APPLICATION FRAMEWORK FOR SUPPORTING PERFORMANCE ISOLATION IN SOFTWARE AS A SERVICE SYSTEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

By
Orhun Alp Oral
January, 2015

APPLICATION FRAMEWORK FOR SUPPORTING PERFORMANCE ISOLATION IN SOFTWARE AS A SERVICE SYSTEMS

By Orhun Alp Oral

January, 2015

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

_____

Assist. Prof. Dr. BedirTekinerdoğan (Advisor)

_____

Assoc. Prof. Dr. HakanFerhatosmanoğlu

_____

Assist. Prof. Dr. ÖmerÖzgürTanrıöver

Approved for the Graduate School of Engineering and Science:

_____

Prof. Dr. LeventOnural

Director of the Graduate School

APPLICATION FRAMEWORK FOR SUPPORTING PERFORMANCE ISOLATION IN SOFTWARE AS A SERVICE SYSTEMS

Orhun Alp Oral
M.S. in Computer Engineering
Advisor: Assist. Prof. Dr. BedirTekinerdoğan
January, 2015

Scalability of a cloud application is the ability of an application that handles the service level agreement (SLA) requirements for all customers. In a *non-isolated SaaS system*, the different clients of a SaaS can freely use the resources of the SaaS. Hereby, disruptive tenants who exceed their limits can easily cause degradation of performance of the provided services for other tenants. To ensure performance demands of the multiple tenants usually elasticity of the SaaS is needed, which supports changing the hardware resources. In case the users' demand change, the system can scale itself up or down according to the minimum necessary computational power to handle all the requests within the minimum requirements of SLA. The main goal of elasticity, however, is not directly to support performance isolation but scalability of the SaaS in general. As such, relying only on elasticity does not explicitly solve the problem of performance isolation.

Performance isolated systems aim to ensure the minimum requirements of the SLA to all of the customers. Ensuring the fair behavior with respect to customers is a need for building a performance isolated system. Various performance isolation approaches have been introduced including artificial delay, round robin, blacklist, and thread pool. However, these approaches assume a SaaS system with thin clients whereby all the tasks are handled by the SaaS.

This thesis introduces the "Tork Framework", for developing scalable and performance isolated enterprise SaaS applications. Tork Framework provides a new way for developing a cloud application from scratch or converts existing software applications to scalable cloud applications. Tork Framework introduces built-in features applicable for the application layer, database layer and client layer. These features include techniques for security controls, logging, and multi-tenancy on relational database management systems (RDMS). Built-in server components of Tork Framework provide service oriented architecture for all business objects in the RDMS, and client components make the application work on different platforms. Also, this framework provides a cloud based IDE for configuring parameters and writing custom codes without the need of any other individual tool via online graphical user interface (GUI).

The framework as such provides an approach for supporting the design and realization of performance isolated cloud system by also considering the usage of resources of thick clients. To illustrate the framework and validate our approach we have adopted a real SaaS environment in which a case study is implemented using the Tork framework. Different Implemented performance isolation techniques are tested in this case study using both thin and thick client types.

Keywords: SaaS framework, performance isolated SaaS architecture, multi-tenant cloud application, performance isolation, tork framework, tork application development.

BULUT MİMARİLİ VE PERFORMANS YALITIMI SAĞLAYAN YAZILIM GELİŞTİRME PLATFORMU

Orhun Alp Oral
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: Yrd. Doç. Dr. Bedir Tekinerdoğan
Ocak, 2015

Bir bulut yazılımının ölçeklenebilir olması için, herhangi bir müşterisinden herhangi bir zamanda gelen isteklere vereceği cevabı servis sözleşmesinde belirtilen kriterler içerisinde yapabilmesi gerekir. İzole olmamış bir servis yazılımında farklı müşteriler yazılım aracılığıyla sunucuya istek yollayarak, sunucunun donanımsal gücünü meşgul edebilirler. Donanımsal gücün aşırı istek gönderen kullanıcılar tarafından meşgul edilmesi sistem performansının düşmesine sebep olur ve düşen bu performanstan diğer kullanıcılar da olumsuz olarak etkilenirler. Müşterilerden gelebilecek yoğunluklara karşı sistem performansını iyi durumda tutabilmek için genellikle bulut yazılımlarında esneklik kavramı kullanılır. Ancak esneklik kavramı sistemin daha fazla donanımla birleşmesini sağlayarak performansın arttırılması veya daha az donanımın kullanılarak performansın azaltılması olarak kullanılabilir. Gereken donanım miktarını müşteriyle yapılan servis anlaşması belirleyebilir, ancak bu durum yine de performansın yalıtımını sağlayan bir çözüm değildir.

Performans yalıtımıyapılmış olan sistemlerde amaç, müşterilerle yapılan servis anlaşmalarındaki minimum gereksinimlerin müşteriye mevcut donanımlarla sağlanabilmesidir. Bir sisteme performans yalıtımıuygulayabilmek için öncelikle bu sistemdeki kullanıcıların adil kullanım ilkelerine uymasını sağlamak gerekir. Bu ilkelere bağlı olarak farklı performans yalıtımıuygulayan yöntemler geliştirilmiştir. Bu yöntemler sanal bekletme, sırası gelene cevaplama, kara listeye alma ve işlem havuzu yöntemleridir. Ancak bu yöntemler hizmet yazılımlarında bulunan müşterilerin zayıf donanıma sahip olduğu varsayımıyla geliştirilmişlerdir.

Bu tez *Tork Yazılım Geliştirme Platformu*'nu tanıtmaktadır. Bu platform ile ölçeklenebilir ve performans yalıtımısağlayan hizmet yazılımları geliştirebilmek mümkün olmaktadır. Bu platform yeni bir yöntem ile en başından bir bulut tabanlı yazılım geliştirebilmeyi veya mevcutta olan bir yazılımı bulut tabanlı ölçeklenebilir bir yazılma dönüştürebilmeyi sağlar. Platform aynı zamanda veri tabanı, uygulama havuzu ve istemci tarafları için yazılım geliştirme kütüphanesi sağlamaktadır. Bu kütüphane sayesinde düzenlenen yazılımın güvenlik, işlem geçmişi tutma, veri tabanları için çoklu-kiracı altyapısı kullanabilme gibi özellikleri otomatik olarak sağlanabilmektedir. Bu özellikler nesne tabanlı bir mimaride geliştirilmiş ve istemci tarafının da farklı platformlarda da çalışabilecek bir yapıda çalışabilmesi sağlanmıştır. Aynı zamanda platformda bulunan bütünleşmiş yazılım geliştirme ortamı sayesinde yazılımın

ayarlamaları ve yazılım mantığında değişecek yerler için kod yazabilmek mümkündür. Dolayısıyla yazılımı geliştirmek için sağlanan çevrimiçi, grafiksel kullanıcı ara yüzü, geliştiricileri başka bir yazılım geliştirme aracına ihtiyaç duymadan yazılımları geliştirebilmelerini sağlar.

Bu platform yazılım tasarımını ve performans yalıtımıgerçekleştirebilen; aynı zamanda güçlü donanımsal istemcilerle çalışırken gereken işgücünü bu donanımlara yükleyebilmeye olanak sağlayan bir araçtır. Bu platform anlatmak ve kullanılan yöntemleri de onaylamak amacıyla örnek bir bulut yazılımının test edilmesi olayı ele alınmıştır. Bu yazılımın kullandığı farklı yalıtım teknikleri bu örnek olay üzerinde zayıf ve güçlü istemci senaryolarıyla test edilip değerlendirilmiştir.

*AnahtarSözcükler:*Performansyalıtımı, performansizolasyonu, bulutyazılım, bulutyazılımgeliştirme, bulutyazılımgeliştirme platform, torkplatformu, torkyazılımgeliştirme, buluthizmetyazılımı.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my supervisor Assist. Prof. Dr. Bedir Tekinerdogan, for the continuous support of my master study. He encouraged me and showed guidance with his sound suggestions. His guidance helped me at the time of research and writing of this thesis all through.

Besides my advisor, I would like to thank the rest of my thesis committee: Asst. Prof. Dr. Ömer Özgür Tanrıöver and Assoc Prof. Dr. Hakan Ferhatosmanoğlu for their encouragement and valuable comments on my thesis.

A special thanks to my family. Words cannot express how grateful I am to my mother, father and grandmother for all of the sacrifices that they have made on my behalf. I would also like to thank all of my friends who supported me in writing, and incented me to strive towards my goal. At the end I would like to express appreciation to my beloved friend Ahu who spent sleepless nights with me, and was always my support in the moments when there was no one to answer my queries.

Contents

List of Figures

List of Tables

1. Introduction

When we compare the legacy software architectures, such as a single client desktop software to web based SaaS (Software as a Service) architectures and cloud based SaaS architectures, the development style shows that the frequency of users does not seem to increase in the host application resource pool [1,2]. On desktop software, one user hosts its application; whereas web based SaaS applications may use internet and service for an entire enterprise company, enabling multiple users to use one host resource in this case. Cloud based SaaS architectures on the other hand are capable of servicing to thousands of users and companies [3]. This trend shows how the computational resource pool is shared via a cost-effective manner. Computational resource pool requirements such as RAM and CPU power for the application host are dependent on the demand of the application user requests. Computational power of the host nodes can be used to satisfy all requests from users, whereas clients only demand for functionality, and compute minimally for the process. These types of clients are called thin clients and they heavily depend on another computer to fulfill their computation roles. On the other hand, clients can also fulfill their own requests and send the minimal data transfer to the host node for a consistent cloud application. These types of clients are called the thick clients. The decision of the user's role as thin or thick clientaffects the architecture of the application. This decision is also influenced by the desired revenue model of the application. Some popular cloud applications give both free service and paid service. Paid users can benefit from more resources and functionalities, whereas free users are only able to use standard functionalities (such as Dropbox [4], iCloud [5]).

In one case, an application may consider a single baseline requirement for all customers. In this case, hardware resources have to be shared equally to each client for consistent performance among users. In a second case, an application has to adjust its performance according to the demand coming from the clients. Some clients may require a higher level of performance, whereas some clients may require more functional software modules. Both of these different requirements influence the general performance of the application.

To get a cloud optimized application for both cases, scalability has to be addressed during designing decisions. Trusting to share only a single resource of

hardware may lead to a system crash, especially when providing some free services to clients; the number of the concurrent customers is a critical parameter for system crashes. Therefore, scalability of the cloud application should consider the computational power of the resources.

Novel cloud applications predict the system load on the runtime and serve the clients by scaling the system upward or downward [6]. This approach changes the computational power of the application in a cost-effective manner. In order to optimize this approach reducing each node's workload is an important factor. For instance, using one host per 1000 clients is a 10 times better design decision than using one host node for 1000 clients within the same response time. Because it means that the quality of service (QoS) stays same as the workload of the application is increased to 10 times more. One way to achieve this performance is to utilize computational power of the client nodes [7].

Utilizing the computational power of client nodes reduces the load per host,and thus contributes to scalability. Some novel mobile applications use this approach especially for caching purposes [8] (such as Facebook [9]). By doing this, cloud operations as well as network traffic is reduced - which is also profitable for the mobile carrier who is charged for data traffic. A crucial point of this approach is to consider the computational capacity of clients. Some bad implementations may result in a larger battery usage, or in preventing the client to run other applications. Thus, application developers consider their implementation requirements according to general client distribution. Sometimes applications stop giving service to existing clients because of the new features implemented in the application which requires more computational power than the current client.

Cost-effective price of the cloud requires clients to update their system in the future. Hardware development in general, effects the general client distribution of applications and existing clients will have to adjust their systems to get support from more computational power; if not, developers will stop providing support for the new versions on old hardware systems due to the changing implementation requirements regarding new client distribution [13].

2. Background

In this section a literature background about cloud computing models is provided with a focus on SaaS architectures, approaches to derive SaaS application architectures, and performance monitoring principles on SaaS applications.

2.1. Software as a Service Architectures (SaaS)

In general, three types of cloud computing models are defined [14, 15, 16]. These are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). The IaaS model shares hardware resources among the users. Cloud providers typically bill IaaS services according to the utilization of resources by the users. The PaaS model is thebasis for the computing platform based upon hardware resources. It is typically an application engine similar to an operating system or a database engine which binds the hardware resources (IaaS layer) to the software (SaaS layer). The SaaS model is the software layer which contains the business model. In the SaaS layer, clients are not allowed to modify the lower levels such as hardware resources and application platform. Clients are typically the end-users, and they use this kind of software at an on-demand basis. Cloud clients are devices which use a network to connect cloud computing. Some of them are especially designed to depend on the computational power of the cloud. These kinds of clients are called thin clients. Generally, browser-based devices tend to be thin clients, since the computing and HTML rendering is done by the cloud servers and the browser displays the rendered content. By using novel technologies such as Ajax and HTML5, designed web user interfaces can achieve similar, and may have even better results than the native applications. However, utilizing these technologies also means utilizing client computation power, instead of cloud computational power.Therefore, SaaS providers generally provide their services with a subscription fee and service level agreements according to the application requirements.

In this thesis we will focus on SaaS context [17, 18, 19]. SaaS context is widely discussed in the literature and different definitions for SaaS architectures are proposed. However, there is not a specific SaaS architecture, rather general components and modules for the SaaS structures are defined. Based on literature reviews [20,21,36], Figure 1 shows one of the generic reference architecture for a SaaS system. This architecture is a three-tiered architecture with multiple

clientand host nodes. Usersbelong in the client tier, whereas the application is generally run on the application tier and communicates with the DB tier.

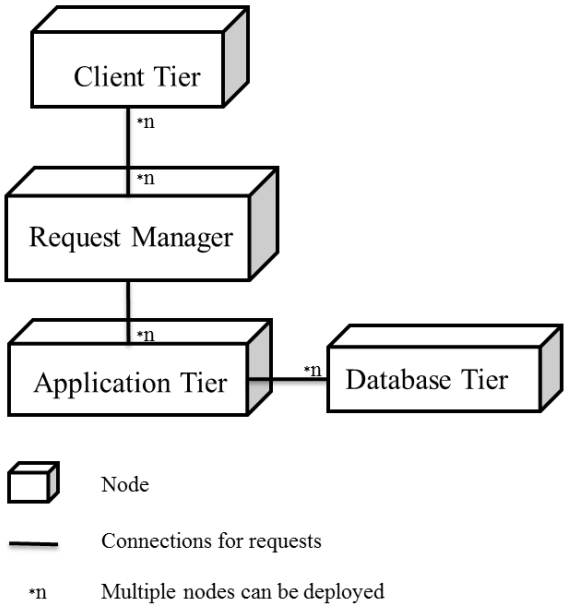

Figure 1 Generic SaaS Deployment Diagram

In this reference diagram three tiers are defined as client tier, application tier and database tier. Generally, for thin clients, the client tier depends on the application tier for running the processes. Therefore modules for user interactions and capturing the events usually contains in the thin clients. On the other hand, dataset modules are containedin thick clients. These modules are used to receive and send data requests to the application tier forprocessing the request. For some cases; business objects modules may be used for caching or running the processes inside the client tier.

Insidethe application tier, the request manager layer is responsible for the distribution of incoming requests. It may be used for caching or load balancing to perform a better service according to application needs. A Meta-Data manager layer may exist on the application tier and, it has a similar behavior as a cached-database inside the application layer for shortening the processing time on frequently incoming requests. Some of the frequently used services such as identity management can be serviced through this layer for better performance considerations. Application threads are the layer inside the application tier which are responsible for processing the requests by communicating with the DB tier and calculate the response.

The Database tier generally uses two general layers, the meta-data layer and the data storage layer. The Meta-data layer is used for cache operations for different

tenants or frequently using operations and the data storage layer is the layer which stores the data of the general application.

Figure 2 shows the basic cloud SaaS decomposition models regarding the deployment diagram. These composite modules described above can belong to different nodes for deployment. For instance, the thin client based cloud applications may use the visualization modules into the application tier, computes the visualized content, and migrates to the client tier. On the other hand, thick client applications may only fetch the raw data from the application tier and use their client device for rendering computation and visualize the content.

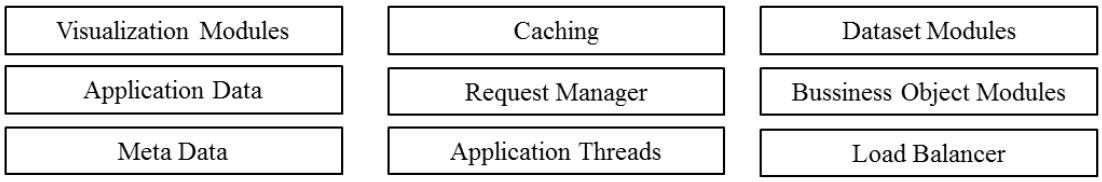

Figure 2 Basic SaaS Decomposition Layers for the Deployment

In literature, Koziolek [20] examined the current multi-tenant architectures, and regarding on thatwork he represented a reference software architecture model that can be used for running multi-tenant applications similar to deployment diagram in Figure 1 and decomposition layers in Figure 2. A later study in [21] analyzed this model to describe the possible performance changing points. These points are as follows:

1. Request control: By controlling incoming requests, it is possible to distinguish among tenants. Preventing disruptive customers may be one way to increase the performance of the overall system.
2. Caching: Some systems may use caching by sharing the same resource pool to different tenants' data. Isolation or limitation of this cache resource pool for active tenants may positively impact the overall system.
3. Load Balance Control: It is possible to redirect requests coming from different tenants. Therefore, disruptive and abiding customers can be separated by service over different application layers.
4. Thread Priorities: By determining the tenant utilization, it is possible to reduce the thread priorities which are serving for highlyutilized customers or disruptive customers.

5. Thread Pools: Similar to item 4, highly utilized requests may be forwarded to lower prioritized thread pools for a more reasonable resource distribution.
6. Database requests: Controlling the requests from the database layer may be another approach for controlling the resource utilized by the given tenant. Customers given higher quality of service (QoS) may utilize more resources from the database cache and other resources with regards to the database.

Based on these points, section 2.2 gives background information related to the performance monitoring in the SaaS architectures. Section 2.2.1 defines the general concepts related with performance isolation. Section 2.2.2 introduces the existing approaches, and finally section 2.2.3 shows the scientific methods on how to measure the performance isolation in multi-tenant SaaS architectures.

2.2. Performance Monitoring in SaaS

Scalability of a cloud application is the ability of an application that handles the SLA requirements forall customers. As long as the minimum requirements of SLA are maintained for all users, that application can be considered as a scalable cloud application. Since the user size is a variable in cloud SaaS applications, performance isolation among the users is an important factor for maintaining scalability of the application.

Lower levels of the cloud systems such as IaaS and PaaSlayers handle performance isolation by handling the hardware resource allocation. Since there is not any physical resource allocation in the SaaS layer, it causes a bigger challenge to ensure performance isolation. With respect to performance isolation we can distinguish between non-isolated systems and performance isolated systems.

In a *non-isolated system*, an action of one user can cause a performance difference and it affects all other users. Thus, all users may observe performance degradation due to a lack of resources reserving to them. Non-isolation may be a good application for the users belonging to the same customer, because reserved resources for a customer can be utilized with non-uniform distribution on different users of the same customer group. In a perfect scalable system, we assume defined limits for customers are sufficient to service with the minimum requirements of SLA. Therefore, the non-isolation state is expected when a client exceeds its defined limits and causes performance degradation to others.

However, in the practical use of SaaS applications, the actions of users are not perfectly isolated, and therefore disruptive customers who exceed its limits may cause suffering to the regular users who demand the service within their limits.

Performance isolated systems aim to ensure the minimum requirements of the SLA to all of the customers. Sometimes it is possible to increase or decrease the performance for some customers, as long as this will not violate other customer's minimum SLA requirements. Isolation of computational resources is one way to achieve performance isolated systems since it is guaranteed that different users do not interfere with each other due to resource sharing. However, it is not a cost-optimized solution due to misusing of idle resources, and it is not a practical solution for SaaS since it requires the solution to be handled on lower layers (PaaS, IaaS).

To ensure performance demands of the multiple tenants usually elasticity of the SaaS is needed, which supports changing the hardware resources. In case the users' demand change, the system can scale itself up or down according to the minimum necessary computational power to handle all the requests within the minimum requirements of SLA. From a general perspective both performance isolation and elasticity support the realization of SLA, but their objectives are different. The main goal of elasticity, however, is not directly to support performance isolation but scalability of the SaaS in general. As such, relying only onelasticity does not explicitly solve the problem of performance isolation.

2.2.1. SystemFairness

Ensuring the fair behavior with respect to customers is a need for building a performance isolated system. Unfair behavior may be caused by the minority of users who may utilize the system. Since they allocate the resource of other users, the system becomes trustless for a majority of the users. To build a fair system, the following conditions must be satisfied:

1. Customers working within their defined limits should not be affected by other customers exceeding their limits.
2. Customers exceeding their limits and causing a negative impact of others should be reduced by performance degradation. This eventually makes the system light and responsive toevery customer.
3. Customersthat havebetter limits should have a better performance compared to customers with lower limits.Performance here can be defined as input/output related parameters such as response time, request rate, etc.

The term *limit* here is referred to as the workload size a tenant is agreed to run on cloud server.Workload can be considered as amount of requests, given a time period. The concept of *performance isolation* is based on the fairness concept given in the first condition.

2.2.2. Performance Isolation Approaches

There are four different architectural solutions proposed for the performance isolation [21] (Figure 3). The assumed application for the performance isolation is an interactive web application which receives a frequent number of requests from customers and is supposed to send responses within a small amount of time as an output. Requests coming from the customers do not intend to utilize computational power after the response, thus they do not tend to compute batch processes. They also assume to have similar workload behavior among other tenants.

In these proposals,response times are considered the primary parameter for the QoS. Therefore, to justify all customers' SLA requirements, comparing average response time with respect to request rates will be used to identify QoS for each customer. On each proposal there are two layers of components, which can be labeled as the request manager and the application server. These layers are used in order to enhance the request control and thread pools structure in the multi-tenant system. The request manager handles incoming requests firstly and redirects them to the application server which is responsible for processing the requests. For the first three approaches (Figure 3.1, 3.2 and 3.3), it is assumed the application server has only one thread pool to process all incoming requests.

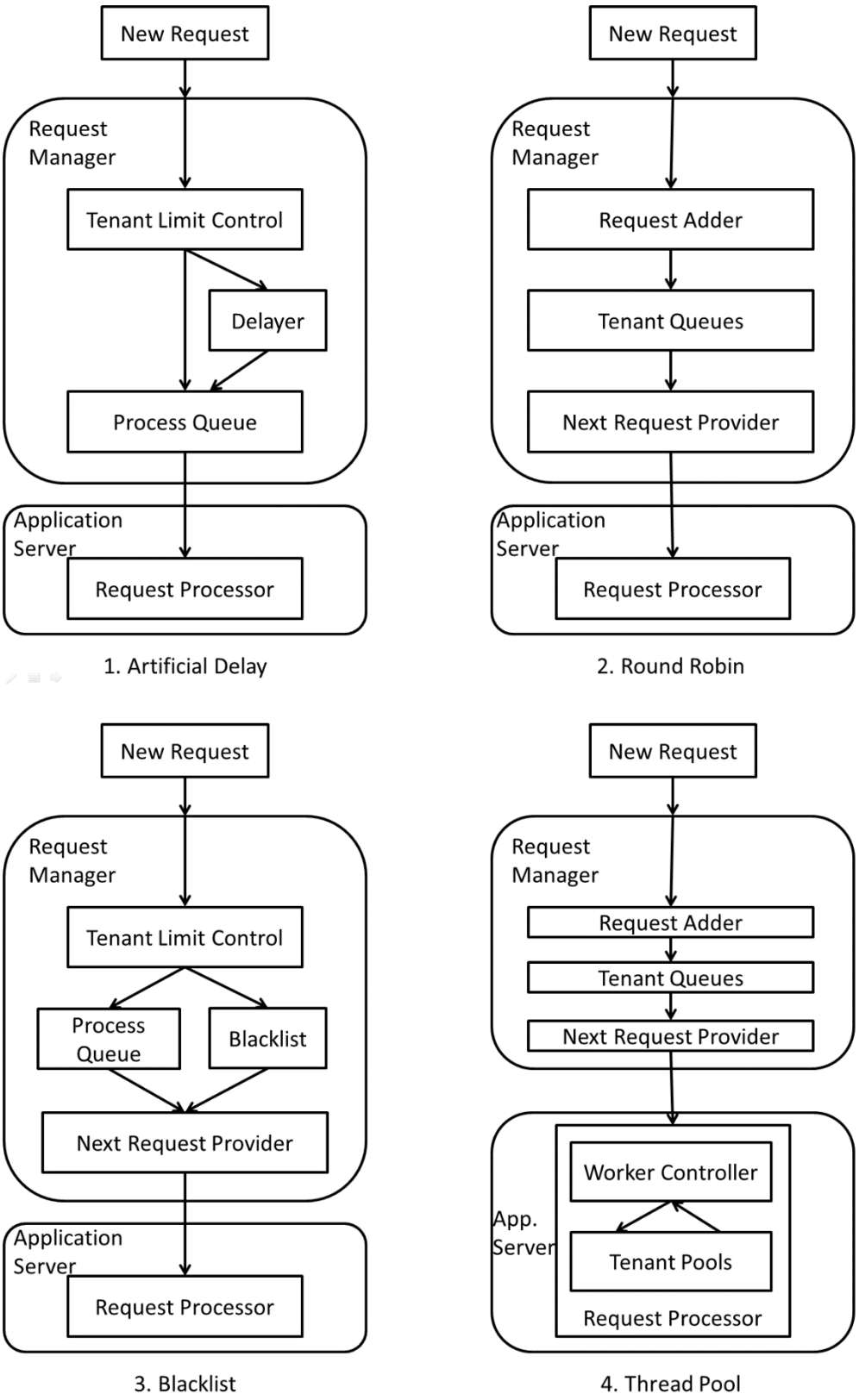

Figure 3Proposed performance isolation approaches in Multi-Tenant Applications

Artificial Delay

This approach generates artificial delay on tenant requests by considering the tenant limitations and its corresponding request rate. The reason for this delay is due to generating backpressure to the tenant, which is disruptive. This method expects increased response times for the disruptive tenants. Delay time for responses can be calculated dynamically considering the limitations of tenant, or it may be a constant value for all disruptive tenants. However, experiments show that the constant amount of artificial delay cannot maintain performance isolation since it does not prioritize the disruptive requests compared to abiding requests. Thus, the same workload coming from the disruptive requests is handled by the system in a delayed manner.

Round Robin

This approach creates FIFO-queues for each tenant and provides the requests first coming out by the queue in each turn. Since these queues are handled by the request manager layer, the application server does not need to consider prioritizing the requests or tenants. The application layer only takes the incoming request from the next request provider. The expected outcome of this technique is to distribute utilization of the system among the waiting tenants. When some queues are empty, and there are only few active tenants on the system, the empty queues are skippedand therefore the active tenants served as much workload capacity of thatthe system allows. Moreover, they use the workload capacity of the offline tenants, which provides a cost-optimized solution while distributing the workload.

Black List

The black list approach uses two FIFO queues for processing the incoming requests. Normally, the first queue is used to fetch the requests and send to the application server. The second queue is the black list queue and is only served in two cases:

1. When there are no incoming requests from the first queue.
2. Every n^{th} request is fetched from the first queue, where n is some large number to process the blacklist queue slowly.

When some tenants exceed the request limits, their requests will be redirected to the black list queue to push the distribute tenant back and process them slowly.

Thread Pool

This approach separates the workload resource pool reserved for each tenant by separating each tenants thread pools. Incoming requests first queued in the request manager layer, are to be ordered in a FIFO basis. Then thread pools in the application server processes the next incoming requests according to its available workload capacity.

2.2.3. Performance Isolation Metrics

Most of the isolation metrics in the literature with regards to shared environments are based on single aspects. These aspects include databases [22], cloud features like elasticity, [23] and traditional throughput metrics in a virtualized environment [24]. One of the recent studies presents a different kind of metrics to measure the performance isolation capability of a system. Feasibility of these metrics is discussed in detail [21] and case studyis further discussed in this thesis. To compare the isolation results according to the metrics given, in [21] it is necessary to consider the workload scenarios of the overall system. Isolation of system is a variable, and basic measurable variables are related with system workload and response time. For instance, only the response time does not supply enough information without considering the workload of the overall system. Similarly, only the workload of the system – number of requests - does not provide any information about isolation of a system without considering further aspects like number of customers with exceeding limits. In our case study, we consider further aspects to such customers with exceeding limits and how they change the performance results of the framework. By selecting the appropriate parameters for the workload and quality of service (QoS), defined metrics can also measure any measurable QoS related system for shared resource environments [21]. To evaluate the results of the case studyaccording to these metrics, we propose specific work load groups to distinguish the groups of disruptive and abiding customers. In traditional performance measurements there is only one group defined to show performance change according to the changing on the group size of customers. However, the major difference of the measurement used in this thesis is that we examinechanging onperformance inone groupwhile changing the workload of the other group.Theset of symbols defined in [21] are used in Table 1.

Table 1.Overview of variables and symbols

Symbol	**Meaning**
t	A customer in the system
D	Set of disruptive customers exceeding their request rate limits (e.g. defined in the SLA)|D|>0
A	Set of abiding customers not exceeding their request rate limits (e.g. defined in the SLA) |A|>0
w_t	Workload of the customer t. $w_t \in W$, $W = \sum_{t \in AUD} w_t$
W	Total system workload. It can be found by the addition of disruptive and abiding customer workloads.
$z_t(W)$	Reflects the QoS provided to customer t, represented as real number. QoS observation of the individual customer, which is the function of overall system workload. Lower values correspond to better qualities (low latency response time).
I	Isolation degree of the system. Index is used to separate different isolation values.

These metrics are dependent on at least two measurements. The first measurement is the reference point whereas;the second measurement is the disruptive case point. In the reference point, the quality of service results for all users in the system ($t \in A$) is measured as W_{ref}. Thena subset of abiding customers challenges the system isolation by increasing their workload, which this point is called the disruptive case point and is measured as W_{disr}. During the measurements of these points, the overall customer set does not change, and the union of A and D stays same. Relative differences of these two measurements are defined as QoS of Δz_a, and this is also defined as the reference quality of service compared to the disruptive quality of service.

$$\Delta z_A = \frac{\sum_{t \in A}\left[z_t(W_{disr}) - z_t(W_{ref})\right]}{\sum_{t \in A} z_t(W_{ref})} \quad (1)$$

In addition to QoS, relative difference of the workload is calculated as follows:

$$\Delta W = \frac{\sum_{w_t \in W_{disr}} w_t - \sum_{w_t \in W_{ref}} w_t}{\sum_{w_t \in W_{ref}} W_t} \quad (2)$$

Relative change in the service quality and workload gives a basis for how QoSis influenced with respect to work load:

$$I_{QoS} = \frac{\Delta z_A}{\Delta W} \quad (3)$$

This metric provides a basic overview for the system isolation. As it represents a high value, this metric shows bad isolation due to how low change of workload and affects QoStoo much. When it represents thelower value, it shows good

isolation since a change in the workload does not change the given QoS for all customers.

These metrics defined above may not be sufficient to define the overall isolation for a system since they derived from only two reference points (W_{ref}, W_{disr}) to calculate system isolation indices. Therefore,the selection of these two reference points is crucial to make theexperiment, more than one reference point tuples may be necessary to give better results. For this purpose these metrics are enhanced with arithmetic means of I_{QoS} results for m different W_{disr} workload.

$$I_{avg} = \frac{\sum_{i=1}^{m} I_{QoS_m}}{\mathrm{m}} \tag{4}$$

These changes in the reference and disruptive workloads are given lower and upper bounds as seen in Figure 4.I_{avg}defines a value for the system isolation considered by the set of customers and m variations of different workload cases. It is possible for a developer to focus only on the measurement of isolation under specific circumstances. Therefore, lower and upper bounds of the metrics will be determined by the provider who guarantees the SLA requirements under these circumstances.

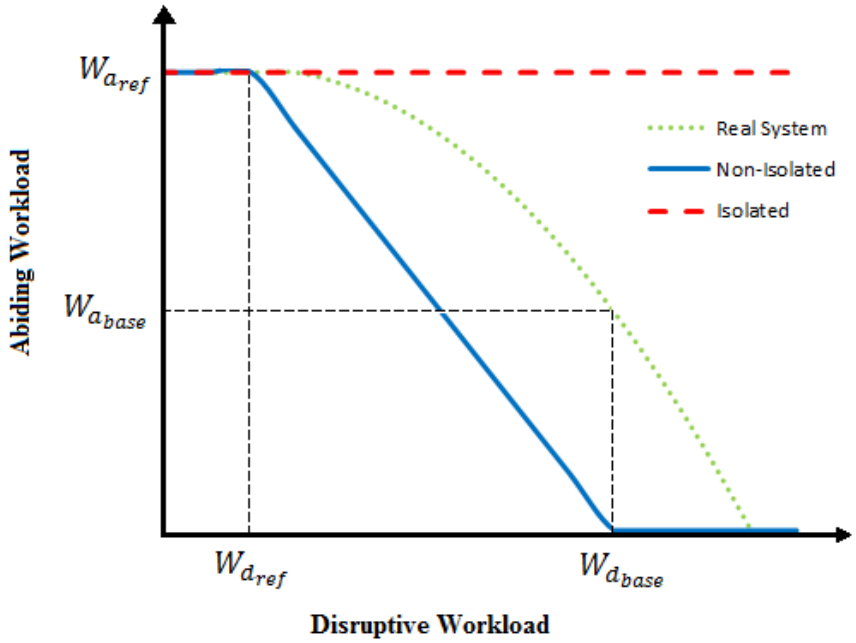

Figure 4Isolation curve with respect to workload bounds given in [21]

Workload Ratios

The key idea of these metrics is to keep constant QoS for abiding customers as the disruptive workload increases. However, supporting only for abiding

customers is not enough for a productive system. In a non-isolated system, one can measure the disruptive workload by increasing the disruptive customers and expect to observe that QoS is decreased for all customers. However,QoS for the abiding customers would remain the same since the workload of the abiding customers is also decreased, and is compensated on the increase of the disruptive workload. Therefore, function of $W_a$resulting fromW_d provides a Pareto Optimum point for the sum of all workload for stablequality of service for the system.

Figure 4 shows how systems maintain the abiding workload given disruptive workload. In a non-isolated system,QoS for abiding workload does not change at the beginning due to sufficient amount of workspace provided by the system. However, after a point, W_{dref}, workload for abiding customers is linearly decreased to compensate the total workload. Linearity inthe non-isolated systemis due to abiding customers that are also using the same workspace with disruptive customers. For an isolated system,the increase of disruptive workload does not affect the abiding QoS because their workspace is completely isolated. Given these baseline rules, some critical points in the figure for measuring the isolation are defined below.

Significant Points

Table 2.Significant points marked in Figure 4

Symbol	**Meaning**
W_d	Total work induced by disruptive tenants.
W_{dbase}	Disruptive workload as the abiding workload in the non-isolated environmentreducesto none.
W_{dend}	Disruptive workload that the abiding workload in the real system is reduces to none during the test.
W_{dref}	Disruptive workload point that the abiding workload just starts to decrease in the non-isolated system.
W_c	Total workload generated by the abiding tenants
W_{cref}	Abiding workload when W_{dref} occurs.
W_{cbase}	Abiding workload when W_{dbase} occurs.

Points marked in figure 4 and the definitions above provide several new ways to define new measurement metrics. In the non-isolated system we know that after point W_{dref}, W_a and $W_d$are linear inverse proportional. Therefore, we can derive the condition of $W_{aref} = W_{dbase} – W_{dref}$. To find the point where the abiding customer workload decreased to 0 for compensating the disruptive workload I_{end} can be defined as follows:

$$I_{end} = \frac{W_{d_{end}} - W_{d_{base}}}{W_{a_{ref}}} \tag{5}$$

Similarly, referencing the W_{abase} and the relation of W_{aref} results another isolation metric with a I_{base}; it has a value between [0,1]:

$$I_{base} = \frac{W_{d_{base}}}{W_{a_{ref}}} \tag{6}$$

Disadvantages of these metrics are that they assume the linearity and do not take the curve progression. This case causesthe following problems:

1. The system behaves lineardirection before it comes to a short distance from W_{dbase}, but afterinstantlydecrease to $W_a = 0$;I_{base} would equal to I_{end} which is unfair for the non-isolated system.
2. For an isolated system dropping $W_a = 0$ may require very high disruptive workload, thus experimenting this case is hard to settle.
3. I_{base} only represents the isolation of the environment between the value of W_{dref} and W_{dbase} that may lead to a misinterpretation of the results, since after W_{dbase} to W_{dend} points may result in differently.

Integral Metrics

To solve the problems caused by the metrics defined above, [21] defines the integral metrics that is calculated by the area under the curve (real system workload) subtracted by the area under the triangle (non-isolated environment workload).

$$I_{intBase} = \frac{\left(\int_{W_{d_{ref}}}^{W_{d_{base}}} f_m(W_d)dW_d\right) - W_{a_{ref}}^2/2}{W_{a_{ref}}^2/2} \tag{7}$$

$I_{intBase}$, represents the ratio of the isolated area and non-isolated area. When the value is 0, the system is considered to be completely non-isolated, and when it is 1,the system is considered to be completely isolated. One disadvantage of this metric is that it only considers the isolation within the workload limits of W_{dref} and W_{dbase}. To extend this metric further, the following metric which determines the real-world system isolation can be used.

$$I_{intFree} = \frac{\left(\int_{W_{d_{ref}}}^{p_{end}} f_m(W_d)dW_d\right) - W_{a_{ref}}^2/2}{W_{a_{ref}}\left(p_{end} - W_{d_{ref}}\right) - W_{a_{ref}}^2/2} \tag{8}$$

In this metric, p_{end} represents the maximum artificial limit of the injected disruptive workload. For feasible measurement of isolation, injected disruptive workload can be changed according to the case scenario. By using $I_{intFree}$ it is possible to determine the isolation degrees of the system under the maximum workload of p_{end}.When the result is 1, it represents for the perfect isolation, whereas, 0 for the non-isolation.

3. Problem Statement

In this section we describe the problem statement regarding the design of SaaS systems with performance isolation support. Section 3.1 describes the problems related to deriving application architectures from the earlier defined SaaS reference architecture. Section 3.2 identifies the problems of current performance isolation approaches in SaaS.

3.1. Deriving Application Architectures

Recent studies show that SaaS based applications have common architectural patterns [31,48], and require high re-usability [49, 50] during the domain analysis and implementation stages of the projects. This commonality causes efforts forsimilar patterns to test, code, and analyze each functional requirement of the application to implement them in a performance optimized and errorless way.

A sample SaaS architecture pattern is provided in section 2.1,however it does not specify any application architecture. The number of tiers, allocation of layers given nodes, and data storage nodes can be changed from application to application. On the other hand by using this pattern, it is possible to generate multiple SaaS application architectures. Each application architecture design differs for different requirements and constraints, thus alternatives of such application architectures affect the performance isolation and algorithm of the application. Currently, the design of the SaaS architecture is largely defined manually, and is applied on the system according to the application logic. However, for an application, manually configuring the SaaS architecture is not a feasible process for two reasons. First, developers have to choose answers for optimizing their application architecture; such questions include, “How many tiers should be used?”,“What kind of logging and security mechanism will be selected?”, and “Which protocols should be used for data migration for each layer?”. Second, to validate the optimality of these answers,performance isolation has to be measured under different circumstances. These circumstances may consist of different layers of the application tiers as well as on existing approaches for performance isolation techniques.

Consequently, for both developing and validating of the application architecture from reference architecture, manual implementation of these circumstances may cause implementation faults and may require additional development time for

customization of each different application. Therefore, an automated support for guiding the SaaS applications is necessary to build the application architecture from the reference architecture. In addition to this, measuring the performance and scalability effects of the given applicationis necessary for validating purposes.

Figure 1 is one of the reference SaaS architectures; however, for a feasible implementation of the SaaS application, the application architecture has to be changed from the reference architecture. Figure 5 shows alternative application architecture derived from the reference architecture in Figure 1. This type of application architecture may be needed for low database;and high compute utilized applications for load balancing the tenants to separate application tiers, whereas they may use the same database for consistency.

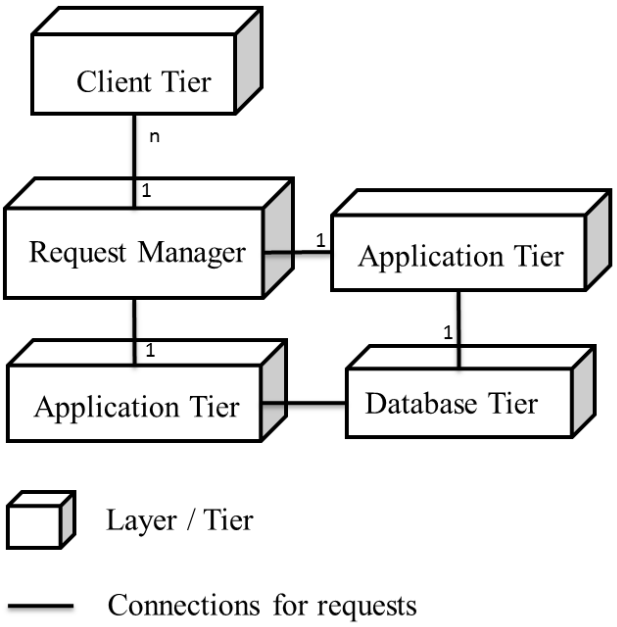

Figure 5 Alternative SaaS architecture with multiple application tiers

Similarly, Figure 6 shows other alternative application architecture derived from reference architecture. In this case, the application may use high database related operations, and utilize a database greater than the computational operations done in the application tier. Therefore, separate database tiers may be suitable for separate tenants for a consistent database application, or the application may divide the overall data into separate database tiers and may not need to consistently store the data according to the needs of the application.

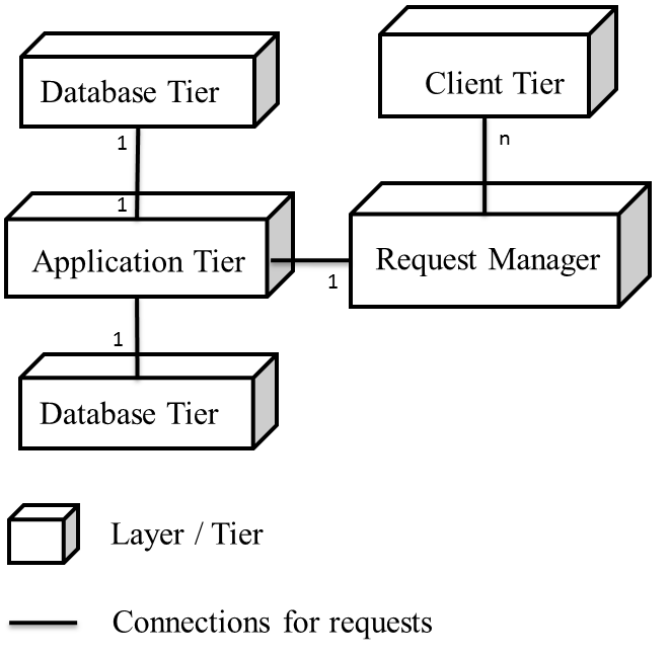

Figure 6Alternative SaaS architecture with multiple DB tiers

3.2. Architecture Design for Performance Monitoring

In section 2 we have provided the background on performance isolation approaches in SaaS. An important concern in performance isolation approaches is to meet the fairness requirements. To supply fairness criteria, workloads of the different tenants should not violate other tenants' limits. In SaaS architecture, to ensure workload limitation for a tenant, the required workload size for a given request has to be known. Since there are multiple different requested actions in one application, the actions required for these workload sizes are also different. This comparison of the workload size is not feasible in real world applications. Furthermore, the existing approaches assume that the server tasks are executed atomically;however, this is not the case especially for thin clients that depend on the cloud server resource pool. Because thin clients execute all the actions directly on the cloud resource pool it causes different amounts of workloads on the same resource pool, which is not a feasible case for performance measuring. In alignment with the main application and expectations for cloud computing, existing performance isolation approaches assume these kinds of thin clients.

As such, for achieving the fairness criteria, the performance isolation approaches tend to focus on controlling the cloud resource pool only, whereby it is assumed that the client cannot take over some of the required total workload. This assumption causes the following problems on the existing approaches:

Approach 1 – Artificial delay

In the artificial delay approach, a tenant that has exceeded the assigned resource quota is deliberately delayed to meet the overall fairness criteria. This situation is based on the assumption that the client cannot take over its own workload. In the case where the tenant is able to handle some actions on its own workload, the tenant will not be unnecessarily delayed. Furthermore, since this knowledge is based on similar workloads for different tenants; in the case when different tenants request different actions, performance isolation will be violated by the tenants who send less request rates with high workloads. The application layer of the SaaS architecture is only able to evaluate the request rates and not evaluate the required physical workload for a given process.

Approach 2–Round Robin

The round robin approach is considered fair since it processes the action of the next tenant one by one,at a time. However, this condition is based on the assumption that the tenants may have similar workload sizes. In the case when some tenants require actions which need high utilization of workload, request weight will not equal with other tenants, therefore, the one-by-one at a time approach is not considered fair anymore. Furthermore, when using this approach on a thin client that uses the cloud server resource pool to execute requested actions, workload space cannot be divided on the client side. This generates incomparable workloads by requesting big and small actions for tenants which indirectly violates the fairness criteria. On the other hand, by using the client resource pool efficiently,the workload divide may provide similar workload size to justify fairness criteria.

Approach 3 – Black List

Blacklisting the disruptive tenants is another approach to justify the performance isolation. However, disruptive tenants are considered on request rates, and thus this approach may cause problems by blacklisting the abiding tenants which sends high request rates with low workload need, instead of tenants that utilize high amount of computational power of the server. Another problem of this approach is that by assuming the tenants as they are thin client who use the cloud server resource pool only. When the tenant exceeds its quota and is therefore put in the blacklist, it is blocked even in case it could use its own resources to continue the required tasks.

Approach 4 – Thread Pools

First step of this approach is same as the round robin approach. First, it orders the requests by using FIFO queues for each tenant, and then makes these requests available to the process by application threads instead of single application process in the round robin approach. Therefore the main work of choosing the correct tenant's request is done by the application threads. However, application threads handled by the request workloads from lower levels than the SaaS because it depends on the operating system, which makes this approach out of the scope of the SaaS based performance isolation approach.

Table 3. The basic performance isolation approaches and the related problems

Performance Isolation Approach	Problems
Artificial Delay	• Variation of workloads is not considered, thus tenants request rate statistics violates fairness criteria. • In case of thick clients unnecessary delay of the tenant.
Round Robin	• Variation of workloads is not considered, thus equal turns violates fairness criteria.
Black List	• Variation of workloads is not considered, thus tenants request rate statistics violates fairness criteria. • In case of thick clients unnecessary delay of the tenant.
Thread Pools	• Applied for PaaS. • Not suitable for SaaS

4. TorkApplication Framework

In the previous chapter we have identified two important problems related to the design and development of SaaS application architectures. First, it is not easy to derive the SaaS application architecture from the SaaS reference architecture based on the scalability requirements. Second, existing performance isolation approaches seem to have primarily focused on thin clients with similar workload requests and do not explicitly consider the computing power of the clients and various workload requests.

In this chapter, we provide an application framework [27]to support the derivation of the SaaS application architecture and for performance isolation. The application framework implements several modules for developing SaaS systems. Within the framework, we provide modules for supporting performance monitoring and performance isolation.

The chapter is organized as follows. In section 4.1, we first describe the requirements for the application framework. Section 4.2 describes the overall design of the application framework. Section 4.3 focuses on the performance isolation using the framework modules.

4.1. Requirements for the Framework

The following criteria are not only, but also include some of the necessary steps for satisfying the framework that provides easily deriving new applications and supports performance isolation.

This first criterion set is to provide easily deriving new applications:

1. To support multiple deployments, framework should support models for multiple nodes per tier.
2. To support easy initialization of re-usable SaaS level components, the framework should provide core components for visualization, dataset, forms, grids, logging, security etc.
3. To support easy development cycle and customizations, the framework should provide an integrated development environment for writing the code or changing the parameters.
4. To support different decompositions of the applications, the framework should support changing of component belongings to other tiers. This

will provide services for the thin client based applications and thick client based applications.

The first criteria focused on the elasticity of the cloud application. By supporting the multiple nodes per tier in the framework, framework eventually provides elasticity solutions.

The second criteria focused on the reduction time of the initialization of the application; for an existing system it is possible to initialize the done work via re-usable components rather implement from the stretch.

The third criteria focused on the reduction of the development time after initialization. In the framework environment, after initialization, most of the time the developer may only need to adjust parameters of the existing components, or may need to write their custom codes. To enhance this, an IDE may provide solutions for versioning the code files and provide environments for writing code online.

The fourth criteria focused on support for the algorithmic details of the cloud application. Some applications or some actions inside the application may consider thin clients and compute the process at the cloud. On the other hand for some applications or actions, it is possible to compute at the client side, which provides scalability and performance isolated solutions. The framework should consider these different implementation styles and be able to support both of these scenarios.

This second criterion set is to support performance isolation at the SaaS level:

1. To separate the resource pool as much as possible, application logic has to be run on the client side as much as possible.
2. To assume similar workload capacities requested by the application tenants, requests have to be as atomic as possible.

The first consideration pushes SaaS applications to support thick clients. Since each tenant uses the different resource pool, different hardware is easier to perform performance isolation. Minimum hardware requirements can help determine different client types, specifically to satisfy SLA requirements.

The second consideration pushes SaaS functionalities to divide smaller functions. By dividing one request to smaller atomic requests, the application provides a better quantification method for workloads on different functional use

cases. For instance, F_i is a functional use case in the application and T_i is the tenant requiring the F_i. In a situation, F_1 requires n atomic operations to complete and F_2 requires m atomic operations to complete. It is possible to compare workload of tenants by T_1 requested n units of workload whereas T_2requested m unit of workload with the assumption of atomic requests cause the same workload. These atomic requests can be done in parallel as well as in serial, based according to the functionality. For serial cases, constant network delays may affect the performance; however, in parallel cases, functionalities are done in one unit of time if the given limit to the specified tenant does not exceed. Therefore, this design approach is beneficial to measure performance isolation and find the isolation degree of the application by reducing the workload difference in different use cases in the application.

4.2. Framework Architecture

The most basic architecture of the Cloud based SaaS enterprise applications use client–server architecture [12], where the server serves the application layer and the database layer if any database is available for the application. On the other hand, client/s would request functional operations to the server. They can be separate nodes, and may use different platforms, hardware, etc. For example, it is also possible to use a server as a client node for executing the scheduled jobs.

In theTork Framework, the main goal is to maintain scalability for one server and the multi-client environment first, then scale out the node-pairs horizontally to keep scalability proportional to the customer size. Therefore, the key idea is that the maximum number of client demands should be served with minimum amount of server computation. General architectural design of the Tork Framework consists of the application layer with RDMS and client layer (similar to Figure 1).

Tork Framework has the motivation of using the client computation power inside the cloud based SaaS architecture development. Therefore, most of the functionality of the application can be deployed into the client nodes as the newer versions are released.

In order to serve different platforms and web browsers at the same time within the same codebase, HTML5, JavaScript and CSS3 technologies are used. The major advantage of this is that, the code is processed on the JavaScript engine and uses web view to visualize the software interface. These technologies are

now supported by a wide range of hardware clients from mobile phones to IP televisions [10, 11].

To process the data on RDMS, restful service architecture is used. Restful requests migrate data from the DBMS layer to the client dataset. This provides a service oriented architecture between the client and host layers.

4.2.1. Application Layer

The application layer is the middle layer between the client and server. The main purpose of this layer is to maintain the migration of the data from server to client for read requests, and client to server for update, delete, insert, and execute requests. Request details are specified from client nodes. These are the main duties of this layer, and the basics for all cloud SaaS applications. There are optional features that can be performed from the application layer. These features are user authorization controls, multi-tenant filtering controls, logging, etc.

If the cloud application works on the private cloud with no tenancy and different users, then the user security feature may not be necessary for the system. On the other hand, if the application is served from the public cloud for different users, anonymous clients can change private data for others via create, read, update, delete and execute (CRUDE) operations. This may cause a security leak for an enterprise application. In the Tork Framework, there are two ways to authorize requests coming from the users. These are session-less authorization, and session authorization.

In session-less authorization, each request coming from any client should indicate its credentials, and it must be authorized before starting the processing. This authorization can be done from the RDMS or from the application layer by using the pre-cached user data-table.

For session authorizations, only at the first request, the client indicates its credentials to server. Then, the server gives a unique session key to the client if the credentials are correct. After the first request, the client does not send credentials, but the session key as validation of grant.

Differences, between these authorization mechanisms have some effects on scalability. In session-less authorization for every request there is a credential validation, which uses the computation power of the server nodes and maybe RDMS. On the other hand, with this technique, it is possible to spread the client

requests to different application layers (making a load balance along multi servers) natively, since there are not any givenunique session key that belongs to one application layer. In this way, clients can be supported to make parallel CRUDE requests. For session based authorization systems, it is a good way to skip the authorization part of the processing after processing it once per distinct client. Skipping the authorization will reduce the response time of each request. On the other hand, the server has to memorize each session key for different clients. Parallelism of requests in a session model can also be another option. In this case, the client has to map each session key with the associated server and send parallel CRUDE requests to these servers with session key. An authorization of this method should be chosen according to the application requirements. Wrong design decisions may cause scalability problems if too much concurrent clients take session keys from the server.

Multi-tenant filtering is another optional control that can be used from the application layer. Multitenant filtering provides an easy development cycle for SaaS developers. In the Tork Framework, queries coming from clients do not consider multi tenancy, because they are sent as there are not any tenants in the RDMS. The application layer satisfies the filtration requirements of data by modifying the request. The Tork Framework assumes that specified tenant separator fields exist on each table that is shared with multi-tenants. For instance, thecompany table indicates the tenant source, and theuser table are shared among different companies' users. In this case CRUD operations on the user table are filtered according to the requested user's tenant identification.

In the configuration file or security table in the RDMS (for multi-domain tenancy), theTork Framework asks the tenant and the user table to find relations of the multi-tenancy among other tables in the application. After querying these tables once at the Javascript business object model (JBOM) file creation step, related scheme results are cached into a meta-file at the application layer for quickly processing the requests. Incoming read, update, delete and execute operations to the tenant related tables are directly filtered by the requesting user's tenant ID even if the user explicitly indicates another value for the tenant field, it is overridden by the application layer. This configuration maintains multi-tenant security and reduces the complexity of the application development cycle.

Some cloud applications may use different tenant dimensions because of the application logic. In this case, different domains are running on the same cloud application, but they are viewing the data from another perspective. For instance,

a cloud based property management system (PMS) may use the multi-tenancy among the domain of property owners. Therefore, they may use the propertyId field to separate each property for the shared tables among the properties. The same application can start to service for the travel agent domain. Since agencies can have multiple properties in the system, they would like to see all of their properties without filtering any of single one instead. In this case,agencyId field would be the tenant separator and it should be used to filter the tenant during the filtering process.

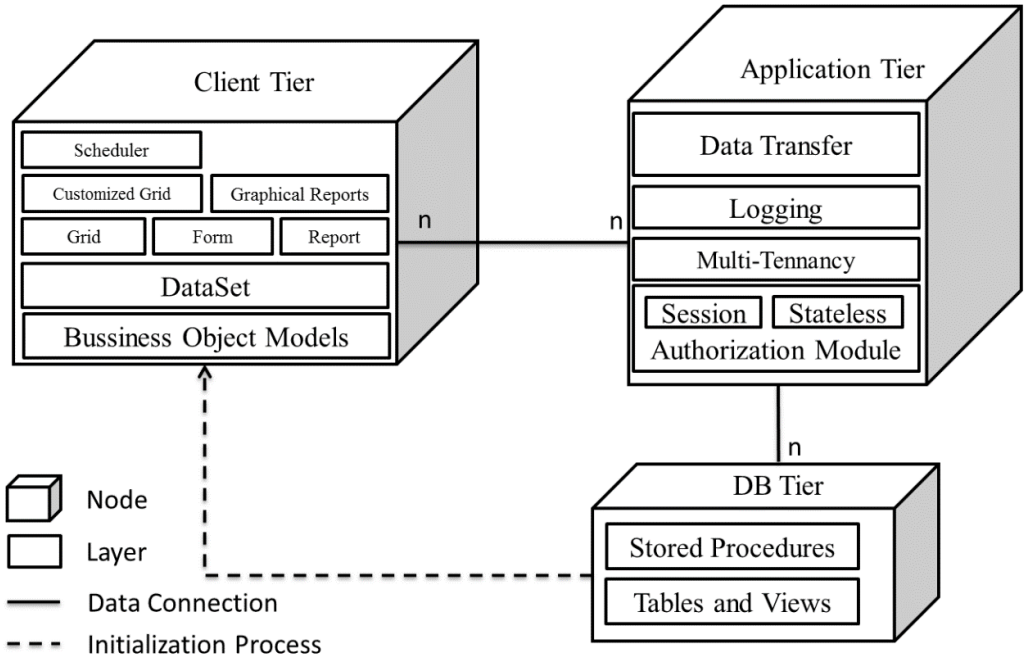

Figure 7Architectural Approach of Tork Framework

In some cases one domain such as agencyId may include another domain such as property. In those cases, the wider domain's tenant field should exist on all tables that have the included tenant’s filtering fields, otherwise application logic would break down in Tork Framework due to inconsistent data retrieval.

The second feature of the multi-tenant filtering component is making opposite filters to show read-only data if it is shared among some tenants. In order to reduce the duplicate data in the RDMS and reduce computations, some records can be related with more than one tenant. This case occurs when tenants interact with each other. The examples may be service requests, or balance transactions between two application-users from other companies. A record owner can edit the transaction record, whereas a shared company only reads it. This feature can be enabled by adding share field on desired tables. Read-only tenant sharable

fields are shown for shared tenants by expanding read queries of the client requests.

After the multi-tenant filtering stage is complete, therequest is redirected to the logging stage if the logging module is enabled. A standard log table of the framework logs with the following fields of the CRUDE operations which are outlined in Table 4. The *ACTION* feature indicates the type of the CRUDE request. It can be one of the C, R, U, D, E characters. The*MODELNAME* feature indicates the name of the business object model on which the action has been operated. This can take values of table names, view names, or stored procedure names. Tork Framework assumes that these names are no longer than 50 characters. The*BEFOREDATA* and *AFTERDATA* are used for logging the actual data on which the operation will take. These fields contain the JSON formatted record data. The*BEFOREDATA* contains the original record before the update, or delete operations. The*AFTERDATA* contains the modified version of the record after the update, insert, or delete operations are complete. These fields are useful to save versioning for the records in the RDMS. In some cases, by showing the record history, it is possible to rollback records into one of their previous versions. The *MULTITENANTID* stores the tenant identification of the requester. The reason why the data type is nvarchar(50) is because theTork Framework assumes that identification formats are *int, bigint, nvarchar, or uniqueidentifier*. All of these types can be cast to nvarchar. Similar to *MULTITENANTID,LOGOWNERID* stores the identification of the requester. The*RECORDOWNERID* identification of the first creator is the record that the action is taken. This value is the same as *LOGOWNERID* during create operations. The*RECORDID* is the identification, the primary key of the record in RDMS. The *ISATTEMPT* field is used to trace whether the request completes successfully or not. When RDMS gives an exception and rollbacks the operation, this exception is logged as a request attempt, and the*ISATTEMPT* field contains 1 in this case. The*ACTIONTIME* contains the specific date and time of the request when it arrives to the server. The*IPADDRESS* field contains the IP address of the client which sends the request.

Table 4.Features of the Standard Log Table

Fields	Data Types
ACTION	char(1)
MODELNAME	nvarchar(50)
BEFOREDATA	nvarchar(max)
AFTERDATA	nvarchar(max)
MULTITENNANTID	nvarchar(50)
LOGOWNERID	nvarchar(50)
RECORDOWNERID	nvarchar(50)
RECORDID	nvarchar(50)
ISATTEMPT	bit
ACTIONTIME	datetime
IPADDRESS	varchar(20)

Some fields take empty value because of the action types. For execute requests, BEFOREDATA and AFTERDATA are not defined, because context of the execute requests are not known from the application layer therefore, they may not be acting on a single record but may act on entire database. The log table given in Table 1 shows the regular logging fields for giving idea to developers. The log table can be reducible or expandable according to the needs of application. Tork Framework initially enables logging feature for all tables and procedures, but it can also be disabled for some tables and procedures parametrically. This can be done, when some procedures are executed from anonymous clients and there is no need for logging. As a result, the optimization of the logging process can be achieved.

The last step of the application layer is the data transfer step (see Figure 7). This step connects to RDMS layer and fetches the data. Initial fetching query comes from the client side and may be modified from the application layer for security settings and multi-tenant filtering status. There are some connection-types that RDMS supports. Some of the basics are connections over TCP/IP protocol and shared memory access. TCP/IP protocol is widely used to connect to databases over the internet. It offers advance security features and could be preferred when the DB and the application layer are on the separate nodes. On the other hand, shared memory access protocol is the simplest one with, no configurable settings, and it uses the shared memory to connect DB layer with the client.

When the DB and application layer are on the same node, this protocol would provide performance optimization.

Queries are executed after connecting to the DB layer. For the read and execute operations, the Tork Framework assumes the fetched data is in the list/table form. For execute operations multiple resultsets, therefore multiple lists/tables may return from the RDMS. Each resultset returned from the DB layer is processed to serialize as JSON format. This JSON string is used to transfer data to the client layer as the response of the Restful service. For delete, update, and create requests after the operation is done, the application layer simulates read request to DB according to the client's requested parameters. After performing the read operations, the Tork Framework generates one more query for optimizing the client's caching performance. This query fetches the number of record count with the final filtering conditions. Final filtering conditions are created by the initial client filtering and multi-tenant filterings. In this case, when the number of record size is equal to the fetched record size, then the client will be able to run offline for some time. Because all the specified data is already migrated to the client side, the client does not need to make requests for sorting, further filtering and paging tasks. This provides additional scalability advantage to the overall application.

Each component of the application layer can be deployed into another node as well as in the same node. Components can run parallel for concurrent requests for the sake of scalability. Except the data-transfer component, none of these components have to wait for the RDMS layer. Only the transfer layer and the logging layer use the RDMS. However, the application may prefer eventual-consistency for the logging part, so logging requests can be done asynchronously. The application can also prefer to use another DB node for the logging part of the system. In this case, the application layers can communicate with another log database asynchrony. This case reduces the bottleneck caused by the logs on RDMS of the application.

4.2.2. RDMS Layer

To easily maintain the –codebase-, the Tork Framework makes two design approaches assumptions for the RDMS layer of the cloud applications. First, it assumes that all tables which are capable of being updated or deleted have to need at least one unique primary key. These primary keys are used to map the objects on the client-side with the DB side. In some cases, the mapping is used to synchronize the data between client and the DB. These primary keys are used as foreign key columns in related tables; in this case, related tables have the belongsTo relation, whereas primary table has hasMany relation for the each related table-column pairs (See Figure 8).

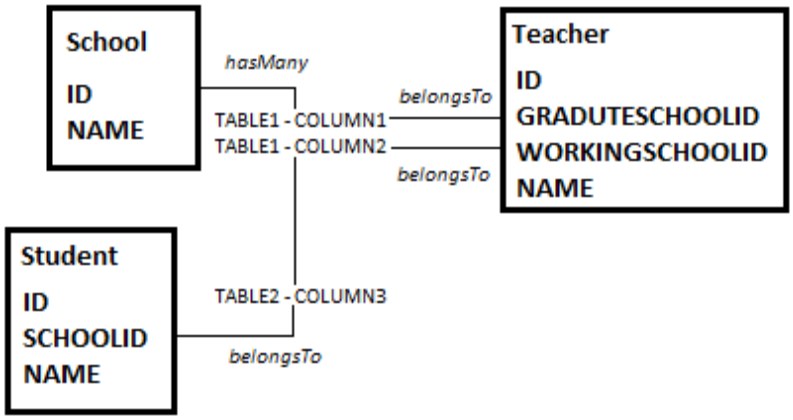

Figure 8 Sample hasMany – belongsTo relations

These relations are automatically extracted (to the JBOM file to simulate the view's logic behind the DB layer into the client-side (See Figure 9) via master form and the detailed grid representation. Detailed extraction algorithm for this process is explained at Section 5.1.

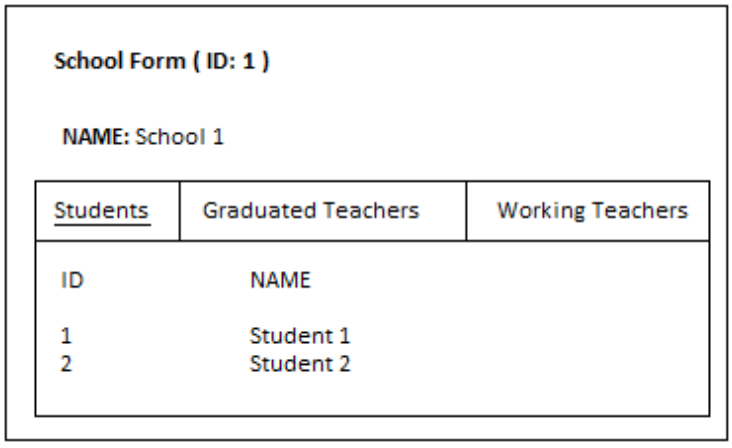

Figure 9Master form and detail grid relation

Second, all the tables that are shared among the tenants need to have a tenant-field. This field indicates the relation of the records with the tenant table. For these tables, it is wise to use a clustered index since the tenant-filtering process always partite these records according to the tenants first and then applies the client's filtering criteria. With clustered indices this process will be more optimized.

Consistent replications of the RDMS layer are possible and are relatively a hard design problem, since availability and responsiveness of the system would decrease during the real-time synchronization of DB layers for consistent data management due to the CAP theorem [25]. To simplify this problem, some architectural changes can be applied according to the application logic. For the multi-tenant cloud applications, there are two different changing data regions from the database perspective. These are shared data and tenancy data. Shared data is available and usable for all tenants. These are generally unchangeable system constants and most multi-tenant applications do not give security grants for these tables to be changed by tenants. Country codes-names, languages codes-names, currency code-name etc. are some examples for these tables. These kinds of tables can be transferred into another separate RDBMS layer, and this layer would be easily replicable since most of the tenants use it as read-only. The second changing data region of the RDMS layer is for the tenancy data tables. These tables can be changed by the all tenants in the application. However, only the owner tenants use their own records inside these tables. This is caused by the tenant-filtering process in theTork Framework. To optimize the processing time for these tables, the tenant filtering process can be reduced by clustering the same tenant’s data into different RDMS layer. This process can eliminate the tenant filtering process and would provide horizontal scalability to the system. Tenant count would increase or decrease proportional to the tenant-clustered RDMS layer without bottlenecking to the DB layer and so for the application availability and system responsiveness. On the other hand, this optimization concept would not be useful if the application is using more than one tenancy domain. In this case, tenancy tables cannot be separable with only one column. More than one column and,more than one domain will relate with the owner of the record. Only when the tenancy domains are inheritable, the tenant column of the most general domain can be selected to separate the records.

4.2.3. Client Layer

Client layer of the Tork Framework consists of the basic functionalities of an enterprise application. These functionalities can be categorized on four criteria: interface elements, view systems, data management, and server input/output operations. Architecture for these functionalities in Tork Framework is based on model view controller (MVC) pattern.

Model part of this pattern handles the data management, and server input/output operations. Data management on the client side is based on the JBOM file which also represents the actual DB scheme of the application, and dataset component of the Tork Framework (Figure 7). Dataset component is responsible for data migration and for server input/output operations during the CRUDE operations. These operations are then mapped on the client-side prototyped JavaScript objects which help the JBOM scheme definition file. Datasets can be synchronized with actual DB in two ways. In the first way, client-side modifications can be updated, inserted or deleted. The second way is the synchronization of the server-side modifications. These are migrated to the client via read operation. Datasets are the base components due to their basic functionality of the data migration. They mostly belong to other view components such as data-forms and data-grids.

The view part of the pattern responsible for the interface elements and view system consists of layout managers, localization modules, themes, styles, and user interactions such as gestures and events. Interface elements of the Tork Framework make the application platform independent. Since these elements are designed to be responsive for different screen sizes and touch devices, they can be widely used on most of the HTML supported devices. These devices may include smart phones-tablets; smart TVs and PCs. Layout managers maintain the form and list designs of the application with the JSON object format. In the runtime of the application, this format is converted into HTML, CSS, and JavaScript code to display the view elements and response according to the device's screen. The most basic framework components for the view part of the pattern are grid component (data-list component) and form component (Figure 9). These components can be customized, and are inheritable by more sophisticated components such as scheduler view and graphical charts, reports,and module.

The controller part of the pattern is mostly designed by the developer of the application. In this part, developers can implement new actions. An action is basically a method that processes the HTML event parameters and produces a

result to be issued by the client. An HTML event can be something that the browser, user or Tork Framework does. Tork Framework events are triggered within the components of the framework. To customize existing framework actions, these events are triggered along the way of execution on all different functional methods. Therefore, developers may also override the existing framework actions by overriding the specified events which are necessary to be triggered before executing the actions.

Developers implement their actions,and then bind them to some events. The Tork Framework is responsible to trigger these events. Binded actions to the triggered events are then executed. With this software pattern, custom developer code is transferred into client-side and organized into different actions. Since JavaScript natively supports the HTML event listeners, it is a useful approach to code the actions in this way. Since developer crafted code is downloaded into the client and executed using the client computational power, it provides general cloud scalability. Because newer versions of the application can be distributed through different content distribution networks (CDNs), this can reduce the network delays and supports the easy software versioning update. On the other hand, some non-cloud optimized frameworks use the application layer to deploy their controller actions. General reasoning of this action is to support thin clients. However, this can cause two issues. First, it consumes the application layer's computation power while reducing the performance of the overall application relative to the client-side controller versions. Second, each event execution on the client side will need to go to the application layer to check if there is an existing action related to the event to execute. In some cases due to network speed, these controls may significantly lower the application responsiveness and may increase the CPU of the application layer which also results in the reduction of the overall application performance.

4.3. Performing Isolation Solutions

In section 2.2.2, we examined four proposed solutions for the performance isolation in the literature. In this section we will focus on the first three solutions, and adapt these solutions into the Tork Application Framework. Since the fourth solution is not directly related with the SaaS layer due to using the application pool, threads are controlled through lower levels such as IaaS or PaaS. We did not implement this solution into the Tork Framework.

Application tier of the Tork Framework managed with IIS, application codes are written with ASP.NET and C#. Since the proposed solutions are based on the request managing and balancing the request priorities, we implement these to the global.asax file. This file is used for writing codes which runs on the system level events during the application lifecycle. In this case, we write codes for the following events:

- Application_Start:

  This event is triggered when the application is initialized for the first time. In this event, we coded the selection of the isolation algorithm and initialization of its depended components. These components are necessary to keep track of the tenant quotas and statistics.

- Application_BeginRequest:

  This event is triggered each time when a new request comes in. In this event, we coded the delaying methods related to the initialized isolation algorithm. These delaying methods are executed based on the request arrival time and the previous quota records of the given requesting tenant.

Global.asax file uses the following codes for implementing the proposed solutions. This code file includes three of these solutions, and any multi-tenant C# application may use this file similar to a performance isolation plug-in. Code file is attached as Appendix A.

Table 5. Methods Used in Performed Approaches

Artificial Delay	1. initDelay
	2. waitAsDelay
	  1. checkLimit
	  2. insertRequestStatistic

BlackList	1. initBlackList
	2. waitAsBlackList
	  1. checkLimit
	  2. insertRequestStatistic

Round Robin	1. initRoundRobin 2. waitAsRoundRobin

Performed methods for the approaches are described with stack order in Table 5. For all approaches the first methods are used to initialize the related algorithm. This is necessary to initialize the related data structures if they did not initialized when the time application has been started. These methods are *initDelay*, *initBlackList*, *initRoundRobin*.

The second methodsin the performed approaches are the waiting methods that are used to wait the related request. In the BlackList and RoundRobin approaches, this methods use first in first out (FIFO) queues for storing the registered tenants. One FIFO queue is used for RoundRobin, and two FIFO queues are used for RoundRobin approach.

For the Artificial Delay approach, there is not any data structureto register waiting tenants. However, both BlackList and Artificial Delay approaches use data structure to register tenants for controlling the tenant limits to wait accordingly. Thus both delay related approaches use*checkLimit* and *insertRequestStatistic* methods in common.

5. Tool Support

In section 5.1, we described the initialization of a cloud application from the Tork Framework. This includes automated code extraction from the DB layer to the client layer for fast prototyping. In section 5.2, the IDE tool for Tork Framework was represented. Its graphical user interface, versioning, and deployment capabilities were described.

5.1. Initial Construction of the Application

Baseline auto-generated project code in the Tork Framework comes from the DBMS layer. The DBMS layer is customized for each application according to the requirements of the application logic. Tork Framework assumes that business object models are defined as tables; customized lists are defined as views or stored procedures, customized, and atomic functional actions are defined as stored procedures in the RDBMS.

Tork's built-in code extraction feature is executed to extract the database scheme into a JavaScript business object model (JBOM) code file (is presented in Figure 10). The JBOM file can also be generated from existing RDMS applications. This file indicates DB relations, field validations and DB schemes of the tables, views and stored-procedures in the JavaScript Object Notation (JSON) format.

There are two baseline DB relations supported from the code extraction feature: hasMany and belongsTo relations. These are gathered from the foreign keys of the related tables. Once the baseline relations are defined, virtual hasMany, belongsTo relations can be defined upon the application. Instead of using foreign keys from the DB layer, *uniqueidentifier joins* can be defined upon application. Once the client side knows the scheme relations, it can prepare the execution plan for the each read request, by considering the designs of master-detail forms and relations between tables. Therefore, it reduces custom code and reducesthe computational time of the application layer on fetching the tables and deciding the join relations.

Field validations are the basic database validation rules that are supposed to be satisfied during the migration of the data to DBMS layer. Most basic validations are presence, minimum and maximum value, or length validations. Custom validations or constraints can also be defined at the DBMS layer. However, it is not practical to convert these customized validations into the JavaScript code.

Therefore, in some cases during the data migration, the DBMS should check and give an error message if there are some validation exceptions. Validation checks at the client layer make the system more responsive and scalable by reducing the network traffic and computational operations. In some cases, data-validation is important only during operational time, for instance, during the insert of a new record and not important for consistency in the DB layer. In those cases, it is possible to add custom validation rules on top of the baseline validation constraints generated from the DBMS. Implementation of these constraints at the client layer may affect huge performance gain for the application. Clients can check thousands of records from different tables, maybe from pre-cached records to decide whether the insert operation is valid or not. It will be instant operation which will not affect the whole application performance because only the client's computational power is used. If desired, these validations can also be double checked from the DBMS layer, and they will still provide performance gained by reducing the failures, and roll-back actions on the DBMS layer.

Finally, table, view, and stored-procedure schemes are auto-generated from the extraction feature. For tables and views these are field names, types, and default values. For stored procedures, these are parameter names, types, and default values. Field names and types are necessary during the preparation of the data migration queries. Data types and default values are also used for selecting correct client components and values at the user interface of the application. Downloading the RDBMS scheme to the client side would be dangerous due to some security requirements. For some tables or fields, it may cause security leaks in order to show the real database name of the given fields. Therefore, it is possible to exclude some of the tables or fields from the code extraction feature if they are not necessarily used by the client; or mapping the alias field names with real field names can be implemented at the application layer for hiding the real field names.

Sometimes, clients use some table columns solely for read-only or write-only purposes. In this case, the security module of the framework will check the migration query on the application layer, and decides the grant of permission which is another security topic.

The JBOM file can be generated from scratch, or incrementally according to the last updatetimes whenever the database scheme is changed.

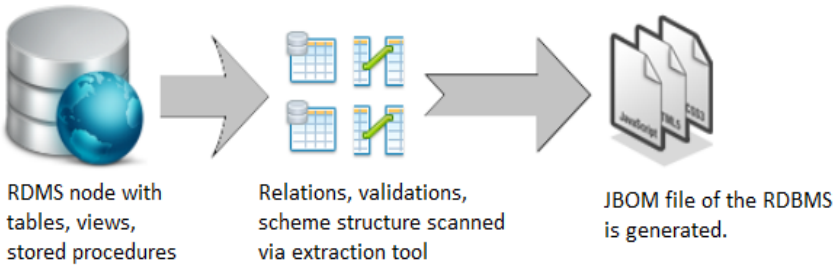

Figure 10Extraction of JBOM files from RDBMS

5.2. IDE for Configuration and Deployment

The Tork Framework has a built-in SaaS IDE tool for customization of the application after initial construction. By using this IDE tool, it is possible to customize the following:

- Customization of the Business Object models (BOM).
- Customization of the list views and detail forms.
- Customization of the application menu and, user group permission grants for the menu.
- Writing codes and binding to an event which is fired by the framework.

Another role of this IDE tool is to make partial deployments from the test environment to the production environment and versioning these changings. For a cloud application, especially for the multi-tenant SaaS applications, partial deployment is very useful for development purposes. Development of a simple functionality just for one tenant should not require a pause ofthe overall application for all tenants. Therefore, partial deployments make it possible to versioning the application without any loss of availability in the production environment.

The IDE tool has a GUI which is necessary to write the XML codes for customizations, and the JavaScript codes for implementing the custom event codes. All these implemented codes are built as JavaScript or XML files. When a developer needs to customize a module in the application, the IDE tool extracts the related partial code of that area, and then visualizes the code in the GUI, see Figure 11. In this state, the developer has the following options for versioning the code.

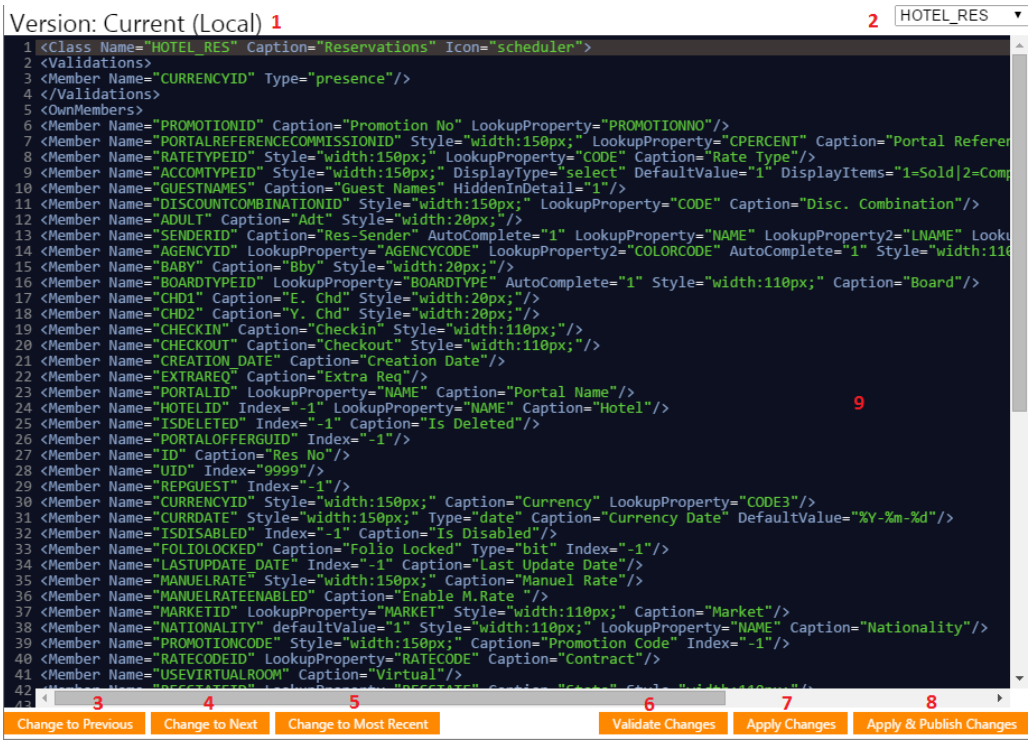

Figure 11 GUI for the IDE tool

- The developer can see the version number of the partial code from the label marked as 1 in Figure 11.
- The developer can change the selected BOM for customization from the select box marked as 2 in Figure 11.
- The developer can request the previous version of the partial code; this will fetch the previously written partial code and visualizes to the GUI. This action is done by the ‘Change to Previous’ button marked as 3 in Figure 11.
- After changing the few previous versions,the developer may want to track the specific partial changes by the time or reach to the current one. These actions are done by the ‘Change to Next’ (marked as 4) and the ‘Change to Most Recent’ (marked as 5) buttons in Figure 11.
- After changing and before saving the code, the developer may want to validate and take information of the errors in the syntax of the code. This action is done by the ‘Validate Changes’ button marked as 6 in Figure 11.
- The developer can change and save the code; this will change the test environment of the new partial so that the custom code will be back merged to its related section of the XML file or JavaScript file. This will also update the new version into the DB of this partial code change. This action is done by the ‘Apply Changes’ button marked as 7 in Figure 11.

- After testing the written code or without testing it, the developer may want to deploy the written code partially. In this case, both files in the test environment and the production environment will be partially changed. The related sections from both environments will be fetched and changed accordingly. This may cause it to run different versions on the test and production environments at the same time. This action is done by the 'Apply & Publish Changes' button marked as 8 in Figure 11.

Figure 12 shows the partial deployment logic; steps during the deployment process are marked with numbers in the figure. Step 1, 2, 3 and 4 are used for deployments only applied on the test environment, which is the same action marked as 7 in figure 11. Step 5is used for deployments for the production environment. In this case, the related section of the production code file is changed according to the developer copy of that file.

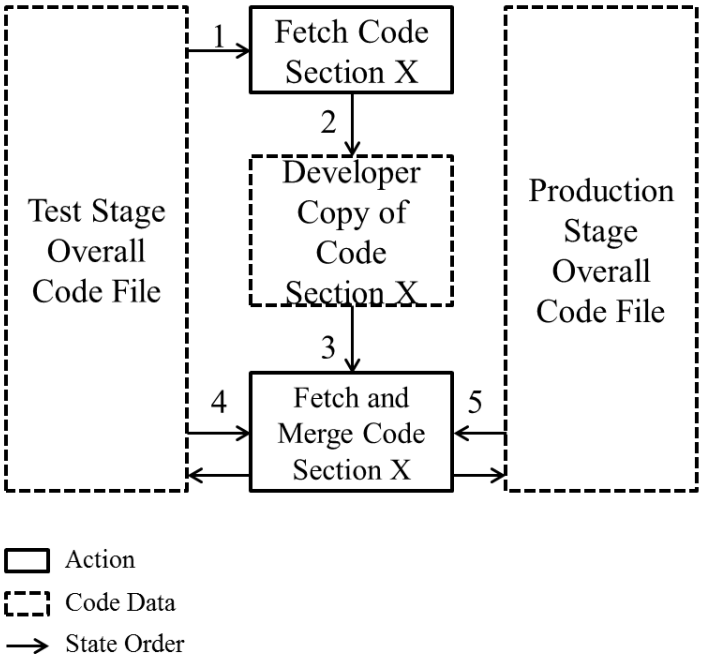

Figure 12 Partial Deployment Logic in Tork Framework

6. Evaluation of the Framework

This chapter describes the case study and behaviors of the client requests in the Tork Framework. In section 6.1, behaviors of the thick and thin clients in the Tork framework are described. To compare the isolation effects, these behaviors are examined, and activity diagrams for the case study are observed.

In section 6.2,we examine a case study related with a travel portal application. This is a cloud application implemented with theTork Application Framework. In this study we describe the example scenario for testing the framework performance.

6.1. Case Study: Travel Portal

In this case study, we focused on a cloud application [51] which provides online travel services such as hotel, flight and tour booking. This application is being used by 79 online travel agencies, and their hundreds of end users are operating from the same database and the same application server for high availability. In this case study, we focused on the functionalities of the end users. Most of the functionalities are related with searching for hotel prices and searching for room prices of the specific hotels. To simulate the real-time traffic and test the framework performance, we replicated the real application which includes the DB and application server. We used m3.medium type of instances for each node; M3 instance types provide a balance of compute, memory, and network resources, and are advised by Amazon Web Services (AWS) [26] for enterprise applications. In our use cases, end users are concurrently sending price requests from different travel portals. Price requests are calculated through a stored procedure based on the DB layer, therefore, only execute operations are used for this purpose; because of high availability requirements, one database server is used, and since the execute operations are done on the database layer, no need for additional application servers is necessary for processing these requests in this use case. Different travel portals in this case mean different tenants of the cloud application; therefore, performance isolation of the separate travel portals is necessary.

In this use case, we evaluated the network traffic with 3 different tenants (travel portals), and 5 concurrent users per tenant (portal end users). We compared the performance isolation metrics with three proposed solutions implemented on the

Tork Framework and one non-isolation case. Our use case has requests for different workloads and their selecting frequencies on the simulation are follows:

- Request for searching hotels and its prices: frequency of 30%.
- Request for detailed prices of the given hotel: frequency of 40%.
- Request for searching package tours: frequency of 10%.
- Request for detailed prices of the given tour: frequency of 20%.

These frequencies are approximate observed frequencies based on the requests coming from the end users of all tenants in the real system application [51].

Table 6. Scenario Parameters to be used in the Evaluation Framework

Scenario Parameter	**Description**
Number of tenants	Number of travel portals deployed to one application server.
Number of disruptive tenants	Popular travel portals that have more number of concurrent end users.
Number of abiding tenants	Travel portals that have normal or low number of concurrent end users.
Concurrent clients per disruptive tenant.	Number of end users takes service from a high loaded travel portal.
Concurrent clients per abiding tenant.	Number of end users takes service from a normal loaded travel portal.
Service Level Agreement for requests.	Promised request count per tenant per second.
Waiting time between client requests.	Calculated automatically from SLA parameter. 0 for disruptive tenants, client size / SLA for normal tenants.
Adopted Performance Isolation Approach	Selected approach for evaluating the results, it can be RoundRobin, Delay, BlackList or Non-Isolated.

This case study is evaluated for thin and thick client scenario environments. In the thin client scenario of this case study, data visualization modules and offer calculation algorithm is running on the cloud server. This overall functionality is requested by the thin client to render pre-calculated response to the screen. For the thick client scenario, the raw data that is necessary to process via offer calculation algorithm is downloaded from the cloud server - if it is not cached inside the client device.Then the thick client calculates the final offer according to this calculation algorithm implemented in the client layer in this case. Finally thick clientmakes a server request to log the calculated offer detailsto the cloud server.

Offer calculation algorithm is used on the raw data, which are the hotel and the tour contract records. They are used to calculate final prices, discounts and

conditions for the each offer. These offers may be depended on various factors on the client request such as checkin/checkout dates, number of adults in the guests, discount conditions in the contracts, ages of the children and combination of these rules that are written in the contract data record. In section 6.1.1, thin client scenario, we call this offer calculation process as F_x, and in section 6.1.2, thick client scenario, we call F_x for the logging action of the offer details from the thick client to cloud server.

6.1.1. Thin client scenario

In this case, we assume that users are constantly requesting the functionality of F_x. F_x functionality is executed on cloud server and then the result of execution is given to the client as response. Let the execution time of the cloud process is called P_x. After the result comes from the server, the user requests the functionality of F_x after waiting sometime, we named ast_x see action diagram in Figure 13 for a more detailed explanation.

Cloud software developers assure that in the SLA, F_x functionality can be serviced a maximum of k times per second.

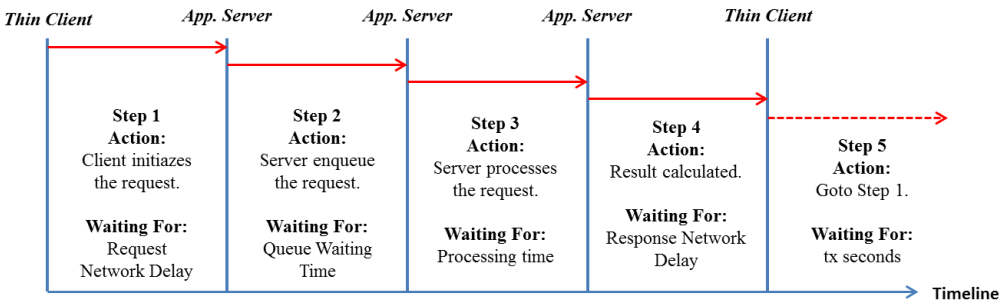

Figure 13 Action diagram for thin client scenario

We know that in step 1 and step 4, waiting times are network delays in Figure 13. Since the network delay depends on the lower layers of SaaS (such as IaaS) and varies from the tenant geographical distance, and is constant for individual tenants, we consider these delays as 0 for the simplicity of the evaluations. Therefore, we do not include these values into our formulations.

In this scenario, the total request processing time is the sum of the queue waiting time and we call it W_t, and the processing time P_x. By combining this formula with the SLA requirements we can say that for abiding tenants, they can send $k / (W_X + P_X)$ request per second. However, in this scenario, tenants will send a request every $W_x + P_x + t_x$ seconds, therefore, the following equation is satisfied for abiding tenants:

$$W_x + P_x + t_x > \frac{k}{(Wx + Px)} \rightarrow (W_x + P_x + t_x)\,(W_x + P_x) / k > 1$$

When this equation is less than one, the tenants should be considered as disruptive tenants. Because in this case, they will send requests more than the limit pre-defined in the SLA which is k requests per second per tenant. Thus, the server may able to prevent their requests due to violation of other tenants' resource rights with respect to performance isolation solutions.

For experimenting purposes, feasible values for$P_x < 1/k$, otherwise, it is guaranteed that every request justifies the SLA in this scenario.

We expect the following behavior when t_x starts to decrease. W_x, the queue waiting time, firstly will not increase due to availability of server resources which does not prevent processing. When t_x starts to decrease for all tenants, queues become full, and processing power of the cloud server will be fully utilized. To balance the overall requests with queues, waiting time in the queue will be increased. This increase may be linear or non-linear depending on the queue waiting algorithms.

Similarly P_x, the processing time firstly will not be affected by t_x due to the available resource pool on the cloud server, which is not yet fully utilized by the tenants. After a point,P_x may start to increase due to utilization on resource pool. Therefore, processing time may increase proportional to incoming request size. However, this increase should not violate the SLA requirements, and the maximum value of P_x should stay constant by balancing with W_x time.

Table 7. Adopted Example Scenarios – Thin Client

Scenario Parameter	**Scenario 1 (Low Workload)**	**Scenario 2 (Heavy Workload)**
Number of tenants	3	3
Number of disruptive tenants	0 ➔ 3	0 ➔ 3
Number of abiding tenants	3 ➔ 0	3 ➔ 0
Concurrent clients per disruptive tenant.	4	4
Concurrent clients per abiding tenant.	4	4
Service Level Agreement for requests.	2 req / sec	4req / sec
Waiting time between client requests.	2 sec	1 sec
Adopted Performance Isolation Approach.	RoundRobin, Delay, BlackList, Non-Isolation	RoundRobin, Delay, BlackList, Non-Isolation

6.1.2. Thick client scenario

In this case, similartothe thin client experiment, users constantly request the functionality of F_x. However, this time, the main computational operations related with $F_x$are executed in the thick client. After that, a server request is executed for logging the result of the functional request.

Similar to thin client experiment let the SLA requirements for F_x functionality be a maximum k times per second. In thick client concept, SLA requirements are also dependent on the client hardware. Therefore, to justify SLA requirements, an appropriate hardware system should be used by clients. This is stated as the minimum hardware requirements for clients on the SLA.

In the action diagram (see Figure 14), the waiting times for this scenario are follows:

- 1-2: Processing time on the client called C_x.
- 2-3: Networking delay for requesting the logging action on cloud server.
- 3-4: W_x the queue waiting time (similar to thin client experiment).
- 4-5: P_x, cloud server processing time for logging the functional result.
- 5-6: Network delay for the response given by the cloud server.
- 6-7: Waiting time for re-requesting the functionality F_x.

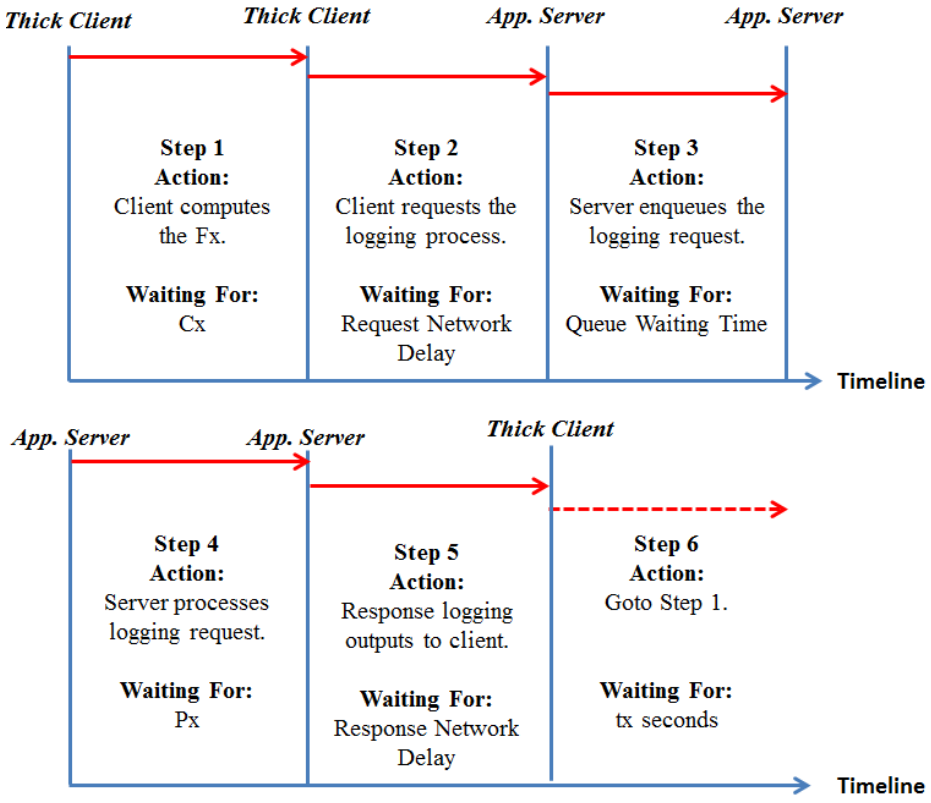

Figure 14 Action diagram for thick client scenario

Items 2-3 and 5-6 are ignored in the following calculations because of the similar reasoning in the thin client experiment; network delays are constant times for each tenant, and therefore do not change the characteristics of the individual tenant requests.

When we sum the waiting times for processing the F_x:

$C_x + W_x + P_x$is the overall time for processing the F_x, according to the SLA requirements, each tenant can send $k / (C_x + W_x + P_x)$ request per second. Since each tenant will send request per $C_x + W_x + P_x + t_x$ second, to justify SLA requirements, following equations must be hold:

$$C_x + W_x + P_x + t_x > k / (C_x + W_x + P_x)$$

$$= (C_x + W_x + P_x + t_x)\,(C_x + W_x + P_x) / k > 1$$

When this value is greater than 1, the tenant is considered as abiding, less than 1 is considered as a disturbing tenant according to the SLA requirements.

Compared to thin client experiment, the thick client experiment has two expected advantages in terms of performance isolation of tenants:

1. C_x is a new variable, the processing time which is a part of P_x in the thin client experiment. It is now fully isolated from the tenants shared resource pool because it is processed on the client hardware, client resource pool only. Therefore, this computation workload does not affect overall system performance.
2. P_x is now decreased due to isolation of C_x. Since in the thin client experiment, P_x can be varied with respect to the workload necessary to the execution of F_x, however, in the thick client experiment it is a constant workload. Because P_x in the thick client is only the processing time for logging the result of the process, and does not relate with the workload of F_x. Thus, tenant requests now have a similar workload to the cloud server which is beneficial implementation for applying the performance isolation techniques on the cloud server.

Table 8. Adopted Example Scenarios – Thick Client

Scenario Parameter	**Scenario 1 (Low Workload)**	**Scenario 2 (Heavy Workload)**
Number of tenants	3	3
Number of disruptive tenants	0 ➔ 3	0 ➔ 3
Number of abiding tenants	3 ➔ 0	3 ➔ 0
Concurrent clients per disruptive tenant.	4	4
Concurrent clients per abiding tenant.	4	4
Service Level Agreement for requests.	2 req / sec	4 req / sec
Waiting time between client requests.	4/2 = 2 sec	4/4 = 1 sec
Adopted Performance Isolation Approach.	RoundRobin, Delay, lackList Non-Isolation	RoundRobin, Delay, BlackList, Non-Isolation

6.2. Evaluation Environment

To evaluate the request counts and response times from the real environment, we have developed simulation software. This simulation software is coded with HTML and JavaScript. It can run on any web browser that is supported byAJAX technology. Requests are sent as restful service in AJAX to the application server of the Tork Framework.

The most important thing to consider while simulating the software is that most of the web browsers use a limit for active TCP socket per host, which generally rangesfrom 2 to 8. In this case, since we simulate different tenants and different users concurrent socket connections are a must. Therefore, we used the Mozilla Firefox web browser to change the default socket size to 10,000 sockets, which is sufficient for this experiment.

To start the simulation process, the developer needs to select the number of tenants, users, workload scenarios, and proposed algorithms issued for the requests. To parameterize these variables,weseparated them as four sections inside the parameters menu. These are:

1. Selecting the workload scenarios
   In this selection, there are two options, thin client or thick client scenario. In the thin client scenario, randomized workloads are heavily dependent on the cloud server. In the thick client scenario, the randomized workloads are heavily dependent on the client; therefore, the overall workload for the cloud server is less in this case.
2. Abiding Tenant Configurations
   In this selection, a number of tenants and users who are in the limits of SLA are defined. According to this selection there will be *tenant x user* amount of abiding clients whose waiting time for the next requests are long enough to satisfy the SLA defined in section 4.
3. Disruptive Tenant Configurations
   Similar to abiding tenant configurations, a number of disruptive tenants and users belonging to these tenants are defined. The difference between disruptive clients and the abiding ones is that disruptive clients do not have any internal waiting time between the requests. Therefore, they may not justify the SLA requirements according to the response time of the server to these clients.
4. Evaluation Criteria Configuration

This section gathers the information required for the evaluation. This information is SLA information, evaluation time period, and evaluated algorithm. The SLA information is provided as the number of requests allowed for one tenant in the units of seconds. The evaluation time period is the period necessary to calculate the request counts and response times; after this amount of seconds end, simulation stops and waits for the responses of all requests. Evaluated algorithm information is the option that is one of the proposed solutions which will be issued for requests when the simulation starts.

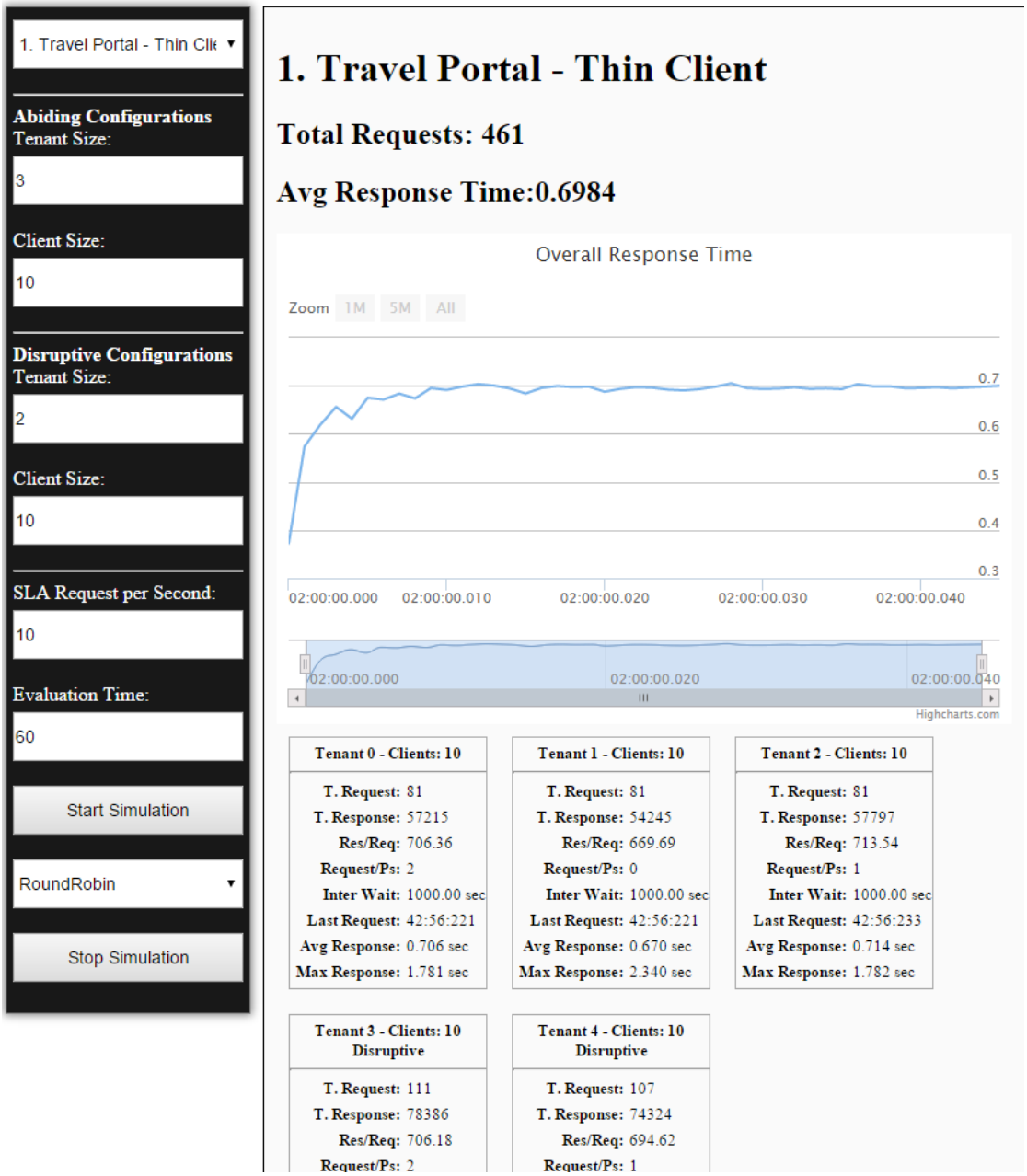

Figure 15Snapshot of the simulation environment

After selecting these parameters, the simulation software triggers the application server and settles the selected algorithm automatically. The SLA configuration is also settled, and it is a necessary parameter for delay related algorithms. Figure 15, shows a snapshot from the simulation environment during the simulation process. In this snapshot it is possible to see real-time events such as abiding and disruptive tenant behaviors, overall system request count, and average response time. These events are giving some clue about application server to understand whether the system is working on the full computational limits or not.

6.3. Evaluation Results

To evaluate the performance isolation we used the quality of service metric and evaluate the response time. This time is examined as the request sent from a client and to the time when the client fetchesback its response. Therefore,function of $z_t$for Wcalculates the average response duration for *t*. Workload amount is taken as the request frequency of the clients and client size which are belongs to each tenant.

We expect the cloud system will run under heavy utilization due to economic reasons. Another reason of heavy utilization is for measuring the performance of the system under full capacity. Therefore,to limit the application server, we design our cloud system to serve for 12 users by limiting the CPU percentage of the application server.

In our simulation, we consideredtwo thin client scenarios and two thick client scenarios. We have used 3 tenants in both scenarios, and 4 users for each tenant. In both scenarios we decreased the number of abiding tenants to 0, as the disruptive tenants were increased to 3. In each step of this increase, we measured the change in the quality of service given to tenants and enumerated them as I_{QoS1}, I_{QoS2} and I_{QoS3}. For the averaged isolation metric I_{avg}, we measured the average of these results.

Table 9. Thin Client Performance Isolation Indices

Approach	Thin Client Scenario 1 (Low Workload)				Thin Client Scenario 2 (Heavy Workload)			
	I_{QoS1}	I_{QoS2}	I_{QoS3}	I_{avg}	I_{QoS1}	I_{QoS2}	I_{QoS3}	I_{avg}
Non-Isolated	4.78	9.44	9.83	8.02	7.49	9.64	9.50	8.88
Round Robin	1.51	3.57	8.14	4.41	2.85	5.56	10.71	6.37
Delay	3.33	5.32	7.26	5.30	3.22	6.33	7.17	5.57
Black List	5.43	6.42	11.22	7.69	-6.84	-4.34	2.63	-2.84

Table 10. Thick Client Performance Isolation Indices

Approach	Thick Client Scenario 1 (Low Workload)				Thick Client Scenario 2 (Heavy Workload)			
	I_{QoS1}	I_{QoS2}	I_{QoS3}	I_{avg}	I_{QoS1}	I_{QoS2}	I_{QoS3}	I_{avg}
Non-Isolated	0.99	2.71	3.21	2.31	3.86	3.12	5.34	4.11
Round Robin	0.18	0.46	0.58	0.41	0.44	1.23	2.57	1.41
Delay	1.32	1.37	1.28	1.32	1.25	1.34	1.24	1.28
Black List	2.26	3.34	-6.29	-0.23	-8.72	-5.44	0.55	-4.54

Tables 8 and 9 show the performance isolation indices with the givenproposed solutions and scenarios. Lower values represent better performance isolated systems. In the *BlackList* approach some indices are negative meaning that performance isolation is against the disruptive clients by prioritizing the process of abiding the client workload.

- *Thin Client, Low Workload*

For thin clients in the low workload environment, non-isolated results show that performance isolation seems to be a problem.In the low workload, abiding tenants do not send too much requests, whereas disruptive ones send non-stop requests. Since it is thin client scenario, workload distribution is relatively wider compared to thick client scenarios. Thesefactors lower the accuracy of the true tenant selection for processing. However, at the overall case,*RoundRobin* providesthe best solution in the average. However, in the most disruptive case,I_{QoS3} delay shows the best result. This is due to workloads of different tenants not being equal,and therefore *RoundRobin* does not provide accurate results.*BlackList* does not provide a good solution in this scenario; this is caused by the lower workload and the clients have a more frequent request demand, which is a blocking factor in the *BlackList* solution.

- *Thin Client, High Workload*

Similar to the thin client with low workload scenario, in this scenario, the difference is the higher request demand coming from the abiding clients. Therefore,we observe more non-isolation in *RoundRobin*. This is because all tenants having a round due to higher frequency of request demand however, their process does not have the same time. This causesmore of an unfair situation compared to a lower demand from abiding tenants. Similarly, in the *BlackList* higher frequency demand in the requests caused opposite degree of isolation.

Since some tenants have a low workload and therefore tend to have more requests, they are blocked against the *BlackList*. Delay also resulted in an increasing degree since the request rate and workload have opposite relation; delay is mostly applied on the lower workload requests. That causes non-isolation and is linearly related with the overall request rate.

- *Thick Client, Low Workload*

For low workload in which the clients can take over many tasks, we can observe a substantial improvement of the performance isolation degree with respect to the case of thin clients, whereby all the tasks are carried out at the server side. Since most of the calculations are done at the client side, the overall workload does not prevent computational power of the cloud server; therefore, all of the requests coming to the server have a similar workload and have a similar response time. This results as the lower performance indices compare to thin client scenario. The reason how *RoundRobin* performs better is due to similar workloads. However, delay based approaches tend to have a negative impact since they consider the request rate as a quota, and they exceed since the response times are relatively short.

- *Thick Client, High Workload*

The thick client with a high workload scenario demands the most requestscomparedtoother scenarios. Since most of the calculations are done at the client side, the individual requests have low workload; therefore response times for individual requests are smaller. Thus, the outcome of this scenario is more isolated than the outcome of thin client scenarios due to the similar workloads of the individual requests. On the other hand, this scenario results worse than the *thick client with low workload scenario* because of the higher utilization of the application server. *Delay* based approaches show a greater impact than the low workload case because of the high request frequency, and the*RoundRobin*approach is the best solution in this scenario due to similar workloads of the requests.

6.4. Discussion

The overall evaluation results show that from thick to thin client and low to heavy utilization, the performance isolation degree increases, which means that the system becomes non-isolated. However, there are some threats to validate these conclusions. These are mostly due to simulation of the real environment. These threats are listed below:

1. One computer is used to simulate all client requests.

   For sending all client requests, the evaluation environment is used. This environment is executed on a single computer. This means that all of the client requests are sent from the one computer. However, in a real life situation, these client requests will come from different computers and may belong to different geographical distances to the application server.

2. Small group of tenants and users are used for the tests.

   Since all requests can only be sent from one computer, there are limiting factors coming from the testing computer. According to these limitations we used three tenants and four clients for each tenant, thus totaling 12 concurrent users. However, in a real life scenario, these numbers may increase, and the behavior of the application server may change due to different utilization criteria.

3. DNS lookup times for client requests are assumed to be zero.

   Since all requests are sent from a single computer, the DNS lookup time is computed once for all requests. This is the first reason the DNS lookup times are not included in the response times. The second reason is we assumed that in real world scenarios for each client, there will be only one DNS lookup compute time therefore; this case will not affect the behavior of the overall cloud system. However, in some real life scenarios, client behaviors may vary and active clients may tend to be unique clients, and may be from different geographical areas. In this situation, since they are connecting to the application server at the first time, the DNS lookup time may also effect to response time which may change the results.

4. Application server is artificially limited for test scenarios.

   Since we have limited computational power to simulate all clients, and since we want to examine the performance isolation solutions under heavy

utilization, we limited the application server. This limitation is based on CPU throating of the process. Therefore, only a limited amount of hardware is used to work for processing the requests. To settle this limitation, first we tried different values of CPU percentages, and selected the best value according to the utilization of the response time. When the response time is increased and stabilized, responding to all the client requests, we understood that limited hardware amount is utilized for the test cases. However, in a real life situation, finding the system limitation given client requests is not a feasible thing to calculate because of the varying workloads and different operating system limitations.

7. Related Work

This section is separated into three parts. The first part is the related frameworks that provide data layer abstraction for software application development. Second is related with the multi-tenancy supporting cloud SaaS architectures, and the third part is related with the performance isolation on cloud SaaS applications.

Data Layer Abstraction Frameworks

Data layer abstraction plays an important part during development of the data centric applications. First, a representation of the data in the applications tends to be independent from the database of the applications. As the application grows the data representations may evolve and underlying database should not be changed to represent different kind of views. Second, different architectural approaches tend to use different data management mechanisms. Separate databases or different nodes can be involved during the data migration processes. Thus to process and represent the data in whole, first it has to be uniformed by the application logic. There are popular alternative solutions to customize data access logic. Hibernate [53] and Oracle TopLink [54] provide middleware mapping technologies. These mapping technologiesprovide a separation between database and the application logic. However different views from same data or providing dynamic calculated services are built-in features of these frameworks.

ADO.NET Entity Framework [52], is a more enhanced framework, in addition to middleware mapping it provides a conceptual data model in the application logic with a manipulation language (Entity SQL) and provides dynamic services moreover the mapped objects.

However none of thesestudies consider the visualization layers but the application layers and the database layers. Thus, for the SaaS based systems, these frameworks do not consider the client tier for manipulating the objects. Therefore built-in features for listing the data via grids, editing the data via forms or representing the customized data via visualization components are not natively related with the data layer. In Tork Framework we provide entity mapping model from database to client. Therefore data manipulation, dynamic data request services, and visualization of the data is working on the client tier instead of the application layer. Thus this work provides an additional layer abstraction (the application layer) for easing the SaaS development and manipulating the data from the user layer. Moreover this abstraction is done

especially considered by the common used approaches for the multi-tenant SaaS applications which are another specific focus of the Tork Framework.

Multi-Tenancy Supporting Cloud SaaS Architectures

Multi-tenant application development is a popular concept and ithasa needfuldemand in SaaS applications. Related works regarding to this concept focuses to specific multi-tenancy problems such as multi-tenant databases [33, 40], security in multi-tenant applications [32, 41]. Other works related with multi-tenancy examines more wide concepts such as their demand and their potential practical capabilities [34, 35, 36].

Regarding to the existing cloud application development frameworks, Intercloud [37], Appscale [38], CloudScale[43] and EasySaaS [42] are recent popular frameworks and publications that are focused on scaling multi-tenant cloud applications. However none of these are focused especially into the performance isolation concept, and they do not focus on thick client for scaling the application performance, instead they provide elasticity solutions for supporting QoS of the cloud application which is another concept and does not ensure the performance isolation of the multi-tenant systems.

Performance Isolation and Efficiency

In literature, performance isolation works generally concerned on resource efficiency. Regarding works can be separated as *performance isolation metrics* and *performance isolation in multi-tenant architecture* subjects. Most related work with regards to performance isolationmetrics on multi-tenant architectures is based on resource sharing by optimal placement of customers given a set of resources, and SLA requirements [28, 29]. Optimal placement solutions enhance the performance isolation in the application; however it cannot completely maintain the performance isolation.

Another work [44] shows a tenant aware model that can be reconfigurable automatically. This leverages the isolation by placing disruptive tenants into one single node or adding a new node to the system. Nevertheless, this work does not focus on performance isolation due to the use of elasticity concept.

Lin et al shows, a better approach to achieve performance isolation on multi-tenant applications [45]. Different quality of service to tenants is provided in this approach and one test case is evaluated as the workload changes. However, this approach mainly built to differentiate QoS, and minimal admission rate settings

for each tenant have to be provided. This approach does not provide any evaluation scenarios as the customerlimits have to work in a utilized system.

Wei et al. [46] proposed a technique for isolation of resources by estimating the demands coming from tenants. It achieves isolation by preventing the disruptive tenants who degrades performance on other tenants. However, this solution requires dynamic adaptions whereas this study uses a static control for the performance isolation. Second, this technique does not consider SLAs, since controlling resources does not guarantee for fulfilling the SLAs as mapping of these are still an open research question [47].

8. Conclusion

In this thesis, we described the concept of software as service architectures and their difference deployment methods. Next, we described thesystem fairness and system isolation concepts. Regarding this knowledge, two key challenges have been identified while building a cloud based SaaS application. The first one is deriving application architectures, and the second one is architectural design considerations to support performance isolation.

To overcome these problems, we described the main motivation and requirements for solving the related issues and then we introduced theTork Framework. We described this framework to show how it enhances building the cloud based SaaS applications. To show that, we described each layer of the framework, and their built-in features that can be initialized in the produced application. We described the existing proposed isolation solutions in the literature and how their application is done at the Tork Framework. At the tool support part, we described the development of a cloud application using framework and online development features of the framework.

Lastly, we applied the previously proposed performance isolation solutions into this framework and showed how they are performedusing a case study.In the case study, we select an existing application, a cloud based travel portal application. We choose different travel portals as tenants and their customers as the clients. Since our proposed framework is motivated from thick client perspective, we choose two extreme client types, thin and thick to evaluate this case study.

Then we performed an evaluation environment to make pre-defined traffic from the clients to the application server. To measure the performance isolation, we used quality of service based performance isolation metrics that are proposed in the literature. We have measured 16 different combinations of isolation indices by selecting different client types, different solutions, and different workloads.

Finally, we showed that thick client scenarios perform better than the thin client scenarios which make the motivation of*Tork Framework*usable for supporting performance isolation on the cloud based SaaS applications. From a development perspective, those applications can be easily deployed via existing components and be supported by an integrated development environment. From the performance isolation perspective, theycan support performance isolation

solutions, and thus assure a more fair system by justifying the service level agreements.

References

[1] Frey, Sören, and Wilhelm Hasselbring. "Model-based migration of legacy software systems to scalable and resource-efficient cloud-based applications: The cloudmig approach." CLOUD COMPUTING 2010, The First International Conference on Cloud Computing, GRIDs, and Virtualization. 2010.

[2] Fox, Armando, et al. "Above the clouds: A Berkeley view of cloud computing."Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS 28 (2009): 13.

[3] Weiss, Aaron. "Computing in the clouds." networker 11.4 (2007).

[4] Official website of DropBox, https://www.dropbox.com

[5] Official website of iCloud, http://www.icloud.com

[6] Soni, Mitesh. "Cloud computing basics—platform as a service (PaaS)." Linux Journal 2014.238 (2014): 4.

[7] Hung, Pham Phuoc, et al. "Optimal collaboration of thin–thick clients and resource allocation in cloud computing." Personal and ubiquitous computing 18.3 (2014): 563-572.

[8] Dinh, Hoang T., et al. "A survey of mobile cloud computing: architecture, applications, and approaches." Wireless communications and mobile computing 13.18 (2013): 1587-1611.

[9] Official website of Facebook, https://www.facebook.com

[10] Silva, Esdras Caleb Oliveira, Joel AF dos Santos, and Débora C. Muchaluat-Saade. "NCL4WEB: translating NCL applications to HTML5 web pages." Proceedings of the 2013 ACM symposium on Document engineering.ACM, 2013.

[11] Serrano, Nicolas, JosuneHernantes, and Gorka Gallardo. "Mobile web apps." Software, IEEE 30.5 (2013): 22-27.

[12] Zhang, Liang-Jie, and Qun Zhou. "CCOA: Cloud computing open architecture." Web Services, 2009.ICWS 2009.IEEE International Conference on.Ieee, 2009.

[13] Yoshikawa, Chad, et al. "Using smart clients to build scalable services." Proceedings of the 1997 USENIX Technical Conference. 1997.

[14] Luo, Jun-Zhou, et al. "Cloud computing: architecture and key technologies." Journal of China Institute of Communications 32.7 (2011): 3-21.

[15] Mell, Peter, and Tim Grance. "The NIST definition of cloud computing." (2011).

[16] Lin, Jimmy, D. Fu, and J. Zhu. "What is Cloud Computing?." IT as a Service 11.2 (2009): 10.

[17] Gold, Nicolas, et al. "Understanding service-oriented software." Software, IEEE 21.2 (2004): 71-77.

[18] Dubey, Abhijit, and DilipWagle. "Delivering software as a service." The McKinsey Quarterly 6.2007 (2007): 2007.

[19] Papazoglou, Mike P. "Service-oriented computing: Concepts, characteristics and directions." Web Information Systems Engineering, 2003.WISE 2003.Proceedings of the Fourth International Conference on.IEEE, 2003.

[20] Koziolek, Heiko. "The sposad architectural style for multi-tenant software applications."Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on.IEEE, 2011.

[21] Krebs, Rouven, ChristofMomm, and Samuel Kounev. "Metrics and techniques for quantifying performance isolation in cloud environments." Science of Computer Programming 90 (2014): 116-134.

[22] Cooper, Brian F., et al. "Benchmarking cloud serving systems with YCSB." Proceedings of the 1st ACM symposium on Cloud computing. ACM, 2010.

[23] Kuperberg, Michael, et al. Defining and quantifying elasticity of resources in cloud computing and scalable platforms. KIT, FakultätfürInformatik, 2011.

[24] Makhija, Vikram, et al. "VMmark: A scalable benchmark for virtualized systems." VMware Inc, CA, Tech. Rep. VMware-TR-2006-002, September (2006).

[25] Brewer, Eric. "Pushing the CAP: Strategies for consistency and availability." Computer 45.2 (2012): 23-29.

[26] Official website of Amazon Web Services, http://aws.amazon.com

[27] Heninger, Kathryn L. "Specifying software requirements for complex systems: New techniques and their application." Software Engineering, IEEE Transactions on 1 (1980): 2-13.

[28] Fehling, Christoph, Frank Leymann, and Ralph Mietzner. "A framework for optimized distribution of tenants in cloud applications."Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on.IEEE, 2010.

[29] Zhang, Yi, et al. "An effective heuristic for on-line tenant placement problem in SaaS." Web services (icws), 2010 ieee international conference on. IEEE, 2010.

[30] Koziolek, Heiko. "The sposad architectural style for multi-tenant software applications."Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on.IEEE, 2011.

[31] Tekinerdogan, Bedir, K. Özturk, and Ali Dogru. "Modeling and Reasoning about Design Alternatives of Software as a Service Architectures." Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on.IEEE, 2011.

[32] Jasti, Amarnath, et al. "Security in multi-tenancy cloud." Security Technology (ICCST), 2010 IEEE International Carnahan Conference on.IEEE, 2010.

[33] Jacobs, Dean, and Stefan Aulbach. "Ruminations on Multi-Tenant Databases."BTW.Vol. 103. 2007.

[34] Bezemer, Cor-Paul, and Andy Zaidman. "Multi-tenant SaaS applications: maintenance dream or nightmare?." Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE).ACM, 2010.

[35] Guo, Chang Jie, et al. "A framework for native multi-tenancy application development and management." E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007.CEC/EEE 2007.The 9th IEEE International Conference on.IEEE, 2007.

[36] Mietzner, Ralph, et al. "Combining different multi-tenancy patterns in service-oriented applications." Enterprise Distributed Object Computing Conference, 2009. EDOC'09.IEEE International.IEEE, 2009.

[37] Buyya, Rajkumar, Rajiv Ranjan, and Rodrigo N. Calheiros. "Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services." Algorithms and architectures for parallel processing.Springer Berlin Heidelberg, 2010.13-31.

[38] Chohan, Navraj, et al. "Appscale: Scalable and open appengine application development and deployment." Cloud Computing.Springer Berlin Heidelberg, 2010.57-70.

[39] Calheiros, Rodrigo N., et al. "Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services." arXiv preprint arXiv:0903.2525 (2009).

[40] Aulbach, Stefan, et al. "Multi-tenant databases for software as a service: schema-mapping techniques." Proceedings of the 2008 ACM SIGMOD international conference on Management of data.ACM, 2008.

[41] Wilson, James, and Mark Richey. "Methods and apparatus for securely collecting customer service agent data in a multi-tenant environment." U.S. Patent No. 6,823,384. 23 Nov. 2004.

[42] Tsai, Wei-Tek, Yu Huang, and Qihong Shao. "EasySaaS: A SaaS development framework." Service-oriented computing and applications (soca), 2011 ieee international conference on. IEEE, 2011.

[43] Shen, Zhiming, et al. "Cloudscale: elastic resource scaling for multi-tenant cloud systems." Proceedings of the 2nd ACM Symposium on Cloud Computing. ACM, 2011.

[44]Schroeter, Julia, et al. "Towards modeling a variable architecture for multi-tenant SaaS-applications." Proceedings of the sixth international workshop on variability modeling of software-intensive systems.ACM, 2012.

[45] Lin, Hailue, et al. "Feedback-control-based performance regulation for multi-tenant applications." Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on.IEEE, 2009.

[46] Wang, Wei, et al. "Application-level cpu consumption estimation: Towards performance isolation of multi-tenancy web applications." Cloud computing (cloud), 2012 ieee 5th international conference on. IEEE, 2012.

[47] Emeakaroha, Vincent C., et al. "Low level metrics to high level SLAs-LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments." High Performance Computing and Simulation (HPCS), 2010 International Conference on.IEEE, 2010.

[48] Tekinerdogan, Bedir, and KarahanÖztürk. "Feature-Driven Design of SaaS Architectures."Software Engineering Frameworks for the Cloud Computing Paradigm.Springer London, 2013.189-212.

[49] La, Hyun Jung, and Soo Dong Kim. "A systematic process for developing high quality saas cloud services."Cloud Computing.Springer Berlin Heidelberg, 2009.278-289.

[50] Li, Qian, Shijun Liu, and Ying Pan. "A cooperative construction approach for SaaS applications." Computer Supported Cooperative Work in Design (CSCWD), 2012 IEEE 16th International Conference on. IEEE, 2012.

[51] Official website of HotelAdvisor, https://www.hoteladvisor.net/

[52] Adya, Atul, et al. "Anatomy of the ado. net entity framework." Proceedings of the 2007 ACM SIGMOD international conference on Management of data.ACM, 2007.

[53] King, Gavin, and Christian Bauer. "Java Persistence with Hibernate." Revista ed. Greenwich, CT: Manning Publications (2006).

[54] Oracle TopLink. www.oracle.com/technology/products/ias/toplink/

Appendix A

```
<%@ Application Language="C#" Debug="true"%>
<%@ Import Namespace="System" %>
<%@ Import Namespace="System.Threading" %>
<%@ Import Namespace="System.Diagnostics" %>
<%@ Import Namespace="System.Configuration" %>
<script runat="server">
publicstaticstringselectedAlgorithm;

publicstatic Queue<int>standardQueue;
publicstatic Queue<int>blackListQueue;
publicstaticintprocessedRequestCounter;
publicstaticHashtable tenants;
publicstaticintrequestCounter;
publicstaticinttenantIndex;
publicintrequestId;

publicintnextRequestProviderRR(){
      List<string> keys = tenants.Keys.Cast<string>().ToList();
while(true){
tenantIndex = tenantIndex % keys.Count;
if (((Queue<int>)tenants[keys[tenantIndex]]).Count > 0)
return ((Queue<int>)tenants[keys[tenantIndex]]).Peek();
else
tenantIndex++;
      }
   }

publicvoidwaitAsRoundRobin(String tenant){
tenant = tenant.ToString();
requestCounter++;
//Assign new requestid.
requestId = requestCounter;
//Initialize new tenant queue if not exists
if (!tenants.ContainsKey(tenant)){
         Queue<int>requestQueue = new Queue<int>();
tenants.Add(tenant, requestQueue);
      }
//Add request to the tenants queue
      ((Queue<int>)tenants[tenant]).Enqueue(requestId);

//Wait until round robin algorithm gives back the requestId
while(true){
intcurrentRequest = nextRequestProviderRR();
//If round robin gives the requestIddequeue and return
if (currentRequest == requestId){
            ((Queue<int>)tenants[tenant]).Dequeue();
return;
         }
//Sleep while waiting the next request for not utilizing CPU
Thread.Sleep(50);
      }
   }

publiclonginsertRequestStatistic(String tenant){
tenant = tenant.ToString();
requestCounter++;
//Assign new requestid.
requestId = requestCounter;
//Initialize new tenant timestampt if not exists
if (!tenants.ContainsKey(tenant)){
         List<long>timeStampts = new List<long>();
tenants.Add(tenant, timeStampts);
      }
//Save request timeStamp
```

```
long time = Convert.ToInt64(DateTime.Now.ToString("yyyymmddHHmmss"));
      ((List<long>)tenants[tenant]).Add(time);
return time;
   }

publiclongcheckLimit(String tenant, intrequestCount){
if (((List<long>)tenants[tenant]).Count >requestCount){
         ((List<long>)tenants[tenant]).RemoveAt(0);
//Calculate the time difference between requests
longtimeDifference = 0;
for(inti = ((List<long>)tenants[tenant]).Count-1;i>0;i--){
timeDifference += ((List<long>)tenants[tenant])[i] -
((List<long>)tenants[tenant])[i-1];
         }
returntimeDifference;
      }
return Int64.MaxValue;
   }

publicvoidwaitAsDelay(String tenant, longminimumTimeFrequency,
intrequestCount, int delay){
//Save request timeStamp
long time = insertRequestStatistic(tenant);
//Only consider the requestCount amount of timeStampts
longtimeDifference = checkLimit(tenant,requestCount);
if (timeDifference<minimumTimeFrequency)
Thread.Sleep(delay);
   }

publicvoidnextRequestProviderBL(){
while(true){
intcurrentRequest = -1;
//Dequeue from blacklist
if ( ( processedRequestCounter % 30 == 0 ||  standardQueue.Count == 0)
&&blackListQueue.Count>0)
         {
currentRequest = blackListQueue.Peek();
if (currentRequest == requestId)
            {
blackListQueue.Dequeue();
processedRequestCounter++;
return;
            }
         }
elseif (standardQueue.Count> 0){
currentRequest = standardQueue.Peek();
if (currentRequest == requestId)
            {
standardQueue.Dequeue();
processedRequestCounter++;
return;
            }
         }
Thread.Sleep(50);
      }
   }

publicvoidwaitAsBlackList(String tenant, longminimumTimeFrequency,
intrequestCount){
//Save request timeStamp
long time = insertRequestStatistic(tenant);

//Only consider the requestCount amount of timeStampts
longtimeDifference = checkLimit(tenant,requestCount);

//Insert to standard queue if frequency higher than minimum
if (timeDifference<minimumTimeFrequency)
blackListQueue.Enqueue(requestId);
else
```

```
standardQueue.Enqueue(requestId);
//Wait until blacklist algorithm gives back the requestId
nextRequestProviderBL();
    }


publicvoidinitBlackList(){
//Stores the requests for abiding customers
standardQueue = new Queue<int>();
//Stores the requests for disruptive customers
blackListQueue = new Queue<int>();
//Enum the incoming requests
requestCounter = 0;
processedRequestCounter = 0;
//Stores the different tenants statistics
tenants = newHashtable();
    }

publicvoidinitDelay(){
//Enum the incoming requests
requestCounter = 0;
//Stores different tenants statistics
tenants = newHashtable();
    }

publicvoidinitRoundRobin(){
//Enum the incoming requests
requestCounter = 0;
//Cursor for the current tenant
tenantIndex = 0;
//Stores different tenants
tenants = newHashtable();
    }

publicvoidinitAlgorithm(){
if (selectedAlgorithm.Equals("RoundRobin"))
initRoundRobin();
elseif (selectedAlgorithm.Equals("Delay"))
initDelay();
elseif (selectedAlgorithm.Equals("BlackList"))
initBlackList();
    }

publicvoidApplication_Start(object sender, EventArgs e){
selectedAlgorithm =
System.Configuration.ConfigurationSettings.AppSettings["selectedAlgorith
m"];
initAlgorithm();
    }

publicvoidApplication_BeginRequest(object sender, EventArgs e){
long start =
Convert.ToInt64(DateTime.Now.ToString("yyyymmddHHmmssffff"));
       String changeAlgorithm = Request.QueryString["changeAlgorithm"];
if (changeAlgorithm != null){
if (changeAlgorithm == "RoundRobin" || changeAlgorithm == "Delay" ||
changeAlgorithm == "BlackList"){
selectedAlgorithm = changeAlgorithm;
initAlgorithm();
          }
       }
       String tenant = Request.QueryString["tenant"];
if (tenant != null){
if (selectedAlgorithm.Equals("RoundRobin")){
waitAsRoundRobin(tenant);
          }
elseif (selectedAlgorithm.Equals("Delay")){
//Wait 5seconds if request more than 10 requests in 5 seconds..
waitAsDelay(tenant,5L,10,5000);
```

```
        }
elseif (selectedAlgorithm.Equals("BlackList")){
//Insert to blacklist if request more than 10 requests in 5 seconds..
waitAsBlackList(tenant,5L,10);
        }
      }
long end = Convert.ToInt64(DateTime.Now.ToString("yyyymmddHHmmssfff"));
Response.Write("{\"algorithm\":\""+selectedAlgorithm+"\",\"duration\":"+
(end-start)+",\"requestId\":"+requestId+"}");
    }
</script>
```

Publications Produced in This Thesis

1. O. A. Oral & B. Tekinerdogan. “TORK APPLICATION FRAMEWORK FOR SUPPORTING PERFORMANCE ISOLATION IN SOFTWARE AS A SERVICE SYSTEMS” submitted toThe 8th IEEE International Conference on Cloud Computing, June 2015.
2. O. A. Oral & B. Tekinerdogan. “TORK FRAMEWORK – SCALABLE CLOUD SaaS BUILDER for ENTERPRISE APPLICATIONS” submitted to The 8th IEEE International Conference on Cloud Computing, June 2015.