Computer Science and Information Technologies
Vol. 1, No. 2, July 2020, pp. 39~46
ISSN: 2722-3221, DOI: 10.11591/csit.v1i2.p39-46

Efficient and scalable multitenant placement approach for in-memory database over supple architecture

Arpita Shah[1], Narendra Patel[2]
[1]Faculty of Technology and Engineering, Charotar University of Science and Technology (CHARUSAT), India
[2]Department of Computer Engineering, Birla Vishvakarma Mahavidyalaya Engineering College-GTU, India

Article Info

Article history:

Received Jan 23, 2020
Revised May 1, 2020
Accepted May 20, 2020

Keywords:

Best-fit greedy algorithm
In-memory database
Multi-tenant placement
Multitenancy
Supple architecture

ABSTRACT

Of late Multitenant model with In-Memory database has become prominent area for research. The paper has used advantages of multitenancy to reduce the cost for hardware, labor and make availability of storage by sharing database memory and file execution. The purpose of this paper is to give overview of proposed Supple architecture for implementing in-memory database backend and multitenancy, applicable in public and private cloud settings. Backend in-memory database uses column-oriented approach with dictionary based compression technique. We used dedicated sample benchmark for the workload processing and also adopt the SLA penalty model. In particular, we present two approximation algorithms, multi-tenant placement (MTP) and best-fit greedy to show the quality of tenant placement. The experimental results show that MTP algorithm is scalable and efficient in comparison with best-fit greedy algorithm over proposed architecture.

Corresponding Author:

Arpita Shah
Faculty of Technology and Engineering
Charotar University of Science and Technology (CHARUSAT)
Changa-388421, Gujarat, India
Email: arpitashah.ce@charusat.ac.in

1. INTRODUCTION

Conventionally, in-memory databases have been in use for applications which were performance sensitive such as financial services markets. In-memory database claim to provide an alternative to the OLAP. Instead of pulling the data from a disk, keeping it in memory (RAM) speeds up the processing and response time of data by order of magnitude. This is the reason why in-memory Database is booming in industry these days. With the expeditious increase of software-as-a-service (SaaS), it has become important to operate services at a faster response time for SaaS providers. With the aim of to reduce operational cost, multi-tenancy provides methods for combining multiple tenants of hosted application into the same system. Multi tenancy can be employed in the database layer in such a way that a single database can be used by multiple customers i.e. tenants. A cloud uses technology of multitenancy to share IT resources among multiple applications and tenants securely. Virtualization-based architectures is used by some clouds to isolate tenants and some uses custom software architectures to get the job done. In this paper we have shown the proposed architecture for standing tenant placement for query request with sample HR benchmark design combined both approaches in memory and multitenancy. To improve sever utilization and resource profit, tested two algorithm(s) (1) best fit greedy (2) MTP. In Supple architecture it consists mainly three components (1) Cluster head: maintain placement information over in-memory database. (2) Router: based on cluster map it forwards query request to the suitable instance manager and (3) Instance Manager: distribution of requests across the tenant user. The

supple architecture adopt microsoft azure platform and also provide physical machine that turned into virtual disk pool.

2. IN-MEMORY DATABASE

Earlier in-memory databases have been used in the performance sensitive applications which were performance sensitive like financial services markets. However now a days in-memory databases have become more generally adopted. At the same time, the SaaS model has become popular and customers are gaining interest in this model, as it decreases their the burden of the hassle of operating the system, which requires provisioning the hardware as well as maintaining, operating and configuring application servers and databases. The key difference between in-memory database (IMDB) and disk resident database (DRDB) is that data in IMDB resides in main memory permanently and in DRDB, data resides permanently on disk. Since the chip density is increasing day by day and semiconductor memory is becoming cheaper so it's possible to store huge amount of data in memory [1]. IMDBs can provide performance by an order of magnitude faster as compared to DRDBs. In-memory which is one of the memory resident systems and compares its processing time with a typical disc resident database. Of course the IMDB gave better performance and response time. Complexity in IMDBs is reduced as the number of machine instructions reduced, buffer pool management is not required, and no need of extra data copies, index pages decreases, and their simplified structure is possible. As a consequence the design becomes simple and more concise, and requests are performed faster. IMDSs have also lower memory and CPU requirements. Also the design and arrangement of data on disk is much more unfavorable than arrangement of data on main memory. Real time applications require fast response. So IMDBs play crucial role for real time applications. "Can entire database fit into main memory?" The answer depends on the application. If size of application is limited and is growing at slower pace than it is possible to have entire database in main memory. For example database employee details of some small company. But for real time applications it is must that data reside in main memory. If size of database is too large to fit into the main memory for example an application with satellite image data the data can be categorized as hot and cold data. The data which is frequently required is categorized as hot and data which is rarely required and is voluminous is cold data. The hot data lay in in-main memory and cold is stored on disk.

2.1. Challenges with in-memory systems

In-memory is liable to change rapidly while disk storage is nonvolatile. So regular backups must be taken (on disk) and at the same time it is to be taken care that performance of IMDB is not affected. If disk fail, data on other disks can be secured and recovery from disk is easy but if memory fails, entire database is lost. The performance gain of IMDB can be limited by the application operating it, layout and implementation of database itself, the hardware on which the database is running and the association with external devices. Large volumes of data with lower frequency reads are not much more efficient with IMDBs. Many papers have discussed the impact of memory residency of some important functional components of database management system (DBMS) like concurrency control, access methods, commit processing, query processing and performance etc. Many papers have discussed the issues related to IMDB recovery and briefly examine some of the solutions [2].

2.2. In-memory architecture

To understand the architecture of in-memory, architecture of oracle database is considered here, which is categorized under the in-memory cache architecture. The elements of architecture include database processes, memory-resident data structures, shared libraries and administrative programs. Indexes, system tables, tables, cursors, locks, temporary indexes and compiled commands together make up the memory resident data structures. Through direct link and client/server connections, the application can be linked to the database or IMDB cache. Information is received by replication agents from master databases and is sent to subscriber databases. Asynchronous data transfers between oracle database and cache groups in the in-memory database cache are performed by cache agent. Figure1 shows oracle's in-memory database cache architecture [3]. External memory is accessed only in three cases:

- To load copy of main memory during system startup.
- Checkpoint over writing, recovery and on Logging.
- To persists data about data and configuration changes.

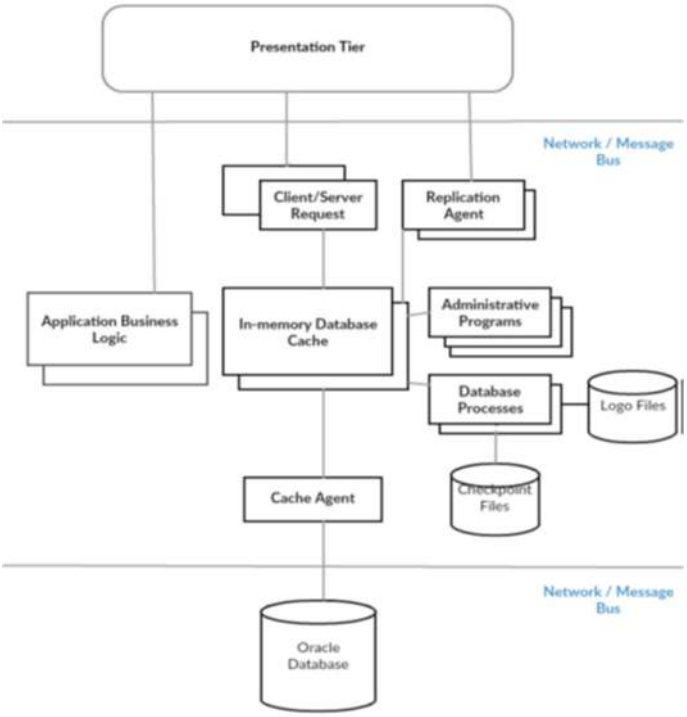

Figure 1. In-memory database cache architecture

So for these type of tasks IMDBs rely on paged data handling while all other operations run purely on main memory. The database community is going to experience a great shift in market in the coming years as in-memory databases are becoming more effective and affordable. Although in-memory database mainly consists two storage approach namely row-oriented (database re-structuring) and column-oriented. Many databases can use both approaches row-oriented as well as column-oriented.

2.3. Multitenancy

With the aim of reducing operational cost, multi-tenancy provides methods for combining multiple customers (i.e. tenants) of deployed application which run on the same infrastructure. Database layer can be employed in the Multi tenancy architecture can be utilized in the database layer in such a way that a single database can be shared by multiple customer. A cloud uses technology of multitenancy to share IT resources among multiple applications and tenants securely. Virtualization-based architectures is used by some clouds to isolate tenants and some uses custom software architectures to get the job done. Depending on requirements of customer such as security, high availability, customizability, the choice of appropriate tenancy model is decided. There are several possible multi-tenant schemes like shared design, VM based design etc. For tenant applications are a well-known example of a type of application whose data and workloads can be partitioned easily. For instance, with tenant applications, data and workload can typically be partitioned along tenant boundaries since most requests are within the confines of a tenant. So, by considering a framework which takes the tenant workloads as input, their performance service level objectives (SLOs), and the server hardware which is obtainable to the SaaS provider.

3. A SUPPLE INFRASTRUCTURE FOR MULTITENANCY OVER IN-MEMORY DATABASE

Proposed architecture for multitenancy using in-memory is shown in Figure 2 which consists major three components.

3.1. Component(s)

3.1.1. Cluster head

There is single cluster leader exists in a supple infrastructure and it assigns one or more tenant to server. Each tenant's replica is assigned to the individual instance handler, so that each instance handler must process different data from multiple tenants. The cluster head maintains the placement information over in memory database performance with hard disk based shown in Figure 2, which it propagates to the route. In

addition the cluster head also observe active nodes, starts and stops severs, placement assessment and migration of tenant between servers. Request processing cannot assessed directly by the cluster head. The cluster head process has to run on all active severs to handle single point of failure to make highly availability that can be possible by running a cluster head process on all active servers.

3.1.2. Router

It forwards query request to the suitable instance manager based on cluster map information and also from outside the cluster. Also, it provides location transparency of a tenant's database. The job of router need to be balanced the load across the tenant's replicas in round-robin pattern.

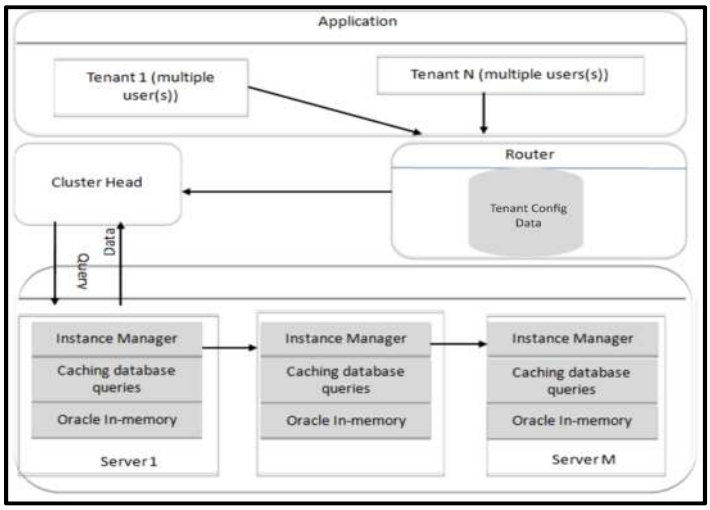

Figure 2. A supple infrastructure for multi-tenancy over in-memory database

If a replica of a tenant becomes unavailable in consequence on failure of a server [4, 5] requests need to be balanced amongst the remaining live replicas. The query results send back to the application through the router.

3.1.3. Instance Manager

The job of instance manger is to manage the distribution of requests across the tenant user. When instance manager receive a write request, it write to caching database [6] (if no space over main memory) and also forward the request to its successor node in a cluster. While, handling the database over no. of servers concurrent requests may cause performance issues and it effect on application execution. To resolve this problem we have experimented on oracle sharding architecture to support elastic scale. Request for the write operation need to be obstinately written on i node out of n nodes and then asynchronously replicated to the nodes $i + 1$ to n. Each tenant consists two consistent replicas and Individual instance manager is coupled up with a local oracle instance, which is shared amongst multiple tenants. Oracle instance support the approach of multitenancy [7] and in-memory. A local oracle instance needs to be paired with instance manger.

3.2. Benchmark design

We have used dedicated sample benchmarks for mixed workload processing that benchmark called HRSB-MT shown in Table 1. Our testing experiment is on oracle in-memory column database, which runs for mixed workload processing application. In column-oriented approach, database keep every attribute for in a separate column structure and is ideal for analytics, since it allows for speedy data retrieval when only a few columns are selected but the query accesses a huge portion of the data set. When DML operation (insert, update or delete) occurs on both methods then row-oriented format is extremely effective for processing DML as it manipulates an entire row or record in one go. A column approach is not so proficient as compared to row format at processing row-wise DML but for OLAP, larger chunk of data column-oriented gives simultaneous data execution. HRSB-MT model also opted dictionary-based compression techniques.

Table 1. HR schema benchmark-MT

Table name	Table size (MB)
Region	951955
Countries	220224
Locations	332
Departments	5171
Jobs	1032567
Employees	1956879

3.3. Resource consumption of multiple homogeneous tenant

So, by considering a framework which takes workloads of the tenant as input, their performance SLOs [8], [9] and the server hardware which is obtainable to the SaaS provider, and result into a cost-effective recipe which specifies utilization of hardware to deliver and how the tenants are scheduled on available hardware resource. Each tenants contain the same size and request rate on sever. Total no. of users is distributed uniformly among all tenants and the server is filled up only 15-20% of its main memory is used. The tenant size ts is divided by the resultant amount of memory. As a result depending on the chosen value for ts server may contain few or more tenants, so we vary ts from 20 to 198 MB (from 400,000 to 4,000,000 rows). Request per tenant is denoted by TR and it may increase until SLO violated [10], [11]. Figure 3 shows that when the number of tenants is increased by a factor of 10, throughput decreases by 10%. A SLO perspective [12] violates whether the server scan function is utilized by small number of large tenants or several small tenants.

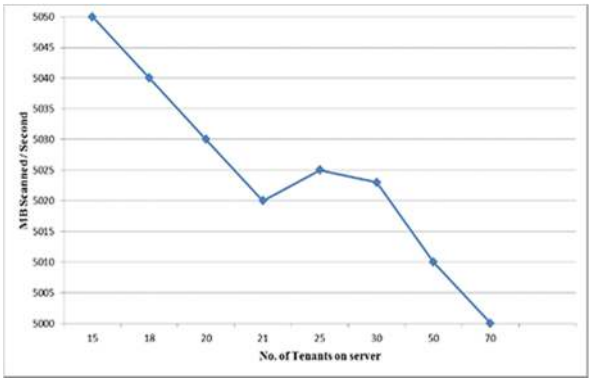

Figure 3. Maximum throughput without violation the performance SLO

4. STANDING TENANT PLACEMENT

For standing tenant placement, we have considered following general data as input.

A valid tenant assignment is performed using binary decision $x \in \{0, 1\}^{S \times T \times R}$, where

$$x_{t,i}^{(y)} = \{1, \text{if tenant copy } y \text{ is on sever } i, \text{ otherwise } 0\} \quad (1)$$

$$s \in \{0,1\}^{N} \quad (2)$$

Where s_i=1 indicates that specific server i is active, otherwise server is inactive.

Now, as performing tenant placement also needs to be checked that replica of tenant is allocated to a server once or not and no two copies of the same tenant. So, to check the specific condition we have applied some constraints.

$$\sum_{i \in N} x_{t,i}^{(y)} = 1 \qquad \forall t \in T, \forall y \in R \quad (3)$$

$$\sum_{y \in R} x_{t,i}^{(y)} \leq 1 \qquad \forall t \in T, \forall i \in N \quad (4)$$

The server in-memory capacity needs to fit on full amount size of all tenant on that server.

$$\sum_{t \in T} \sum_{y \in R} \sigma(t) \bullet x_{t,i}^{(y)} \leq c_{\sigma}(i) \bullet s_i \quad \forall i \in N \tag{5}$$

One of the important key features of greedy heuristics is that they are computationally less rigorous than meta-heuristics. Greedy algorithms are loosely based on the well-known best-fit algorithm. It also delivers good results for the related bin-packing problem. Although the problem in a best fit algorithm consist constant approximation ratio over bin packing. So, when the tenants are small it inclined to bundle lots of tenants on a server. Best-fit algorithm is used for tenant placement, which finds the server with the least remaining resource that can accommodate each tenant, and in case such a server cannot be found, either relaxes the constraints gradually or use a new server. Starts with a random placement. It assumes a fixed number of tenant types and, on each server, it assigns tenants of the same type to the same database server instance with a fixed amount of main memory allocated [13].

4.1. Proposed multitenant placement (MTP) algorithm

When tenant's quantity is increasing then we need to apply tenant placement algorithm to allocate server resources [14], [15]. Parameters for MTP algorithm as shown in Table 2 are illustrate the tenant, server capacity, replica of tenant and other resource parameters.

Table 2. MTP parameters

Symbol	Meaning
$T \subseteq NT$	Shows no. of tenants, Ti represent ith tenant
$S \subseteq IS$	No. of servers (to show resource allocation)
$\sigma : T \rightarrow NT^{+}$	Function returns Main memory requirement of a given tenant
$c_{\sigma} : S \rightarrow IS^{+}$	Function returns Main memory capacity of ith server

The purpose of mentioned algorithm which implemented on proposed architecture is to improve utilization of server to the set of tenants. Allocating proper resources over the tenants, we have deployed the tenant placement algorithm in Cluster head to improve server utilization. To fulfil tenant's service quality as its basis requirement, the performance testing use basic function swapping as a result it is considering minimum number of servers as and when it is required. On the base of meeting the requirements of service quality of the tenant, experimental test use best fit heuristic algorithms which is compared with the proposed MTP algorithm. In a resulting outcomes MTP requires less number of servers.

Algorithm: MTP

```
Input: T Tenants, Server in-memory capacity c_σ , Replica y
Find binary value thru eq. (1) (0,1)
T={t1,t2.....tn}= sorted list of tenant in decreasing order with respect of size
for server i in S = { 1,2,3} do
   for each t_i ∈ T  x_{t,i}^{(2)}
        if x_{t,i}^{(2)}  then
            locate other server j such that x_{t,j}^{(1)}
            call swap(t, i, j)
        else
            place t_i on server
        end if
   end for
  call σ(T[i])
  if (T[i]< c_σ ) then
  S[j]=S[j]-T[i]
  end if
end for
```

5. EXPERIMENTAL SETUP AND RESULTS ANALYSIS

The experiments were carried out on a cluster of three servers which is shown in Figure 2, each having 16GB memory, VM(s), running centOS and oracle in-memory database. We produced up to 60 tenants. Each tenant runs with mix workloads in proportion to the generated query pattern. Concerning the sizes of the tenants, we measured the in-memory database with multitenancy settings existing in microsoft SQL azure [16], [17]. Figure 4 shows that number of tenants placed on each server (i.e. no of server = 2, 3 and 4) for resource allocation [16]. If there are 8 tenants labeled as A, B, C, D, E, F, G, H and for three server numbered as 1, 2 and 3. If tenant B, C, D placed on server 1 then 1= {B, C, D} likewise for others tenants too. When the number of servers is not enough, both algorithm best fit and MTP uses different strategies. Figure 4 shows no. of tenants placed on each server numbered as 1, 2 and 3. In this paper we adopt the SLA penalty model [17], if queries arrived during server overload will fail to notice their SLA deadlines and the penalty need to be paid by service provider; and other queries will meet their SLA. Reason is using the load of a tenant will change very frequently for opting SLA model. SLA penalties occur mostly due to prolonged system overload instead of a temporary burst in arrival of query for a short period (example arrival during 10 millisecond).

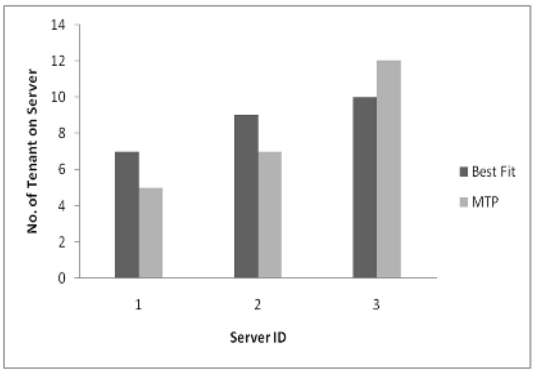

Figure 4. Number of tenant on sever

As it shown in Figure 5 shows that how query processed using two different approaches: Best fit greedy and MTP, in which best fit worked on hard disk based query processing while MTP processed query over in-memory database cluster. Cost of query processing through tenant is comparatively low in MTP than best fit greedy and also works effectively than best fit. We have also experimented scalability of tenant placement. Figure 6 shows running time of MTP which is for 50 tenants. Running time for best fit greedy is insignificant below 0.3 second even for 30 tenants, whereas MTP takes bit longer time than best fit greedy approach.

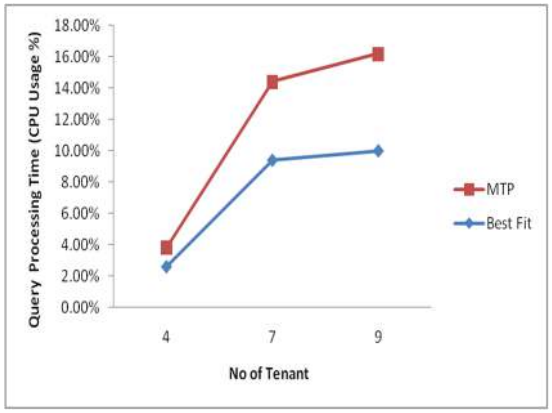

Figure 5. Effectiveness of tenant placement

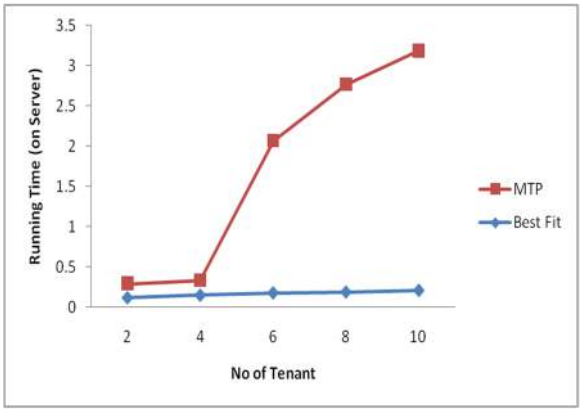

Figure 6. Scalability of tenant placement

6. CONCLUSION

Multitenancy with in-memory database (opted in oracle) speeds up the processing and response time of data request. For in-memory database we have tested over different platform like SAP HANA, grid grain and oracle. In this paper we have proposed multitenancy architecture (supple) using in-memory database with proposed MTP algorithm. From the perspective of efficiency the paper shows proposed MTP algorithm in comparison with Best fit Greedy approach with database benchmark (HRSB-MT) over few tenants to improve the quality of tenant placement. While this paper focuses on supplement architecture for multitenant with in-memory database, in future will work on dynamic placement approach.

REFERENCES

[1] T. V. Hung and H. Chuanhe, "An Effective Data Placement Strategy in Main-Memory Database Cluster," *2011 Second International Conference on Networking and Distributed Computing*, 2011, pp. 93-98, doi: 10.1109/ICNDC.2011.27.

[2] S. M. Chung, "Parallel main memory database system." In *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied computing: technological challenges of the 1990's* (SAC '92), Hal Berghel, Ed Deaton, George Hedrick, David Roach, and Roger Wainwright (Eds.). ACM, New York, NY, USA, pp. 273-282, 1992, doi: 10.1145/143559.143652.

[3] Oracle, "Oracle Database In-Memory with Oracle Database 12c," [Online] Available: https://docs.oracle.com/cd/E16662_01/doc/timesten.1121/b56058/arch.htm, 2017.

[4] J. Schaffner, B. Eckart, D. Jacobs, C. Schwarz, H. Plattner and A. Zeier, "Predicting in-memory database performance for automating cluster management tasks," *2011 IEEE 27th International Conference on Data Engineering*, 2011, pp. 1264-1275, doi: 10.1109/ICDE.2011.5767936.

[5] Y. Huang, Y. Zhang, X. Ji, Z. Wang, & S. Wang, "A Data Distribution Strategy for Scalable Main-Memory Database," In *Web and Network Technologies, and Information Management, APWeb/WAIM 2009 International Workshops: WCMT, RTBI, DBIR-ENQOIR*, *PAIS*, Suzhou, China, pp.13-24, Apr 2009.

[6] D. Jacobs, and S. Aulbach, "Ruminations on Multi-Tenant Databases," In *BTW*, vol. 103, pp. 514-521. 2007.

[7] N. Zhang, J. Tatemura, J. Patel, and H. Hacigumus, "Re-evaluating designs for multi-tenant OLTP workloads on SSD-basedI/O subsystems," In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (SIGMOD '14), ACM, New York, NY, USA, pp. 1383-1394, 2014, doi: 10.1145/2588555.2588567

[8] F. Farber, C. Mathis, D. D. Culp, W. Kleis and J. Schaffner, "An in-memory database system for multi-tenant applications," *Proc. BTW*, pp. 650-666, 2015.

[9] Z. Liu, M. S. Squillante, and J. L. Wolf, "On maximizing service-level-agreement profits," In *Proceedings of the 3rd ACM conference on Electronic Commerce* (EC '01). ACM, New York, NY, USA, vol. 29, no. 3, pp. 213-223, Dec 2001, doi: 10.1145/501158.501185.

[10] Y. Chi, H. J. Moon, and H. Hacıgümü, "iCBS: Incremental Cost-based Scheduling under Piecewise Linear SLAs," *PVLDB*, vol. 4, no. 9, pp. 563–574, 2011.

[11] Y. Chi, H. J. Moon, H. Hacıgümü¸s, and J. Tatemura, "SLA-Tree: A Framework for Efficiently Supporting SLA-based Decisions in Cloud Computing," In *EDBT*, pp. 129–140, 2011, doi: 10.1145/1951365.1951383.

[12] J. Csirik, D. S. Johnson, "Bounded space on-line bin packing: best is better than first," In *Proceedings of the1991 second annual ACM/SIGACT-SIAM,* San Francisco, vol. 31, no. 2, pp. 115-138, Mar 1991, doi: 10.1007/s00453-001-0041-7.

[13] F. Wang, J. Li, J. Zhang and Q. Huang, "Research on the Multi-Tenant Placement Genetic Algorithm Based on Eucalyptus Platform," *2016 12th International Conference on Computational Intelligence and Security (CIS)*, 2016, pp. 382-385, doi: 10.1109/CIS.2016.0093.

[14] H. Koziolek, "The SPOSAD Architectural Style for Multi-tenant Software Applications," *2011 Ninth Working IEEE/IFIP Conference on Software Architecture*, 2011, pp. 320-327, doi: 10.1109/WICSA.2011.50.

[15] Y. Zhang, Z. Wang, B. Gao, C. Guo, W. Sun and X. Li, "An Effective Heuristic for On-line Tenant Placement Problem in SaaS," *2010 IEEE International Conference on Web Services*, 2010, pp. 425-432, doi: 10.1109/ICWS.2010.65.

[16] D. Strockis, "Microsoft Azure," [Online] Available: https://azure.microsoft.com/en-us/resources/samples/active-directory-dotnet-webapp-multitenant-openidconnect.

[17] Y. Xiaoyong, T. Hongyan, L. Ying, J. Tong, L. Tiancheng and W. Zhonghai, "A Competitive Penalty Model for Availability Based Cloud SLA," *2015 IEEE 8th International Conference on Cloud Computing*, 2015, pp. 964-970, doi: 10.1109/CLOUD.2015.142.