International Journal of Electrical and Computer Engineering (IJECE)
Vol. 10, No. 3, June 2020, pp. 3216~3226
ISSN: 2088-8708, DOI: 10.11591/ijece.v10i3.pp3216-3226

An efficient resource sharing technique for multi-tenant databases

Pallavi G. B.[1], P. Jayarekha[2]
[1]Department of Computer Science and Engineering, BMS College of Engineering, India
[2]Department of Information Science and Engineering, BMS College of Engineering, India

Article Info

Article history:

Received Mar 27, 2019
Revised Nov 9, 2019
Accepted Nov 23, 2019

Keywords:

Cloud computing
Mult-itenancy
Resource management
SLA
TPC-C

ABSTRACT

Multi-tenancy is a key component of Software as a Service (SaaS) paradigm. Multi-tenant software has gained a lot of attention in academics, research and business arena. They provide scalability and economic benefits for both cloud service providers and tenants by sharing same resources and infrastructure in isolation of shared databases, network and computing resources with Service level agreement (SLA) compliances. 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. 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. This work presents a model referred as Multi-Tenant Dynamic Resource Scheduling Model (MTDRSM) embracing a query classification and worker sorting technique enabling efficient and dynamic resource sharing among tenants. The experiments show significant performance improvement over existing model.

Corresponding Author:

Pallavi G. B.,
Department of Computer Science and Engineering,
BMS College of Engineering,
VTU, Bangalore, India.
Email: pallavi.cse@bmsce.ac.in

1. INTRODUCTION

Cloud computing currently is an emerging and most promising technology, on which varied research has been carried by various communities [1]. It has been adopted by various organization and IT industries to build and deploy custom made application in various fields as genetic science, healthcare and so on. Cloud technologies is driven by economies of scale by providing large scale distributed computing infrastructure in which resource such as computing power, storage, platform etc. and services are provided on demand through internet [2], The service offered by cloud technologies are broadly classified into three categories. They are 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 [3] referred to as Multi-tenant Data Management System (MTDBMS) where 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. Many organizations export their data to third party MTDBMS in order to provision data management services. A MTDBMS 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) [4]. 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 is effective utilization of resources [6]. 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) [7] of tenants.

There are several dire consequence for both tenant and service provider such as inefficiency in data centre and revenue, limited cloud applicability and unpredictable application performance [8]. However these issues are behind the scope of this paper. In state-of-art 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. This paper explores resource management architecture composed of architecture and scheduling strategies to address multitenancy issues, particularly sharing of resources among tenants in order to compute intensive queries and scalability for workflow execution. To provide scalability, the MT-DBMS should run on low cost commodity hardware and scale out to a many servers to provide service to large consumers.

Workflow scheduling is a process of identifying and managing the execution of certain task on a distributed network. It allocates certain amount of appropriate resource to a task and completes the task within user's defined deadline or objective time. Developing an efficient scheduling model will aid in improve the overall system performance. Scheduling distributed task is considered to be NP-hard problem [9], as a result no optimal solution is found within polynomial time. To achieve near optimal scheduling many heuristic scheduling has been presented. However, these techniques are not suitable for scheduling workflow in multi-tenant cloud computing environments. To address this issue, the authors of [10] presented an efficient workflow scheduling where a proof-of-concept experiment of real-world scientific workflow applications has been performed to demonstrate the scalability of the scheduling algorithm, which verifies the effectiveness of the proposed solution. However they did not consider the impact of resource failure and dynamic SLA requirement of Tenants. Moreover, efficient resource allocation and load balancing technique is required, because there is uncertainty in resource and load which changes over time. Request for resources changes over time and the resource itself undergo several changes (i.e. resource can join or leave a network).These dynamic uncertainties might lead to performance bottleneck.

This work presents a dynamic scheduling technique for Multi-Tenant SaaS cloud environment overcoming the above challenges. Firstly, architecture of the proposed Multi-Tenant Database System is presented. Secondly for dynamic scheduling, the query (load) and resource information is collected according to Memory, I/O and CPU. Thirdly the query and resource are divided into three queues according to Memory, I/O and CPU intensive. Lastly, the scheduler utilizes the overall resource available and schedule to resource with lighter loads. This aid in balancing the load and make full use of idle instances. The paper is organized as follows: In section 2, a study of related work is been carried out. A simple multitenant database architecture and related algorithms and flowcharts are discussed in section 3. Experimental set up and results are discussed in section 4. Finally, section 5 concludes the paper.

2. RESEARCH METHODOLODY

The issues pertaining to scheduling task on multiple workers has been widely studied in distributed, parallel, grid and cluster computing and in recent year the same kind of study is been carried out considering virtual workers on cloud environment. The techniques adopted by these models differ from characteristic of workload, resources, performance metric and scheduling in multiagent architecture [11]. All these methodologies are designed based on Heuristic Algorithm, Meta-Heuristic Algorithm, Scientific Workflows Execution, Deadline-aware Scheduling and Multi-tenant SaaS Applications, which is extensively researched in the presented work.

Heuristic Algorithm: Many existing approaches have considered heuristic methods for clustering, task duplication and scheduling. Few examples are: In [12] Jing-Chiou Liou et al, presented a task clustering algorithm with no duplication namely CASS-II. They compared their algorithm with DSC in terms of both speed and solution quality. In [13] R. Bajaj and D. P. Agrawal presented task duplication based scheduling mechanism for heterogeneous network (TANH). In [14], a Heterogeneous Earliest finish time (HEFT) scheduling technique for single work flow was presented by H. Topcuoglu, S. Hariri, and M. Y.Wu and in [15] H. M. Fard, et al., presented a multi-objective heuristic scheduling for grid and cloud environment. However these models are not suitable for multi-tenant cloud environment, due to unpredictable performance (throughput). Since some tenant may opt for best effort behavior [16] and some may prefer performance isolation.

Meta-Heuristic Algorithm: To minimize workflow execution cost in cloud environment, the authors in [17, 18] have adopted particle swarm optimization (PSO) based scheduling technique and in [19] an optimization of genetic algorithm (GA), Ant colony optimization (ACO) and PSO has been implemented. In [20] H. M. Fard et al., has implemented a dynamic scheduling and pricing model for single query for multi-cloud platform and has compared with traditional model multi-objective evolutionary algorithms, i.e., NSGA-II and SPEA2. These entire models are designed to optimize in grid environment and induce computing overhead. Hence these models are not suitable for large workflow application.

Scientific Workflows Execution: In [21] the authors have studied the performance and cost involved in computing in public cloud environment. They showed that amazon EC2 is not suitable for I/O intensive application (NASA HPC cluster) due lack of parallel heterogeneous computing platform. To improve system performance the authors of [22] presorted locality aware scheduling. However evaluation on dynamic real world workload is not carried out. Similarly D. Yuan et al, in [23] presented a data placement strategy in scientific cloud workflows by adopting k-mean clustering.

Deadline-aware Scheduling: The authors of [24] have studied dynamic resource allocation for adaptive application on cloud platform. They adopted Q-learning based learning model to meet the user define deadline for particular application requirement. A grid based scheduling model for deadline constraint weather forecasting system and a heuristic model to meet deadline for scientific application workflow has been presented in [25, 26] respectively. In [27] S. Abrishami et al. presented scheduling strategies for single workflow instance for IaaS cloud platform. However none of these models considered multi-tenant cloud environment.

Multi-tenant SaaS Applications: Many approaches have been presented for multi-tenant SaaS applications. A two-tier multitenant architecture has been presented in [28]. A model to determine optimal allocation policy and a resource allocation model for SaaS applications has been presented in [29, 30] respectively. In [31] S. Walraven et al. presented an adaptive middleware design for efficient multi-tenant SaaS applications. The authors in [32] 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 the same time it reduces the system performance as well. In [33] Vivek Narasayya et al. 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 [34], Ying Hua Zhou et al has introduced a DB2MMT (massive multi-tenant database platform) high level architecture. The author has addressed key technical challenges, including resource, tenant and offering management, monitoring, scalability and security. They have compared the economics of DB2MMT and traditional solution with precise data showing acceptable performance.

To conclude, extensive survey and the study of related work showcase that scheduling and load balancing plays an important role in improving the performance of multi-tenant cloud architecture. Many approaches adopt various heuristic, Meta heuristic, clustering and optimization techniques to classify user quires and resource classification. All these approaches are time consuming processes, induce computation overhead and are and may not be applicable for dynamic workflow provisioning. To overcome these challenges, we present an efficient scheduling technique for multi-Tenant cloud architecture that fully utilizes the system resources with SLA guarantee.

3. ARCHITECTURE OF MULTI-TENANT DATABASE SYSTEM

3.1. Modelling of multi-tenant system

An overall architecture of Multi-tenant database system is presented in Figure 1. The Tenant Manager maintains the service level agreement received from the tenants. These SLA based tenant requirement is considered for designing a multitenant system and maintaining the system QoS (Latency).

The other input to Tenant Manager is the tenant configuration file where tenant specific settings are established. Tenants request for the task execution or data base accessing. Tenant Manager checks the load and schedules the tenant as per availability of the workers based on SLA constraint of the corresponding tenant. Workers execute the task. 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 scheduling system for assigning jobs is introduced in the next section.

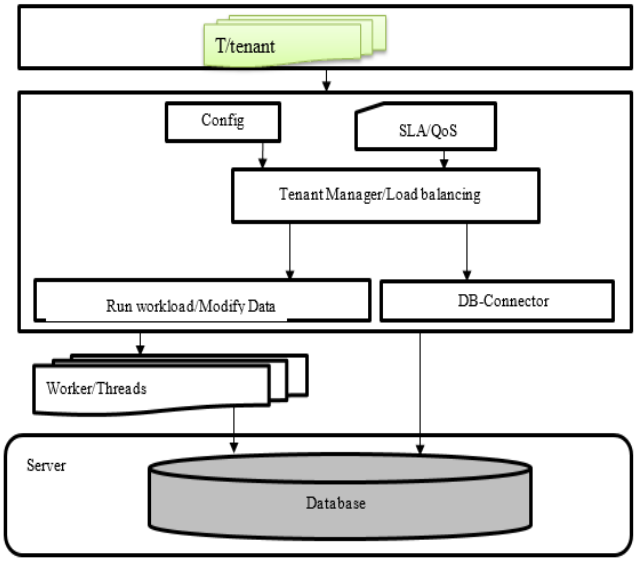

Figure 1. Architecture of multi-tenant database system

3.2. Multi-tenant dynamic resource scheduling model

The objective of proposed dynamic resource scheduling system is that the Memory, I/O and CPU usage do not conflict each other in order to improve scheduling performance and utilizing resource efficiently. Let's consider a case where some query execution requires less I/O or Memory resources, but it might require higher CPU resource to complete the task. This scenario can be effectively solved by proposed dynamic scheduling system, and moreover effective load balancing approach aid in better utilization of idle instances.

The scheduling system comprises of three modules

→Tenant Task Manager(TTM) →Global Tenant Manager(GTM) →Dynamic Scheduler

3.2.1. System model

Architecture of system framework is presented in Figure 2. The Tenant Task Manager (TTM) manages the task/query requested by the tenant. Simultaneously it also processes these request. The processed requests are further divided into separate queues based on the tenant requirement of Memory, I/O and CPU for computation or searching of data. Meanwhile, the Local Worker Manager (LWM) monitors the worker load and updates the information to the Global Tenant Manager (GTM). GTM sorts the available workers based on CPU, I/O and Memory for processing task. Dynamic scheduler works between Tenant Task Manager and Global Tenant Manager. Scheduler takes the request task queue from Tenant Task 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.

3.2.2. System parameters

Let us consider T tenants, H workers and M number of query requests. H Workers are represented as $W = \{w1, w2, w3 \ldots a, wx, \ldots, wH\}$ and M queries are represented as$\{Q1, Q2, Q3, \ldots, Qx, \ldots, QM\}$. The workers in cloud environment represent a set of virtual machines which are threads in our experiments. Each thread's computing capability is defined by its parameter such as Memory, I/O and CPU (i.e. $Lx = (Dx, Vx)$ where Rx defines Memory usage, Dx defines I/O waiting time and Vx defines CPU utilization respectively). The GTM periodically collects and updates this information from LWM.

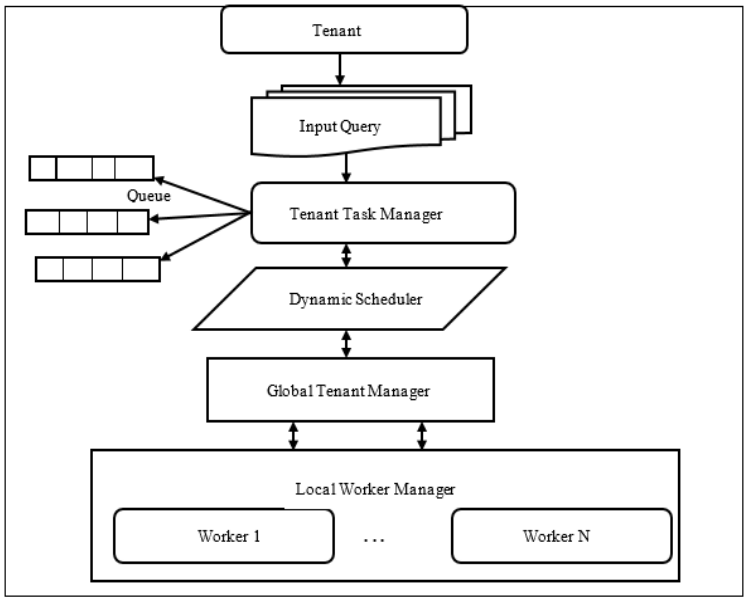

Figure 2. Architecture of multi-tenant dynamic resource scheduling model

3.2.3. Query classifier

Initially, the tenant task manager collects tenant submitted query along with the resource required information to process the query and SLA requirement. The query specifies query size Sy, required CPU Vy, memory Ry ,time required for the execution By [This information is obtained from config file for each tenant shown in Figure 1]. The information is gathered in order to cater the queries demanding diversified resources. Henceforth, a query requested by tenant y is represented as Qy = (Ry, Vy, Sy, By).The TTM further determines the I/O required as:

$$D_y = \frac{S_y}{V_y} \tag{1}$$

The I/O usage is directly dependent on the query size and CPU capability and is therefore computed by the rate of Sy and Vy. Further, the received queries are classified and queued up. In order to classify the received query, the cloud resource parameters (system parameters in our case) R_k, D_k and $V_k$of Memory, I/O and CPU are defined. Then, for each query Qy, weights of R, D and V are computed by its value Ry, D_y and V_y and R_k, D_k and V_k. The maximum of these three weights are considered as query group Qyg. If Qyg = D, the query is portioned into queue of I/O intensive, if $Qyg = V$, the query is portioned into queues of CPU intensive and so on. In the proposed model the queries in these three queues are equal to the total number of queries (i.e. each one of three queues makes up only one part of all queries).

$$Q_{yg} = \max(R, D, V) = \left({}^{R_y}\!/_{R_k}, {}^{D_y}\!/_{D_k}, {}^{V_y}\!/_{V_k} \right). \tag{2}$$

Finally, total M queries which are partitioned into three queries are represented as LQD, LQV and LQR of i I/O intensive, j CPU intensive and $M - j - i$ Memory intensive respectively by the query group Qyg.

$$L_{QD} = \{Q_{j+1}, Q_{j+2}, Q_{j+3}, \dots, Q_{yD}, \dots, Q_{j+i}\} \tag{3}$$
$$L_{QV} = \{Q_1, Q_2, Q_3, \dots, Q_{yV}, \dots, Q_j\} \tag{4}$$

$$L_{QR} = \{Q_{j+i+1}, Q_{j+i+2}, Q_{j+i+3}, \dots, Q_{yR}, \dots, Q_{M-j-i}\}. \tag{5}$$

A detailed diagram is shown in Figure 3.

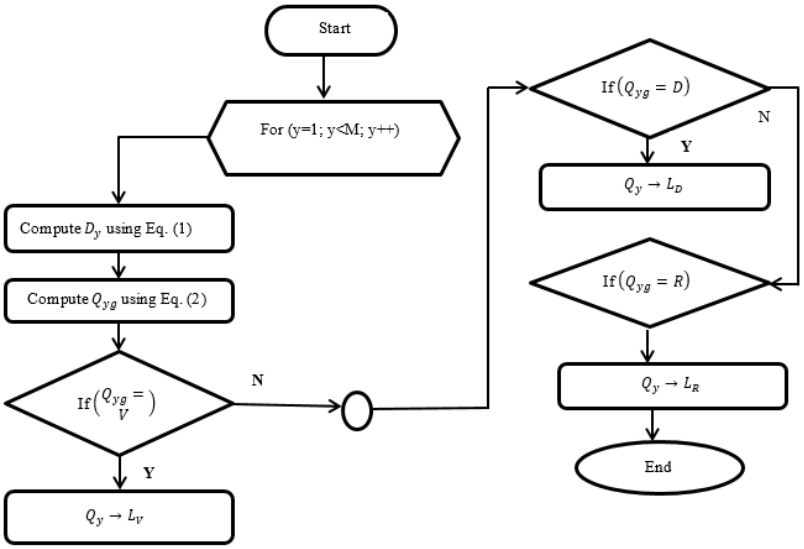

Figure 3. Query classification technique

3.2.4. Worker sorting technique

The worker in cloud environment consists of set of virtual machine (threads). Each virtual machine computing capability is defined by its parameter such as Memory, I/O and CPU (i.e. $L_x = (R_x, D_x, V_x)$. This parameter defines, Memory usage, I/O waiting time and CPU utilization. The GTM periodically collects and updates this information from LWM. LWM gathers Memory usage, I/O waiting period and CPU utilization information from local workers either periodically defined by user or when 50% of the task is completed in a particular thread. LWM transmits this information to the GTM. Next, GTM sorts these workers from small to large considering Memory, I/O and CPU resources and forms queues *LR*, *LD* and *LV* respectively i.e., LR holds the workers in the increasing order of their memory capacity, LD holds the workers in the increasing order of their I/O capacity and LV holds the same workers in the increasing order of CPU available respectively.

$$L_R = \{W_1, W_2, W_3, \dots, W_{yR}, \dots, W_H\} \quad (6)$$

$$L_D = \{W_1, W_2, W_3, \dots, W_{yD}, \dots, W_H\} \quad (7)$$

$$L_V = \{W_1, W_2, W_3, \dots, W_{yV}, \dots, W_H\} \quad (8)$$

All the workers are sorted rather than classifying, due to size and resources dynamics. As a result, these three queues are composed of workers with all the resources, unlike the queries queue. Consequently, the proposed model comprises of two types of queues. The query queues representing Memory, I/O and CPU intensive queries and the worker queues, which are formed by sorting Memory, I/O and CPU load from small too big.

3.2.5. Dynamic scheduling approach

Lastly, the scheduler assigns query (based on its type,weight and SLA)from queues of Tenant Manager to workers sorted by GTM. i.e., based on weight (CPU) assigned to a query say q1 a high or low CPU utilization worker is allocated. If a query has less weight, then it is assigned a worker with less processing power and for higher weight query a worker with high processing power is assigned.A query q2(memory or i/o intensive) in accordance with its weight can be assigned to a worker which is already executing another query if it has enough resource to handle the query and also SLA of the query is met. If neither fails a new worker is assigned to query q2.

Besides, for maximizing resource utilization, the query is assigned to a worker with less load (i.e. assigning query corresponding to its type and load). For example, the Memory, I/O and CPU intensive queries are assigned to worker with low Memory, I/O and CPU usage respectively. Moreover, the scheduler will assign each query from each queue to a different available worker for simultaneous execution. This aids in reducing the load and enhance system efficiency.

3.2.6. Dynamic scheduling approach

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 as shown in Figure 4. It makes one batch of queries from sub queries, queues it as g=M/G, where G represents the number of queues created. Remaining M-g queries will be considered in next group. If M-g>H 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.

3.2.7. Dynamic scheduling adaptivity method

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 as shown in Figure 4. It makes one batch of queries from sub queries, queues it as g=M/G , where G represents the number of queues created. Remaining M-g queries will be considered in next group. If M-g>H 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.

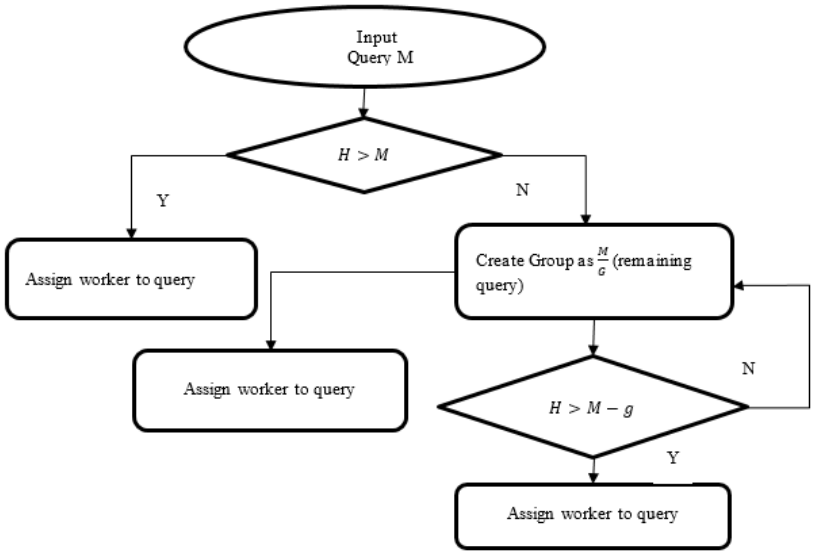

Figure 4. Flowchart of dynamic scheduling adaptivity method

4. EXPERIMENTAL RESULT AND ANALYSIS

We have conducted several experiments to evaluate the performance of proposed model over existing Mute Bench approach [4] in terms of latency and throughput (transaction per seconds). For experiment evaluation OLTP and YCSB benchmark is used. The Mute Bench model is designed using java framework in which the authors have attempted to upgrade OLTP-Bench into a Multi-Tenant Database Benchmark Framework. In the presented work, we have incorporated proposed Multi-Tenant Dynamic Resource Scheduler Model (MTDRSM) into [4]. We further extended model [4] to support workload execution for different benchmarks and multi-tenant workload execution on different database such as MySQL, Oracle, and H2DB etc. by using Hibernate framework.

The MTDRSM is developed using JAVA programming language on eclipse neon framework. The system environment used for workload execution is I-5, 3.2 GHz, quad core Intel class processor with 16 GB RAM. We have considered workload execution of TPCC and YCSB benchmark on H2 database. The workload execution is carried out for both with and without SLA compliances. Each tenant is given a set

of worker (threads) for workload execution. The number of worker is varied as 10, 20 and 50. The tenant ID is incremented by 3 (i.e. for 10, 20 and 50 worker there are 4, 7 and 17 tenants, respectively) and 6 tenant per execution is considered. Each tenant executes its workload with unlimited data rate. The OLTP and YCSB workload mix is composed of 25% read record and 15% for each other transaction types.

4.1. SLA and SLA breach

In the Query $Q_y = (K_y, V_y, R_y, T_y)$, if the first three parameters represent query size, CPU utilization and memory that the tenant applies to use, then T_y is the SLA breach of the query. These parameters come from the Tenant task manager and are submitted by tenants. If the query $Q_y$fail to meet T_y defined by tenant to its service provider, then the SLA is considered to be breached. The SLA is measured as follows:
Query retrieval time is calculated

$$q_{\text{Retrieval}} = \frac{\sum(q_{y-w}+q_{y-\text{processed}})}{H} \quad (9)$$

where q_{y-w} is the waiting time, $q_{y-\text{processed}}$ is the processing or query completion time and H is total number of queries. Check if $q_{\text{Retrieval}} > T_y$. If yes query is breached.

4.2. Latency performance evaluation

In Figure 5 the latency performance considering different worker without SLA compliances is shown. It is seen from graph the MTDRSM performs better than MuTeBench in term of latency performance considering varied worker. The MTDRSM reduce latency by 23.87%, 11.82% and 46.63% considering 10, 20 and 50 worker respectively, over MuTeBench. An average latency reduction of 27.44% is achieved by MTDRSM over MuTeBench. Similarly, In Figure 6 the latency performance considering different worker with SLA compliances is shown. It is seen from graph the MTDRSM performs better than MuTeBench in term of latency performance considering varied worker. The MTDRSM reduce latency by 23.08%, 11.7% and 45.83% considering 10, 20 and 50 worker respectively, over MuTeBench. An average latency reduction of 28.2% is achieved by MTDRSM over MuTeBench. It is seen from Figure 5 and Figure 6 that provisioning SLA to tenant induces a slight overhead in latency performance.

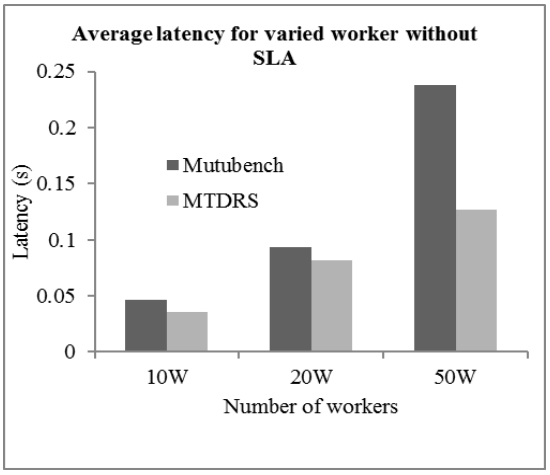

Figure 5. Average latency achieved for varied worker without SLA

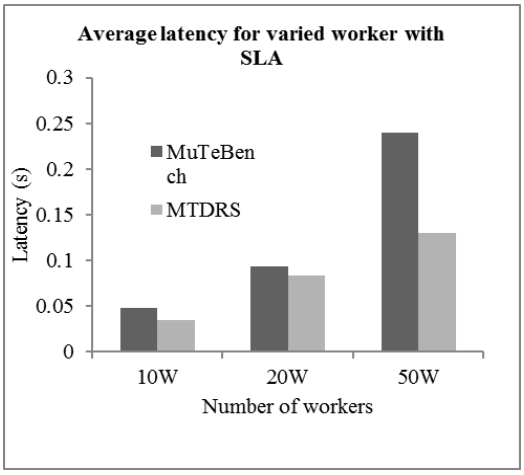

Figure 6. Average latency achieved for varied worker with SLA

4.3. Throughput (transaction per second evaluation) performance evaluation

Tables 1 and 2 describes the transaction status without and with SLA respectively. The transaction status is composed of following type:

- Completed transaction: this shows the transaction is successfully completed,
- Aborted transaction: this show the transaction is aborted by user/system,
- Rejected transaction: this shows the transaction is rejected due to wrong information entered (.i.e. non-existent account number) during transaction and
- Unexpected error: this is due to unexpected scenario such as server/network down.

It is seen from Tables 1 and 2, MTDRSM achieves high number of transaction per second (TPS) as compared to MuTeBench. In Figure 7 the throughput performance considering different workers without SLA compliances is shown. It is seen from graph the MTDRSM performs better than MuTeBench. The MTDRSM improves throughput by 5.87%, 3.03% and 2.63% considering 10, 20 and 50 worker respectively, over MuTeBench. An average throughput improvement of 3.84% is achieved by MTDRSM over MuTeBench. Similarly, in Figure 8 the throughput performance considering different worker with SLA compliances is shown. It is seen from graph the MTDRSM performs better than MuTeBench in terms of throughput performance considering varied worker. The MTDRSM improves throughput by 7.24%, 7.4% and 7.1% considering 10, 20 and 50 worker respectively, over MuTeBench. An average throughput improvement of 7.25% is achieved by MTDRSM over MuTeBench. It is seen from Figure 7 and Figure 8 that provisioning SLA to tenant induces an overhead in throughput performance of MuTeBench, where us MTDRSM is efficient when provisioning SLA to tenant.

Table 1. Transaction status without SLA

Transaction Status without SLA								
Number of worker	Completed Transaction		Aborted Transaction		Rejected Transaction		Unexpected error	
	MuTeBench	MTDRSM	MuTeBench	MTDRSM	MuTeBench	MTDRSM	MuTeBench	MTDRSM
10	6711	6849	10	7	74911	79901	31	5
20	13203	13288	11	8	149122	154134	59	45
50	10507	12873	9	7	148255	151225	116	102

Table 2. Transaction status with SLA

Transaction Status with SLA								
Number of worker	Completed Transaction		Aborted Transaction		Rejected Transaction		Unexpected error	
	MuTeBench	MTDRSM	MuTeBench	MTDRSM	MuTeBench	MTDRSM	MuTeBench	MTDRSM
10	10799	12894	12	8	144222	154264	37	21
20	10906	13108	10	8	142508	152592	64	50
50	11062	12960	8	8	140554	150260	39	27

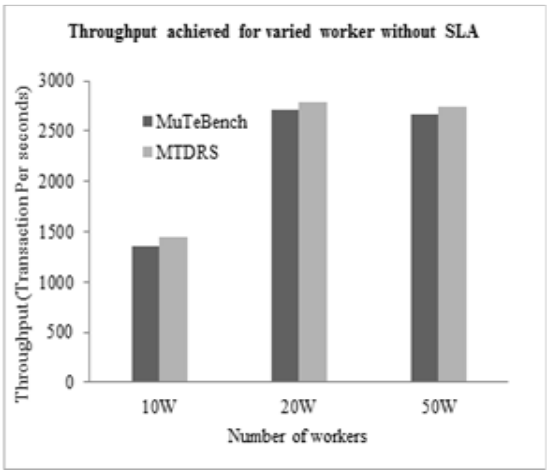

Figure 7. Throughput achieved for varied worker without SLA

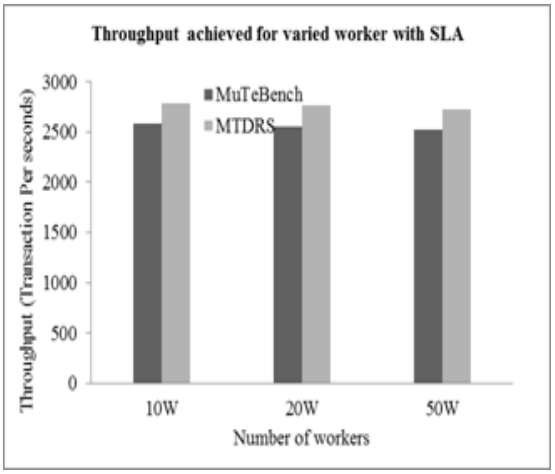

Figure 8. Throughput achieved for varied worker with SLA

5. CONCLUSION

Multitenant database management on cloud environment has attained huge interest among various organizations, due to scalability and cost benefits. The wide survey carried out shows the existing scheduling technique suffers due to NP-Hard problem. Therefore an efficient scheduling and load balancing mechanism is required for dynamic resource allocation. Here we presented query classification and worker sorting technique for dynamic resource allocation and handling idle instance efficiently. Experiments are conducted to evaluate the performance of MTDRSM in terms of latency and throughput with and without SLA compliances. The experiments are conducted considering varied tenant, worker and workload such as TPCC and YCSB benchmarks. The experimental outcome shows the MTDRS reduces average latency of 27.44%

and 28.2% over Mute Bench without and with SLA compliance respectively. The MTDRSM improves average throughput by 3.84% and 7.25% over Mute Bench without and with SLA compliance respectively. The overall result achieved shows that when SLA is given to tenant there incur an overhead for Mute Bench model, as a result affect the performance of throughput and induce latency for tenant. This shows the efficiency of handling idle instance by MTDRSM model. The overall result achieved shows that the MTDRSM can provision SLA without incurring latency to tenants and performs significantly better than Mute Bench. Provisioning security to database access in multi-tenant cloud environment is a critical factor in increasing wide adoption. The future work would consider provisioning security to multi-tenant cloud SaaS environment.

ACKNOWLEDGEMENTS

The authors would like to acknowledge and thank Technical Education Quality Improvement Program [TEQIP] Phase 3, BMS College of Engineering, Basavanagudi, Bangalore.

REFERENCES

[1] B. P. Rimal and E. Choi, "A service-oriented taxonomical spectrum, cloudy challenges and opportunities of cloud computing," in *International Journal of Communication Systems*, vol. 25, no. 6, pp. 796–819, Jun. 2012.
[2] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," *2008 Grid Computing Environments Workshop*, Austin, TX, pp. 1-10, 2008.
[3] Stefan Aulbach, Torsten Grust, Dean Jacobs, Alfons Kemper, and Jan Rittinger, "Multi-tenant databases for software as a service: Schema-mapping techniques," In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1195–1206, Jun. 2008.
[4] Andreas Gobel, "MuTeBench: Turning OLTP-Bench into a Multi-Tenancy Database Benchmark Framework," *The fifth International Conference on Cloud Computing, GRIDs and Virtualization*, pp. 84-47, 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, Nov. 2012.
[6] Cor-Paul Bezemer, Andy Zaidman, "Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare?," *IWPSE-EVOL '10: Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, pp. 88-92, Sep. 2010.
[7] Archana Bhaskar and Rajeev Ranjan, "Optimized memory model for hadoop map reduce framework," in *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 5, pp. 4396-4407, Oct. 2019.
[8] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *ACM SIGCOMM Computer Communication Review*, pp. 242–253, Aug. 2011.
[9] Toan Phan Thanh, Loc Nguyen The, Said Elnaffar, Cuong Nguyen Doan, Huu Dang Quoc, "An Effective PSO-inspired Algorithm for Workflow Scheduling," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 8, no. 5, pp. 3852-3859, Oct. 2018.
[10] B. P. Rimal and M. Maier, "Workflow Scheduling in Multi-Tenant Cloud Computing Environments," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 290-304, Jan. 2017.
[11] F. S. Hsieh and J. B. Lin, "A dynamic scheme for scheduling complex tasks in manufacturing systems based on collaboration of agents," in *Applied Intelligence*, vol. 41, no. 2, pp. 366–382, Sep. 2014.
[12] J.-C. Liou and M. A. Palis, "An efficient task clustering heuristic for scheduling DAGs on multiprocessors," in *Proc., Resource Management, Symp. of Parallel and Distrib. Processing*, pp. 152–156, 1996.
[13] R. Bajaj and D. P. Agrawal, "Improving scheduling of tasks in a heterogeneous environment," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 2, pp. 107-118, Feb. 2004.
[14] H. Topcuoglu, S. Hariri, M. Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260-274, Mar. 2002.
[15] H. M. Fard, R. Prodan, J.J.D. Barrionuevo, and T. Fahringer, "A multi-objective approach for workflow scheduling in heterogeneous environments," *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, Ottawa, ON, pp. 300-309, 2012.
[16] D. Shue, M. J. Freedman, and A. Shaikh, "Performance isolation and fairness for multi-tenant cloud storage," *OSDI'12: Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, pp. 349–362, Oct. 2012.
[17] S. Pandey, L. Wu, S. Guru, and R. Buyya, "A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments," *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, Perth, WA, pp. 400-407, 2010.
[18] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds," in *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 222-235, 2014.
[19] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang, "A market-oriented hierarchical scheduling strategy incloud workflow systems," *The Journal of Supercomputing*, vol. 63, no. 1, pp. 256–293, Jan. 2013.
[20] H.M. Fard, R. Prodan, and T. Fahringer, "A truthful dynamic workflow scheduling mechanism for commercial multicloud environments," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1203-1212, Jun. 2013.

[21] G. Juve, *et al.*, "Scientific workflow applications on amazon EC2," *2009 5th IEEE International Conference on E-Science Workshops*, Oxford, pp. 59-66, 2009.
[22] J. Jin, J. Luo, A. Song, F. Dong, and R. Xiong, "Bar: An efficient data locality driven task scheduling algorithm for cloud computing," *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Newport Beach, CA, pp. 295-304, 2011.
[23] D. Yuan, Y. Yang, X. Liu, and J. Chen, "A data placement strategy in scientific cloud workflows," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1200–1214, Oct. 2010.
[24] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," in *IEEE Transactions on Services Computing*, Fourth Quarter, vol. 5, no. 4, pp. 497-511, 2012.
[25] L. Ramakrishnan, *et al*., "Vgrads: enabling e-science workflows on grids and clouds with fault tolerance," *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Portland, OR, pp. 1-12, 2009.
[26] K. Plankensteiner and R. Prodan, "Meeting soft deadlines in scientific workflows using resubmission impact," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 5, pp. 890-901, May 2012.
[27] S. Abrishami, M. Naghibzadeh, and D.H.J. Epema, "Deadline constrained workflow scheduling algorithms for infrastructure as a service clouds," in *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, Jan. 2013.
[28] W. Tsai, X. Sun, Q. Shao, and G. Qi, "Two-tier multi-tenancy scaling and load balancing," *2010 IEEE 7th International Conference on E-Business Engineering*, Shanghai, pp. 484-489, 2010.
[29] T. Kwok and A. Mohindra, "Resource calculations with constraints, and placement of tenants and instances for multi-tenant saas applications," in *International Conference on Service-Oriented Computing*, vol. 5364, pp. 633-648, 2008.
[30] E. Javier, M. Arturo, J. Guillermo, M. Martn, R. Ral, and C. David, "A tenant-based resource allocation model for scaling softwareas- a-service applications over cloud computing infrastructures," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 273–286, Jan. 2013.
[31] 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 International Conference on Utility and Cloud Computing (UCC)*, Limassol, pp. 112-121, 2015.
[32] Sudipto Das, Vivek R. Narasayya, Manoj Syamala, "CPU Sharing Techniques for Performance Isolation in Multitenant Relational Databaseasa Service," *Proceedings of the VLDB Endowment*, vol. 7, no. 1, pp. 37-48, 2013.
[33] 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*, Jan. 2013
[34] Ying Hua Zhou, *et al.*, "Db2mmt: A massive multi-tenant database platform for cloud computing," *2011 IEEE 8th International Conference on e-Business Engineering, Beijing*, pp. 335-340, 2011.

BIOGRAPHIES OF AUTHORS

Pallavi G. B., working as assistant professor, in the Department of Computer Science and Engineering, BMS College of Engineering,Bangalore. Pursuing PhD in the field of Cloud Computing.

Dr. P. Jayarekha., working as Professor, in the Department of Information Science and Engineering, BMS College of Engineering,Bangalore. Awarded PhD in June 2011in the field of networks. She has published 35 research papers in referred International Journals and also few in national and international conferences. She has been actively involved as an expert member in various committees such as AICTE, NBA.