International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 8 (2016) pp. 266-274
© MIR Labs, www.mirlabs.net/ijcisim/index.html

Advanced Cache Techniques for SLA-Driven Multi-Tenant Application on PaaS

K R Remesh Babu[1], Saranya S[2] and Philip Samuel[3]

[1] Department of Information Technology, Government Engineering College Idukki,
Painavu, Idukki – 685 603, Kerala, India
remeshbabu@yahoo.com

[2] Department of Information Technology, Government Engineering College Idukki,
Painavu, Idukki – 685 603, Kerala, India
saranyasasangan3@gmail.com

[3] Division of Information Technology, Cochin University of Science and Technology,
Kochi – 682 022, Kerala, India
philipcusat@gmail.com

***Abstract*: Multi-tenant application is one of the main characteristics of cloud computing. Today, most of the application uses cache service for getting faster access and low response time. Currently in multi-tenant cloud applications data are often evicted mistakenly by cache service, which is managed by existing algorithms such as LRU. Also, security mechanisms are implemented to avoid data breach when data are accessed improperly by another tenant. SLA Driven cache optimization Approach for multi-tenant application is built on PaaS. It helps to improve the cache performance and cost effectiveness of tenants. This tries to improve the cache utilization by avoiding faulty evictions and unnecessary storage retrievals. It considers both tenant profile and data profile in addition to LRU for weight the evicted data and determines re-cache mechanism. Memcache is currently used to cache data explicitly. It provides only average response time to users. In the proposed work, HashMap is introduced as a new form of internal cache on PaaS for faster access. Tenants can access this application through web service by using PC, Laptop, Mobile etc. The test results shows that the proposed method provides better performance and faster access to multi-tenant users.**

Keywords: Cache Optimization, Cloud computing, Multi-tenant Application, SLA, Cache security, QoS.

I. Introduction

Cloud computing is a model like ubiquitous network, that allows accessing the shared computing resources to handle the application. It delivers the hosted services over the Internet. These services are broadly divided into three categories: Infrastructure as a Service (IaaS), Software as a Service (SaaS), and Platform as a Service (PaaS). Many enterprises are adopting this technology to achieve low cost and high performance computing. Multi-Tenant application is one of the important characteristics of cloud computing which provides better resource utilization for application provider. Its development and adoption are greatly promoted by cloud computing.
A multi-tenant application supports tenants to share one application and database instance while allowing them to configure the application to fit their needs respectively as if, it runs on a dedicated environment [1]. To ensure service qualities for each tenant, more and more people advocate that these applications should abide by Service Level Agreements (SLAs) to perform their computation needs [20],[21]. Several works have offered solutions to help these applications to improve resource utilization during runtime. Most of these works focus on virtual machine (VM) management. Storage and cache are the important cloud services, but the issues in these are less addressed.
Cloud distributed cache [10],[11] service plays a vital role in improving cloud application performance. It reduces latency and improves user satisfaction greatly. In Google App Engine (GAE) Memcache service is used for multitenant application. Memcached service is an open source and widely used by many high-traffic sites, such as Facebook, Live Journal, Wikipedia and Fotolog. In cloud, the time for reading/writing data from/into cache T_tcache and reading/writing data from/into data store T_tround includes transfer time between different nodes. Where T_tround time is much greater than T_tcache [11]. Hence effective utilization of cache service leads to high hit rate and low response time. Therefore, how to improve cache hit rate by reasonably appointing data to cache becomes a key to multi-tenant application success.
Currently different methods and cache strategies are used for managing the tenant oriented resources and cache management in shared environment respectively. SLA-driven optimization approach help the multi-tenant application to better utilize cloud cache service. It can be taken as complementary to the existing work. It considers both Tenant Profile and Data Profile when weighting the evicted data with re-cache method, and then adjust their re-cache priorities. Tenant profile include tenant SLAs and tenant priorities. Data profile including its historical trend and importance to the tenant. Optimization process occurs at the beginning of every cycle. This approach is built on the top of PaaS.
SLA-driven cache optimization approach for multi-tenant application uses external cache for storing data. This will take

average response time for handling user request. Use of an internal cache provides faster access to users. Internal cache can be created using HashMap technique. HashMap stores key value pairs. User can access value by using their keys.

There are several methods to ensure Quality of Service (QoS) in multi-tenant cloud environment such as knowledge base resource allocation [33] and SLA based scheduling [34]. The paper [35] proposed a method for performance measurement of multi-tenant applications in cloud. The paper [36] implemented client-based in-memory caching method for cloud data store.

This paper introduces an advanced cache techniques for multi tenant application. Internal cache is created for multi-tenant application using HashMap. It will take low response time for handling user request. Internal cache is built on PaaS. This application can access through web service from PC, Laptop, and Mobile etc. These approaches provide integrity to original cloud cache service and portability among different cache services. This helps to provide better performance to multitenant users.

II. Multi-tenant Application

Multi-tenant applications development and adoption are greatly promoted by cloud computing [1], which aims for "better resource utilization" for application provider and "pay as you go" for application tenants. Multi-tenant application supports tenants to share one application and database instance while allowing them to configure the application to fit their needs respectively as if it runs on a dedicated environment.

While a number of definitions of a multi-tenant application exist [2][3], they remain quite vague. Therefore, we define a multi-tenant application as the following:

Definition 1.: A multi-tenant application lets customers (tenants) share the same hardware resources, by offering them one shared application and database instance, while allowing them to configure the application to fit their needs as if it runs on a dedicated environment.

Definition 2.: A tenant is the organizational entity which rents a multi-tenant SaaS solution. Typically, a tenant groups a number of users, which are the stakeholders in the organization.

These definitions focus on what we believe to be the key aspects of multi-tenancy:

a) The ability of the application to share hardware resources.

b) The offering of a high degree of configurability of the software.

c) The architectural approach in which the tenants (or users) make use of a single application and database instance.

Figure. 1 show the overview of the multi-tenant application, where tenants share one application and database instance.

Multi-instance approach [4], in which each tenant gets its own instance of the application (and possibly also of the database). Multi-instance approach is the "easier" way of creating multi-tenant like applications from a development perspective of virtualization technology and cloud computing. Multi-tenant application can be built on top of SaaS and PaaS. In SaaS model, users are provided access to application software and database, where cloud provider manages the infrastructure and platform that run the application. In the PaaS model cloud provider deliver a computing platform, typically include, OS programming language execution environment, database and web server. Application developer can develop and run their software solution on a cloud platform without cost complexity of buying and managing the underlying hardware and software. SLA-Driven multi tenant application is provided to tenants in [6],[7].

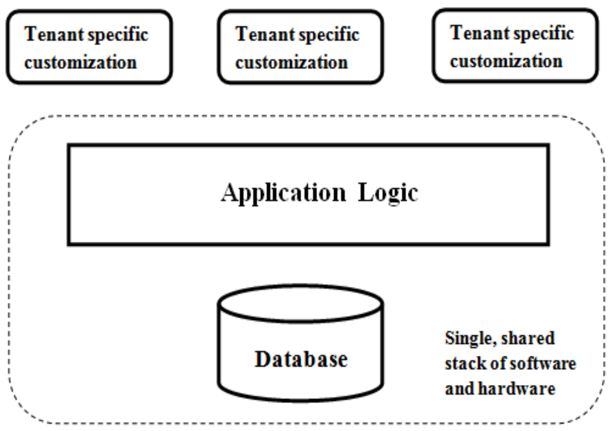
Tenant specific customization
Tenant specific customization
Tenant specific customization
Application Logic
Database
Single, shared stack of software and hardware

Figure 1. Multi-Tenant Application

Advantages of multi-tenancy include hardware resource sharing, high degree of configurability. The application and database instance can also be shared in the multi-tenant cloud environment.

The major challenges of multi-tenancy are good performance maintenance during the computation, scalability issues and security concerns. The maintenance of multi-tenant applications are also an important factor.

A. Cache and Replication

In the existing method Memcached is taken for cloud cache service. Memchaed is an in-memory caching solution. Facebok leverages memcahced as a building block to construct and scale a distributed key-value store [8].

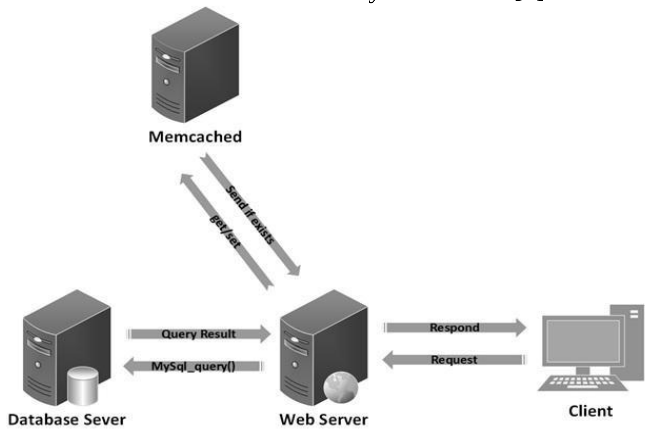

Figure 2. Memcached Operation in Web Application

Figure. 2 illustrates how Memcached works in web application. Request sent by clients for a dynamic web page usually contains data query to database server. Web server will find and get the data in the form of object in Memcached through application server. If the object is found, it will return to web server and respond to the client. However if object is not found in Memcached or 'cache miss', the data will be fetched from database server and it will be set to Memcached as a new item before it is returned to the client. The same process will be repeating all over again. Table1 shows the comparison between multi-tenant application with and without cache.

Table 1. Multi-Tenant Application with and without cache.

Multi-tenant application with cache	**Multi-tenant application without cache**
Better resource utilization	Better resource utilization
Faster access	Less compared with cache
Low response time	High response time
No security mechanism	Different security models used

Currently VM resource allocation has been the hottest area in tenant oriented cache management. To know the cache strategies, first understand the resource management among tenants. Li in [17] put together tenants, SaaS providers and IaaS providers to provide optimizing objective model for each stakeholder respectively, which breaks down the global optimization problem and solve it by an iterative algorithm. Performance regulator based on feedback-control [18] deliver different performance levels based on tenant-specific SLA. The regulator has a hierarchical structure in which high-level controller for managing request admission rates to prevent overloading and a low-level controller manages resource allocation for admitted requests to track a specified level of service differentiation between the co-hosted tenants. A method is introduced [19] for calculations of resource requirements for multi tenants in a shared application instance with applied constraints and propose optimal placement of tenants and instances without violating any requirements. Proposed resource allocation algorithms [20], help SaaS providers to minimize infrastructure cost and SLA violations. It maps customer requests to infrastructure level parameters and handling heterogeneity of Virtual Machines.

Machine learning approach [9] reconfigures the cache strategy online, which is off-line training coupled with online system monitoring. A rule set is trained on the basis of system statistics and the performance results inorder to find which cache strategy is optimal under the current condition. The agent then uses this rule set to identify the right cache strategy for the current condition. An approach [10],[11] is introduced to select optimal cache strategy dynamically using trace-driven simulations, so as to differentiate caching and replication policies for each document based on its most recent trace.

The article in [2] demonstrates the need for continuous dynamic adaptation of replication strategies for Web documents and proposes a techniques for the selection of an optimal replication strategy from a number of candidate strategies with low cost. It evaluates the most likely strategies rather than the entire set of candidate strategies, capturing the history of transitions between different cache strategies.

A cache replacement algorithm for adaptive processor is proposed in paper [13]. It observes the behavior of two (two or more) replacement algorithms such as LIRS, LRU, LFU and Random and then switches to algorithm which performing better policy. Authors in [14] proposed a new two-step method for storage caches management. First approach is an adaptive QoS decomposition and optimization step uses max-flow algorithm to determine application performance optimization. Second approach is a storage cache allocation steps based on feedback control theory to allocate cache space.

Self-adaptive multi-tenant memory management achieve tenant's SLA requirement while minimizing the memory consumption [15]. This method dynamically generates a series of cache replacement units according to the current access model and computes the corresponding I/O yield, and then adopts a greedy algorithm for each tenant to select the corresponding replacement units.

Table 2. Advantage and Limitations of Different cache management policies

Cache approaches	**Advantages**	**Limitations**
Machine learning Approach [9]	Reconfigure cache strategy online	Require large amount of data, Average response time
Different cache strategy for single file in web caching [10-12]	Select optimal cache strategy dynamically	Average response time
Cache replacement algorithm in adaptive processor [13]	Switches between any two algorithm based on workload	Require additional hardware module, Average response time
Adaptive QoS decomposition and optimization [14]	Determine application performance optimization	Average response time
Self-adaptive multi-tenant memory management [15]	Minimizing memory consumption	Average response time
Proportional Hit Rate [16]	Allocate cache space for all clients properly	Average response time
SLA-driven cache optimization approach [21]	Better utilization	Average response time

Proportional Hit Rate method [16] meet clients' SLAs, which tries to allocate cache space properly for every client by combining Isolated Cache Model and control theory for storage. Controller guarantees the relationship of QoS among classes as constant. It provides great contributions in improving cache performance.

SLA-driven optimization approach [21] helps the multi-tenant application to better utilize cloud cache service. Currently in multi-tenant cloud applications, data are often evicted mistakenly by cache service, which is managed by existing algorithms such as LRU. The mistaken eviction leads to wasting time to reload the data back and increase response time. It is because the existing algorithms consider without

Tenant Profile and Data Profile. SLA-driven optimization approach is built on PaaS, so it respects the integrity of original cloud cache service and improves application portability among different cache services. Table 2 shows advantages and limitation of different methods for cache management policies.

III. System Design

A. *Security Oriented Cache Approach*

The main issue in multi-tenant cloud application with cache is internal security due to improper memory access by other tenants. In [31] we have developed an information security mechanism to multi-tenant application with cache by avoiding improper data access by other tenant's. In this method, symmetric DES is used to encrypt the tenant's critical data inorder to prevent improper access by the other active tenants. DES requires lesser computational power compared to other public key encryption methods.

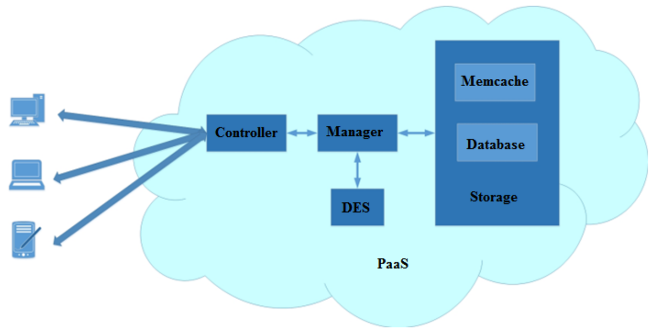

Figure 3. Security Oriented Cache Approach

Figure. 3 illustrates security oriented cache for multi-tenant application on PaaS, to incorporate all devices that are connected to the Internet. Tenants uses web browser to access applications. Tenants can send request to the controller for getting data or inserting data into database. Controller will map the user request to manager. Data encryption and decryption is done by manager. For inserting data into database, manager first encrypts the user data and passes it to storage unit. Only encrypted data is stored in database. For getting data of a particular user, the Manager will find it and get the data in the form of object in Memcached. After getting data from Memcached, manager will decrypt the data and returned it to the user.

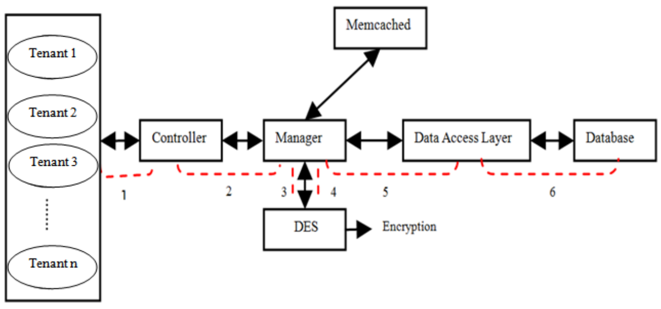

Figure 4. Insertion of user data into database

If the object is not found, data is fetched from the database server and it will be set to Memcached as a new item. Then decrypt the data from database and returned to the user. The same process will be repeating all over again.

Figure. 4 demonstrate the insertion of data item into database. User first sends a request to the controller with data, key and its id (1). Controller will map the request to manager (2). Using DES manager will encrypt the data and it is passed to data access layer (3, 4, 5). Data access layer store these details in database securely (6).

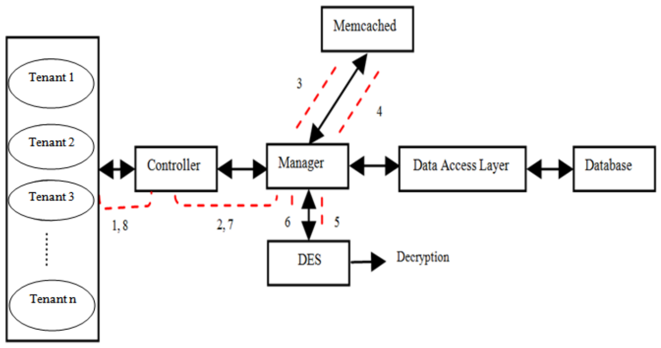

Figure 5. Data Retrieval from Memcached

Figure. 5 illustrates Data Retrieval from Memcached, when user sends a request to controller for retrieving data (1). Controller will map the request to manager (2). The Manager will check and get the data from Memcached (3, 4). After getting data from Memcached, the manager will decrypt it and returned to the user (5, 6).

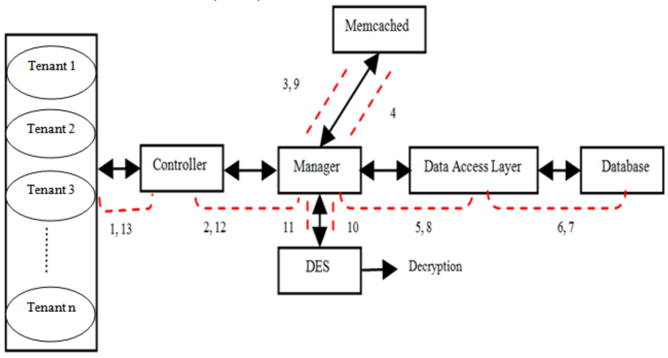

Figure 6. Data Retrieval from database if not found in Memcached

If data is not found in cache, then it is fetched directly from database and it will be set to Memcached before decryption. Then manager will decrypt the data and returned to the user. This procedure is shown in Figure. 6.

B. *Internal Cache*

The main issue in multi-tenant application with Memcached [31] is low average response time. In this proposed method, an Internal Cache is created on application context using HashMap. Internal caching of data makes faster retrieval to users. Users can access this application through web service using PC, laptop and Mobile etc. Internal cache method is automatically generated on all devices from which user access this application. It stores key value pairs, i.e. it keeps tenant key and its data. It provides quick response time and better performance to tenants.

Internal cache and Memcached mechanism for SLA-Driven multi-tenant application on PaaS is shown in figure 7. Tenants can send a request to controller for getting data or inserting data into database. Request is send in the form of URL. Controller will map this request to manager. An internal security is provided at manager by using DES. Only encrypted data is stored at storage unit. Decrypted data is passed to tenants. For inserting data into database, manager first encrypts the data and stored at database.

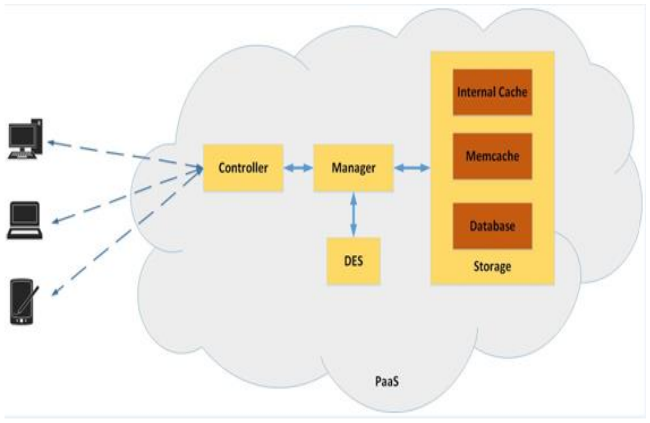

Figure 7. Proposed Internal Cache Method

Data retrieval request is handled by manager. When a data is requested, it first looks into internal cache and gets the data in the form of object in internal cache. If it is not found, check in Memcached and get data and set to internal cache as a new item. If not found in Memcached, data is fetched directly from the database server and it will be set to Memcached as a new item. The same process will be repeating all over again. It is built on the top of PaaS, so as to available to all devices that are connected to the Internet.

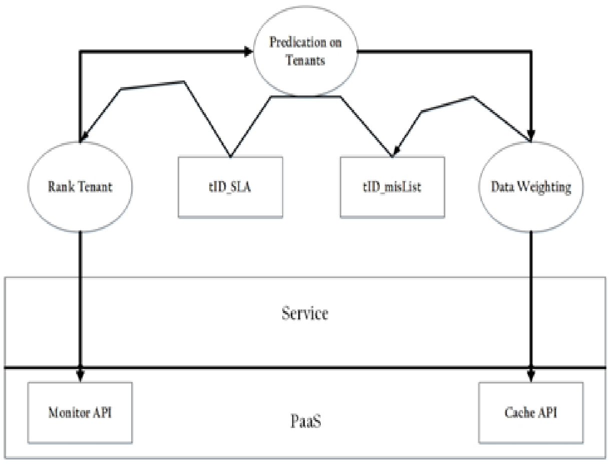

Figure 8**.** Optimization Processing Flow

Cache optimization is done at manager. Figure. 8 illustrates the cache optimization processing flow. An SLA model depicting tenant's expectations and a miss list recording evicted data in past are two core data models. At the beginning of optimization, tenant ranking procedure is invoked to calculate tenant priority for the coming cycle based on tenants' status at the last cycle. And then predication is invoked to predict average response time for every tenant, which also makes an optimization budget. Finally data weight is invoked to weight every data in miss list by considering tenant rank as well as the data relative importance.

Figure. 9 describes how to access service data in our approach. When a data is requested, it first looks up on the internal cache. If cache hit, it directly fetches data and return. If it not found, it looks up on the Memcached. If data found in Memcached, directly fetches data from database and set to internal cache as a new item. If the data is missing, it first checks whether the budget approves data retrieval from database. After that increment the miscount and get data from the storage. Only if the budget permits, it reloads data and caches it in consideration of data likelihood as well as optimization budget.

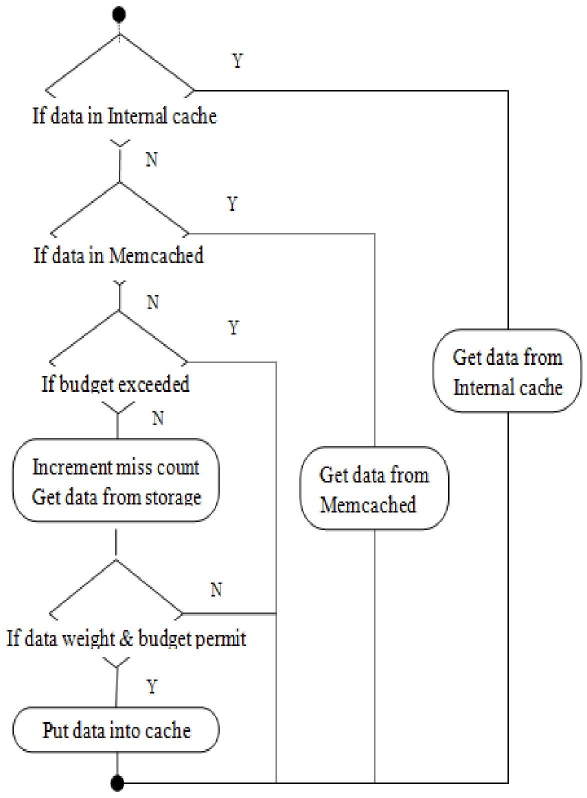

Figure 9**.** Data Access Service

C. Tenant Profile

A tenant profile includes SLA, penalty and optimization budget. A tenant SLA is expressed as a multi-tuple ap_id, tenant_id, tenant_fea, avg_rt_exp, pay_wil, ap_id is application ID, tenant_id is the tenant id, tenant_fea describes tenant features such as tenant priority, avg_rt_exp describes tenant expectation on average response time, pay_wil specifies tenant willingness to pay for the response time expectation in every cycle which is already decomposed from previous budget.

Gompertz function [29] is used to calculate tenant compensations from application providers, which is a ratio to tenant's pay_wil. Gompertz function is a mathematical model for time series, where its growth is the slowest at both the start and the end of the time series. This curve imposes penalty rationally on the application provider: slow growth at the beginning gives the provider a cushion in detecting causes; slow growth at the end indicates although the application continues to violate SLA.

Initiate the formula constants in consideration of function characteristic and SLAs. *u* sets the penalty rate upper asymptote, which is set to 0.5 according to SLA on Google App Engine (GAE) [30]. *v* sets the *y* displacement which is set to -5. *w* determines the growth rate of *y* which is set to -1/2. Besides, we take into account the accumulated relative difference of response time, expressed as $\Delta(rspt)/ rt\_exp_{tID}$ to encourage service provider to improve response time as much as possible. $\Delta(rspt)$ is the difference of response time expectation rt_exp_{tID} and actual average response time $avg\_rt\_exp_{tID}$ for the tenant. Equation (1) specifies the penalty ratio for the tenant. Initial value of n is set to 0 and increases once a violation is detected. Once violation is eliminated, n resets again

$$\mathrm{penlty}_{\mathrm{RattID(n)}} = 0.5 * e^{-5 * e^{-1/2[n + \sum \Delta(\mathrm{rspt})/\mathrm{rt\_exptID}]}} \tag{1}$$

Optimization budget op_budget_{tID} is different from tenant to tenant, and the budget is different under different tenant status as well. Equation (2) describes how to calculate *op_budget* when a violation is predicted to happen or not. Based on current tenant status, that is, a tenant SLA is satisfied or not, $rout\_rat_{tID}$ and $penlty\_Rat_{tID}$ determines *op_budget* respectively. $rout\_rat_{tID}$ indicates the routine budget for the tenant.

$$\mathrm{penlty}_{\mathrm{RattID(t)}} = \begin{cases} \mathrm{pay}_{\mathrm{wiltID}} * (1 - \mathrm{rout}_{\mathrm{rattID}}); & \text{if present tenant status is normal} \\ \mathrm{pay}_{\mathrm{wiltID}} * (\mathrm{penlty}_{\mathrm{RattID(n-1)}} - \mathrm{penlty}_{\mathrm{RattID(n)}}); & \text{otherwise} \end{cases} \tag{2}$$

Tenant ranking is achieved by (3) determining the importance of tenants. At the beginning of t^{th} cycle, every tenant is graded as follows:

$$\mathrm{tenant}_{\mathrm{rnktID(t)}} = \delta * \mathrm{tenant\_featID} + \frac{[1 + \mathrm{penlty\_RattID(t)}]\mathrm{pay\_wiltID}}{\mathrm{rt\_exptID}} \tag{3}$$

where,

$$\mathrm{penlty}_{\mathrm{RattID(t)}} = \begin{cases} 0 & ; \text{if present and predicted tenant status is normal} \\ \mathrm{penlty}_{\mathrm{RattID}(n+1)} & ; \text{otherwise n is accumulated cycle of present violation} \end{cases}$$

D. Data Profile

Miss list mis_list included in data profile, it is vital part of data profile. Every tenant is equipped with one miss list, which is periodically updated to reflect miss hit statistics. Miss list mis_list include {*paramList<>, wght, miscnt, len*}, *paramList* records list of parameters, *wght* is the data weight, *miscnt* is the miss count, *len* is the data size.

Weight of data determining the importance and the cost benefit brought by data caching. Then weight calculation considers both tenant importance $tenant\_rnk_{tID}$ described by (3) as well as data importance. Data importance is relevant to its miss count and its past performance.

if t > 0,

$$\mathrm{wght}_{\mathrm{ij}} = \lambda * \mathrm{wght}_{\mathrm{ij}}(\mathrm{t}-1) + (1-\lambda) * [\mathrm{ztenant}_{\mathrm{rnki(t-1)}} * \mathrm{zmiscnt}_{\mathrm{ij}}(\mathrm{t}-1) \div \mathrm{zlen}_{\mathrm{ij}}]$$

when $t = 0$

$$\mathrm{wei}_{\mathrm{ij}}(0) = (1-\lambda) * \mathrm{ztenant\_rnkt}_{\mathrm{i}}(0) \tag{4}$$

Data weight at the first cycle only depends on tenant rank $tenant\_rnk_{tID}\ (0)$. λ set to $e^{T/2}$ determines the effect imposed on predicted weight by distant past, which is affected by length of cycle. Variables starting with letter "z" represent standardized values of tenant rank, miss count and length respectively.

Beyond that, optimization cost should be considered as well. According to the data accessing process, its major cost arises from two aspects: loading data from database and occupying cache space. To calculate the optimization cost use equation (5).

$$\mathrm{op}_{\mathrm{cost_i}(t)} = \mathrm{unit\_prcs}_{\mathrm{store}} * \left[\sum_{j=0}^{lst\_len} \mathrm{miscnt}_{\mathrm{ij}}(\mathrm{t}-1) * \mathrm{wght}_{\mathrm{ij}}(\mathrm{t})\right] + \mathrm{unit\_prcs}_{\mathrm{cache}} * \sum_{j=0}^{lst\_len} \mathrm{len}_{\mathrm{ij}} \tag{5}$$

In equation (5) *i* indicates i^{th} tenant and *lst_len* indicates the size of mis_list_{tID}, *unit_prcstore* and *unit_prccache* indicate the unit price of storage retrieval operation and cache space respectively. Furthermore, *op_cost* is constrained by the budget. According to (5), it should be checked before fetching data from storage or putting data into cache to satisfy (6).

$$\mathrm{op\_cost}_{\mathrm{tID}}(\mathrm{t}) < \mathrm{op\_budget}_{\mathrm{tID}}(\mathrm{t}) \tag{6}$$

IV. Experimental Setup and Results

In this section, present simulation experiments to evaluate this approach and its performance. Currently, we adopt Memcached to simulate cache service in GAE [8], which is implemented by Memcached. Memcached is a powerful distributed cache management system which already integrates powerful caching policies in its architecture and widely applied in industry such as Facebook.

Internal cache implemented to provide faster access and low response time to users. It is created using HashMap. It stores key value pairs. It is automatically generated to user's application context on PaaS. Internal cache mechanism provides better performance as compared to Memcached.

A. Experimental Design

The experiment environment is constructed by machines with cloud service. Memcached version v 1.4.15 is installed on these machines. The experiment test cases is conducted by using Tomcat version 7.0.39, and available to all the devices that are connected to this network. The experiment case is implemented with a visited data set on [30]. This data set stores application accessible data. Each row includes identifying fields such as a global unique identifier and value field that store value of the corresponding identifier.

The test cases are implemented as a web service. This helps the tenants to access the data through web browser. The business logic using cache service is as follows. The users send requests for data by keys, and the server first looks up to the internal cache. If it is not found, check in Memcached and get data and set to internal cache as a new item. If it is not found in Memcached, it gets from database and set to Memcached. To investigate the relation between internal cache and Memcached, we consider up to four tenants, each of which owns identical amount of users and workloads. The Table 3 shows the response time required for tenants using internal cache and Memcached for data retrieval.

Table 3. User performance with Memcached and Internal Cache

Experiment Cycle	**Memcached**	**Internal Cache**
Tenant 1	0.25	0.033
Tenant 2	0.17	0.033
Tenant 3	0.13	0.017
Tenant 4	0.10	0.017

During experiment, we considered multiple users for every tenant. Every tenant has its own key. This key is same for all users under one tenant. The proposed method controls the data characteristics by using this pre-generated key. Table 4 shows the response time for tenants from without cache, with Memcached and with Internal Cache.

Table 4. Tenant performance without Cache, with Memcached and Internal cache

Tenant	**Without Cache (sec)**	**Memcached (sec)**	**Internal cache (sec)**
Tenant 1	14.610	4.38	1.40
Tenant 2	15.433	4.40	1.50
Tenant 3	13.650	3.40	1.73
Tenant 4	14.200	3.90	1.60

B. *Performance Calculation*

Performance Calculation indicates whether multi-tenant application with internal cache captures trends in response time. Figure. 10 Shows the response time for tenants with Memcached and Internal Cache. Here a tenant contains only one user. So there is a huge difference in response time between Memcached and internal cache. We can see that the response time for Memcached is more than double as compared to Internal Cache. In other words, internal cache takes few milliseconds to respond to user request. Response time of tenants with internal cache is less as compared to tenants with Memcached. So it provides faster performance and QoS to the users.

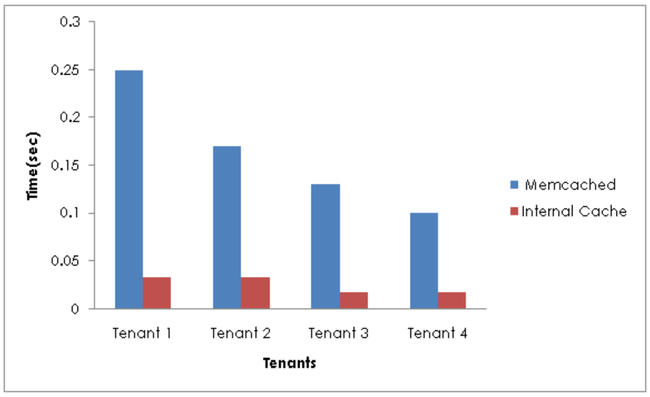

Figure 10. Response time with Memcached and Internal Cache

To provide further evidence for the performance of the proposed method, we consider multiple tenants and their response time with Memcache, internal cache and without cache. Here each tenant contains 100 users. Internal cache service provides a greater difference in response time as compared with database storage and Memcached. Response time without cache means data is fetched directly from the database. It takes more time compared with internal cache. Response time with cloud cache service means data fetched from Memcached. It takes more time compared with internal cache. It is shown in figure. 11.

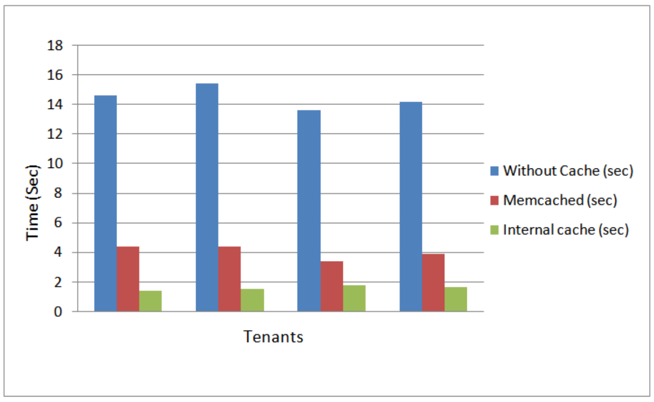

Figure 11. Response time without cache, with Memcached and with internal cache

Figure. 12 shows Response time vs. number of users in tenants. Number of users within the tenants is incremented to study the relation with response time. The number of users in each tenant is incremented periodically and checked their corresponding response time for Memcached and internal cache mechanism. A high number of user leads to high response time. When number of users is less, response time also improves. Internal cache provides better performance than Memcached.

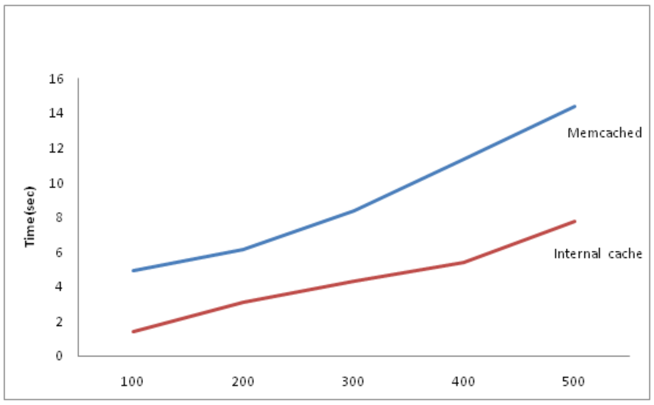

Figure 12. Response time vs. Number of users

Figure. 13 shows the miscount rate for three tenants in each experiment life cycle. Miscount incremented after every experiment cycle, that means, corresponding tenant's data is frequently requested. It first checks in internal cache, if it is not present in the internal cache then checks in Memcached. If not found in Memcached increment miscount. This leads to the caching of tenant data for faster access.

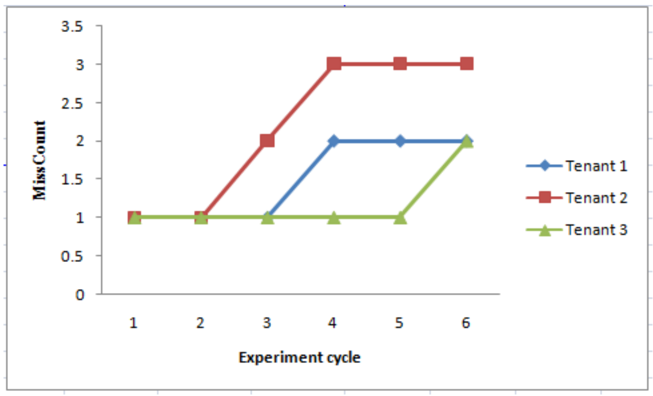

Figure 13. Miss Count vs. number of tenants

V. Conclusion

In cloud multi-tenant application, most of the work focuses on VM migration and load balancing issues. Efficient and secure data storage and retrieval are important issues to improve the resource utilization and response time. Multi-tenant applications have made the need for cache mechanism for effective and faster service. Security is another major concern in cloud computing. In order to address security issues in the proposed cache approach employs DES algorithm for encryption.

Currently in multi tenant cloud applications data are often evicted mistakenly by cache service, which is managed by existing algorithms such as LRU. SLA-Driven cache optimization for multi-tenant application based on PaaS help to improve the cache performance and meet tenant's needs for better and improve cost effectiveness. The proposed method tries to improve the cache utilization by avoiding mistaken evictions and unnecessary storage retrievals. It introduces tenant profile and data profile to weight the evicted data. Tenant profile considers SLA, runtime penalty and budget. The optimization introduces prediction methods to rank tenant and determine budget. This approach satisfying tenant SLA requirements and cost effectiveness. Internal cache is implemented to improve the performance of multitenant application. This internal cache is built on the top of users' application context. It is created using Hashmap. It provides better performance and faster access to multi-tenant users.

References

[1] M., Peter, T., Grance. "The NIST definition of cloud computing", NIST special publication 800.145: 7, pp.1-3, 2011

[2] Bob Warfield. "Multi-tenancy can have a 16:1 cost advantage over single-tenant", *http://smoothspan.wordpress.com/2007/10/28/multitenancy-can-have-a-161-cost-advantage-over-single-tenant*, 2007

[3] Craig D. Weissman, Steve B. "The design of the force.com multi-tenant internet application development platform", *In Proceedings of 35th SIGMOD Int. conf. on Management of data (SIGMOD)*, pp. 889–896, 2009

[4] Bezemer, Zaidman. "Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare?", *In Proceedings of ACM 4th Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, pp.88-92, 2010

[5] Zheng Xuxu, Li Qingzhong, Kong Lanju. "A Data Storage Architecture Supporting Multi-level Customization for SaaS", *In Proceedings of IEEE 7th International conference on Web Information Systems and Applications (WISA)*, pp.106-109, 2010

[6] Wu, Linlin, S K Garg, Rajkumar Buyya. "SLA-based resource allocation for software as a service provider in cloud computing environments", *In Proceedings of 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp.195-204, 2011

[7] Nandi, Bipin B. "Dynamic SLA based elastic cloud service management: A SaaS perspective", *In Proceedings of IFIP/IEEE International Symposium on Integrated Network Management (IM2013)*, pp.60-67, 2013

[8] Rajesh Ni, H Fugal, S Grimm, M Kwiatkowski, H Lee, H C. Li, R McElroy, M Paleczny, D Peek, P Saab, D Stafford, T Tung,V Venkataramani. "Scaling Memcache at Facebook", *In Proceedings of 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI '13)*, pp. 385-398, 2013

[9] Qin Xiulei, "On-line Cache Strategy Reconfiguration for Elastic Caching Platform: A Machine Learning Approach", *In Proceedings of IEEE 35th Annual Conference on Computer Software and Applications Conference (COMPSAC)*, pp.523-534, 2011

[10] Pierre, Guillaume, A S Tanenbaum. "Differentiated strategies for replicating Web documents", *Journal of Computer Communications*, vol. 24, Issue. 2, pp. 232-240, 2001

[11] P Guillaume, Maarten Van Steen, Andrew S. Tanenbaum. "Dynamically selecting optimal distribution strategies for Web documents", *IEEE Transactions on Computers*, vol. 51(6), pp. 637-651, 2002

[12] S Swaminathan, G Pierre, Maarten Van Steen. "A case for dynamic selection of replication and caching strategies", *Web content caching and distribution*, Springer Netherlands, pp. 275-282, 2004

[13] Subramanian, R., Yannis, S., Loh, G.H. "Adaptive caches: Effective shaping of cache behavior to workloads", *In Proceedings of 39th IEEE/ACM International Symposium on Microarchitecture*, pp. 385-396, 2006

[14] Prabhakar, Ramya. "Adaptive QoS Decomposition and Control for Storage Cache Man-agement in Multi-server Environments", *In Proceedings of 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 402-413, 2011

[15] Yao, J.C., Z., Shi-Dong, S., Yu-Liang, L., Qing-Zhong. "Multi-Tenant Database Memory Management Mechanism based on Chunk Folding", *Chinese Journal of Computers*, vol. 34.12, pp. 2320- 2331, 2011

[16] G., Ang, Dejun M., Yansu H. "A QoS Control Approach in Differentiated Web Caching Service", *Journal of Networks*, vol. 6.1, pp. 62-70, 2011

[17] Li, Chunlin., Layuan Li. "Efficient resource allocation for optimizing objectives of cloud users, IaaS provider and SaaS provider in cloud environment", *The Journal of Supercomputing*, pp. 1-20, 2013

[18] Hailue Lin, Kai Sun, Shuan Zhao, Yanbo Han. "Feedback-control-based performance regulation for multi-tenant applications", *In proceedings of 15th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 134-141, 2009

[19] Kwok, Thomas, Ajay Mohindra. “Resource calculations with constraints, and placement of tenants and instances for multi-tenant SaaS applications”, *In Proceedings of Service-Oriented Computing ICSOC*. Springer Berlin Heidelberg, pp. 633-648, 2008

[20] LinLin Wu, Saurabh Kumar Garg, Steve Versteeg, Rajkumar Buyya. “SLA-based Resource Provisioning for Hosted Software as a Service Applications in Cloud Computing Environments”, *IEEE Transactions on Services Computing*, vol.7, no. 3, pp. 465-485, 2013

[21] Huihong He, ZhiYi Ma, Hongjie Chen. “An SLA-Driven Cache Optimization Approach for Multi-tenant Application on PaaS”, *38th IEEE Annual International Computers on Computer Software and Applications Conference (COMPSAC)*, pp. 139-148, 2014

[22] Jose M. Alcaraz Calerot, Nigel Edwards, Johannes Kirschnick, Lawrence Wilcock, Mike Wray. “Towards a Multi-tenancy Authorization System for Cloud Services”, *IEEE Security & Privacy*, vol.8, no. 6, pp. 48-55, 2010

[23] Mohemed Almorsy, John Grundy, Amani S. Ibrahim. “Collaboration-Based Cloud Computing Security Management Framework”, *In Proceedings of IEEE International Conference on Cloud Computing (CLOUD)*, pp. 364-371, 2011

[24] Mandy W., W., Zimmermann. “Controlling Data-Flow in the Cloud”, *In Proceedings of 3rd International Conference on Cloud Computing, GRIDs, and Virtualization*, pp. 24-29, 2012

[25] Y., Liang, Z., Hao, N., Yu, B., Liu. “RandTest: Towards More Secure and Reliable Dataflow Processing in Cloud Computing”, *In Proceedings of Int. Conf. on Cloud and Service Computing*, pp. 180-184, 2011

[26] M Almorsy, J Grundy, AS Ibrahim. “TOSSMA: A Tenant-Oriented SaaS Security Management Architecture”, *In Proceedings of IEEE 5th Int. Conf. on Cloud Computing (CLOUD)*, pp. 981-989, 2012

[27] Eyad Saleh, Ibrahim Takouna, Christoph Meinel. “SignedQuery: Protecting Users Data in Multi tenant SaaS Environments”, *In Proceedings of IEEE International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 213-218, 2013

[28] HMAC RFC 2104. Website. [Online]. Available: *http://tools.ietf.org/html/rfc2104* [retrieved: April, 2015]

[29] Ho-Yu Lam, Song Zhao, Kang Xi, H. Jonathan Chao. “Hybrid Security Architecture for Data Center Networks”, *In Proceedings of IEEE International Conference on Communications (ICC)*, pp. 2939 – 2944, 2012

[30] Multi_Tenant Data: *https://developer.salesforce.com/ pages/Multi_Tenant_Architecture*, January 2016.

[31] Remesh Babu K.R., Saranya S., Philip Samuel. “Secure Cloud Multi-tenant Applications with Cache in PaaS”, Innovations in Bio-Inspired Computing and Applications”, *Advances in Intelligent Systems and Computing*, vol. 424, pp. 15-27, 2015

[32] H. AlJahdali, A. Albatli ; P. Garraghan, P. Townend. "Multi-tenancy in Cloud Computing", *In Proceedings of IEEE 8th International Symposium on Service Oriented System Engineering (SOSE)*, pp. 344-351, 2014

[33] Gongzhuang Peng, Hongwei Wang, Jietao Dong, and Heming Zhang. "Knowledge-Based Resource Allocation for Collaborative Simulation Development in a Multi-tenant Cloud Computing Environment", *IEEE Transactions on Services Computing*, DOI 10.1109/TSC.2016.2518161, pp. 1-13, 2016.

[34] G. Peng, J. Zhao, M. Li, B. Hou, H. Zhang. "A SLA-based scheduling approach for multi-tenant cloud simulation", *In Proceedings of IEEE 19th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pp. 600-605, 2015

[35] A. O. Ayodele, J. Rao, T. E. Boult. "Performance Measurement and Interference Profiling in Multi-tenant Clouds", *In Proceedings of IEEE 8th International Conference on Cloud Computing (CLOUD)*, pp. 941-949, 2015

[36] Jens Kohler, Thomas Specht. "Performance Analysis of Vertically Partitioned Data in Clouds Through a Client-Based In-Memory Key-Value Store Cache", *International Joint Conference, Advances in Intelligent Systems and Computing*, 369, pp. 3-13, 2015.

Author Biographies

K R Remesh Babu holds Bachelor degrees in Mathematics and Information Technology. He received his Masters Degree in Computer Science from Anna University, Chennai. He is currently pursuing the Ph.D. degree at Cochin University of Science & Technology (CUSAT). He is an assistant professor in department of Information Technology, Government Engineering College Idukki, India. His main research interests include distributed systems, grid & cloud computing, Internet of things, WSN, and data mining.

Saranya S Born on 1992 in Thiruvananthapuram, India. She received the B. Tech degree in Computer Science and Engineering from Kerala University, Kerala, India in 2013, and M. Tech degree in Network Engineering from Mahatma Gandhi University, Kottyam, Kerala, India in 2015. Her current research interest includes Cloud computing, Multi-tenant applications, Cache protection, and Cloud security.

Philip Samuel is Reader in Information Technology Division, School of Engineering, Cochin University of Science & Technology (CUSAT). He holds a Masters Degree in Computer & Information Science from CUSAT and Ph.D degree in Computer Science & Engineering from IIT Kharagpur, India. He has more than 17 years of experience in teaching and research as a faculty at CUSAT. He has published more than 35 research papers in International Conferences and Journals. His research interest includes Big Data Analytics, Distributed Computing and Automated Software Engineering.