Arabian Journal for Science and Engineering (2021) 46:9221–9235
https://doi.org/10.1007/s13369-021-05743-z

RESEARCH ARTICLE-COMPUTER ENGINEERING AND COMPUTER SCIENCE

TOSDS: Tenant-centric Object-based Software Defined Storage for Multitenant SaaS Applications

Aditi Sharma[1] · **Parmeet Kaur**[1]

Received: 4 September 2020 / Accepted: 20 May 2021 / Published online: 16 June 2021

Abstract
Enormous amounts of unstructured data such as images, videos, emails, sensors' data and documents of multiple types are being generated daily by varied applications. Apart from the challenges related to collection or processing of this data, its efficient storage is also a significant challenge since this data do not conform to any predefined storage model. Therefore, any enterprise dealing with huge unstructured data requires a scalable storage system that can provide data durability and availability at a low cost. The paper proposes a tenant-centric approach to develop an object-based software defined storage system for SaaS multi-tenant applications. We present TOSDS (Tenant-centric Object-based Software Defined Storage), a system that can efficiently meet the storage requirements of users or tenants with diverse needs who are using a multitenant SaaS application. The experimental verification of TOSDS illustrates its effectiveness in storage utilization as well as tenant isolation.

Keywords Multitenancy · Distributed database · Software defined storage · Location · Bin · Virtualization · Object storage

1 Introduction

The last decade has witnessed a focus on digitalization in all types and scales of industries. With each passing year, companies and individuals are moving their data from physical storage to online storage. Furthermore, in the aftermath of COVID-19 pandemic, even small industries have moved to the digital world, thus adding more data to the existing pool of Big Data [1]. In this digital era, it is more or less impossible to store or process the humungous data on a single machine; thereby leading to the need for distributed systems. For faster processing, designing an efficient distributed storage system is the need of the hour.

The most common existing and popularly used distributed environments involve Cloud based solutions for data storage [2]. Development and maintenance of system resources such as data storage infrastructure are cost-inefficient for individual users. On the other hand, since Cloud services follow a pay-as-you-go model, a customer pays only for the demanded services, thus reducing cost to a greater extent. The various popular models of cloud service delivery include Infrastructure as a service, Platform as a service and Software as a service. Software defined technologies such as SDNs (Software Defined Networks), SDDCs (Software Defined Data Centers) and SDS (Software Defined Storage) are leading to efficient implementation of the cloud services. Of these, SDS is emerging as the focus area for many research efforts due to its obvious importance in the era of Big Data. Software Defined Storage is a well-organized storage approach that provides flexibility and scalability in the system architecture along with better performance and cost efficiency. This is possible because SDS separates the software layer from the hardware of the processor. Software-defined storage (SDS) is a kind of virtualization technology for cloud storage. To improve availability of data, SDS uses an additional control layer having required software to integrate the resources [3].

System architecture of SDS has been depicted in Fig 1. In a traditional storage system like Hard Disk Drive (HDD), Phase Change Memory (PCM), Shingled Write Disk (SWD), Solid State Drive (SSD) etc., there exists a storage interface which needs to be changed depending on the dynamic environment [4]. This implies that this storage interface needs

✉ Aditi Sharma
aditi.sharma@jiit.ac.in

Parmeet Kaur
parmeet.kaur@jiit.ac.in

[1] Jaypee Institute of Information Technology, Noida, India

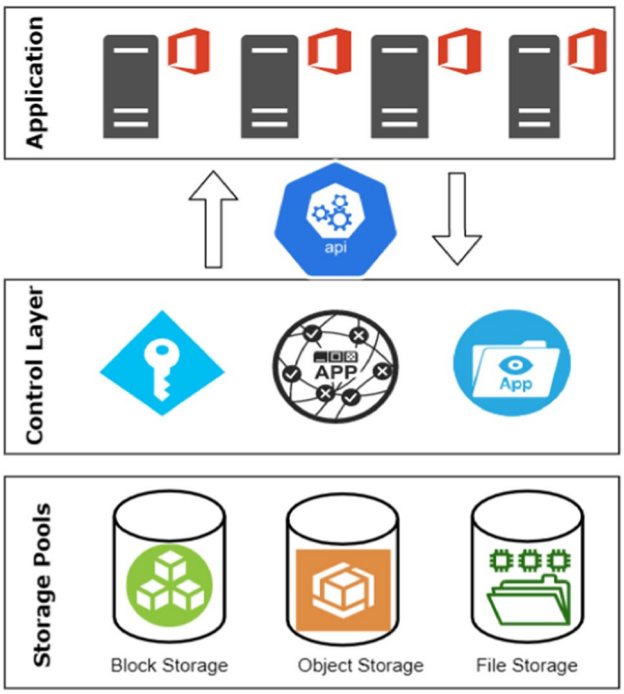

Fig. 1 Software Defined Storage Architecture

to be updated for different user requests involving data of varied types. Whereas, in SDS there is a control layer with defined set of rules for different data types to handle varied user requests with software APIs [5].

Motivated by the importance of SDS, this paper proposes a generic architecture for improving storage in a multitenant system using the concept of object-based software defined storage. Multitenancy is the most commonly used approach to build scalable SaaS applications. In case of SaaS model, multi-tenant architecture can be a cost-effective choice over a single-tenant architecture for building scalable applications. Further, a multi-tenant architecture can be used to build scalable storage systems for shared access to resources [6]. Multitenancy can effectively reduce the redundant data of each system by avoiding storage of duplicate data and it also avoids data inconsistency. Tenants use a common infrastructure but may have varied data attributes or features. Moreover, there can be variations in the data usage patterns, data access frequency rate, priority, etc. among the tenants. Considering this, it is postulated that segregation of these diverse tenants based on their distinguished features will lead to better storage allocation and improved system performance [7, 8].

This paper proposes an implementation of TOSDS, a tenant centric, software defined object-based storage system for SaaS applications. The architecture is illustrated with an example of an educational SaaS application, where it is assumed that universities, colleges, institutions, schools are the individual tenants. Apart from businesses, the educational sector has moved to digital platform during the Covid 19 pandemic [1, 9]. Earlier, it was just limited to Massive Open Online Courses (MOOC); however, online education will be a new normal in the near future. Thus, software as a service (SaaS) model of cloud computing can bring great advantage to the schools and universities. In a multi-tenant architecture, multiple universities or schools, termed as tenants, can work in a shared environment with access to shared study materials and related resources [10]. Apart from education, the presented work is suitable for other multitenant SaaS applications as well, such as those storing data of customers of various branches of banks; Event booking systems with Movie halls, Conference/Workshop, Adventure parks as tenants or a corporate with big data of its geographically distributed offices, etc.

The highlights of TOSDS are as follows:

- Object based Software defined storage (SDS) is used for implementation, thus ensuring durability and high availability.
- A tenant-centric approach for storage allows an enhanced user experience in terms of response time.
- Storage allocation for tenants is possible statically as well as dynamically.

The rest of the paper is structured as follows: The related work is discussed in Sect. 2. Design of the proposed system, TOSDS is presented in Sect. 3. Results of implementation are described in Sect. 4. Consideration of privacy requirements of tenants within the proposed system is discussed in Sect. 5. Finally, we conclude the article.

2 Background and Related Work

2.1 Object Based Software Defined Storage

Data storage was a straightforward task in traditional systems where applications involved limited size of structured data. However, massive influx of documents, files, videos, etc. into the digital world has sparked a debate to choose an appropriate data storage which can handle such exponentially growing volumes of unstructured data. There are three basic types of storage being used now-a-days, namely: File, Block and Object storage [4]. For a small enterprise, with limited storage requirements, *file storage* can be used to store data in a well-organized, hierarchical manner. It is user friendly, easy to operate and hence, generally used for local archiving and file sharing applications. In *block storage* approach, data are divided into blocks of fixed size, where each block has an individual address. Mostly enterprises, databases, email servers,

RAID, virtual machines employ block storage for better scalability as compared to file storage. In contrast to block storage, *object storage* strategy does not split data into blocks [11]. Instead, objects, each with a unique identifier, are used to encapsulate data and metadata. Due to its scalability and flexibility to handle unstructured data, popular online platforms, forums like Facebook, Amazon S3, OpenStack or Spotify [12], etc. use object storage for their implementation.

Unlike traditional storage, software defined storage (SDS) abstracts the hardware layer from the software layer of the system. In addition, it incorporates a control layer for better processing of software with the hardware [4]. Thus, an IT administrator has the flexibility to choose hardware from different vendors in a single storage system. This is in contrast to traditional storage systems where hardware infrastructure of a fixed type was available with any storage service. The advantage of SDS can be illustrated with a simple use case. Suppose there is a single datastore in a system that is used for data storage, but with time, data are exponentially growing, leading to the requirement for scalability in storage. In order to increase the storage capacity, traditional systems used the scale-up technique which focuses on increasing the system capacity by adding a new component such as disk, drive in the existing system. Therefore, this datastore can be upgraded to a more powerful and bigger datastore by adding more drives, making it capable to handle more data. On the other hand, SDS uses scaling-out method instead of scaling-up for achieving scalability. Nodes are added in parallel one at a time to the existing SDS cluster, i.e., scale-out method focuses on adding more components in parallel rather than just increasing a component's capacity [13].

Software defined storage offers the following advantages:(a) *Elasticity:* Since software and hardware layers are independent of each other, it is straightforward to upgrade or downgrade the system capacity. (b) *Cost Effectiveness:* System can be upgraded without replacing existing hardware. (c) *Vendor Neutrality:* IT administrator is free to opt for hardware from different vendors for building a single system. (d) *Flexibility:* Depending on the application requirement, different hardware can be used for different applications

Thus, the decoupling of hardware layer from the software layer in SDS allows upgrading or downgrading hardware without interfering with the storage system. Object storage is seen as the *de-facto* choice when highly scalable systems need to be developed for storage of unstructured data. In comparison, File or Block storage are preferred if data needs to be frequently fetched or updated. Therefore, this work proposes a system for Object based Software defined storage (SDS) system.

2.2 Literature Review

Education plays a very crucial role in the well-being of a nation and hence, various countries are investing in infrastructure such as cloud-based systems for the education sector. Authors of the work presented in [2] have comprehensively surveyed the importance of Cloud Computing in the field of education. They have explored various aspects including role of stakeholders such as faculties, students, parents as well as that of a University in a cloud-based education system. Unlike traditional education resources, SaaS based cloud model has been found to be a cost-efficient option as it follows pay-as-you-go model. It decreases the carbon footprint and provides a highly available and scalable model. The concept of multitenancy is quite popular with cloud storage; an example of which is illustrated in a multi-cloud [14] setup. This setup provides high availability, as different tenants' data with customized requirements is replicated and distributed between multiple clouds. However, this architecture is quite complex and hence, authors have presented a data management middleware platform which is responsible to provide abstraction between different cloud storage and designing policy for data distribution among respective clouds. For designing a multi-tenant architecture, different factors are considered such as data isolation among different tenants, data security, query optimization and query response time. It has been observed by the authors [15] that there is a trade-off between system performance and storage space.

SaaS applications are quite prevalent but still need to address few significant concerns [16]. For enterprise SaaS applications, authors of [16] have tried to explore three such important issues. Firstly, they discuss implementation of tenant context storage and propagation for which they use a thread-specific storage to implement tenant specific customization. The second issue considered in this work is connecting different logical tables together, which the authors address using schema-mapping. Lastly, the authors discuss integration of ORM (object-relational mapping) framework. A SaaS based Multitenant system architecture for educational institutions has been proposed in [9]. It considers various entities related to a University, such as colleges, research institutes, student unions as individual tenants. The proposed distributed database access platform uses SQL, hibernate and J2EE technology for its implementation. A multi-tenant database architecture using a column-based NoSQL datastore has been presented in our previous work in [17]. Different tenants work in a shared environment in the proposed multi-tenant system. In such a scenario, apart from providing high availability and scalability, isolation of these tenants is another important challenge. Therefore, this multitenant system architecture has been implemented using the concept of Materialized views

in Cassandra; thus, providing a shared environment along with a sense of isolation to the tenants.

The design of a scalable multi-tenant SDS system has been proposed in [18]. Authors have introduced a Hierarchical Bin Packing algorithm for dynamic allocation of tenant data load over the available storage space. This is a scalable approach in which tenants are hierarchically clustered based on multiple scenario-specific characteristics. Authors have extended their work in [19], which presents a Multi-tenant system for dynamic tenant load allocation. In this approach, to minimize network latencies, tenants are hierarchically arranged such that the tenants' data are stored in proximity to the designated application server. Instead of relational data, authors have focused on allocation of blob data. Two hierarchical bin-packing approximation algorithms that can work both for static and dynamic loads have been proposed in the paper. First algorithm is Hierarchical First-Fit Decreasing (HFFD) strategy and second one is the Hierarchical Greedy Decreasing strategy (HGD). These algorithms work similar to First- Fit Decreasing (FFD). Both of these algorithms are quite similar in their layout but give distinct results on execution over static data and dynamic data. This work also focuses to minimize the migration of tenants' data over time. Authors have also proposed dynamic variants of these algorithms; dHFFD and dHGD. It was concluded that dHFFD is best suited for static allocation. For dynamic allocation which may involve data migration between different bins, dHGD reduced the number of migrations in comparison to dHFFD. However, average bin usage of dHGD is significantly lower than dHFFD, thus having higher operative cost.

The work in [20] presents an analytical model using multi-core virtual machines hosting cloud SaaS applications. Every SaaS application provides some quality-of-service parameters (QoS) such as availability, durability, performance. Authors have proposed a model that can assess and estimate the count of VM required corresponding to the dynamic workload and can easily satisfy these QoS parameters. In this model, different SaaS applications are allocated separate VMs; thus, providing ease of scalability as it is easy to scale up by merely adding multi-core VMs and to scale down by just removing VM.

In traditional storage systems, available storage space of individual host server was required to be calculated beforehand, prior to initializing the storing process. This situation could become worse when it is required to check storage space for all the hosts storage server, leading to time complexity approximately equal to O(n) for n repositories. In contrast, SDSs use a Function table which stores updated available storage space for each individual host. This function table is kept at network switch. Thus, fetching available storage space is easier in the latter approach, which takes only 0(1) time to complete this task [4]. In this paper, authors have developed a SDS framework: "MinStor" which is based on Mininet emulator-an OpenFlow-based SDN simulator. This work also highlights the concept of "Software deployed" which is another associated concept to SDS. System resources and devices are controlled and managed by a separate software in the software define storage. In comparison, software deployed systems do not use any separate software or API to control and manage the resources; instead the hardware object controls all of these. Unlike traditional systems, in software deployed system the control layer can control all the resources regardless of their vendor disparities. This is possible because in the data layer, resources are substantially placed separately from the hardware.

SuperCell [21] is a SDS based system that uses IaaS (Infrastructure as a Service) model. Authors have presented a dynamic approach for handling growing data. This is a Ceph-based distributed storage system, which initially allocates basic requirements to the application with respect to storage size and availability as well as the desired throughput and response time. Post-deployment, SuperCell recommends consequent changes after observing user's requirement; thus, providing scalability in a cost-effective manner.

With an increase in diverse amount of information, use of traditional storage systems is not justifiable to overcome data challenges of scalability, integration, and flexibility [22]. With each passing year, growing storage space based on estimated demand will be superseded. Another key concern is to lower the overhead involved in handling heterogeneous storage system [23]. Authors have implemented a cloud storage system to accomplish the notion of software defined storage. This was achieved using heterogeneous storage systems like Hadoop HDFS, Ceph and swift on Open Stack, which were integrated using software APIs. A control service was then implemented on the controller to handle these storage services. In another work [24], authors have used cubic spline interpolation and distribution mechanisms to implement a scalable and on-demand cloud based SDS system. This can be achieved by using Nova Compute which is a component within the Open Stack.

Virtual machines in a cloud platform store data in the form of large files on networked storage servers [25]. Software-defined file system (SDFS) is been proposed to provide performance isolation by allocating resources at per-image-file granularity. Like SDS, here SDFS also has control plane and data plane. The control plane defines set of system calls to map tenant's performance requirement into the metadata of image files and data plane has a file-based scheduler which is used to handle storage resources customized according to tenant's requirement.

The work in [26] proposes an application for storing network-based digital evidence for crime scenes. For the implementation, authors have used Ceph which is a software defined storage, thus providing reliability, high scalability and security from tampering the digital evidence.

Another, recent application of SDS is optimizing resources of distributed grid computing model: A Large Ion Collider Experiment (ALICE) [27]. In this work, a SDS model based on cloud and edge computing has been presented to optimize the data storage of ALICE. Another software-defined storage system: *IOFlow* has been proposed in [28] for multitenant data centers. The main idea of this system is that it abstracts the data plane from the control plane. To implement this, the authors have used specific layers termed as "stages" to enforce the input output flow polices from the control plane. It offers low-level routing and classification services to control flows and it also operates under I/O bandwidth limits. *IOStack* [29] is another software-defined storage system for a multitenant object store unlike *IOFlow* which supported file system storage. Along with I/O policies, *IOStack* builds filters at the data plane as discrete components which can be used for data management techniques such as data reduction or data optimization on incoming object requests. In [30], authors have proposed *Crystal:* a multi-tenant software defined storage architecture in object store, implemented in OpenStack Swift. *Crystal* is suitable to handle heterogeneous applications with varied requirements as it decouples control policies from the data planes. In addition to policies and filters, it uses two more abstractions: inspection triggers and controller. Inspection trigger enables automatic execution of filters with the help of real-time metrics and metadata (size, type of the object etc.) from the objects. Controller can either be a simple rule or a complex algorithm to automate the execution of a filter. Distributed controllers, DSL (Domain-Specific Language) and CrystalAPI are used to build a centralized control plane.

The proposed system, TOSDS improves over the related work described in this section as follows:

1) TOSDS is a tenant centric object-based software-defined storage system that can handle both static as well as dynamic data. Software-defined storage systems are implemented to hasten and boost the performance of the system with a minimum cost involved.
2) The proposed system's main objective is to build an efficient storage system for multiple tenants having distinct data load with minimal data migration.
3) Most of the existing systems have not considered the tenant load characteristics, apart from load size, while allocating storage. The work closest to the proposed system is Tenant Defined Storage [18, 19] which has performed clustering of tenants based on only their locations. In comparison to [18, 19], our model focuses on the storage-specific characteristics of the tenants' data loads such as load size, priority and frequency of access of tenant's data for clustering tenants.

A comparison of the features of the proposed system, TOSDS, with existing systems is presented in Table 1.

3 Proposed System

In a multi-tenant system, different tenants with divergent data requirements work in a shared environment. The proposed data storage system follows a Shared Database, Same Schema Multitenancy model. Multitenant system architecture is generally more intricate than that of single tenant architecture. This may be attributed to sharing of resources by multiple tenants, while providing sense of data isolation to each tenant. This paper highlights the application of such a tenant-centric software defined architecture for educational SaaS applications. Different universities, colleges, institutions, schools are assumed to be individual tenants in this system. These tenants use a common infrastructure but may possess varied data attributes or features; data access frequency rate and priority. For instance, in a multitenant SaaS application, some tenants, i.e., schools or colleges, may use only the ERP services like fee collection, admissions, Hostel allotment etc. These are generally one-time services and their data is not fetched frequently from the storage. Whereas, there could be handful of colleges/institutions which are consuming content curation services, grading students, conducting online classes, providing study material etc. In the latter case, data will be frequently fetched from the storage and some services may have higher priority than the others. Considering this, it is essential to segregate these diverse tenants based on their features and requirements.

The work presents TOSDS, a tenant-based storage allocation model for a multitenant system. The underlying idea is to firstly classify the tenants of the application based on their features and storage requirements. Subsequently, storage allocation is performed on the basis of this classification. Each tenant is assumed to possess a set of attributes. Fig 2 shows the structure of the tenant node with the underlying functions that have been used to implement the proposed system. Each tenant has been assigned a unique identifier, referred to as *TNumber*. *Priority* denotes the significance of the tenant's data load. It may be specified by the tenant and vary as 1 (High), 2 (Medium) and 3 (Low); High, Medium and Low, respectively. *Frequency* measures regularity in the fetching of the data load which is scaled from 1 to 10. *Class-label* attribute stores value 1 or 0 depending on the cluster to which the tenant belongs. Dynamic array has been used to store bin numbers assigned to store a tenant's data.

The functionality of the proposed system is categorized according to the type of tenant load. Initial experiments were performed assuming tenant data load to be static. The subsequent experiments also considered dynamic tenant data, i.e., which shrinks or expands over time depending on tenant

Table 1 Comparison of proposed system with existing systems

Related Works	Type of Storage System	SaaS Application	Handles dynamic data load	Special consideration or optimization	Implementation Algorithms	Minimize data migration	Platform for Simulation/ implementation	Performance Measures
Tenant Defined Storage [18, 19]	Blob Storage	Yes	Yes	Tenants are clustered based on their location	Hierarchical Bin Packing algorithm	Yes	C	Average LUN utilization, reallocations metrics, average distance between the nodes
SDStorage [4]	Storage Files	No	No	None	Simulation of SDS	No	Mininet	Memory usage,CPU time
Yang, C. T., [23]	Swift and Ceph	No	Yes	None	Cubic spline interpolation and distribution mechanisms	No	OpenStack	Network throughput &disk writing speed
Crystal[30]	Object Store	No	Yes	None	Simulation of SDS	Yes	OpenStack Swift	IO operations per second, bandwidth usage
SuperCell [21]	Ceph	No	Yes	None	Recommendation engine	No	OpenStack	Response time
Our Model (TOSDS)	Object based	Yes	Yes	Tenants are clustered according to the features of the data load	Bin-allocation algorithm for Static & Dynamic data	Yes	C, Python	Average Bin utilization, Data Split ratio, Tenant Isolation Ratio

requirements; this may lead to migration of data from one bin to another.

3.1 Bin Allocation for Static Load

Firstly, we describe storage allocation for static tenant loads. Fig 3 lists the algorithm showing working of the proposed framework for static load. The proposed framework comprises of the following phases:

3.1.1 Clustering of the Tenants

Different tenants with varying requirements and specifications come together in a multitenant environment. With respect to data storage, each tenant may have different features such as size of data, priority of data, and access rate or frequency of data access. In order to draw a fine line between these tenants, the proposed approach employs a feature-based clustering on the Tenant Load. In this phase, tenants are classified into two groups: ACTIVE and PASSIVE; encoded as "1" and "0", respectively. ACTIVE tenants are those who have higher data requirements than PASSIVE tenants.

3.1.2 Allocation of Bins to the Tenant

After the clustering task is completed, the next phase assigns storage bins to the tenant using a separate Bin allocation policy for Active and Passive tenants.

For Active tenants (Algorithm 1 Fig. 3), the tenant load/ data is compared with a predefined metric, referred to as the *Allocation Threshold* ($\mathcal{T}$). $\mathcal{T}$ determines the maximum amount of space that can be allocated for data storage in a bin. It has been used to cater to dynamic load handling, as discussed in the next section. We have assumed $\mathcal{T}$ as the difference between the size of the bin (Size_{Bi}) for storing a tenant's load and the data value of the Tenant with minimum(MIN_T) data load.

$$\text{ş} = MIN_T \tag{1}$$

$$\text{ٹ} = Size_{Bi} - \text{ş} \tag{2}$$

$\mathcal{T}$ is used to determine how much bin memory can be allocated to these ACTIVE tenants. Remaining bin memory has been termed as "Stash", denoted by ş.

If tenant load is less than the allocation threshold, the proposed approach allocates a dedicated bin to the tenant. In this case, the assigned bin's load is equivalent to the tenant's load (Steps 4 and 5 of Algorithm 1). However, if the tenant's load is higher than $\mathcal{T}$, a dedicated bin is firstly assigned to the tenant, using a value equal to allocation threshold as the bin load (Step 6 of Algorithm 1). The remaining data load of

```
struct node
{
    int data;                  // Tenant Load
    int TNumber;               // Tenant ID
    int Priority;              // specifies data Priority of the Tenant
    int frequency;             // specifies data fetching Frequency of the tenant
    int Classlabel;            // ClassLabel assigned after clustering into ACTIVE(1)or PASSIVE(0)
    int *bin;                  //For storing bin(s)
    struct node *next;         // Points to the next tenant in the list
};
void binallocation(int T);     // Function to allocate bin to the Tenant
int bestFit(int T);            // Function implementing Best Fit Binning Algorithm
```

Fig. 2 Structure of a Tenant Node

the tenant is again compared with T and the above allocation process is repeated (Step 11 of Algorithm 1).

Similarly, bin allocation is performed for the passive tenants by comparing its load with the allocation threshold, T. If a passive tenant's load is higher than T then a dedicated bin is allocated for the load similar to the case of active tenant and the process is repeated (Steps 18-21 of Algorithm 1). However, if a passive tenant's n this case, when the tenant load value is less than that of allocation threshold, instead of assigning a dedicated bin, the best fit bin packing algorithm is applied, as illustrated in Algorithm 2 of Fig 3.

The best-fit bin packing technique [31], listed in Algorithm 2 of Fig 3, can place multiple tenants in a single bin. Here the tenant data is assigned strategically to that bin which will have the smallest empty space after bin allocation. For accomplishing best fit binning, firstly the available space in already existing bins is checked. If adequate space is available in any single bin, the tenant is allocated that bin itself and the bin's space is updated accordingly (Steps 6-11 of Algorithm 2). Otherwise, if the tenant's data is greater than the available space in any of the bins, a new bin is allocated and data are stored in that bin (Steps 12–14 of Algorithm 2). If the bin contains space equivalent to allocation threshold, the value of available space in the bin and information regarding the bin number allocated to the tenant are updated (Steps 15–18 Algorithm 2). In the alternate case, best fitting bin is selected for the tenant data (Steps 19–22 of Algorithm 2)

3.2 Bin Allocation for Dynamic Load

Cloud computing services are characterized by their ability to scale up or down depending on user requirements. Therefore, it is expected that any SDS system will encounter dynamic data storage loads. This section addresses the storage allocation policy of the proposed system when tenant data is considered as dynamic, i.e., it may increase or decrease over time. Dynamic load allocation can trigger data migration during reallocation or deallocation of the tenant data. At this stage, role of previously defined parameter, Allocation threshold (T) comes into picture. The definition of T ensures that adequate space is left unallocated in each bin during static allocation. The size of this unallocated space is equal to the minimum of all existing tenants' load. The assumption used here is that an increase in any one tenant's load will not exceed any other tenant's total load. Thus, to minimize the migration of data, it has been assumed that during expansion of tenant's load, the growth of the data will be adjusted in the stash (s) first and then new bin will be allocated if required.

On the other hand, if shrinking of tenant data occurs, storage needs to be deallocated and this implies that space is now available in the tenant's current bin. Hence, some type of compaction is possible; either data can be migrated from another bin to this bin or this tenant's data can be migrated to another bin. This compaction can lead to freeing up of resources in the system.

To explain it further, let's take a scenario in Fig 4. Suppose, in a Multitenant environment, with bin size of 1000 units and T as 700 units, there is a tenant "X" with data load of 1200 units. Using our approach, this tenant is clustered let's say as an Active tenant. Initially, two dedicated bins are allocated for this tenant with bin load as 700 units and 500 units, respectively, using the static allocation policy.

With time, X's data start shrinking and its data load is reduced to 900 units. Thus, the X's data in the second bin need to be deallocated and is reduced to 200 units. In the proposed approach, at this time data of 200 units will be migrated to Stash storage (which is 300 units in this case) of the first assigned bin. Now, complete data of X is stored in a single bin; thereby, increasing storage utilization.

Algorithm 1: Bin allocation using Best Fit binning

Input:
TenantLoad: Tenant Data Load
Priority and Frequency of Data Usage: Tenant information
Variables:
BinLoad: Amount of space allocated in a bin
LeftData: Data of a tenant remaining to be allocated
Ţ: Allocation Threshold
Remaining_Data_list: list of nodes, each node stores amount of a tenant's data left to be allocated
Output:
Average Bin Usage

```
1.  Analyze Tenant Data (Priority, Frequency of Data) and Cluster Tenants into Active or Passive
2.  For each Tenant do
3.      If (Tenant ==ACTIVE TENANT) {
4.          If (TenantLoad< Ţ) {
5.                  Allocate Dedicated Bin
6.                  BinLoad=TenantLoad}
7.          elseIf (TenantLoad>= Ţ) {
8.                  Allocate Dedicated Bin
9.                  BinLoad= Ţ
10.                 LeftData=TenantLoad-Ţ
11.             If (LeftData>=Ţ ) goto step 8
12.                 else {
13.                     Allocate Dedicated Bin
14.                     BinLoad = LeftData}
15. }  //end ACTIVE TENANT
16.     else If (Tenant == PASSIVE TENANT) {
17.         If (TenantLoad>=Ţ ) {
18.                 Allocate Bin
19.                 BinLoad= Ţ
20.                 LeftData=TenantLoad- Ţ
21.                 If (LeftData>=Ţ ) goto step 18
22.                 else append LeftData to a list,Remaining_Data_list}
23.                     } //end PASSIVE TENANT
24. endFor
25. Apply Best Fit Bin Packing on Remaining_Data_list
26. Calculate Average Bin Usage
```

Algorithm 2: Best Fit binning

Input:
Remaining_Data_list: list of nodes, each node stores amount of a tenant's data left to be allocated
V*ariables*:
bi: current bin number
min: minimum space available in a bin
newnode: pointer to start of the list, Remaining_Data_list
avail_space: list of nodes corresponding to number of bins, each node stores amount of available space in a bin
Output:
binno: number of bins allocated for load

Fig. 3 Algorithm showing working of the proposed framework for static load

```
1.  Set binno = 0;
2.  for each Tenant do
3.  Initialize min = Ƭ + 1
4.  Set bi = 0;
//Initialize minimum space left and index of best bin
5.   for  j = 0 to binno do
6.   if(avail_space[j]>=newnode→data &&(avail_space [j]-newnode→data)<min) then
7.              {bi = j;
8.               min = avail_space [j]- newnode→data;}
9.   end For
10.  if (min == Ƭ + 1) then //if the tenant's data is greater than the available space in any of the bins,
11.             {Allocate a new bin with number as binno
12.              Add newnode->data to it
13.              avail_space [binno] = Ƭ - (newnode→data);
14.              newnode→bin= binno;
15.              binno++; }
16.  else
17.             {Assign tenant's data to the best fitting bin
18.              avail_space [bi]=avail_space [bi] -newnode→data;
19.              newnode→bin=bi;}
20.  endFor;
```

Fig. 3 (continued)

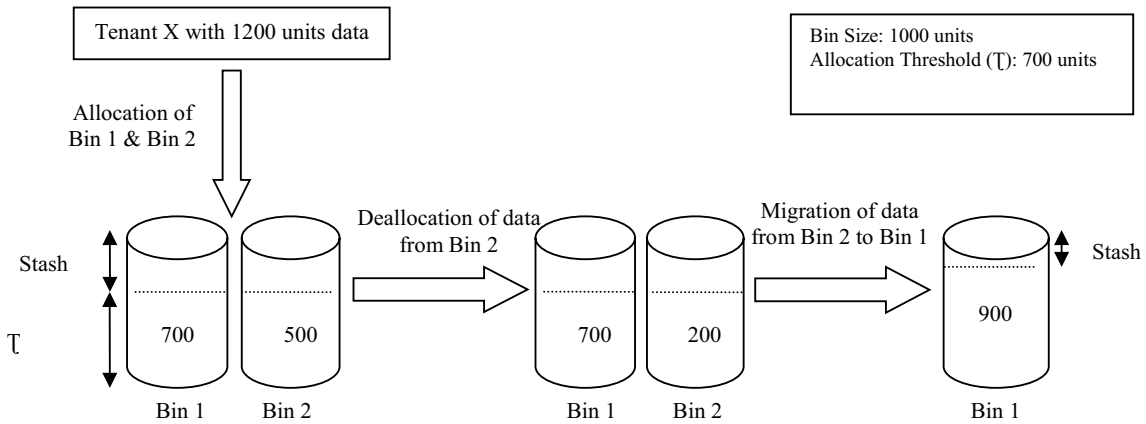

Fig. 4 Migration of Active Tenant data during deallocation of bin storage

4 Simulation Experiments and Results

The performance of TOSDS with respect to effective storage utilization and tenant isolation has been evaluated using simulation experiments. All experiments were executed on a Windows server with an Intel Core i5 CPU (1.60 GHz) and 8 GB of DDR3 memory. A synthetic data set of 250 different tenants with different data loads and requirements or features has been used. The important tenant data load features taken into account are size of data, priority of data and frequency of data fetching. The initial step in the implementation was to cluster the tenants into ACTIVE and PASSIVE categories according to their distinct data load features. Structure of the tenant object has been explained in Fig 2. *K*-means algorithm was used to perform clustering of tenants. Next task was to perform bin allocation for the tenants' data for static as well as dynamic data loads.

TOSDS proposes a bin allocation strategy for Active Tenant data. In this approach, a dedicated bin is allocated to an Active Tenant after comparing the data load with the *Allocation Threshold* (τ). This proposed bin allocation strategy has been designed as such to minimize data migration that may occur during dynamic load allocation. For the bin allocation of Passive tenant data, the Best Fit Binning strategy was used, in which the algorithm can allocate single bin to multiple tenants. This algorithm is designed such that it calculatingly assigns tenant's data to that bin which has the smallest empty space left after the bin allocation.

4.1 Evaluation Metrics for the Proposed System

This section discusses the different evaluation metrics that have been used to evaluate the proposed system.

4.1.1 Average Bin utilization

There can be different measures or metrics to determine how efficiently data storage of the system has been utilized. One of the most commonly utilized evaluation metrics for this is to calculate the average bin utilization which corresponds to the ratio of sum of all tenants' data load(s) to the total bin storage used by the tenants.

$$\text{Avg Bin Usage} = \frac{\sum_1^n TD_i}{\text{binno} \times B} \tag{3}$$

where TD_i is the data of the tenants ($1 \leq i \leq n$), n is the number of tenants in the multitenant system and B corresponds to the size of the bin.

For initial experiment, for static load, tenant data set taken for the simulation ranges from 0–800 units for different tenants, and average data was of 460 units. Considering the average load size and the allocation threshold, bin sizes have been varied starting from 700 units. This ensures that the data of a tenant with average data load will fit into a single bin, thereby avoiding splitting of data. Subsequent experiments have been performed to assess scalability of the system by increasing the average tenant load as well as the bin sizes. The dynamic allocation method ensures that even if the data size increases multiple times, it can be allocated efficiently; as illustrated by the following results

It was observed (Fig 5) that bin usage firstly increased when bin size was increased from 700 units and reached its peak value at 1100 units. Bin size was taken as maximum 2000 units to show the effect of bin size from minimum load size to up to three times the average load, which was approximately 450 units. Even on increasing bin size beyond 2000 units bin utilization did not improved as can be observed in Fig. 5. This is due to the fact that larger space was left unallocated in each bin upon increasing the bin size. We have used 1100 units, which we found empirically best for further experiments. After evaluating this, it was concluded that by merely increasing the bin size, desired average bin usage cannot be achieved. During the experiments performed under the proposed system for static data, the highest average bin utilization of around 76% is achieved for a bin size of 1100 units. It was observed that the smaller to medium bin sizes resulted in higher average bin usage in comparison to larger bin sizes.

Next experiment accessed the scalability of the system by taking dynamically increasing load. The effect of dynamic data and load on average bin usage is depicted in results of Fig 6. As expected, according to the proposed algorithm for dynamic data handling, it was observed that initially, bin utilization increased for smaller bin sizes and gradually started decreasing. This is also illustrated in sect. 3.2, when data load is increased dynamically, the growth of the data will be adjusted in the stash (s). Once the stash is full, only then new bin will be allocated to the tenant. Thus, the proposed system handles dynamic data with minimum migration.

The next experiment evaluated the average bin usage of the system with respect to different types of data loads. Tenant load has been categorized according to its size, namely

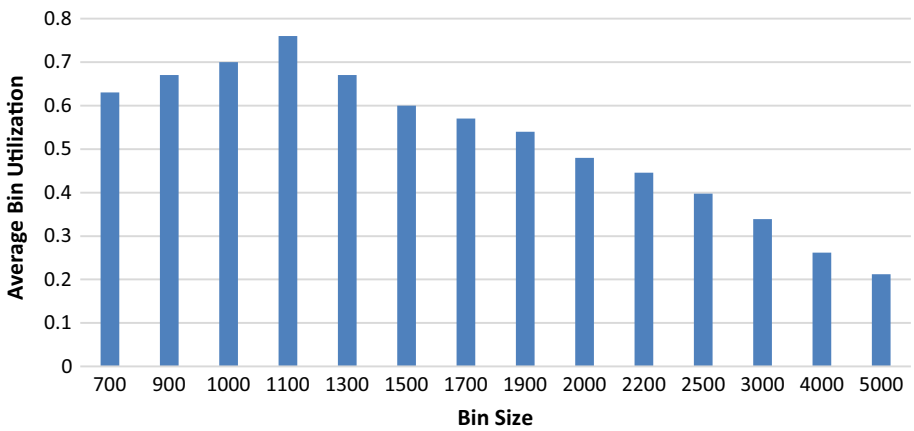

Fig. 5 Average Bin Usage w.r.t varied Bin Size for Static Data

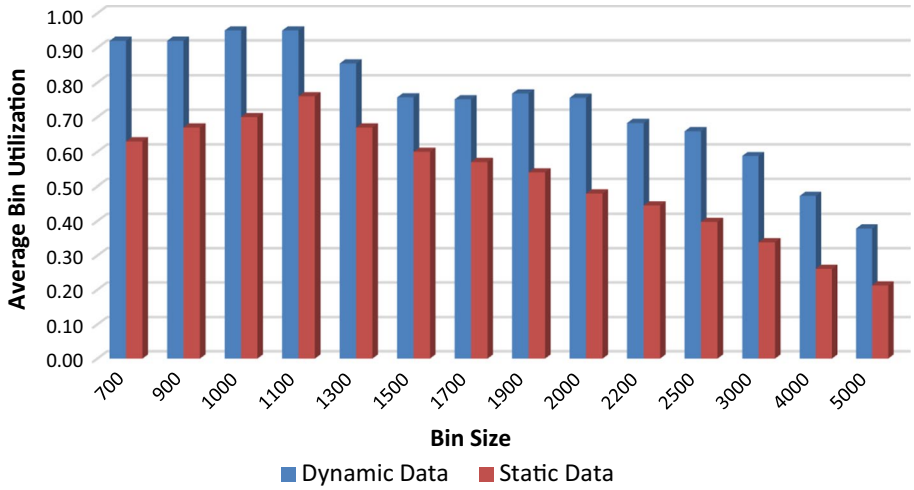

Fig. 6 Average Bin Usage w.r.t varied Bin Size for Dynamic Data v/s Static Data

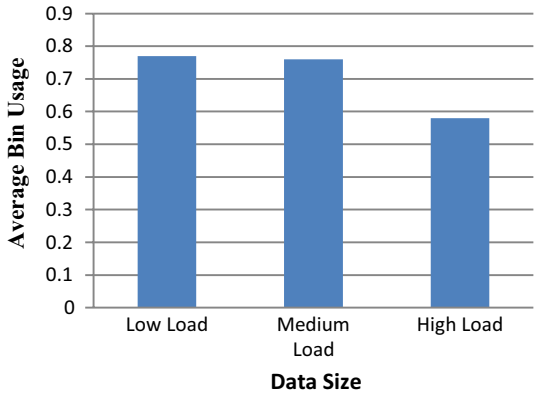

Fig. 7 Average Bin Usage v/s Load Size

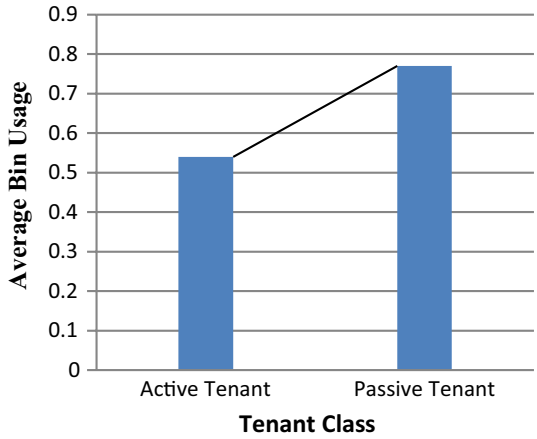

Fig. 8 Average Bin Usage between Tenant Class

High, Medium and Low. From Fig 7, it may be observed that average bin usage of tenants with low and medium data load was approximately at par but was greater than that of tenants having high data load. Subsequently, average bin usage was also calculated for the two categories of tenants, i.e., Active and Passive tenants. Fig 8 illustrates that average bin usage of Active tenants is lower by approximately 22%in comparison to Passive tenants. This is due to the fact that Active tenants are expected to increase their loads with time and hence, they are allocated dedicated bins; thereby, lowering the bin usage. However, this lowered bin usage for Active tenants is compensated by improved data isolation and lower split ratio metrics, as will be seen subsequently.

The proposed system, TOSDS aims to utilize the least number of bins possible for a given load and avoid data migration. To illustrate this, the next experiments were conducted considering data loads to be dynamic. During the experiments, we decreased the Active tenant load, thus resulting in deallocation of data from allocated bins. When the data load of a tenant shrinks, the proposed system attempts to migrate this tenant's data from multiple bins to single bin by using the available stash. Average bin usage was calculated for this experiment and compared with the same for best fit algorithm on the same data. As shown in Fig 9, the average bin usage was approximately similar in both the cases. Comparison between average bin usage for TOSDS, as calculated before migration of active tenant's data and after migration, has also been shown in Fig 10.

4.1.2 Data Split Ratio

Tenants may have data loads greater than the size of a single bin. In such cases, splitting of tenants' data into multiple bins is unavoidable. Data split ratio represents the fraction of tenants whose data have been split into multiple bins to the total number of tenants.

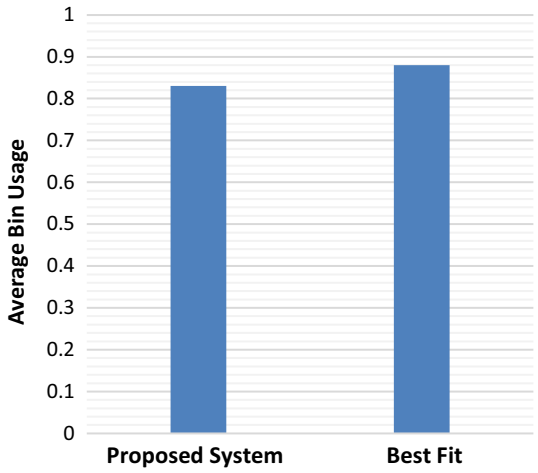

Fig. 9 Average Bin usage after decreased Active Tenant load

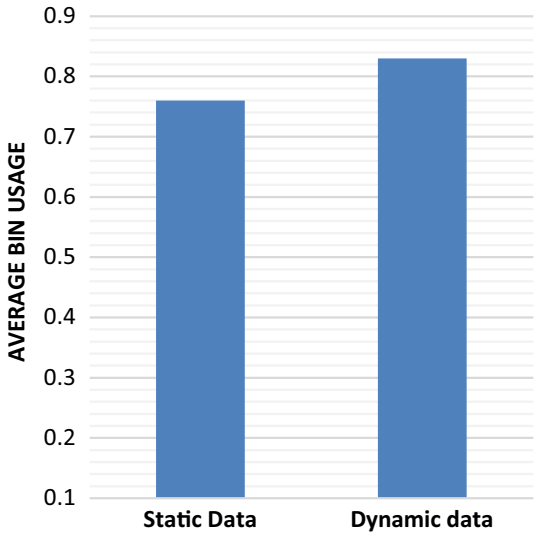

Fig. 10 Average Bin Usage for Static v/s Dynamic Data

$$\text{Ratio}_S = \frac{T_{\text{Split}}}{T_N} \tag{4}$$

where T_{Split} corresponds to the number of tenants having data distributed into different bins and T_N is the total number of tenants in the system.

Split ratio has been considered as one of the important evaluation metrics. Comparison between the spilt ratio of static v/s dynamic data has been depicted in Fig 11, it can be noted that split ratio of dynamic data is 24% higher than that of static tenant's data. In our architecture, there can be multiple applications (tenants) such as different colleges, universities. If the split ratio is more, this signifies that there are more such tenants whose data have been split into multiple bins and it may require more time to fetch complete data

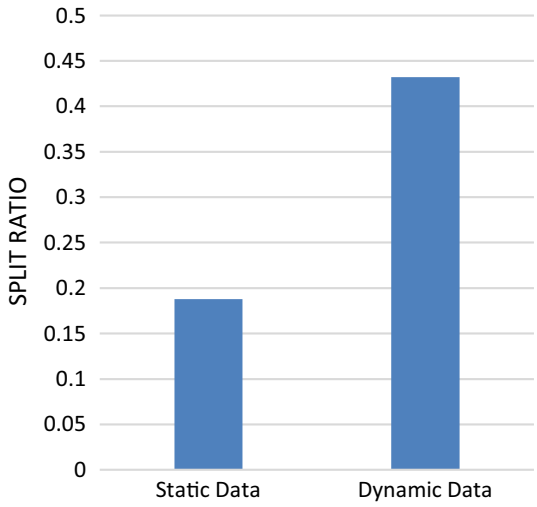

Fig. 11 Split ratio for Static v/s Dynamic Data

of a single University/college from multiple bins. Thus, as the data increases, as in case of dynamic load, split ratio also increases, thereby making it slower to fetch data.

4.1.3 Tenant Isolation Ratio

In the proposed SaaS application, different Universities/colleges are the tenants working in a shared environment. Along with shared data, different colleges may require to store customized data as well. For instance, in a multitenant application, shared services used by multiple tenants can be ERP services like fee collection, admissions, Hostel allotment, etc. Whereas, there could be handful of colleges/institutions which want to use customized services like content curation services, grading students, conducting online classes, providing study material, etc. In such a scenario, isolation of data among these different colleges is of utmost importance. We can calculate degree of isolation between these tenants using this metric.

$$\text{Ratio}_1 = \frac{T_D}{T_s + T_D} \tag{5}$$

where T_S is the number of tenants sharing bins with other tenants and T_D is the number of tenants having dedicated bin for storage. The tenant isolation ratio metric helps us to determine the degree of isolation among different tenants on different data sets considered in the experiment, refer Fig. 12.

Larger is the tenant isolation ratio, more will be the degree of isolation; implying that tenants are more isolated in the data set with higher tenant isolation ratio. It was experimentally observed that for a given dataset, degree of isolation remains unmodified regardless of the bin size or

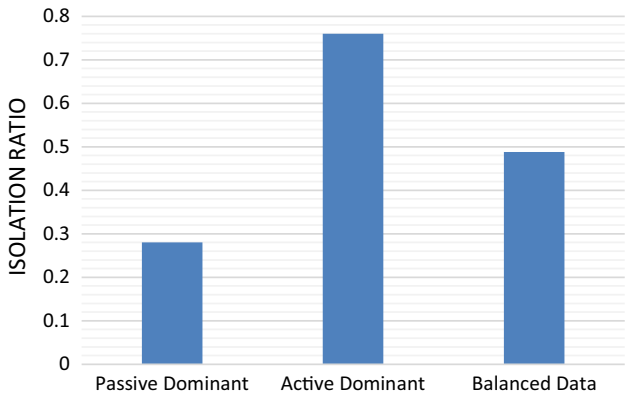

Fig. 12 Tenant Isolation Ratio for varied data sets

the nature of data. Hence, we calculated the tenant isolation ratio for different data sets, varying in fraction of Active and Passive tenants. It was observed that for TOSDS, the tenant isolation ratio was highest for the dataset having more Active tenants and least for the dataset with more Passive tenants. Also, it can be seen from Fig 12 that dataset having Active dominant tenants was 48 and 27% greater than that of Passive dominant tenants and balanced tenants, respectively.

5 SLA Requirement Privacy

TOSDS focuses on the characteristics of data load while allocation of storage to tenants. Another important Service Level Agreement (SLA) objective in a multi-tenant environment is *privacy*; however, it can be ensured only at a cost to the tenant. To provide privacy guarantees, TOSDS services can be offered in three different categories. The tenants can be classified according to their desired level of privacy, namely- free service tier, basic service tier and premium service tier. To avail a higher level of privacy guarantee, a tenant in the free trial phase can upgrade to a higher tier by paying the corresponding price. Table 2 illustrates the difference in privacy policy, bin sharing and data isolation between tenants of different tiers. Free service tier is level 1 tier which is the most vulnerable tier since privacy is not considered. Here, any number of tenants can have a shared storage space, thus providing no isolation at all. On the other hand, basic service tier is level 2 tier that provides privacy since data are shared among some trusted tenants only. Highest level tier is the premium service tier with highly strict privacy policy. Here, each tenant's data are stored in isolation from other tenants' data.

In a multitenant environment different tenants work in a shared environment. Individual tenants can have different SLA requirements like security, privacy, price, characteristics of the data load, location of the tenants, response time or budget, etc. of tenants in a cloud environment. The proposed system currently is limited to consideration of the characteristics of the data load such as load size, priority and frequency of tenant's data. We have not considered the scenario where multiple tenants may have separate SLA requirements. However, using the same approach, TOSDS can be customized easily for satisfying other related SLA requirements such as response time or budget of tenants in a cloud environment.

Table 2 Privacy as the SLA requirement

Type of Tier	Free service tier	Basic service tier	Premium service tier
Privacy policy	No privacy policy defined	Basic privacy provided	Highest level Privacy policy
Bin Sharing	Any number of tenants can share bin	Only trusted or related tenants may share bins	Dedicated bins for all the premium members
Data Isolation	Data isolated not guaranteed among tenants	Data isolation between untrusted or unrelated tenants	Data isolation is assured.Tenants who opted for premium service may have sensitive or confidential data, thus data is isolated among tenants, only authorised tenant can access the data
Example- Consider an application where large amounts of personal and financial data about banks' customers is required to be handled. Here each bank with corresponding branch will be considered as a tenant	Shared bins can be allocated for data of all the bank branches	Shared bins can be allocated for data of different branches of the same bank but data of different banks is not allocated shared bins	Data of each tenant is allocated separately from other tenants

6 Conclusion

The paper presented TOSDS, an object-based software defined storage approach for a multitenant system. In this system, tenants have been placed into two clusters, Active Tenants and Passive Tenants, according to the features of their data loads. The system employs distinct strategies to assign tenant's data to bin(s); bin allocation for Active tenants is performed differently from that of Passive tenants. Novel bin allocation algorithms for determining appropriate bin storage location for static as well as dynamic data loads have been presented. The system performance has been evaluated based on metrics of bin usage, data load's split ratio and tenant isolation ratio. It has been observed that average bin usage of the system is comparable to the Best-fit bin storage algorithm and improves in case of dynamic data loads. Further, experiments performed with respect to different data loads showed that average bin usage of tenants with low and medium data load was approximately at par but was greater than that of tenants having high data loads. For a multitenant system architecture, isolation among different tenants is essential. Simulation experiments verified that within the same dataset, degree of isolation is high and remains constant regardless of the bin sizes or the nature of data. Although the system cannot avoid splitting of data with size greater than bin sizes, only 18% tenants' data is split in case of static load and 42% tenants' data is split when data load is dynamic. The future work will focus on further reducing data split as well as data migration among bins for dynamic loads.

References

1. Teräs, M.; Suoranta, J.; Teräs, H.; Curcher, M.: Post-Covid-19 education and education technology 'Solutionism': a seller's market. Postdigi. Sci. Educ. (2020). https://doi.org/10.1007/s42438-020-00164-x
2. Katiyar, N.; Bhujade, R.: A survey: adoption of cloud computing in education sector. Int. J. Comput. Trends Technol. **60**(1), 15–25 (2018). https://doi.org/10.14445/22312803/ijctt-v60p102
3. Parab, N.N.: Software defined storage. December 2018, 0–6 (2019). https://doi.org/10.13140/RG.2.2.26934.75848
4. Darabseh, A.; Al-Ayyoub, M.; Jararweh, Y.; Benkhelifa, E.; Vouk, M.; Rindos, A.: SDStorage: a software defined storage experimental framework. In: Proceedings—2015 IEEE International Conference on Cloud Engineering, IC2E 2015, pp. 341–346 (2015). https://doi.org/10.1109/IC2E.2015.60
5. Sahu, H.; Singh, N.: Software-defined storage. Innov. Softw.-Defin. Netw. Netw. Funct. Virtualiz. **4874734**, 268–290 (2018). https://doi.org/10.4018/978-1-5225-3640-6.ch013
6. Sellami, W.; Hadj Kacem, H.; Hadj Kacem, A.: Dynamic provisioning of service composition in a Multi-Tenant SaaS environment. Springer, US (2020)
7. Jia, R.; Yang, Y.; Grundy, J.; Keung, J.; Hao, L.: A systematic review of scheduling approaches on multi-tenancy cloud platforms. Inf. Softw. Technol. **132**, 106478 (2021). https://doi.org/10.1016/j.infsof.2020.106478
8. Kalra, S.; Prabhakar, T.V.: Multi-tenant quality attributes to manage tenants in SaaS applications. In: Proceedings–2020 IEEE International Conference on Software Architecture Companion, ICSA-C 2020, vol. **128**, pp. 83–88 (2020). https://doi.org/10.1109/ICSA-C50368.2020.00025
9. Xu, J.; Li, X.; Zhao, X.: Design of database architecture in the saas-based multi-tenant educational information system. In: ICCSE 2011—6th International Conference on Computer Science and Education, Final Program and Proceedings, 128(Iccse), pp. 114–119 (2011). https://doi.org/10.1109/ICCSE.2011.6028597
10. Fithri, D.L.; Utomo, A.P.; Nugraha, F.: Implementation of SaaS cloud computing services on e-learning applications (case study: PGRI foundation school). J. Phys. Conf. Ser. **1430**(1), 012049 (2020). https://doi.org/10.1088/1742-6596/1430/1/012049
11. Contel Bradford.: Storage wars: file vs block vs object. Storage Recovery Zone by StorageCraft (2020). https://blog.storagecraft.com/object-storage-systems
12. Thomas, P.: A guide to clouds: object, file, and block. Backblaze (2020). https://www.backblaze.com/blog/object-file-block-storage-guide\
13. https://www.datacore.com/software-defined-storage/
14. Rafique, A.; Van Landuyt D.; Lagaisse B.; Joosen W.: Policy-driven data management middleware for multi-cloud storage in multi-tenant SaaS. In: Proceedings-2015 2nd IEEE/ACM International Symposium on Big Data Computing. BDC 2015, no. ii, pp. 78–84 (2016). https://doi.org/10.1109/BDC.2015.39.
15. Ahamed, I.; Mohammed, A.; Abu-Elkheir, M.: A hybrid multi-tenant database schema for multi-level quality of service. Int. J. Adv. Comput. Sci. Appl. **5**(11), 132–139 (2014). https://doi.org/10.14569/ijacsa.2014.051123
16. Liao, C.F.; Chen, K.; Chen, J.J.: A service framework for multi-tenant enterprise application in SaaS environments. In: ICSOFT-EA 2014–Proceedings of 9th International Conference on Software Engineering Applications, pp. 5–13 (2014). https://doi.org/10.5220/0004995300050013.
17. Sharma, A.; Kaur, P.: A multitenant data store using a column based NoSQL database. In: 2019 12th International Conference on Contemporary Computing, IC3 2019, ii, 1–5 (2019). https://doi.org/10.1109/IC3.2019.8844906
18. Maenhaut, P.; Moens, H.; Volckaert, B.; Ongenae, V.; De Turck, F.: Design of a hierarchical software-defined storage system for data-intensive multi-tenant cloud applications. In: 2015 11th International Conference on Network and Service Management (CNSM), pp. 22–28 (2015). https://doi.org/10.1109/CNSM.2015.7367334
19. Maenhaut, P.; Moens, H.; Volckaert, B.; Ongenae, V.; Turck, F.D.: A dynamic Tenant-Defined Storage system for efficient resource management in cloud applications. J. Netw. Comput. Appl. **93**, 182–196 (2017)
20. El Kafhali, S.; Salah, K.: Performance analysis of multi-core VMs hosting cloud SaaS applications. Comput. Stand. Interfaces **55**, 1339–1351 (2018). https://doi.org/10.1016/j.csi.2017.07.001
21. Uehara, K.; Xiang, Y.; Chen, Y.F.R.; Hiltunen, M.; Joshi, K.; Schlichting, R.: Supercell: adaptive software-defined storage for cloud storage workloads. In: Proceedings of 18th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing. CCGRID 2018, vol. **128**, pp. 103–112 (2018). https://doi.org/10.1109/CCGRID.2018.00025.
22. Peters, M.; Keane, M.: Key reasons to use software-defined storage—and how to get started. ESG Inc., Milford, Connecticut, White Paper (2015)
23. Yang, C.T.; Chen, S.T.; Chan, Y.W.; Shen, Y.C.: On construction of a cloud storage system with heterogeneous software-defined

storage technologies. Hum.-Cent. Comput. Inf. Sci. (2019). https://doi.org/10.1186/s13673-019-0173-x
24. Wan, J.; Tang, S.; Shu, Z.; Li, D.; Wang, S.; Imran, M.; Vasilakos, A.V.: Software-Defined Industrial Internet of Things in the Context of Industry 40. IEEE Sens. J. **16**(20), 7373–7380 (2016). https://doi.org/10.1109/JSEN.2016.2565621
25. Liu, J.; Wang, F.; Zeng, L.; Feng, D.; Zhu, T.: SDFS: A software-defined file system for multitenant cloud storage. Softw.-Pract. Exp. **49**(3), 361–379 (2019). https://doi.org/10.1002/spe.2663
26. Mohammad Faruq, A.; Mukhammad Andri, S.; Yudi, P.: Clustering storage method for digital evidence storage using software defined storage. IOP Conf. Ser. Mater. Sci. Eng. **722**(1), 012063 (2020). https://doi.org/10.1088/1757-899X/722/1/012063
27. Loncar, P.; Gotovac, S.: Software-defined storage optimization of distributed ALICE resources. In: 2020 28th International Conference on Software in Telecommunications and Computer Networks, SoftCOM 2020, vol. **128** (2020). https://doi.org/10.23919/SoftCOM50211.2020.9238270
28. Thereska, E.; et al.: Ioflow: a software-defined storage architecture. In: Proceedings of ACM Symposium Operating Systems Principles, pp. 182–196 (2013)
29. Gracia-Tinedo, R.; et al.: IOStack: software-defined object storage. IEEE Internet Comput. **20**(3), 10–18 (2016). https://doi.org/10.1109/MIC.2016.46
30. Gracia-Tinedo, R.; Sampé, J.; Zamora, E.; Sánchez-Artigas, M.; García-López, P.; Moatti, Y.; Rom, E.: Crystal: software-defined storage for multi-tenant object stores. In: Proceedings of the 15th USENIX Conference on File and Storage Technologies, FAST **2017**, pp. 243–256 (2019)
31. Korte, B.; Vygen, J.: Bin-Packing. In: Combinatorial Optimization. Theory and Algorithms, vol. 21 of Algorithms and Combinatorics, pp. 407–422. Springer, Berlin (2000)