TOWARDS BENCHMARKING A VIRTUAL SCAFFOLD

What it Should Be, Can, Cannot be

B Nagarajan
Assistant Professor
Department of Computer Applications,
Arunai Engineering College, Tiruvannamalai, India.

Abstract: Cloud computing is becoming vogue among the business firms. The reason behind is - its availability of services over internet anywhere, anytime. Cloud Benchmarking is nothing but fixing standards while developing Multi-Tenant SaaS application, which will result in better and amicable solution for the end- user. Virtualization is an important entity for Cloud based services. The virtual scaffold is virtualized platform to develop Multi-Tenant Software as a Service (SaaS) application. Cloud Pattern is a reusable component which can used in the Virtual Scaffold to develop applications. Benchmarks for Cloud based SaaS application is still involute. This paper proposes a virtual architecture and provides a Pattern for pragmatic implementation of benchmark solution. The aspects discussed in this paper will enlighten the sceptics and researchers to thoroughly understand a set of quantifiers essential to implement the virtual architecture in the seclude tenant perspective and will motivate in tweaking the SaaS applications.

***IndexTerms* -. Benchmark, Cloud, Cloud Pattern, Multi-Tenant Applications, Virtual Scaffold**

I. INTRODUCTION

Benchmarking is defined as a methodology of setting standards to execute a Cloud application and helps the researcher to quantify the results using metrics [3]. The motivation behind this paper is to propose a reasonable benchmark framework, what it should be. This will help the researchers and sceptics, a pragmatic implementation of the System under Test (SuT) and according to the proposed benchmark framework during their implementation. Cloud is based on the virtualization and it provides essential services such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) to the global tenant who login into the system and now a days Cloud has redefined as Everything (X) as a Services (XaaS) [1].

The essential characteristics of the Cloud include on- demand self-service, broad network access, rapid elasticity, measured usage and resource pooling [10]. The resource pooling (or multi-tenancy) is the important characteristic included by the National Institute of Science and Technology (NIST) becoming more popular among the researcher [19]. The Multi-Tenant application helps the tenant to share a single resource and view a discrete view of the resources. Tenant are group of users sharing the same discrete view of a single resource. Thus, the multiple tenant can develop their own SaaS applications using a single shared resource. These Multi-tenant SaaS applications are useful and can be implemented for various groups of people such as medicine, finance, governance, banking and so on. The Multi-tenant applications can be deployed as community Cloud for the usage of end user. To tweak these SaaS applications according to the need of the organization and multiple tenant - a virtual architecture and its relevant benchmark of acceptable parameters is essential.

Virtual scaffold is an architectural framework in which the programmer can specify how their application and database can be interoperable [21]. Scaffold has a set of layers and each layer has set of specific function. This paper elaborately discuss about setting standards for every layer in the scaffold and thus help the tenant to login into the system, develop and implement their Cloud applications and thus mitigation the vulnerability of tenant during the failover.

Patterns is a reusable and implementation component specific to a virtual scaffold [22]. Thus standard can be set to quantify component during SaaS implementation and thus provide a lucid software development at every stage. The proper implementation of benchmark framework will help to provide better solution to SaaS application both in the tenant and organizational perspective.

The benchmarking plays a major role in the software engineering [3]. It helps the Cloud application developer to checklist with various architectural and software requirement to be included for tweaking the SaaS application. Based on the checklist requirement available from the feasibility study and benchmarking, the framework for System under Test (SUT) can be finalized. The SUT will include a detailed study of software and hardware requirements. This will help the developers to quantify the SUT with suitable metrics and construe the results. The SUT can be periodically improved suitably to obtain better performance [9].

In this paper, section two studies on the various literatures related to the benchmarking. The section three elaborately discuss about the framework for the benchmarking. Section 4 propose a SuT and a pragmatic benchmark Pattern for implementation of SaaS application. Section 5 discuss about experimental setup and pragmatic solution for SuT. The last section provides an epilogue with future scope of work to be carried out.

II. STUDY ON RELATED LITERATURES

Benchmarking is still elusive and subtle among the researcher. The study on benchmarking consist of two folds. The first fold studied on basics of benchmarking and its ways of implementing them into Cloud architectural Pattern.

Folkerts E., et al., [2012] have exhaustively studied on benchmarking practice prevailing over the years and in his work. The author have presented a general requirements for writing a benchmark for Cloud based implementation. They provide sufficient information about SuT and its ways of implementation which was extremely useful. The author concluded about the ways of building and running a benchmark in a lucid manner using a case study.

Bermbach D., [2017] gave detailed note on the general requirement to be followed by the researcher while implementing benchmarking into a Cloud based environment. Conflicts are common while implementing such systems and authors have illustrated a detailed study on the ways of resolving on those conflicts. This paper was extremely useful while implementing benchmarking.

The second fold is to study on the readymade tools available for accessing the performance of Cloud based implementation. There are number of readymade tools available namely CloudCmp, HI Bench, Yahoo! Cloud Service Benchmarking (YCSB) and Transaction Processing performance Council (TPC). This study on literatures related to all of the above readymade tools are useful for this work.

The TPC was setup in year 1988 and released user manual with periodic revision for implementing and evaluating the performance of the SuT based on the relational databases. The council revised TPC in 2013 and published TPC – Virtual Measurement Single (VMS) with inclusion of evaluating the Cloud based SuT and TPC-E in 2015, which handled traditional data model using OLTP [Smith W D et al., 2013 and TPC BENCHMARK ™ E 2015].

Cooper B F et al., [2010] has contributed a detailed study on the YCSB and was extended by Patil S., et al., [2011] as YCSB++ with additional advanced features to handle NoSQL data model. Both the authors have provided techniques to handle the OLTP using Cloud data sets. Cooper B F., et al., [2010] have provided a compendium for the researchers to understand YCSB architecture and its ways and means of implementing benchmarking and evaluating datasets.

The elaborate study on the existing literatures have concluded that only a scant authors have implemented a SuT and experimented them with either Structured Query Language (SQL) or Not-only SQL (NoSQL) data model. The results were based on On-Line Analytical Processing (OLAP) with meagre datasets. Thus, selecting a proper data model which handles huge data sets in a Cloud based virtual environment which uses both SQL and NoSQL data under single SuT is a challenge and scant researcher have proposed a benchmark framework to construct and implement SuT and manipulate the data.

III. Framework for System Under Test

3.1. Problem Statement

In Multi-Tenant SaaS application, end user may be from any corner of the globe. Hence a system has to be developed for seclude tenant to login from a dashboard (as View). The resource R (or database) is to be stored in a virtual container a “silo” location. The business logic should be provided to support those resource R with an authenticated user. Manipulate shared resource R using Create Read Update Delete (CRUD) operations. To summarize the system is to be developed as Model View Controller (MVC) and an appropriate benchmark requirements has to be framed according to satisfy the needs in the seclude tenant.

3.2. Key Requirements for Benchmarking

The task of a benchmarking is to report how well a system performs with respect to the augmented priority under a given constraint [3, 9]. The key requirements provides an overall requirements to be satisfied while developing Multi-Tenant SaaS applications.

3.2.1. General Requirements

- Strong target audience discuss about the size of audience who are in need of information
- Relevant to measure the performance of the typical operations. The problem definition and required result should be relevant and should satisfy given constraints
- Economical while implementing the solution. The cost of implementing the solution should be affordable by everyone who implement the solution
- Simple to understand the constraints and implement. The solution should be in lucid manner and ease of use for the users

3.2.2. Implementation Requirements

- Fair and Portable articulate that everyone should participate equally while implementing a solution
- Repeatable is an ability to rerun the process to obtain the same solution
- Realistic and Comprehensive indicates that the audience implements SuT for any sort of problems and get a solution out of it
- Configurable means the SuT should be provide flexibility to customize the solution according to the needs of the audience

3.2.3. Workload Requirements

- Representativeness is very important among the workload. The audience should be able to interact with the workload and adjust them according to the requirements
- Scalable is another important feature which means either vertical or horizontal scalability of software and hardware up-gradation according to the present need of the audience
- Metrics should be a meaningful and understandable and it reports the reaction of the SuT after implementation

The above requirements provides a general benchmark for the problem statement and it should be particularized for the Virtual Scaffold.

3.3. Benchmark Requirements for SuT

The next step is to chalk out benchmark indices based on the key requirements and grouping those requirements with respect to enhancing a virtual scaffold into a SuT as general, implementation and workload. This will help to provide a framework for implementing the Multi-Tenant SaaS applications

3.3.1. General Requirement for SuT

- The target audience for this proposed work were the tenants login into the system
- The target audience were working based on the scaffold Pattern language implementation
- The Multi-Tenant Component Gateway Pattern (MTCG) reduces the cost of implement and thus economical, which results with better solution
- The layers associated with SuT are very simple to understand and implement

3.3.2. Implementation Requirement

- The layer associated with the SuT are rational and portable and has its own purpose
- The scaffold Pattern formulation is repeatable in the SuT and same results were obtained for given set of constraints
- The SuT is implemented for any type of realistic and comprehensive solution related to the SaaS application development.
- The tenant customizes their solution using the SuT

3.3.3. Workload Requirement

- The dataset with different workloads is fed as input into the SuT and produce a report accordingly
- SuT is scalable time to time based on requirements
- A meaningful metrics to report the system performance is used and performance of the system is measured

Based on the above indices, the first step is to chalk out the general requirements for a SuT and then move on into implementation and workload requirements. The final outcome of any benchmarking solution is to present an efficient and effective feasible solution in a given domain.

IV. System Under Test (SuT)

The general requirements to develop virtual scaffold into a SuT should include four aspects namely

- A seclude tenant login into the system is the target audience for this proposed contrivance
- The tenants were working based on the scaffold Pattern formulation
- The MTCG Pattern reduces the cost of implementation and economical to get results
- The layers associated with SuT should be very simple to understand and implement

The necessity for the enhancement of Virtual Scaffold as SuT include

- To devise a viable SuT. The distributed application are developed as a web service which has to provide realistic interoperability among various layers and satisfy the needs of global tenants
- Study on Pattern format creates positive motivation to enhance virtual scaffold based on MVC and thus to provide a rational congregation among various Patterns discussed in MTCG Pattern
- The study on scaffold Pattern language clarify that the Command Query Responsibility Segregation (CQRS) Pattern is used to perform CRUD operations on huge data set in CLOUD environment where the data set comprises of any formats such as XML, JSON, BSON and so on

To satisfy these necessities, System under Test (SuT) has to be implemented which accepts varied data formats. The Pattern formulation is formalized for the SuT and is implemented.as web service which accepts global tenants and provide them a solution.

4.1. Framework for SuT

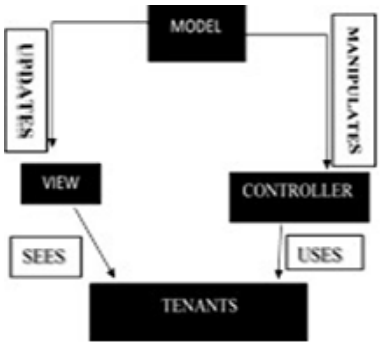

Figure 1. Model View Controller (MVC)

The enhanced virtual scaffold (otherwise SuT) is a composite assemblage of SaaS, IaaS and PaaS and is based on MVC Pattern (Figure 1) which uses RESTful web service and performs OLTP operations on CLOUD. The motivation and necessity for using the MVC Pattern into this enhancement is to provide a smooth flow of information among the global tenants login into the system and to maintain a workload balance among the tenants. MVC motivates interoperability between the various layers and thus the tenant manager implements access control efficiently during authentication and authorization of a tenant via a single customized UI (or View). The tenant uses single UI as Controller to write business logic to manipulate their data and on the flip side uses the single customized UI to update their data of varied formats into the Model and thus a smooth flow of information among the global tenant login into the system is maintained.

In MVC, the Model is the place where the tenant stores /retrieves/updates their data. The seclude tenant writes their business logic in the Controller to manipulate these data through the single secluded isolated View.

In enhanced virtual Scaffold (Figure 2), Tenant self-service layer acts as View. The business layer acts as a Controller where seclude tenant write their snippet code or business logic for invocating the resource R and metadata layer will have (or hold) the data dictionary. The storage layer (or otherwise Model) stores the resultant solutions in varied formats. The Enhanced Virtual Scaffold based on the MVC is illustrated in Figure 2.

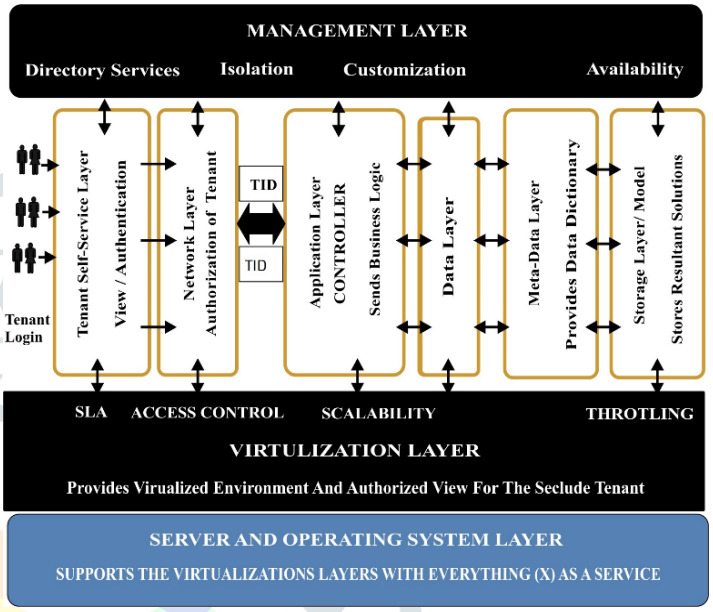

Figure 2. Enhanced Virtual Scaffold (SuT)

4.2. Design for Scaffold Pattern Language

The Pattern format for the virtual scaffold is implemented using a suitable Pattern database language. The scaffold Pattern language concentrates on the business layer of the virtual scaffold and provisions a suitable back-end database and encapsulate them with the application during the Multi- Tenant aware application development. CQRS Pattern is applied in a scaffold to perform CRUD operation. This Pattern segregates the operation that read data (Queries) and the operation that update data (command) by using separate interfaces. Thus the Pattern states there should be complete separation between "command" methods that performs action and "query" method that returns data. This implies that the data models were collaborated with other Pattern to provide results in the scaffold [22].

4.3. Workflow for CQRS

A Snippet code for Tenant identification using CQS was as follows

```
Public class Tenantdatastore {
        // Query method
Public tenant Gettenant1 (int TID) {
      // query data storage for specific tenant by TID
      // return tenant
      // command Method
Public void Insert (tenant TID)
    // Insert tenant into data storage
}
Public void updatename (int id, stirng name) {
    // find tenant in the data storage by TID
   // Update the datebase
}

}
```

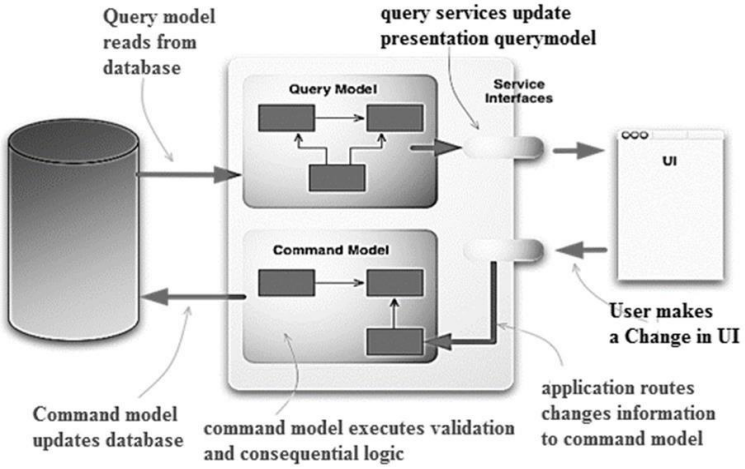

Figure 3. Workflow in CQRS Design Pattern

Event sourcing Pattern use an append-only store to record the full series of event that describe the history of actions taken on data. This Patterns were applied through the Mongo DB. Sharding Pattern is used to divide a data store into a set of horizontal partition. In Mongo DB, a driver is provided through which the tenant accesses their data. Config in the network layer of the scaffold segregates the command and query run by the tenant. The data layer shards using STSI data model instance through the query router. Thus, Sharding is used by the tenant with shared component where the data is stored across number of machines as either XML or JSON document format [22].

V. IMPLEMENTATION OF SUT

5.1. Main Idea for Writing Contrivance

The implementation of benchmark solution concentrates following indices

- The layer associated with the SuT are rational and portable and has its own purpose
- The scaffold Pattern formulation is repeatable in the SuT and same results is obtained for given set of constraints
- The SuT is implemented for any type of realistic and comprehensive solution related to the SaaS application development
  - The tenant can customize their solution using the SuT

Based on the indices specified for the implementation requirements, a main idea for the contrivance was conceived. The scaffold formulation is applied and the result is repeatable with realistic and comprehensive pragmatic solution.

5.2. Experimental Setup

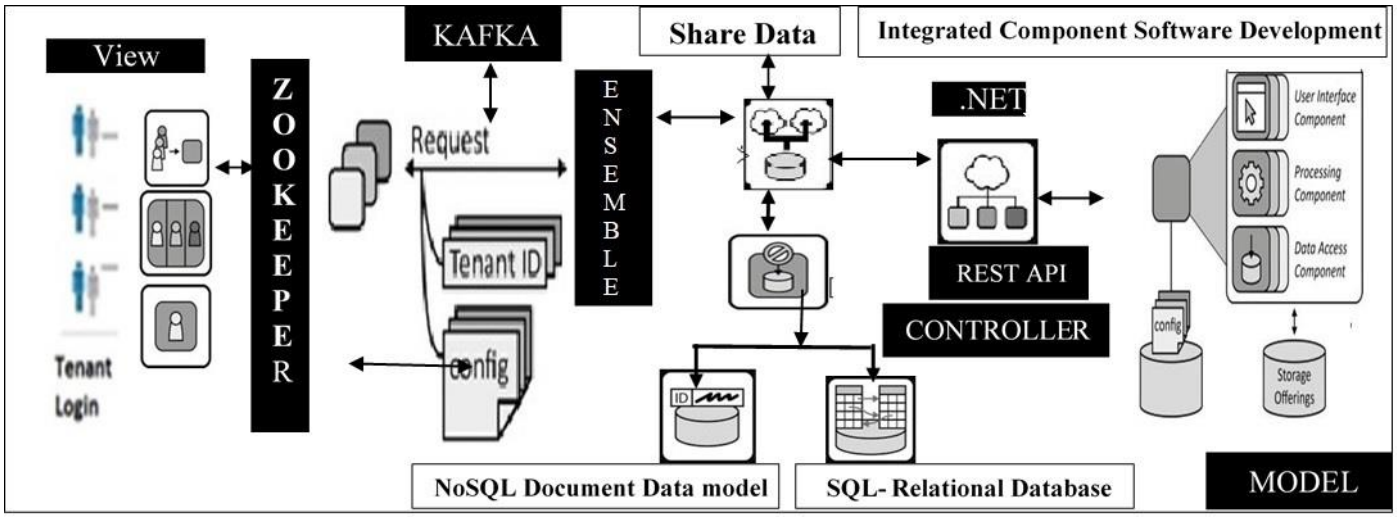

Figure 4. Experimental Setup

The experiment Setup was based on the benchmarks for contrivance discussed earlier and the software used in this setup were open source software. The tenant login into the system as either shared, isolated or dedicated component.

The Apache Zookeeper will coordinate these tenants and authenticate them with Tenant ID. The Config will provide centralized configuration management and can create Znode of category either as persistence, ephemeral or sequential. The persistence Znode is alive even after the tenant was disconnected. The ephemeral Znode are active until the tenant is live. Sequential Zone was combination of both nodes. The sessions with session ID will be created once the Znode was created and heartbeat were sent in particular interval to ascertain whether the tenants are alive.

The Apache Kafka is a distributed publish – subscribe messaging system and can handle a high volume of data and enable the tenant to pass messages from one end- point to another. The Ensemble was a Zookeeper servers with a minimum nodes of 3 to maximum 'n' nodes which fetches the data sets either SQL or NoSQL data model for processing. The enhanced virtual scaffold is used as the system model for this experiment in both cases of existing and proposed new system. The MTCG pattern format formalized is used as reference for conducting experiments.

5.3. Pragmatic Solution

This research uses ActiveX Data Object (ADO).NET to provide data access using .NET Framework. The driver provided in the shard will helps to interoperate with any language to write business logic in application layer. This work is implemented as Representational State Transfer (RESTful) web services.

The benchmark solution for the virtual scaffold is implemented as a pragmatic solution which accepts both for SQL and NoSQL datasets. The tenant login into the system has to register themselves into the tenant registration and login using a login screen The Figure 5 is used as Dashboard (or view) where the tenant can register themselves. The authentication of the tenant can be done using Figure 6.

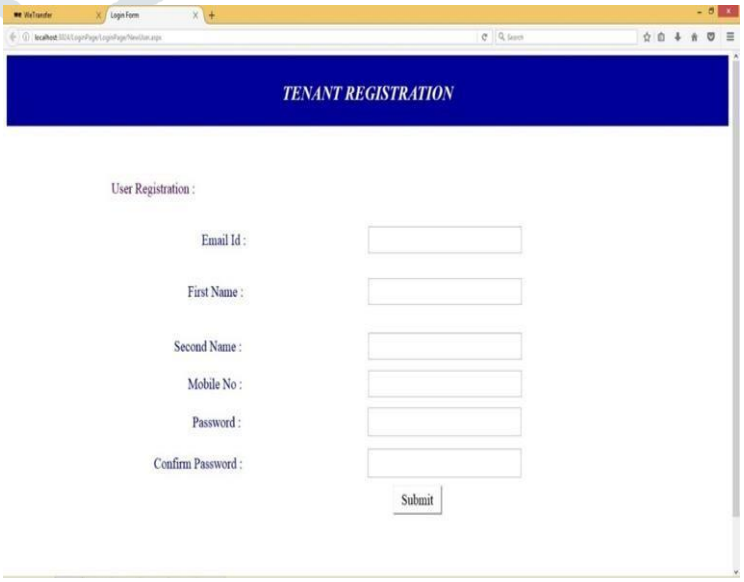

Figure 5. Tenant Dashboard

The genesis of this work is because of the development in the Service Oriented Architecture and its specific advantages in variety of social groups such as medicine, engineering, biotechnology, governance, banking, education and much more.

In present scenario, the services were redefined as online, location independent utility which is gearing up its impetus to support and satisfy the needs of all walks of people in their day to day activities which reduces the upfront cost of maintaining their hardware and software. The services were provided through a Virtual Private Network (VPN) authenticating the stakeholders. This benchmark solution will cater the needs of the sceptic and researchers and provoke a right path for the SaaS application development.

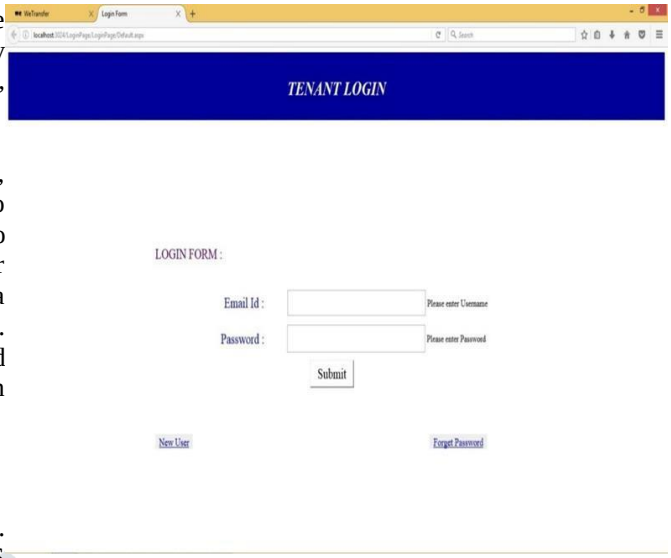

Figure 6. Login and Authentication

VI. CONCLUSION

CLOUD based application development is gearing its impetus. Virtual scaffold is a virtualized environment, where the SaaS application can be developed and delivered.

This paper elaborately discussed about the benchmarking factors and requirement to be considered while developing the Multi-Tenant SaaS application. The future implementation include a Docker and crowdsourcing and improve the solution and storage of the resultant solution.

REFERENCES

[1]Bansode, S. M (2015). Mitigating Cloud Virtualization Vulnerabilities. International Journal of Recent Advances in Engineering and Technology [pp.2347-2812], ISSN [online].
[2] Betts, D., Homer, A., Jezierski, A., Narumoto, M., & Zhang, H. (2013). Developing Multi-tenant Applications for the Cloud on Windows Azure.3rd Edition, Microsoft Press.
[3] Berbatch D et al., (2017) Cloud Service Benchmarking. Springer International Publishing AG [pp.37-45].
[4] Bermbach, D., Wittern, E., & Tai, S. (2017). Cloud Service Benchmarking: Measuring Quality of Cloud Services from a Client Perspective. Springer.
[5] Bezemer, C. P., & Zaidman, A. (2010). 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] [pp. 88-92]. ACM.
[6] Binnig, C., Kossmann, D., Kraska, T., & Loesing, S. (2009). How is the weather tomorrow? Towards a benchmark for the cloud. Proceedings of the Second International Workshop on Testing Database Systems [p. 9]. ACM.
[7] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., & Sears, R. (2010). Benchmarking cloud serving systems with YCSB. Proceedings of the 1st ACM symposium on Cloud computing [pp. 143-154]. ACM.
[8] Fehling, C., Leymann, F., Retter, R., Schumm, D., & Schupeck, W. (2011). An architectural pattern language of cloud-based applications. Proceedings of the 18th Conference on Pattern Languages of Programs [p. 2]. ACM.
[9] Folkerts, E., Alexandrov, A., Sachs, K., Iosup, A., Markl, V., & Tosun, C. (2012). Benchmarking in the Cloud: What It Should, Can, and Cannot Be. In TPCTC (pp. 173-188).
[10]Garg, T., Kumar, R., & Singh, J. (2013). A way to cloud computing basic to multitenant environment. International Journal of Advanced Research in Computer and Communication Engineering [pp.2394-2399], Volume 2, Number 6.
[11] Jacobs, D., Aulbach, S. (2007). Ruminations on Multi- Tenant Databases. In BTW Vol. 103, [pp. 514-521].
[12]Krebs, R., Momm, C., & Kounev, S. (2014). Metrics and techniques for quantifying performance isolation in cloud environments. Science of Computer Programming, 90, [pp. 116-134].
[13]Koziolek H., (2010). Towards an architectural style for Multi-Tenant software applications. Proceedings Software Engineering (SE'10) [pp.200-209]. Volume 159 of LNI, GI,
[14]Koziolek, H., (2011). The SPOSAD architectural style for multi-tenant software applications. ConferenceProceedings of Software Architecture (WICSA), 2011, 9th Working IEEE/IFIP [pp. 320-327]. IEEE.
[15]Leavitt, N. (2010). Will NoSQL databases live up to their promise?. IEEE Computer Society, [pp.12 – 14].
[16]Li, Y., Manoharan, S (2013). A performance comparison of SQL and NoSQL databases. In Communications, computers and signal processing [PACRIM], 2013 IEEE pacific rim conference on [pp. 15-19]. IEEE.
[17]Maenhaut, P. J., Moens, H., Ongenae, V., & De Turck, F. (2015). Design and evaluation of a hierarchical multi- tenant data management framework for cloud applications. 2015 IFIP/IEEE International Symposium on Integrated Network Management [IM]. [pp. 1208-1213].
[18]Mehar, D., Vishwakarma, G., & Jain, Y. K. (2015). Modified Fine-grained Data Access Control Algorithms for File Storage Cloud. International Journal of Computer Applications [online], 116[22].
[19]Mell, P., & Grance, T. (2011). The NIST definition of cloud computing, [pp. 1 – 3
[20]Mietzner, R., Unger, T., Titze, R., & Leymann, F. (2009). Combining different multi-tenancy patterns in service- oriented applications. In Enterprise Distributed Object Computing Conference. EDOC'09. IEEE International [pp. 131-140]. IEEE.
[21]Nagarajan. B., and Jayapal S., (2015). A Virtual Scaffold for Storage Multi-Tenant SaaS Data Models. International Journal of Applied Engineering Research, 10[20], 40775- 40780.
[22]Nagarajan. B., and Suguna, J., (2016). Rumination on Scaffold Pattern Language for Multi-Tenant SaaS Application Development. International Journal of Control Theory and Applications. Volume 9 [16], pp. 8257 – 8265.

[23]Patil, S., Polte, M., Ren, K., Tantisiriroj, W., Xiao, L., López, J., & Rinaldi, B. (2011). YCSB++: benchmarking and performance debugging advanced features in scalable table stores. In Proceedings of the 2nd ACM Symposium on Cloud Computing [p. 9]. ACM.

[24]Smith, W. D., & Sebastian, S., (2013). Virtualization performance insights from TPC-VMS. Transaction Processing Performance Council,[pp. 1 – 13].

[25]Yang, M., & Zhou, H. (2015). New Solution for Isolation of Multi-tenant in cloud computing. In proceeding of 3rd International Conference on Mechatronics, Robotics and Automation [ICMRA 2015], [pp. 334 – 337].

[26] TPC BENCHMARK ™ E (2015), Standard Specification, Version 1.14.0, [pp. 1 – 287].

B Nagarajan completed his Bachelors of Science in Computer Science from Sri Sankara Arts and Science College, Kanchipuram, India in 1995. He has completed his Master of Computer Applications (MCA) from Arunai Engineering College, Tiruvannamalai, India in 1998 and his Master of Philosophy in Computer Science from Bharathiar University in 2006. He is presently a Ph.D., Research Scholar (Part-Time, category B), Department of Information Technology, Bharathiar University, Coimbatore, India and has recently submitted his thesis. His research interest is CLOUD architecture and specific to Cloud Multi-tenant databases and security