and reducing computing cost. However it may induce revenue loss for cloud provider due to usage of idle instance and network I/O delay between tenant and cloud provider and is not considered within the scope of this study.
V. EXPERIMENTAL RESULT AND ANALYSIS

Windows 7 operating system with 3.2 Ghz quad core processors and 16 GB ram are used for the experimental analysis. For performance evaluation we are taking TPCC benchmark which simulate the application of real world [10] and MuTeBench multitentant database benchmark framework [11]. We configured the system with hibernate and used the H2 database for tenant database query execution.

For simulation study we considered the different transaction type such as new order, payment, status of order and delivery. There are 10 number of workers and here we are generating the latency for each work, total number of transaction, total successful transaction, rejected transaction and average time taken in processing of different type of transaction. Here processing time of transaction is computed for CPU utilization.

In Figure 4 average latency in processing the transaction request is presented for each worker starting from work 0, worker 1 up to worker 9. From figure we observed that the latency for each worker is very less. In Figure 5 TPP-C average transaction time for each type of transaction is presented. Transaction time is very less for each type of transaction. Type 2 transactions takes more time as it is a payment related transaction and security is concerned. Time of transaction is presented in ms and it is CPU time.
Figure 6 represents the status of transactions as total number of successful transactions, unexpected errors and rejected number of transactions. Number of successful transactions are 6766, unexpected errors caused at run time are 1478 and rejected number of transactions are 72747. Transactions are rejected due to wrong input parameters, transaction issues etc.

Figure 7 represents total number of requested transaction vs time. In 59.11 micro seconds of CPU time total number of requested transactions are 80991.

Figure 8 represents the TPC-C worker Start and End time. Each worker started at 1487569.9 sec and ends at 1487570.1. Total CPU time for each worker is 0.05911898 ms.

VI. CONCLUSION

Recently, sharing of database in cloud computing environment has become very popular. Low cost solutions are the necessity of end users for which multi-tenant database system are adopted by the cloud service providers. The multitenant database systems need to be improved in terms of latency, fairness and isolation. With this regard, we have presented a schema based multitenancy approach and designed a dynamic resource sharing scheduler. In this approach tenants requests are accepted by the tenat manager and based on query request tenant manager creates queue. Dynamic scheduler schedules the query to the worker depending upon the availability of the worker and system load. We observed the system performance in terms of worker latency, transaction type and time, number of generated transaction and overall successful transaction, rejected transaction and unexpected error at the time of transaction processing. Load is efficiently managed and makes system more stable. However, the system works only for schema based approach and not for shared machine or shared table approach of multitenancy. In future we implement the scheduling approach for creating more number of tenants maximizing the system throughput.

ACKNOWLEDGMENT

The authors would like to acknowledge and thank Technical Education Quality Improvement Program [TEQIP] , BMS College of Engineering.

REFERENCES

[1] R. Krebs, C. Momm, and S. Kounev, “Metrics and Techniques for Quantifying Performance Isolation in Cloud Environments,” Elsevier Science of Computer Programming Journal (SciCo), 2013.

[2] Sudipto Das, Vivek R. Narasayya, Manoj Syamala,” CPU Sharing Techniques for Performance Isolation in Multitenant Relational Databaseas a Service”. September 5th 2014, Hangzhou, China. Proceedings of the VLDB Endowment, Vol. 7, No. 1

[3] S. Walraven, W. D. Borger, B. Vanbrabant, B. Lagaisse, D. V. Landuyt and W. Joosen, "Adaptive Performance Isolation Middleware for Multi-tenant SaaS," 2015 IEEE/ACM 8th
[4] W. Lang, S. Shankar, J. M. Patel and A. Kalhan, "Towards Multi-Tenant Performance SLOs," in IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 6, pp. 1447-1463, June 2014.

[5] Li heng, Yang dan, and Zhang xiaohong. Survey on multi-tenant data architecture for saas. IJCSI International Journal of Computer Science Issues, Vol. 9, Issue 6, No 3, November 2012, 2012.

[6] Stefan Aulbach, Torsten Grust, Dean Jacobs, Alfons Kemper, and Jan Rittinger. Multi-tenant databases for software as a service: Schema-mapping techniques. In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD ’08, pages 1195–1206, New York, NY, USA, 2008. ACM.

[7] Vivek Narasayya, Sudipto Das, Manoj Syamala, Badrish Chandramouli, and Surajit Chaudhuri. Sqlvm: Performance isolation in multi-tenant relational database-as-a-service. In 6th Biennial Conference on Innovative Data Systems Research (CIDR ’13), 2013.

[8] Ying Hua Zhou, Qi Rong Wang, Zhi Hu Wang, and Ning Wang. Db2mmt: A massive multi-tenant database platform for cloud computing. In e-Business Engineering (ICEBE), 2011 IEEE 8th International Conference on, pages 335–340 Oct 2011.

[9] X. Zhou, D. Zhan, L. Nie, F. Meng and X. Xu, "Suitable Database Development Framework for Business Component Migration in SaaS Multi-tenant Model," 2013 International Conference on Service Sciences (ICSS), Shenzhen, 2013, pp. 90-95.

[10] JFreeChart, http://www.jfree.org/jfreechart/, retrieved: March 28, 2014.

[11] Andreas Gobel,"MuTeBench: Turning OLTP-Bench into a Multi-Tenancy Database Benchmark Framework".2014, The fifth International Conference on Cloud Computing, GRIDs and Virtualization, ISBN: 978-1-61208-338-4

[12] https://github.com/MuTeBench/MuTeBench/blob/master

[13] Cor-Paul Bezemer, Andy Zaidman.: Multi-Tenant SaaS ApplicationsMainte-nance Dream or Nightmare?”, Report TUD-SERG-2010-031, Delft University of Technology Software Engineering Research Group Technical Report Series (2010)