Hierarchical Multi-tenancy in Business to Business Software Services

Adeniyi Abdul[1]
University of Salford
Salford, UK
Email: A.Abdul1@edu.salford.ac.uk

Julian M. Bass[2]
University of Salford
Salford, UK
Email: j.bass@salford.ac.uk

***Abstract*—The introduction of cloud computing has changed the provisioning and consumption of IT resources. Software-as-a-Service (SaaS) applications provide functionality using internet connectivity and enable centralised management of executable code resources. Multi-tenancy is an architectural pattern for sharing a single instance of executable software while isolating the data and business process serving each tenant. In a multi-tenant application, a tenant is a logical grouping of end-users who share common code, data and process instances. Multi-tenancy can cut across all the layers of an application software architecture.**

Conventionally, multi-tenancy is implemented as a flat structure. All tenants are treated in the same way by application software. However, large enterprises are hierarchically organised with defined boundaries between business and functional units. A SaaS solution, aimed at large enterprise users, needs to reflect these hierarchical boundaries to eliminate duplication of functional software but enforce data and business process separation.

This paper introduces a novel hierarchical multi-tenancy architecture for an enterprise-scale business-to-business (B2B) cloud-hosted software service application. Organisational hierarchy plays a vital role in grouping end users into tenants and sub-tenants in our software. A new reference architectural style for implementing hierarchical multi-tenant application is presented here for the first time.

We use a case study approach to empirically evaluate the latency of this architecture in comparison to known flat multi-tenancy patterns. Our experimental evaluation supports the hypothesis that the hierarchical multi-tenancy approach meets the needs of our application and improves its performance.

I. Introduction

Cloud computing is changing how software services and IT infrastructures are provided as a result of benefits such as elasticity, scalability, ease of maintenance, on-demand provisioning etc. [3]. Software-as-a-Service (SaaS) solutions relieve end-users of the need to upgrade and maintain their applications. To create affordable solutions, SaaS providers have re-engineered and provisioned applications on shared resources and multiplex usage among many end-users.

By scaling an instance of software to serve many groups of end-users known as tenants, SaaS can maximize hardware usage, minimize operational cost and render services at affordable rates. Multi-tenancy coupled with other technologies are employed by providers to deliver a single instance-like solution on shared resources. Multi-tenancy delivers a high return on investment by utilizing the same hardware and software resources to serve multiple organisations. A tenant is an organisational representation of a group of users employing the use of shared software to meet their business needs.

One of the prevalent usages of multi-tenancy is in the design of data management system for SaaS solutions. Data management systems are programs used to merge, extract and filter data from various data stores for use in other applications. Previous research on multi-tenancy for data management system represent organisational data as a flat structure, i.e. grouping of tenants data based on functional requirements without defined relationship among data or data stores.

Flat structure multi-tenancy makes it hard to store data such that it reflects business boundaries that capture the business constraints without data or process duplication. For example, storing related geographical data in a flat system requires extras application level implementation to establish proper authorization and business rules for each region.

Most organisations operate in a hierarchical way and embed this into their business process and data management system to enforce different business constraints and policies. The hierarchical organisational structures place new demands on B2B multi-tenant applications. As a consequence of observing these organisational structures, we propose a new concept of hierarchical multi-tenancy for B2B SaaS application.

This paper introduces a new concept of hierarchical multi-tenancy in an organisation with hierarchical structure and proposes a novel reference pattern on how to design and implement hierarchical multi-tenant B2B applications. We employed this pattern in our case study for the development of a multi-tenant solution for asset and integrity management for an organisation with hierarchical business structure.

This research makes three main contributions: (i) a novel concept of hierarchical multi-tenancy for enterprise-scale B2B software-as-a-service, (ii) a new reference architectural style for cloud-hosted applications with hierarchical multi-tenant requirements, and (iii) performance evaluation of this architecture compared with known flat multi-tenancy patterns.

The rest of the paper is as follow section 2 presents pre-

[1]Adeniyi Abdul is with the School of Computing, Science & Engineering, Newton Building, University of Salford, M5 4WT, United Kingdom email:A.Abdul1@edu.salford.ac.uk

[2]Julian Bass is with the School of Computing, Science & Engineering, Newton Building, University of Salford, M5 4WT, United Kingdom email:j.bass@salford.ac.uk

vious research contributions in the field of cloud computing, software-as-a-service, multi-tenancy, data management system and business to business systems. Section 3 presents our motivating scenario and establishes the concept of flat multi-tenancy and hierarchical multi-tenancy. Section 4 presents the research methods employed for empirical evaluation of the architecture. Section 5 details architecture outline and experimental analysis employed in the case study. Section 6 presents the results and findings of our empirical studies. Section 7 discusses the results obtained, and lastly, section 8 presents the conclusion of the research work

II. Related Work

Many organisations have seen a good return on investment after adopting cloud services for their IT operations. This has led to many solutions that helped increase business agility and profitability. Cloud solutions provide organisations with scalable, secure, on-demand and utility-like solutions with minimal start-up and operating cost. One of the emerging cloud solutions is Software-as-a-Service (SaaS). SaaS is a utility-like cloud-hosted software application provided via the internet for many users to consume on-demand. A well-designed SaaS application must be multi-tenant efficient, easily configurable and scalable [4].

Affordable SaaS applications are actualized with the help of advances in cloud technologies such as multi-tenancy and virtualization. Virtualization creates an abstracted layer between the application and hardware, making it possible for many instances of the application to share pool computing resources [17].

In Multi-tenancy, tenants share an application instance, while data and business process instances are isolated. There are three multi-tenancy design patterns: dedicated, tenant isolated and shared component [5]. The dedicated model dictates that application data are hosted in separate instances for each tenant without any form of sharing with other tenants. In the Tenant isolated model, each tenant operates its own set of data grouped into a name-space, however, all the tenant data tables are co-hosted in the same instance. In the shared multi-tenancy model, application data are stored in the same tables with row level isolation for all tenants.

Although multi-tenant applications deliver customized solutions with shared resources, the tenants must be confident that their data and process are protected and not accessible by other tenants on the same platform. The multi-tenant application must ensure that a tenant usage does not impact the performance of others or denied others resources when needed. Isolation is also crucial when billing tenants on their application usage. Previous research has evaluated the degree of isolation among tenants using component-based approach [13]. Dedicated components, without any form of sharing among tenants, offer a high degree of isolation with the highest level of security and low interference. SaaS application provisioning using dedicated approach can be extremely expensive and not economically viable for applications with hierarchical tenant structures.

Flat multi-tenancy with the shared model is known to experience conflicting preference [6]. The author proposed a solution to overcome conflicting preferences among tenants using service configuration techniques. The proposed solution is a dynamic user-centric adaptation in a shared multi-tenant environment to increase tenant satisfaction.

In multi-tenant applications, the data size tends to grow as the number of tenants increases [9]. The paper establishes the premise that a tenant could comprise of multiple sub-tenants and proposes a tree-like structure for grouping tenants in a SaaS application. This work further proposed splitting tenant's data based on the logical representation and creating a data store to hold each sub-tenant data. Three models for storing the tenants' data have been proposed: monolithic, distributed and hybrid [9]. Where the number of tenants is limited, a monolithic model will be adequate, while the distributed model is used to store data across many data stores and the hybrid model is a fusion of monolithic and distributed to reduce the cost of hosting.

This is the first research that considers hierarchical structure in a multi-tenant application. However, this work only introduces abstraction layer into the widely used three-tier architecture. The proposed layer was used to dynamically stored data to the hierarchical data stores using Linear and Non-Linear (Permutation based) algorithms. Their results show that the Linear algorithm has low execution times when the search space is small and use cases are limited. However according to their permutation-based algorithm especially the improved permutation algorithm (Fast Data Allocation algorithm) proposed can perform better in a large search space [10].

In a SaaS multi-tenant application, a large number of tenants and sub-tenants precipitates a high number of data stores, with data duplication, increased server management overhead, high server utilisation and increased provisioning costs. For example, Amazon web services only permit 40 relational database system instances to be provisioned per account [15].

Our paper proposes grouping tenant and sub-tenant data in a hierarchical way that reflects the relationship among the tenants' data and storing them in a cluster of data stores. This approach differs from previous hierarchical database solution proposed by [10] that creates a data-store for each tenant and sub-tenants and mapped the data stores in a hierarchical way. Our research shows that the hierarchical nature of the organisation and among tenants data plays a vital role when designing multi-tenant application for B2B enterprise hence the need for this research.

III. Motivating Scenario

There are two categories of SaaS solutions presented by [4]. These are line-of-business and consumer-oriented service. The line-of-business service is what we refer to as business to business service (B2B). These services are offered to enterprises with large, customizable business solutions aimed at facilitating business processes. While consumer-oriented service is referred to as business-to-consumer (B2C) service and consumed by the general public. B2C services do not

address any organisation needs, rather they are provisioned to the public as a software to meet daily needs [4]. Our scenario captures multi-tenancy in a SaaS provider that develops multi-tenant applications for use in large enterprise scale environment.

A. Flat Multi-Tenancy

Employing multi-tenancy in B2B has gained little attention in the field of SaaS. In flat structure multi-tenancy, there is no defined hierarchical relationship among tenants. A flat tenant is a virtual or logical grouping of organisational users. In this scenario, a user can only belong to a tenant as shown in figure 1.

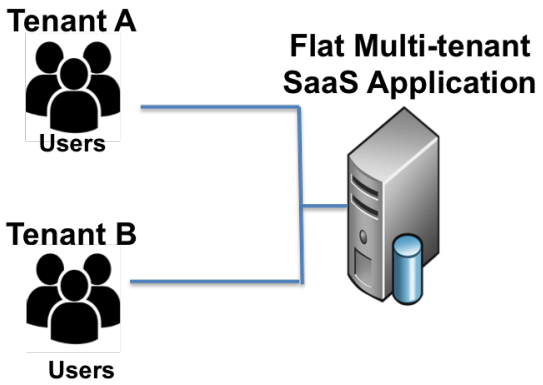

Fig. 1: Flat Structure Multi-Tenancy

B. Hierarchical Multi-Tenancy

In hierarchical multi-tenancy, there are defined hierarchies among tenants and sub-tenants for the purpose of setting business boundaries and functional isolation as shown in figure 2 below. The tenants and sub-tenants exhibit a tree-like structure, grouped in a logical way to represent organisation structures. In this tree-like structure, the SaaS provider is the parent node. Tenants at one level might be corporations, where each of its sub-tenants has private data and application requirements which are clearly different from top-level tenants.

At the next level of the tree are business units or territories that form organisational structures within the multi-national corporation. These can still be subdivided into different branches or sites. Each site has users that share common features and perform similar business operations. Example of this tree-like hierarchical organisation was employed in our case study research.

The case study described in the methodology section is developed by Add Energy, a SaaS provider in the energy industry, specialising in asset management consultancy services and software solutions.

Add Energy was tasked to provide a cloud-based service to an International Power Generation company with two business units located in the United Kingdom.

The cloud service was designed to help the units operate efficiency and transparently in accordance with the industry standard.

Unit A and Unit B are part of business units of a broader energy company. Each unit has an operational team lead for overseeing operations and maintenance data. They also manage a group of engineers and contractors who rely on the cloud service provided by Add Energy for operational decision in their day to day operational and maintenance work orders.

The team leads operate at the business units level and report to the operational excellence manager at the head office, who monitor, manage and report, using the same cloud service, to senior management of the operation of the various business units.

Add Energy has developed a hierarchical multi-tenant application called AimHi that embeds these business hierarchies. This multi-tenant application will help organisations with a hierarchical structure in the energy sector to better optimise, manage and visually evaluate the asset performance across many regional stations. Figure 2 shows a hierarchical relationship as described in the case study.

Using Hierarchical multi-tenancy in AimHi helps Add Energy's clients to establish defined business rules and manageable security around their data and processes. This helps to define and group tenants access control at the data layer. Another benefit of a hierarchical approach to Add Energy compared to flat approach is a performance improvement. Proper segregation of data in a defined way helps to eliminate large scanning of dataset when performing operations for sub-tenants.

IV. Research Methods

This exploratory research was conducted in the context of an embedded case study [14] approach over a 5 year period. This research employs case study approach to empirically evaluate the performance of various methods of storing data in a hierarchical multi-tenant application. [16] detailed the suitability of using case study approach in the empirical investigation into a contemporary phenomenon.

A scrum-like software development method formed the basis of an innovation process in collaboration with the partner company and their clients [1]. This type of industrial case study is found in the collaborative research tradition [8].

A series of workshops, comprising key company domain experts, was used to identify and elaborate software requirements. The scrum method includes product demonstrations at the end of each sprint, which was used to gain feedback on the solutions produced and to help prioritise future directions. The functional and non-functional requirements were used to identify architectural software structures. These software structures help in the development of the hierarchical multi-tenancy architectural style which is the subject of this research. This architectural style was then experimentally characterised, tested and evaluated.

V. Case Study

A. Architecture Outline

This section details the architectural design adopted when implementing our case study hierarchical multi-tenant appli-

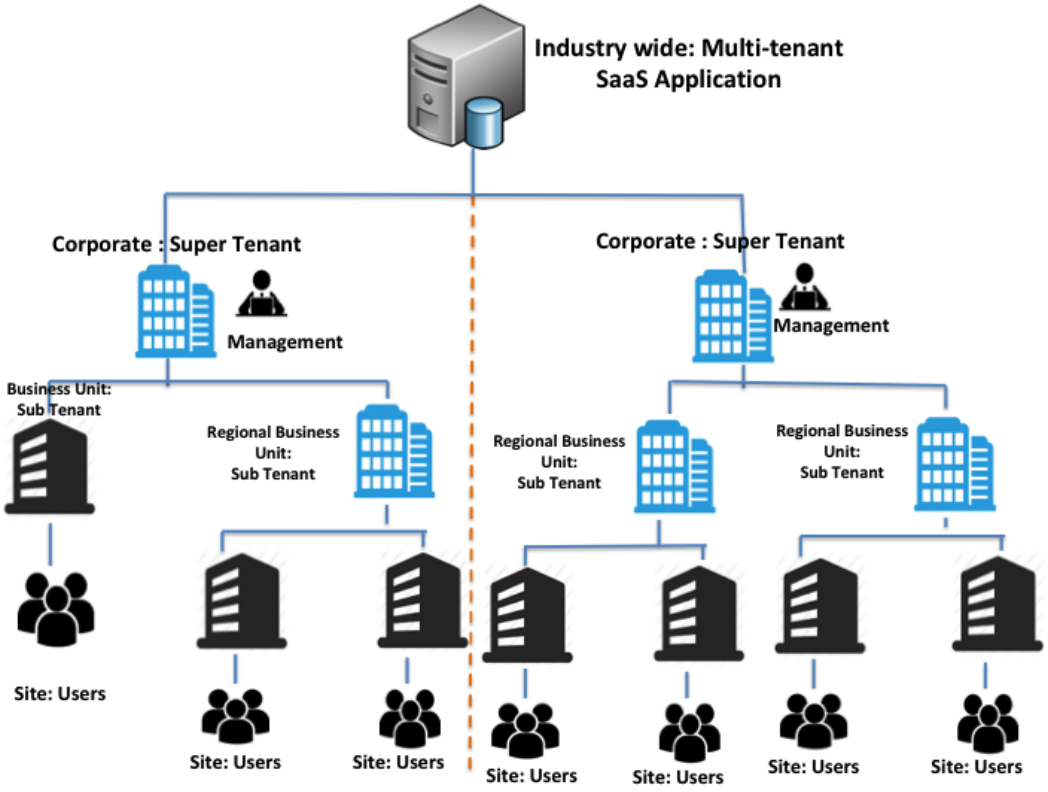

Fig. 2: Hierarchical Multi-Tenancy

cation. The design decision was as a result of the uniqueness of the system, hierarchical nature of the organisation, and business requirements. The tree-like diagram in figure 3 is an example of tenants structure adopted in our application data design. The tenants are sub-grouped into different levels, where level 0 is the highest level and level 5 is the lowest level.

Users in a tenant can only access data within their level or lower level. Tenants are grouped into levels which set the boundaries of their operations. For example, a departmental head in level 4 will have access to level 5 data and manage all level 5 tenants if acting in a managerial role. However, other tenants at level 5 would not have access to level 4 data. And also application data, process, roles and privileges are also grouped into levels. where level 0 stands for restricted data and level 5 applies to a more general data that can be accessed by all the tenants.

Data in level 3, 4 and 5 that are accessible to all the tenants can be stored in a shared table. While restricted data can be stored in separate isolated tables further grouped by name-space. Tenants data are grouped by access level and partitioned using list partitioning algorithm. Partitioning data this way, helps our queries to access only a fraction of the data at a given time hence result in faster queries.

The concerns itemized below became apparent during re-quirement gathering phase. The elicitation techniques em-ployed are series of workshops, unstructured interview with

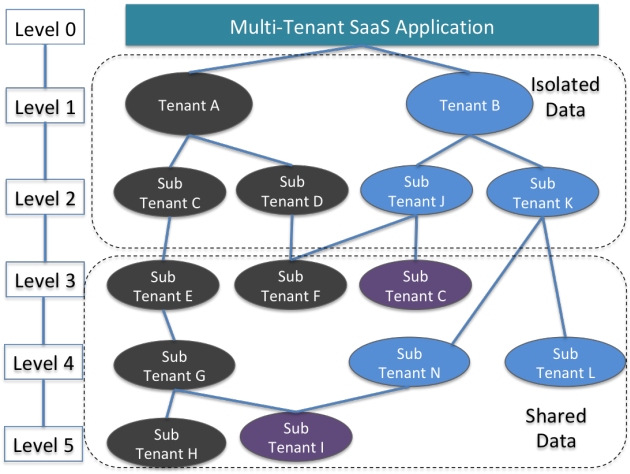

Fig. 3: Tree Like structure

domain experts and product prototyping. Our architecture design aims to address some of the concerns raised in the hierarchical multi-tenant solution.

- Data Access Across Tenants - Master and transactional data need to be stored or configured such that sharing is made possible and easy without compromising the integrity of the data or privacy across the organisation. However, the application must maintain access control

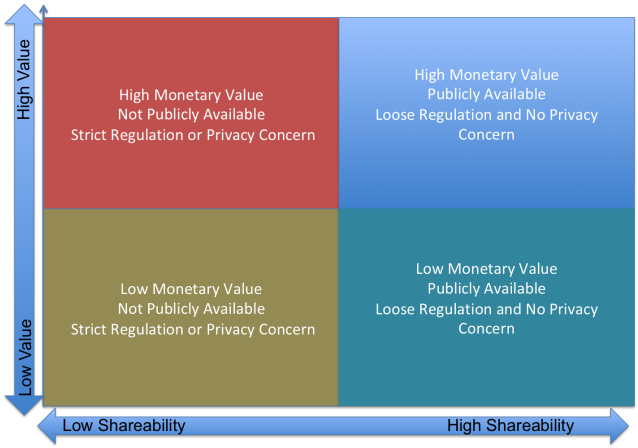

Fig. 4: Data value vs Shareability

among tenants.

- Data Inheritance - For data to be used within the right domain and among interested parties, data relationship and boundaries must be established, showing how a parent or top-level tenant data can be abstracted from the lower-level tenant data. Also, data could be represented such that it belongs to multiple tenants when there is a relationship.
- Tenant Discontinuity - In a traditional flat multi-tenant application, a tenant discontinuity only results to deletion of data related to that tenant and removing the configurations for that particular tenant. However, in hierarchical multi-tenancy, discontinuity of tenant may not result in total removal of data or configurations of this tenant because other tenants may depend on this information to perform their operations. In our case, the architecture must address such possibilities without disruption to the application. Tenant and Data dependency is a crucial part of a hierarchical multi-tenant application; our implementation employed soft dependency such that it can be modified in case of a tenant discontinued.
- Elasticity and Scalability - One of the advantages of the cloud is the ability to accommodate spikes in load demand by provisioning or de-provisioning resource without human intervention. In a multi-tenancy, there is a tendency that customer data can grow beyond the allocated storage capacity. The application must be able to re-allocate resources or move data belonging to some sub-tenants to another storage without interruption to an existing system. In hierarchical multi-tenant application, this becomes challenging because these data may be interrelated or belong to other users in other tenant domain. Hence the use of domain and partitioning techniques in our system design.
- Data value and Shareability - Data are becoming an important asset of an organisation. The advantage of big data for business cannot be overemphasized. All data are valuable, but some are valuable than others in certain context. For example, credit card details are more valuable than customer service feedbacks survey. The more valuable data is to a business entity, the less likely it will be shared among other cloud tenants. Figure 4 shows the value of data should be used to determine how it's shared. This diagram was used when determining how data are grouped into shared and isolated domain within our implementation.
- Service level Agreement - Hierarchical multi-tenant application needs to strike a balance in the midst of conflicting Service Level Agreement (SLA). This is out of the scope of this research.

B. Experimental Setup

AimHi is a cloud-based SaaS application developed to optimize and manages a large number of assets across global energy sector firms. It aims to provide maintenance engineers with an online platform to improve maintenance planning and reduce the cost of breakdowns.

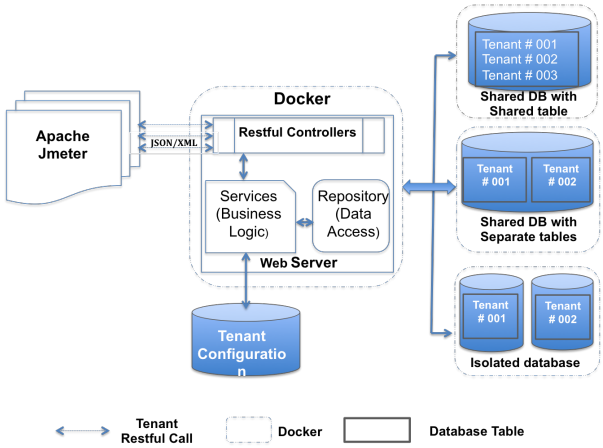

Fig. 5: AimHi Architectural Diagram

Our case study (AimHi) application is a richly designed hierarchical multi-tenant application that gathers maintenance data from different tenants and transforms them into comprehensive charts that provide insight into the performance of the assets monitored and carefully report the effect of the maintenance strategy deployed. One of the benefits is to provide management and stakeholders overview on how the maintenance strategies have improved the safety of engineers, raised the maintenance quality and result to cost saving for the organisation.

The first version of AimHi was deployed on Amazon elastic beanstalk with load balance across Amazon availability zone. We adhered to three-tier architecture design, with the front end of an application implemented as a single page application powered by angular, the middle tier is known as the application layer is implemented using spring framework and hosted using Apache tomcat wrapped in a docker environment, and the database layer powered by MySQL provision on Amazon RDS platform.

In order for application portability and experimental repeatability, a containerised technology called docker is employed. Docker platform is a container technology that abstract application into an image that is easily portable across compatible hosting platforms. Docker is an opensource, lightweight, portable and consumes low resource compare to other types of container technology solutions [12].

The experiment is carried out on docker container running on a machine with 16GB memory, 3.1GHz Intel Core i7 processor and mac0S Sierra. The application is deployed on Apache web service running tomcat and further connected to a backend database instance. The application exposes restful endpoints to be consumed by JMeter for experimental purpose. Apache JMeter is favoured because of its numerous capabilities and also its an open source software written in Java for performing a load test on a web-based application. See figure 5 for our architectural diagram for AimHi.

The hierarchical nature of the tenants in our multi-tenant application and the size of data being processed by the application requires extensive testing to evaluate the performance impact of the application. The objective of our experimental test is to evaluate the performance of our proposed hierarchical architecture and compare its performance with flat multi-tenancy architecture introduced in our previous paper [1].

These tests measured the latency of each of the restful endpoints in our case study. In the first test, the application data is held constant while the number of concurrent users is varied. And in the second test, the number of concurrent users is held constant while data size of the application is varied. The former helps to understand the performance of the application as the number of concurrent users increases. While the latter test shows the latency of the system as the data size increases.

Both experiments were performed across both the flat multi-tenancy and hierarchical multi-tenancy. To avoid skew results and establish consistency across many test cycles, database caching was disabled in the experiment. The test captures only the performance of read operations because the case study application does more reading and data aggregation across various assets.

The table 1 and 2 show the total number of users, the total records used and the operation type employed during the testing.

VI. Results And Findings

This section details our finding after performing various experiments using the table 1 and table 2 parameters. The findings in figure 6 and figure 7 show results for flat, non-partitioned hierarchical and partitioned hierarchical model. In the flat multi-tenant application, all the tenant data are stacked in a related table based on relational rules. In non-partitioned hierarchical multi-tenant application, data are grouped into subtype tables based on the access level without any partitioning algorithm applied. And for partitioned hierarchical multi-tenant application, tenants' data are grouped and partitioned based on access level and operational units.

TABLE I: Latency test of increase Users

First Experimental Test		
total Concurrent Users per min	total table records	Operation type
50	97656	Read
100	97656	Read
150	97656	Read
200	97656	Read
200	97656	Read

TABLE II: Latency test of increase data size

Second Experimental Test		
total Concurrent Users per min	total table records	Operation type
200	53732	Read
200	73716	Read
200	82763	Read
200	97656	Read

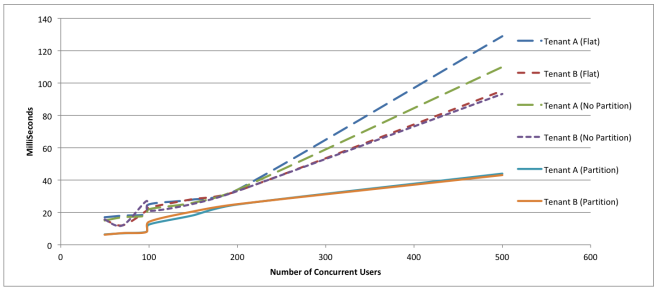

Fig. 6: Shared Data Performance (latency) vs Concurrent Users

Figure 6 shows performance (measured in latency) against an increase in concurrent users. This shows there is no significant difference in latency across all the test cases (flat, non-partitioned hierarchical and partitioned hierarchical model) when the number concurrent is small, in our case when the number of concurrent users is below 200 users. However, as the number of concurrent users increases above 300, a significant increase in latency became evident. The high latency and degradation in the system performance for flat and hierarchical non-partitioned multi-tenancy are a result of intensive database scan and filtering across tenant data when performing complex operations. While performance degradation is not conspicuous in the hierarchical model as the number of users increases, this performance improvement can be attributed to the logical grouping of the data and partitioning of the data.

The second results in figure 7 show the latency against an increase in data size. As stated earlier, the number of users in this test case is held constant. All the models performed reasonably well in terms of low latency when data size is about 53732. The latency of flat and non-partitioned hierarchical

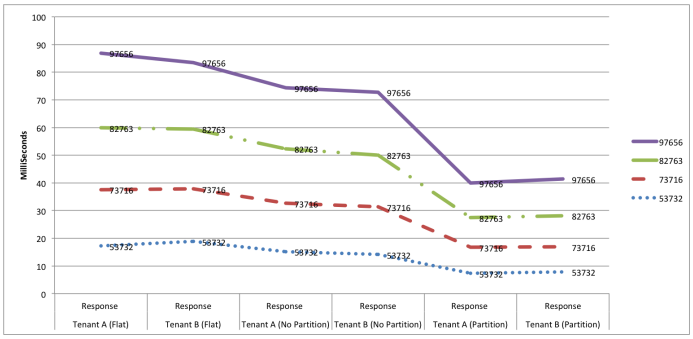

Fig. 7: Shared Data Performance (latency) vs Increases in Data Size

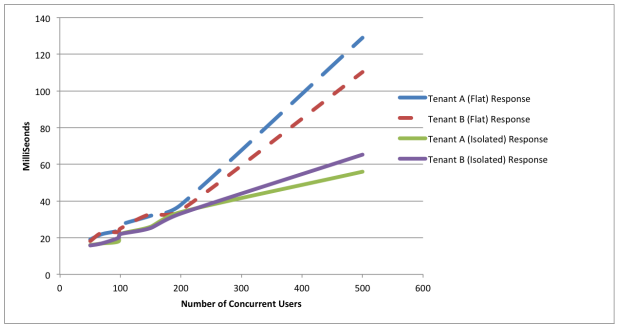

Fig. 8: Isolated Data Performance (latency) vs Increases in Data Size

model increases as the data size increases. The hierarchical partitioned application still performs better in terms of latency as the data size increases. A further experiment was performed to know the latency over an isolated data. In this experiment, each sub-tenant data is stored in a separate table hence there was no need to partition the data. The result in figure 8 shows the performance of isolated hierarchical multi-tenancy against flat data. The isolated hierarchical multi-tenancy model still produces lower latency when compared to flat multi-tenancy models.

Our results also show that there is interference in flat multi-tenancy when two tenants with a large number of concurrent users. Some tenants suffer high latency when a tenant is performing a large table scan. However, the interference is not evident in the case of hierarchically partitioned application.

VII. Discussion

This research was motivated by challenges encountered when building the cloud hosted software-as-a-service (AimHi) application for asset maintenance optimisation in the energy sector. AimHi has been developed to provide a dashboard of key performance indicators for maintenance optimisation in power stations [2] [7] with hierarchical business units.

During the development of AimHi, we observed the hierarchical organisational structure of multi-national companies created new data sharing and tenant isolation requirements for our software-as-a-service application.

The term 'hierarchical multi-tenancy' is not new. However, we approach hierarchical multi-tenancy from a new perspective and view our concept as a novel. The previous approach to hierarchical multi-tenancy focuses on creating hierarchical data stores mapped to tenants and sub-tenants [11]. This work did not approach hierarchical multi-tenancy from organisation structure perspective, rather they propose hierarchical data stores similar to the tree-like structure of tenants and subtenant. This approach is not feasible in a large enterprise-scale B2B environment with many tenants.

Hierarchical Multi-tenancy in our case can simply be described as a tree-like relationship among various tenants data and processes with defined boundaries that mirror the business structures. Hierarchical Multi-tenancy is targeted at large-scale enterprise with a hierarchical structure.

Our application can easily handle hundreds of tenants concurrently and with large datasets. One way to improve the efficiency of queries and scalability is through data partitioning [4] to further reduce expensive database scanning. Our experimental result shows that partitioning the multi-tenant application data and process based on the tenant hierarchical structure helps to improve queries and performance in terms of latency of the application. In this case, the partitioning is applied based on the hierarchical nature of our potential SaaS consumers.

The empirical experiment shows the performance of our concept in the midst of increasing concurrent users, varying data size and performance latency over isolated data. This shows the viability of our concept in handling data requests across different level of tenants without impacting the performance of the system. The performance gained in terms of latency is a step in the right direction that hierarchical multi-tenancy concept with enhanced partitioning techniques can better suit for multi-tenant application with tenants and sub-tenants.

VIII. Conclusion

This research addresses the issue of multi-tenancy in cloud-hosted software-as-a-service applications. Specifically, we propose a novel concept of hierarchical multi-tenancy for enterprise-scale B2B applications.

We adopted a case study approach in this study to experimentally evaluate our hierarchical multi-tenancy reference architecture compared with previously published dedicated, tenant-isolated and shared component multi-tenancy patterns. Our case study was derived from a commercial project developed for an asset management and optimisation for the energy sector called AimHi.

The paper also presents some concerns associated with hierarchical multi-tenancy, these became apparent during the requirement gathering phase. We addressed some of the concerns by structuring tenants data in a hierarchical way. The data value and shareability table were used to structure data into different access level.

Our findings show that the hierarchical multi-tenancy reference architecture is better suited to the requirements of

our application than other multi-tenancy models. Specifically, the paper shows through our findings that hierarchical multi-tenancy architecture can help achieve hierarchical structure similar to organisation hierarchy and still experience better performance. The research shows that hierarchical structure used by business-to-business (B2B) to capture business rules and governance, can be enforced in a multi-tenant application.

Our future work will look into the impact of the write operations in hierarchical multi-tenant application, and scalability of hierarchical tables across a multi-cloud environment.

IX. Acknowledgement

The funding for this research was provided by Add Energy Ltd and Innovate UK under a Knowledge Transfer Partnership with the University of Salford, Manchester UK. The technical expertise on how to implement SaaS application for oil and energy sector was provided by Peter Adam and Hossein Ghavimi of Add Energy Ltd Aberdeen

References

[1] Adeniyi Abdul et al. (July 2017a). "A performance evaluation of multi-tenant data tier design patterns in a containerized environment". In: *International Conference on the Information Society (iSociety)*. Dublin, Ireland: Infonomics Society, pp. 115–120.

[2] Adeniyi Abdul et al. (July 2017b). "Product Innovation with Scrum: A Longitudinal Case Study". In: *International Conference on the Information Society (iSociety)*. Dublin, Ireland: Infonomics Society, pp. 21–26.

[3] Michael Armbrust et al. (2009). *Above the Clouds: A Berkeley View of Cloud Computing*. Tech. rep. University of California at Berkeley.

[4] Frederick Chong and Gianpaolo Carraro (2006). *Architecture Strategies for Catching the Long Tail*. Tech. rep. Redmond, WA (USA): Microsoft Corporation. URL: http://msdn2.microsoft.com/en-us/library/aa479069.aspx.

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

[6] Jesus Garcia-Galan et al. (2014). "User-centric Adaptation of Multi-tenant Services: Preference-based Analysis for Service Reconfiguration". In: *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS 2014. Hyderabad, India: ACM, pp. 65–74. ISBN: 978-1-4503-2864-7.

[7] Gareth Hollyman (Feb. 2018). *Computer scientists optimise safety at UK power stations*. URL: https://www.salford.ac.uk/news/articles/2018/computer-scientists-optimise-safety-at-uk-power-stations.

[8] Mathiassen Lars (2002). "Collaborative practice research". In: *Information Technology People* 15 (4), pp. 321–345.

[9] Pieter-Jan Maenhaut et al. (2014). "Characterizing the performance of tenant data management in multi-tenant cloud authorization systems". In: *Network Operations and Management Symposium (NOMS), 2014 IEEE*.

[10] Pieter-Jan Maenhaut et al. (2015a). "Design and Evaluation of a Hierarchical Multi-Tenant Data Management Framework for Cloud Application". In: *International Workshop on Management of the Future Internet*.

[11] Pieter-Jan Maenhaut et al. (2015b). "Design of a Hierarchical Software-Defined-Storage System for Data-Intensive Multi-Tenant Cloud Applications". In: *11th International Conference on Network and Service Management(CNSM, 2015)*.

[12] Dirk Merkel (Mar. 2014). "Docker: Lightweight Linux Containers for Consistent Development and Deployment". In: *Linux Journal* 2014.239. ISSN: 1075-3583.

[13] Laud Charles Ochei, Andrei Petrovski, and Julian M. Bass (2015). "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)*, pp. 59–66.

[14] Per Runeson et al. (2012). *Case Study Research in Software Engineering: Guidelines and Examples*. 1st. Wiley Publishing.

[15] Amazon Web Services (2014). *Amazon RDS DB Instances User Guide API*. URL: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Overview.DBInstance.html (visited on 06/10/2018).

[16] R.K. Yin (2003). "Case Study Research: Design and Methods". In: *3rd edition. SAGE Publications*.

[17] Qi Zhang, Lu Cheng, and Raouf Boutaba (2010). "Cloud computing: state-of-the-art and research challenges". In: *Journal of Internet Services and Applications* 1.1, pp. 7–18. ISSN: 1869-0238.