Multi-tenancy Design Patterns in SaaS Applications: A Performance Evaluation Case Study

Adeniyi O. Abdul[1], Julian Bass[1], Hossein Ghavimi[2], Natalie MacRae[2] and Peter Adam[2]
[1]*School of Computing, Science and Engineering, University of Salford*
[2]*Add Energy Ltd.*

Abstract

Utility-like computing has emerged as the future of computing for many organizations seeking to remain competitive in today's business environment. Promising features such as rapid elasticity, low cost provisioning, pay-as-use model, layered security, measured service, resource pooling, are the reasons companies are opting for this technology. Cloud technologies are provided as services ranging from Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS) and Data as a Service (DaaS). SaaS has become one of de facto approach for deploying cloud base services or applications for many businesses. At the core of SaaS is Multi-tenancy; multi-tenancy gives customers (i.e. tenants) and providers vast opportunities to leverage the power of cloud infrastructure by consolidating operational entities. The drive toward multi-tenancy in SaaS application is a result of the economic benefit derived by shared development and maintenance cost. This paper presents different multi-tenancy models at the data layer as dedicated, isolated and shared. The paper further empirically evaluates the performance of these models in a containerized environment. Our results show that under a containerized environment dedicated and isolated schema performed reasonably well in terms of latency when compared to shared model. Although the shared model proved to more resource efficient, it performance is greatly affected by finite resources shared by many concurrent tenants.

1. Introduction

Many businesses today are looking for economic viable approach to scale their information management system and manage the current explosion of business data. Building a system of data analytical application with a scalable and elastic environment such as cloud will bring maximum return on investment for enterprise. Cloud Computing offers business on-demand, utility-like computing provisioning with minimal start-up cost and operating cost. One of the bedrocks of cloud environment is resource sharing. Resource sharing, at different software levels, helps businesses to take advantage of the benefits of cloud with minimal cost burden. Multi-tenancy has emerge as a technology that can help businesses effectively share cloud resources while maximizing their sales profit and reducing the cost of application hosting, software development and maintenance. On this premise, multi-tenancy in a SaaS application is a architectural pattern that entails several tenants (customers) to share resources at different tiers of an application. This might range from tenants sharing the same user interface components, business logic right down to data layer, however each tenant only access functions or components and data belonging to their business entity. In contrast to multi-user model (traditional), where an application is designed and developed with the same functionalities but with limited configuration and hosted in a single instance. The advantage of multi-tenant application is its ability to support multi-faceted functionalities which maybe similar or varying and has the capability to scale horizontally.

A similar architecture to multi-tenant is Multi-instance, this leverage the power of virtualization technology to host the same application code on many different instances to accommodate spike in demand for resources. These instances are pooled such that it acts as a single resource server to handle increasing users work loads. [8] Argued that multi-instance architecture can be used to deploy multi-tenant application where the number of tenants is remain relatively low.

According to Antonio [14], a tenant in a multi-tenant environment subscribes or pays to use the SaaS application however a tenant comprises many end-users. In other words, a tenant represents a group of users of an organization that employs a set of SaaS customized functionalities in multi-tenant environment to achieve organizational goals. The business requirements, configurability, scalability, security requirements and the costing model, among others, will determine which multi-tenancy model will be employed when implementing SaaS application. SaaS applications architect must be able to determine which multi-tenancy model will best serve different business requirements without compromising on performance and security of each tenant. [18] Stated that multi-tenancy can be realized at the application level, the business logic

(middleware), virtualization layer and data layer. The next section presents multi-tenancy at the data layer.

2. Multi-Tenancy Model

2.1. Dedicated Model

In a dedicated model, the tenant applications and database are isolated at the instance level or database level. Each instance or database instance entirely host different tenant application and nothing is shared among tenants. This scenario can be employed when each client is perceived as entity without any related business logic or where the data privacy and regulations are of most concerns. Dedicated model gives the highest level of isolation compared to others, however hosting large number of databases can be costly and practically impossible for many SaaS providers because of the number of customers they aim to serve.

In dedicated mode, extending or restoring individual tenant database instance can be easily achieved without any disruption to other tenants. However extra costs such as software licensing, hardware and development cost will be per tenant and considerably high because of strong data isolation and extra security and customization put in place. Please see Figure 1.

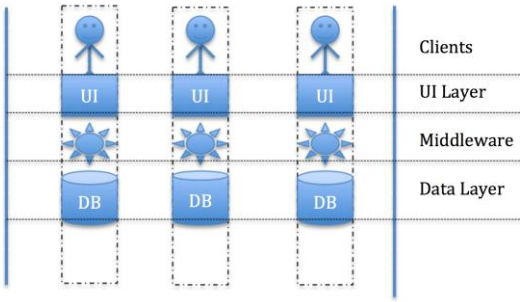

Figure 1. Dedicated database

2.2. Isolated Schema Model

In an Isolated Schema, the tenant tables or database components are group under a logical schema or name-space and separated from other tenant schemas, however the schema are hosted in the same database instance. Though there is some level of logical isolation however it's not instance level isolation, this might cause some down time for tenants when migrating or during maintenance for other tenant schema. This is very suitable where the schema of each tenant is different and changes often. It mitigates the cost of dedicated model and the security limitation of shared model.

A significant drawback of this model is that extending or restoring a database instance might disrupt some tenants' with data co-hosted on the same database and the limit of number tables permitted by the database. However having a scalable database with robust redundancy can help mitigated and reduce the down time. The model is appropriate for tenants with few numbers of tables because of the limitation on the number of tables that a database instance can accommodate. Please see Figure 2.

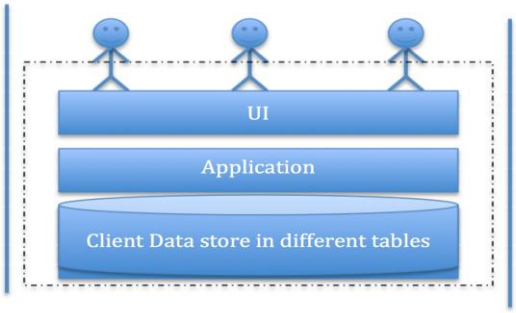

Figure 2. Tenant Isolated

2.3. Shared Schema Model

In a shared schema, the tenant shared common components of the application especially at the data layer with the degree of isolation provided at row level. Sets of tables are used to store all the data belonging to all the tenants of the application. Developers must architect their database access security such that tenant cannot access other tenant data. Same tables are used to store tenant related data with tenant identifier to differential tenant data. This model is commonly used in applications where the client data are closely related or using meta-data architecture for data remodeling. Data extensibility is a great challenge with Shared model compared to isolated schema.

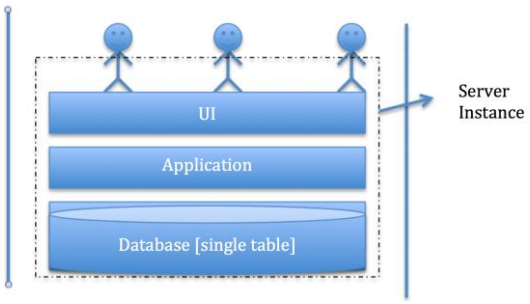

Figure 3. Shared Model

Shared schema model [14] provides data extensibility using multi-tenant metadata and multitenant indexes. This type of design requires complex schema and creation of pre-allocated fields (dummy columns) violating relational database normalization rules [1]. However, it offers the lowest cost in terms of hardware, software licensing, development and backup cost. This model is appropriate when the application is design to accommodate large number of tenants willing to

forfeit strong level of isolation for lower cost of usage. Please see Figure 3 above.

2.4. Partially Isolated Component / Hybrid Model

This is a bridge between shared model and tenant isolated schema model. In this model, components that have common functionalities are shared among tenants while components with unique or unrelated functions are isolated. At the data layer, common data such as data that identify tenants are grouped or kept in single table while tenant specific data are isolated at table or instance layer.

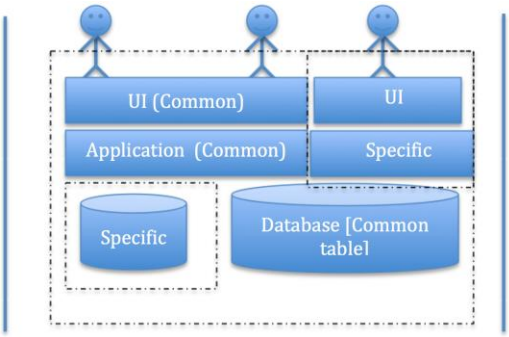

Figure 4. Partial Isolated

The contribution of this paper is to empirically detailed the performance of each multi-tenancy model using a real life example and bridge the knowledge gaps relating to architecting a multi-tenancy application and show how he choice of multi- tenancy model can affect the performance of SaaS application. The case study chosen for this evaluation is a multi-tenancy solution for asset and integrity management application designed to optimized maintenance strategies for various clients in oil and energy sector. To properly carry out this evaluation, an in-house application has been developed to address each multi- tenancy model and deployed using a containerized platform for experimental repeatability.

This paper aims to address this research question - ”How does multi-tenancy models affect the performance of cloud-hosted asset integrity application”. Providing empirical answers to this question will elucidate how each multi-tenancy model can affect the design of this application in terms of latency and throughput.

The rest of the paper is organized as follow - section 2 presents related works in the field of cloud application, SaaS application, and multi-tenancy and software containerization. Section 3 introduce the research methodology and expatiate on the case study employed for the empirical evaluation. Section 4 presents experimental and results of our study. Section 5 presents the conclusion of our work.

3. Related Work

In this section, we present previous research, models and deployment patterns related to cloud computing, SaaS application and multi-tenancy. Also we reviewed deployment pattern using containerization approach to further understand how this approach can be deployed in repeatable multi-tenant environment. This will give us insight into how researchers and software engineers have tried to proffer solutions to some of the challenges and difficulties of resource sharing using multi-tenancy and containerization.

In today’s business climate, cloud offering promises cost-effective realization for many businesses striving to remain competitive in their industries. Promising features, such as interoperability, scalability, on-demand resource provisioning, pay-as-use model, central resource management, resource sharing etc., have changed how information technology (IT) solutions are consumed by many business [16]. Prior to cloud computing, businesses have to provision software applications that meet their needs and also provision in house hardware to support these legacy applications. This mode of IT provisioning has its own pros and cons, but in this ever-changing environment couple with data explosion and severe business competition, businesses are turning to cloud computing for IT solutions.

Different models of cloud computing have emerged as services to cater for different business needs. The emerging models are Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), Data as a Service (DaaS) [2]. Software as a Service has gained wider attentions because this directly relates to end-users. [18] Stated many limitations of legacy software application can be addressed by distributed software services with real time configuration and provisioning. SaaS becomes economic viable for cloud providers and affordable for SaaS consumer through resource sharing. Resource sharing is achieved in SaaS via multi-tenancy. The idea of multi-tenancy started as a result of the success achieved through the use of virtualization technologies in (aaS architecture. The advent of cloud computing and desire to maximize server utilization and minimize provisioning cost, led to multi-instances which gave birth to multi-tenancy. SaaS providers want solutions that can handle many tenants with similar or disparate business requirements with minimum code maintenance and high degree of configurability.

[4] And [3] highlighted some of the challenges and concerns of multi-tenancy are conceptual framework, configuration and customization, data separation, security, performance, deployment, zero-downtime, maintenance and metering. Due to

resource sharing, multi-tenant application must tackle interference and ensure security isolation among tenants. [19] argued that multi-tenant isolation, configuration and customization increase the engineering complexity of SaaS application. Multitenant configuration support pre-defined parameters that users can set on the data model, interface and the business level of the application while customization according to [19] is implementing variations of software that meet specific tenant requirements and deployed at the run-time. The authors proposed customization of multi-tenant application using dependency injection for injecting different software variations at the middle layer. [19] Discussed how ontology can be used to create multi-layered customization for tenant across layers.

[12] Evaluates the degree of isolation between tenants and proposed component-based approach to multi-tenancy isolation through request re-routing. This research highlights an interesting result in response time between shared component and the two other models (tenant isolated and dedicated component). Principle of improving isolation of security, performance at the application-level was presented by [9].

[10] States that one of the major obstacles to cloud adoption is lack of performance benchmarking especially in multi-tenant applications. The proposed a benchmark to evaluate the maximum throughput and the amount of tenants a multi-tenant can accommodate. This work motivate our research to evaluate the performance in terms of latency for a containerized multi-tenant application using different multi-tenancy models

4. Methods

Our research employs the use of case studies approach to methodically evaluate the performance of multi-tenancy model. Case study has shown it best suitable for conducting empirical investigation into contemporary phenomenon within its real-life context [20]. A business application (AimHi) developed to be used by multiple tenants in cloud-based environment was selected as our case study. The application is engineered to optimize maintenance management and improve asset integrity in the field of oil and energy. This will enable tenants and application users to improve maintenance planning and reduce cost of maintenance. This application is chosen because of it requirements as a multi-tenant application and the data size that needs to be processed to perform its functionalities. Tenants of our application share requirements of high similarity (common functions) and requirements that are distinctive to each tenant. In this case, a tenant is a group of users representing an organization that utilize the application to actualize the business goals of the organization. AimHi is developed to interface with CMMS system such as SAP used across oil and gas field. After many years of consultations and development, AimHi is a fully functional application deployed on Amazon web service.

The case study application is a cloud-based application called AimHi that gather and transform maintenance data into comprehensive rich visuals to better provide insight into maintenance strategy employed by our clients. These needs arise mainly for managements and stakeholders to be informed on what maintenance is been carried out, what its benefit for the business in terms of cost and resources saving and providing insight into how operational performance can be improved. Currently the application is been used by an energy company (named withheld for confidentiality) to monitor and improve their power generation plants. This application is selected as a case study because of its multi-tenant capabilities to serve organizations with multi-faceted requirements in the field of gas and energy.

The first version of AimHi was released in July 2015 and deployed on Amazon elastic beanstalk with load balancer and Amazon RDS (MySQL) as the back end. Stress and Integration test were carried out to ascertain its behaviour under different load behaviour. JMeter and Amazon cloud watch were used to monitor the latency, throughput and utilization of the application. This preliminary test gave insight into the performance of the chosen multi-tenancy architecture, which led to reimplementation of some of the data component of the application. This research work will expose how the application will perform under different architectures and properly lead to a robust implementation of the application.

Case studies are suitable for exploratory research in which a hypothesis describing some phenomena is developed [28]. The longitudinal embedded case study approach was employed in order to provide a holistic, in-depth, analysis of one setting and are characterized by production of rich and detailed descriptions [29].

4.1. Experimental Setup

As discussed in the case study section, our application (AimHi) is a SaaS web-based application for asset and integrity management deployed on Amazon cloud. In order to ensure experimental repeatability, the application is developed, containerized and deployed using docker. The objective of our performance test is to evaluate the performance of three variants of multi-tenancy model using Aimhi as the case study. These are the scopes of testing:

1) The testing is carried to evaluate the latency when processing asset integrity key

performance indicators (KPI) such as preventive, corrective maintenance.

2) Performance test does not include the front or GUI part of the application; instead performance latency will be evaluated by conducting restful calls to the endpoints provided by the application. This will help evaluate multi-tenant model and eliminate the processing time used to render the application in the browser.

3) This test evaluates the multi-tenancy at the data tier level therefore the same application calculations or algorithm were used with only changes carried out at the data access level (Hibernate) and database.

4) The test carried out on incremental concurrent users while the data size remains constant. This helps to evaluate the performance behaviour as the concurrent user base increases.

5) The test carried also evaluate the incremental data size while the number of concurrent users is set constant. This helps to evaluate the performance behaviour as the data size increases. This is important because the application was designed to be used by tenants with varying data size.

Three variants of the same application were designed and implemented to address each multi-tenancy model. Each variant is hosted in docker container running on a machine with 16GB memory, 3.1 GHz Intel core i7 and mac0S Sierra. Inside each is an Apache web server connecting to java servlets container running the AimHi application and connected to MySQL relational database system as the database system. The application is developed using a restful technology exposing endpoints returning JSON data, which are consumed by JavaScript front-end code (angularjs). Please see the Figure 4 below of architecture of the system:

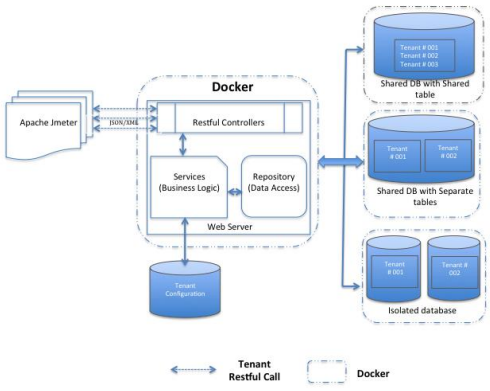

Figure 4. Aimhi Test Architectural Diagram

The implementation of multi-tenancy in each of the variant has been done at the data tier layer. Each data layer variant implementation corresponds to each multi-tenancy model.

Each data layer variant implementation corresponds to each multi-tenancy model. In the separated database model, each tenant set of tables is created in a separate database hosted in their own docker container instance.

In the tenant isolated, each tenant set of tables is created separately (group in name-space) but resides in the same database hosted in a single container instance. While in shared a set of tables are created that stores all the data related to all the tenants, for example a table storing maintenance details contains all the maintenance data for all the tenants of the application. Lastly, in the partially isolated model, data that highly related such as authentication and authorization for each users are stored in a single table while data that highly disparate in representation, meaning and requirements are store in tables associate to that tenant only. The last model might helps cater for tenants that require data isolation as a result of data protection law for example EU data protection law. The containerisation of the data instance for this model is highly dependent on the requirement and size of the tenants and data. See section I-A for each diagram of multi-tenancy model.

Our goal in this research paper is to test the performance of each of the variant of multi-tenancy model in a containerised environment. In order to do this, Apache JMeter application is used to conduct load testing and measure performance in terms of response time. Apache JMeter is open source software written in Java for conducting load test to evaluate functional behaviour and measure performance. Our choice of load testing software is influenced because of its multi-threading capabilities, high extensibility and ability to test many different application and protocol. The Apache JMeter tool was used to simulate heavy concurrent workload usage against Tomcat environment housing our multi-tenant application in a cloud environment.

Table 1. Test – For Each Variant of Multi-tenancy Model

Test – For Each Variant of Multi-tenancy Model		
Concurrent Users per min	***Total table records***	***Operations type***
50	87976	Read
100	87976	Read
150	87976	Read
200	87976	Read

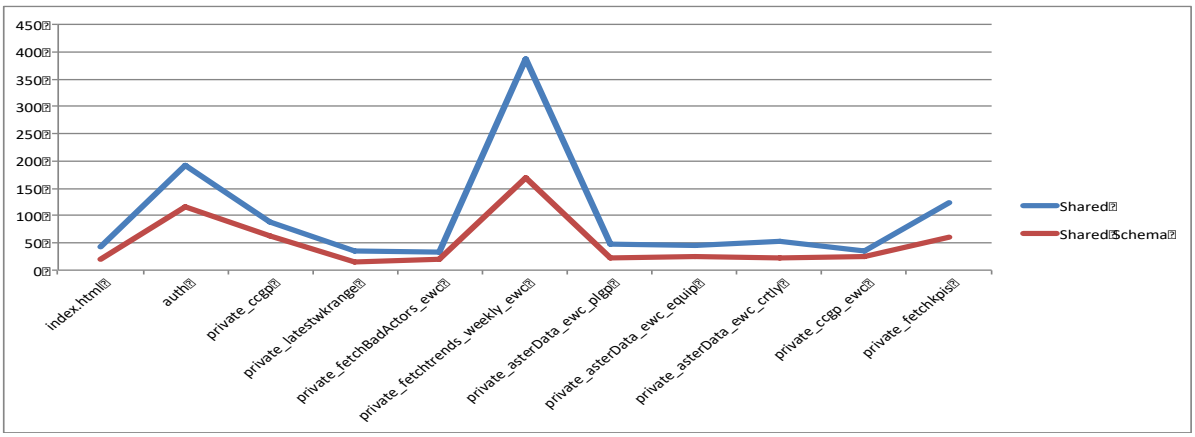

Figure 5. Latency across all endpoints

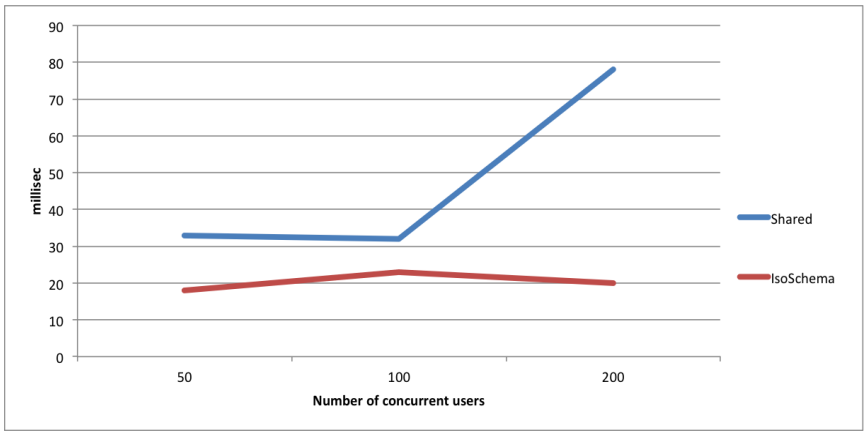

Figure. 6. Performance latency below 200 Users

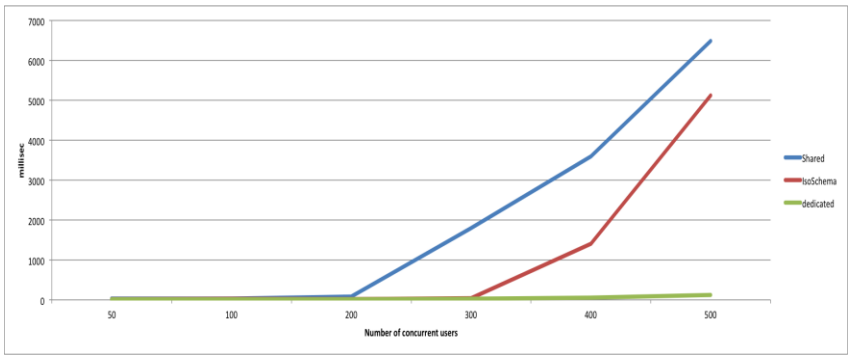

Figure 7. Performance latency above 200 Concurrent users

These tests were carried out to perform the base test on the restful endpoints provided by our case study application (AimHi). One variable is varied while the other was held constant to further understand how this variable affects the behaviour of the system. The number of tenants used in this test is three and the number of restful endpoints tested is ten. The test only covered a read operation both at the application level and data level, other types of operations such as delete, write /put will be carried out in our further research which is not covered in this paper.

5. Results

As stated in the last section, the application is configured to handle three tenants with varying number of concurrent users targeting various restful endpoints. The first set of tests for three models were performed using the same number of endpoints and dataset. Please see the results as shown in Figure 5.
The diagram above shows comparison between the shared component, isolated schema and dedicated component. The result shows that across all the endpoints tested, the shared model experience high latency compared to the isolated schema and dedicated. In some endpoints where calculations over large dataset are required, for example weekly fetch trends, the difference in latency is in magnitude of 2. However, the behaviour of the three models are consistent across all the endpoints as seen in the Figure 5.

The dedicated latency was lower compared to isolated schema model, the difference is not noticeable in this test however as the number of tenants and processing increases, dedicated will tend to perform better in terms of latency because of dedicated resources without any interference from other tenants. Figure 6 shows tests across the same number of endpoints using the three models when the user base is varied.

The figure shows up to 200 concurrent users, the dedicated and isolated schema performed reasonable well compared to shared component. This might be due to the records/rows selection process at the data layer when retrieving records belonging to a tenant or context switch between tenants in the shared component. However, there is a big leap in the latency when the number of user base hit 200 for the shared model. This occurs as more requests were queued and this causes delay in the processing.

6. Discussion

Another interesting result emerged when the three models were processed over increasing number of users as seen in figure 7. It was observed that although shared model is resource efficient, it reaches is maximum capacity around 200 concurrent users while isolated shared component reaches its maximum capacity at about 300 concurrent users, and dedicated was able to process the same dataset without any delay or errors. As the number of concurrent users increases in shared and isolated, there is exponential increase in latency across all endpoints.

Our presumption about the exponential increase in the two models might be as a result of saturated connection pooling size, which was set to 2000 and has to be shared among many tenants. Please note in this test, all form of data caching was disabled in order to show the models behaviour with any enhance algorithms.

Our empirical results are in consonant with the assertion made by [12] that there is a specific fraction of concurrent users a system can handle before the system starts queuing request or, in extreme cases, request can be dropped. Universal Scalability law introduced by [9] and Amdahl's law introduced by [6] both detailed functions to determine capacity of a system in terms of user load. [12] Asserted that resident time (latency time) of a system will start growing exponentially when the capacity is less than the concurrent users it can handle which is what is observed in both cases of multi- tenancy models experimented.

7. Conclusion

This research work systematically evaluates the performance of different multi-tenancy models at the data tier layer. Three (shared, isolated schema, and dedicated component) types of multi-tenancy models were presented. A case study approach was adopted to examine the performance of two the listed multi-tenancy models. The empirical experiments compared shared model against isolated schema using the same case study scenario and under the same circumstance. Our results show that the performance in terms of latency of isolated schema is better than the shared model. However, both models reach their full capacity at different concurrent number of users.

Shared model reaches maximum capacity under a lower concurrent user loads compared to isolated schema model. However when designing SaaS application such as relational database service where the number of tenants are considerable large, the use of partial isolated schema that creates a meta table that describe each tenant data structure and store shared data together for all the tenants data might be best options for ease of maintainability and economic standpoint. In conclusion, the number of concurrent user loads, the size of tenants, the duration of queries performed and the size of data store should be critical element when choosing which multi-tenancy model to adopt when designing an application. Our research has shown insight how the performance of

SaaS application can be impacted by the choice of the multi-tenancy model employed at the data tier layer. Our future works will explore how to best configure a multi-tenant application using Meta data pattern such as to increase isolation and customization.

8. Future Works

This research work has open up further works to better capture the performance of multi-tenancy under different scenario. Our future works will look into performance and interference that may occur when writing and deleting tenant records in a shared model. Another interesting area is hierarchical multi-tenancy; many organisations are structured in hierarchical way with various business rules and data governance. Our next work will study how multi-tenancy can implemented in this kind of organisation and provide challenges when developing hierarchical multi-tenant application. This will be extended into containerized technologies for portability across various platforms.

9. References

[1] Mohemed Almorsy and John Grundy. "SMURF: Supporting Multi-tenancy Using Re-Aspect Framework". In: 17th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2012) (2012).

[2] Michael Armbrust et al. Above the Clouds: A Berkeley View of Cloud Computing. Tech. rep. 2009.

[3] Cor-Paul Bezemer and Andy Zaidman. "Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare?" In: In Proceeding of 4th International Join ERCIM/IWPSE Symposium on Software Evolution (IWPSE-EVOL) (2010), pp. 88–92.

[4] Ngo Huy Bien and Tran Dan Thu. "Hierarchical Multi- Tenant Pattern". In: 2014 International Conference on Computing, Management and Telecommunications (ComMan Tel) (2014), pp. 88–92.

[5] Christoph Alexander Fehling et al. Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud. The Science of Microfabrication. Springer, 2014.

[6] Eduardo B. Fernandez. "Patterns for operating system access control". In: In Proc. of PLoP 2002 (2002).

[7] Amdahl G.M. "Validity of the single processor approach to achieving large scale computing capabilities". In: In American Federation of Information Processing societies: Proceedings of the AFIP '67 Spring joint Computer Conference (1967).

[8] Chang Jie Guo et al. "A framework for native multitenancy application development and management". In: In Proc. Int. Conf. on E-Commerce Technology (CEC) and Int. Conf. on Enterprise Computing (2007), pp. 551 –558.

[9] Changjie Guo et al. "A Framework for Native Multi- Tenancy Application Development and Management." In: CEC/EEE. IEEE Computer Society, 2007, pp. 551– 558. ISBN: 0-7695-2913-5. URL: http://dblp.uni- trier. de/db/conf/wecwis/cec2007.html#GuoSHWG07.

[10] Rouven krebs, Alexander Wert, and Samuel Kounev. "Multi-tenancy performance benchmark for web application platforms". In: ICWE 13 Proceedings of the 13th International conference on web Engineering 7977 (3 2013), pp. 424–438.

[11] Guthaus M.R et al. "Mibench: a free, commercially representative embedded benchmark suite". In: In Proceeding of International Workshop on Workload Characterization (2001), pp. 3–14.

[12] Laud Charles Ochei, Andrei Petrovski, and Julian M. Bass. "Degree of Multitenancy Isolation for Cloudhosted Software Services: Synthesis of findings from three Case Studies". In: International Journal of Intelligent Computing Research (IJIR) 6 (3 2015).

[13] Laud Charles Ochei, Andrei Petrovski, and Julian M. Bass. "Evaluating degrees of tenant isolation in multitenancy patterns: a case study of cloud-hosted version control system (VCS)". In: International conference on information society (i-society) (2015), pp. 59–66.

[14] Davy Preuveneers et al. "Systemic scalability assessment for feature oriented multi-tenant services". In: The Journal of Systems and Software 116 (2016), pp. 162– 176.

[15] Antonio Rico et al. "Extending multi-tenant architectures: a database model for a multi-target support in SaaS applications". In: Enterprise Information Systems 10.4 (2016), pp. 400–421.

[16] Bhaskar Prasad Rimal, Eunmi Choi, and Ian Lumb. "A Taxonomy and Survey of Cloud Computing Systems". In: Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC. NCM '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 44–51. ISBN: 978-0-7695-3769-6. DOI: 10. 1109 / NCM.2009.218. URL: http://dx.doi.org/10.1109/NCM. 2009.218.

[17] SalesForce.com. The Force.com Multitenant Architecture Understanding the Design of Salesforce.com Internet Application Development Platform. White Paper. Salesforce.com, 2001.

[18] Mark Turner, David Budgen, and Pearl Brereton. “Turning Software into a Service”. In: Computer 36.10 (Oct. 2003), pp. 38–44. ISSN: 0018-9162. DOI: 10.1109/MC. 2003.1236470. URL: http://dx.doi.org/10.1109/MC. 2003.1236470.

[19] Stefan Walraven, Eddy Truyen, and Wouter Joosen. “A Middleware Layer for Flexible and Cost-efficient Multi-tenant Applications”. In: Proceedings of the 12th ACM/IFIP/USENIX International Conference on Middleware. Middleware’11. Lisbon, Portugal: Springer- Verlag, 2011, pp. 370–389. ISBN: 978-3-642-25820-6. DOI: 10. 1007 / 978 - 3 - 642 - 25821 - 3 19. URL: http: //dx.doi.org/10.1007/978-3-642-25821-3 19.

[20] R.K. Yin. “Case Study Research: Design and Methods”. In: 3rd edition. SAGE Publications (2003).

