International Journal of Research Studies in Computer Science and Engineering (IJRSCSE)
Volume 2, Issue 10, December 2015, PP 1-12
ISSN 2349-4840 (Print) & ISSN 2349-4859 (Online)
www.arcjournals.org

Testing a Cloud Service Application Using Combinatorial Testing

Lakshmi Prasad Mudarakola[1], Venkata Laskhmama[2], T.Keerthana[3], T.Nirosha[4] S.Lokesh [5], P.V.BhanuPrakash[6]

[1] Dept. of Computer Science & Engineering, NBKRIST, Nellore, AP, India.
prasad.hinduniv@gmail.com

[2,3,4,5] Dept. of Computer Science & Engineering, NBKRIST, Nellore, AP, India.
venkatalakshmma6113@gmail.com[2], keerthanasanju24@gmail.com[3] , prabhas.lokesh@gmail.com[5] , mail2tnirosha@gmail.com[5] , bhanuprakash141996@gmail.com[6]

Abstract: *Cloud computing introduced many new software techniques as well as new issues. New requirement engineering processes, design techniques, code generation, and testing techniques need to be developed for cloud applications. Combinatorial testing is needed to test the cloud applications. Numerous numbers of combinatorial testing techniques are available, most of them produce static sequences of test configurations and their goal is often to provide sufficient coverage such as 2-way interaction coverage. But the goal of pairwise testing is to identify those compositions that are faulty for cloud applications. Currently, cloud computing has been applied to share computing resources to achieve coherence and economies of scale similar to a utility over a network. Testing-as-a-Service (TaaS) in a cloud environment can leverage the computation power provided by the cloud. Specifically, testing can be scaled to large and dynamic workloads, executed in a distributed environment with hundreds of thousands of processors, and these processors may support concurrent and distributed test execution and analysis. Pair-wise testing is a combinatorial test criterion technique based on specification, which requires that for each pair of parameters, every combination of their valid value should be covered by at least one test case in the test set. The generation of minimal pairwise test sets has been shown to be an NP-complete problem and there have been several deterministic algorithms published. Some of pairwise test techniques like IPOG, AETG, MIPOG, and ACO etc. are available to generate the pairwise test cases. The proposed approach is to generate the optimized pair-wise test cases for any cloud applications.*

Keywords: *Combinatorial testing, pairwise testing, AETG, IPO, IPOG, Automated Test case generation, Genetic algorithms, ACO, Practical swarm optimization.*

1. INTRODUCTION

Cloud computing introduced many new software techniques as well as new issues. New requirement engineering processes, design techniques, code generation, and testing techniques need to be developed for cloud applications. Combinatorial testing is needed to test the cloud applications. Numerous numbers of combinatorial testing techniques are available, most of them produce static sequences of test configurations and their goal is often to provide sufficient coverage such as 2-way interaction coverage. But the goal of pairwise testing is to identify those compositions that are faulty for cloud applications.

Currently, cloud computing has been applied to share computing resources to achieve coherence and economies of scale similar to a utility over a network. Testing-as-a-Service (TaaS) in a cloud environment can leverage the computation power provided by the cloud. Specifically, testing can be scaled to large and dynamic workloads, executed in a distributed environment with hundreds of thousands of processors, and these processors may support concurrent and distributed test execution and analysis.

Pair-wise testing is a combinatorial test criterion technique based on specification, which requires that for each pair of parameters, every combination of their valid value should be covered by at least one test case in the test set. The generation of minimal pairwise test sets has been shown to be an NP-complete problem and there have been several deterministic algorithms published. Some of pairwise test techniques like IPOG, AETG, MIPOG, and ACO etc. are available to generate the pairwise test

cases. The proposed approach is to generate the optimized pair-wise test cases for any cloud applications.

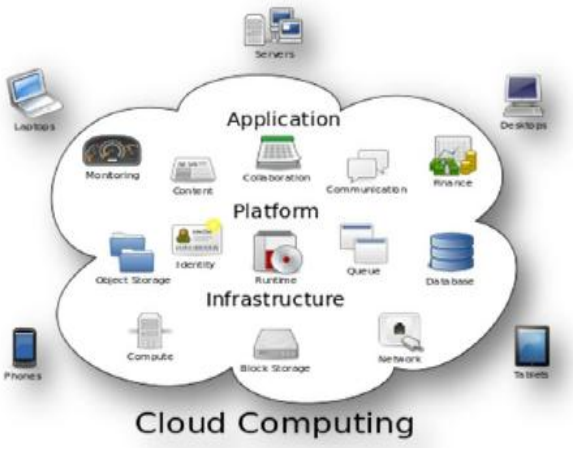

Fig: 1 *Structure of cloud computing*

(a) Architecture of cloud computing

The Cloud Computing architecture comprises of many cloud components, each of them is loosely coupled. We can broadly divide the cloud architecture into two parts: 1.Front End and 2.Back End.

Each of the ends is connected through a network, usually via Internet. The following diagram shows the graphical view of cloud computing architecture.

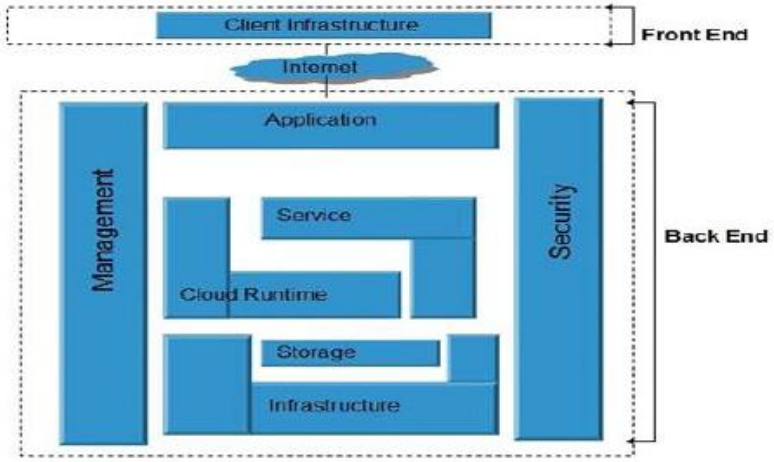

Fig: 2 *Architecture of cloud computing*

Front End: Front End refers to the client part of cloud computing system. It consists of interfaces and applications that are required to access the cloud computing platforms, e.g., Web Browser.

Back End: Back End refers to the cloud itself. It consists of all the resources required to provide cloud computing services. It comprises of huge data storage, virtual machines, security mechanism, services, deployment models, servers, etc. It is the responsibility of the back end to provide built-in security mechanism, traffic control and protocols.

The server employs certain protocols, known as middleware, helps the connected devices to communicate with each other.

(b) CLOUD DEPLOYMENT MODELS

Clouds can be classified in terms of who owns and manages the cloud. They are

1. Public Cloud
2. Private Cloud
3. Hybrid Cloud
4. Community Cloud

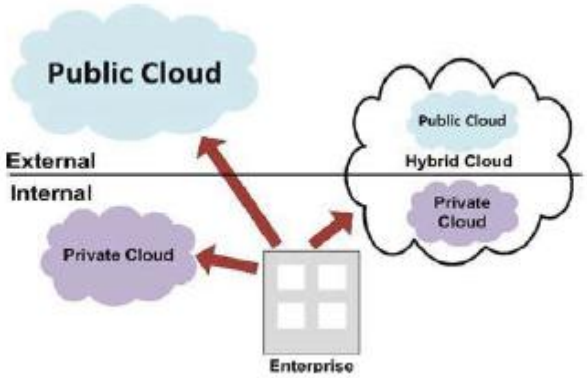

Fig: 3 *Types of Cloud*

1. Public Cloud: A public cloud, or external cloud, is the most common form of cloud computing, in which services are made available to the general public in a pay-as-you-go manner. Customers – individual users or enterprises – access these services over the internet from a third-party provider who may share computing resources with many customers. The public cloud model is widely accepted and adopted by many enterprises because the leading public cloud vendors as Amazon, Microsoft and Google, have equipped their infrastructure with a vast amount of data centres, enabling users to freely scale and shrink their rented resources with low cost and little management burden. Security and data governance are the main concern with this approach.

2. Private Cloud: A Private Cloud, or internal cloud, is used when the cloud infrastructure, proprietary network or data center, is operated solely for a business or organization, and serves customers within the business fire-wall. Most of the private clouds are large company or government departments who prefer to keep their data in a more Controlled and secure environment.

3. Hybrid Cloud: A composition of the two types (private and public) is called a Hybrid Cloud, where a private cloud is able to maintain high services availability by scaling up their system with externally provisioned resources from a public cloudwhen there are rapid workload fluctuations or hardware failures. In the Hybrid cloud, an enterprise can keep their critical data and applications within their firewall, while hosting the less critical ones on a public cloud.

4. Community Cloud: The idea of a Community Cloud is derived from the Grid Computing and Volunteer Computing paradigms. In a community cloud, several enterprises with similar requirement can share their infrastructures, thus increasing their scale while sharing the cost. Another form of community cloud may be established by creating a virtual data center from virtual machines instances deployed on underutilized users machines.

(c) CLOUD SERVICE MODELS

Cloud Computing comprises three different service models. They are

1. Infrastructure-as-a-Service (IaaS)
2. Platform-as-a-Service (PaaS)
3. Software-as-a-Service (SaaS)

1. Infrastructure-as-a-Service (IaaS): Infrastructure as a Service (IaaS) is one of the "Everything as a Service" trends. IaaS is easier to understand if we refer it as Hardware as a Service (i.e. instead of constructing our own server farms, a small firm could consider paying to use infrastructure provided by professional enterprises). Companies such as Google, Microsoft and IBM are involved in offering such services. Large-scale computer hardware and high computer network connectivity are essential components of an effective IaaS.

2. Platform-as-a-Service (PaaS): Platform as a Service (PaaS) cloud systems provide a software execution environment that application services can run on. The 5 environment is not just a pre-installed operating system but is also integrated with a programming-language-level platform, which users can be used to develop and build applications for the platform. From the point of view of PaaS clouds' users, computing resources are encapsulated into independent containers, they can develop their own applications with certain program languages, and APIs are supported by the container without having to take care of the resource management or allocation problems such as automatic scaling and load balancing.

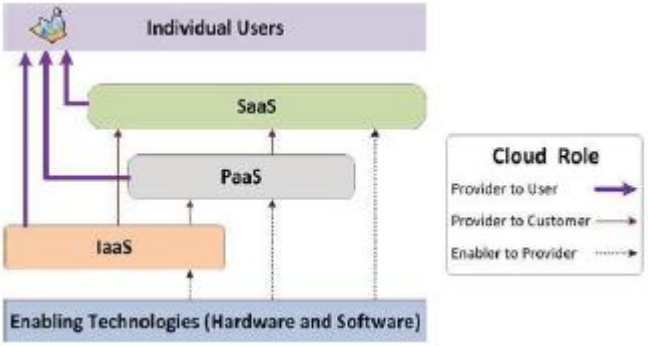

Fig 4: *Structure of Cloud Service Models*

3. Software-as-a-Service (SaaS): Software-as-a-Service (SaaS) is based on licensing software use on demand, which is already installed and running on a cloud platform. These on-demand applications may have been developed and deployed on the PaaS or IaaS layer of a cloud platform. SaaS replaces traditional software usage with a Subscribe/Rent model, reducing the user's physical equipment deployment and management costs. The SaaS clouds may also allow users to compose existing services to meet their requirements.

(d) TYPES OF COMBINATORIAL TESTING

Combinatorial testing is a specification based sampling technique that provides a systematic way to select combinations of program inputs or features for testing. It is an effective testing technique to test hardware/software that reveals failures in a given system based on input or output combinations. It has been applied over the years to test input data, configurations, web forms, protocols, graphical user interfaces, software product lines etc. The pair wise testing can detect possible t-way combinatorial interactions for t=2, 3, 4, 5, 6 or more.

(i)Each-used (also known **one-wise**) coverage is the simplest coverage criterion. 100% each-used coverage requires that every value of every parameter is included in at least one test case in the test suit.

(ii) Pair wise (also known **2-wise**) coverage requires that every possible pair of values of any two parameters is included in at least one test case in the test suite.

(iii)T-wise coverage is the natural extension of pair wise (2-wise) coverage, which requires every possible combination of values of t parameters be included in some test case in the test suit.

(e) MOTIVATION OF COMBINATORIAL TESTING

Pair wise testing (or 2-way testing) is a specification based testing criterion, which requires that for each pair of input parameters of a system, every combination of valid values of these two parameters be covered by at least one test case. Empirical results show that pairwise testing is practical and effective for various types of software systems. By seeing the graph we have found that there are more number of errors is covered by applying pair wise testing in different applications. Hence pair wise testing is to be used to apply in Cloud applications.

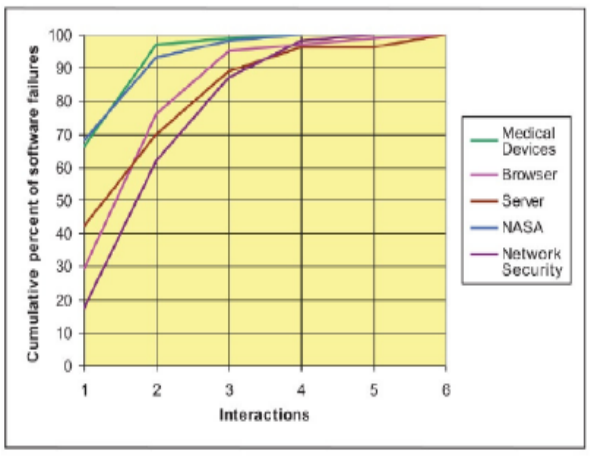

Fig: 5 *Percentage of failures triggered by t-way interactions*

2. SURVEY OF THE LITERATURE

Ahmed discussed TaaS and illustrated that the normal functions of manufacturing can be applied to commoditize software testing service to achieve consistent quality [1]. In 2010, Candea [5] discussed three applications for TaaS: a) a programmer's sidekick enabling developers to thoroughly and promptly test their code with minimal upfront resource investment; b) a home edition on-demand testing service for consumers. In 2011, a TaaS workflow process was introduced in [9]. In addition, the paper also reviewed TaaS systems. In 2011, Bai [3] provided a survey about recent results on testing service and tools in different aspects. It covered issues and research results in TaaS including TaaS infrastructure, test platforms, and environment. Yu proposed reference architecture of TaaS based on ontology, process automation, and SOA (Service-Oriented Architecture) [31]. This TaaS framework included 5-layers: a) test service tenant and contributor layer, b) test task management layer, c) testing resource management layer, d) test layer, and testing database layer. Their work focused on TaaS task management, scheduling, and processing. In addition, unit testing results were reported based on VM (Virtual Machine). Ciortea proposed a symbolic execution engine called Cloud9 that scales to large clusters of machines to support automated software testing [7]. It used symbolic execution to support on-demand testing.

Similar discussion was also given in [19]. Mathew [14] discussed the Apex test framework in the Force.com platform. This framework enabled developers to focus on testing without worrying about additional testing infrastructure. It provided built-in support for test creation and execution, so Apex tests can be used to automate unit tests and GUI-based functional tests with simulated user actions.

SaaS performance evaluation and scalability analysis require powerful simulation and environments. Buyya [4] discussed an extensible simulation toolkit (known as CloudSim) that enables modeling and simulation of cloud computing environments for computing resources in data centers. Tsai [23] discussed SaaS scalability testing. Gao [12] discussed SaaS performance testing and scalability, and presented a graph-based model to define a set of performance and scalability evaluation metrics. Moreover, SaaS performance validation approach was presented based on Amazon EC2 cloud technology with a case study report. Chen [6] proposed their scalability testing and analysis system (called STAS), and introduced a scalability metric (known as isospeed-e). SaaS testing is a new research topic [27], [10], [22]. Using policies and metadata, test cases can be generated to test SaaS applications. Testing can be embedded in the cloud platform in which tenant applications are run [27]. Gao proposed a framework for testing cloud applications [10], and proposed a measure for testing scalability. Another scalability measure was proposed in [22].

Software testing may be used to represent the variability in an expressive and practical way. Domain-specific languages, feature diagrams, and other modeling techniques can be used to express variability [20]. Furthermore, it may need to generate test cases automatically using a description of the variability to reveal faults effectively. Testing all combinations of inputs and/or configurations is infeasible in general [13], [15]. The number of defects in a software product can be large, and defects occurring infrequently are difficult to find [28]. Testing regimes balance the needs to generate tests quickly, to employ as few tests as possible, and to represent as many of the potential faults in tests as possible. *Combinatorial interaction testing* (*CIT*) ensures that every interaction among *t* or fewer elements is tested, for a specified *strength t*. *AETG* [8] and other algorithms are often used in combinatorial testing.

3. EXIXTING SYSTEM

A cloud often has three major components: IaaS (Infrastructure-as-a-Service), PaaS (Platform-as-a-Service), and SaaS (Software-as-a-Service). While different cloud environments have different techniques, most of them share features such as dynamic provisioning, metadata-driven computing, automated redundancy and recovery, automated migration with load balancing, Multi- Tenancy Architecture (MTA) [24] with tenant customization, virtualized computing, and databases such as NoSQL with Big Data analytics. These new features create new issues in testing applications running in a cloud environment.

• ***Testability:*** Application execution involves many parties including infrastructure providers, service providers, service consumers, and end users. Each party has limited visibility and controllability. For example, a SaaS system may not have direct control of scheduling and resource allocation if it runs on top of a PaaS.

• ***Test scale:*** Testing cloud-based applications requires scalability including scalable software architecture [23], scalable data storage and analysis, and scalable test execution and simulation.

• *Test environment:* Modern software needs to be tested in a variety of platforms including mobile devices and cloud platforms.

• ***Test process:*** In cloud computing, engineers may have access to service specifications such as APIs or WSDL only, and they have no direct control of dynamic scheduling, provisioning, and reliability mechanisms.

Testing-as-a-Service (TaaS) in a cloud environment can leverage the computation power provided by the cloud. Specifically, testing can be scaled to large and dynamic workloads, executed in a distributed environment with hundreds of thousands of processors, and these processors may support concurrent and distributed test execution and analysis. TaaS may be implemented as SaaS and used to test SaaS applications.

TaaS is important due to the following reasons:

1) **Cost-sharing of computing resources among production lines and teams**: This reduces the upfront costs and increases resource sharing and utilization.

2) **Scalable test environments**: This provides a scalable cloud-based test environment with auto-provision and de-provision using virtual and physical computing resources

3) **On-demand testing service in 365/7/24**: This enables TaaS vendors to offer diverse large-scale testing services online at anytime and anywhere.

4) **Pay-as-you-test**: This allows customers to receive on demand testing services using the pay-as-you-test model.

5) **Quality certification by third parties**: This leverages scalable cloud system infrastructures to test and evaluate SaaS and cloud-based applications as a third-party.

SaaS provides users flexibility to compose their own services. SaaS application with complicated functions can be composed by different basic services. Testing SaaS application is an extremely heavy work, due to the extremely large number of possible combinations. SaaS applications need to have reliability and availability before publishing.

Combinatorial testing technique can be used in SaaS application testing. Due to the large number of possible combinations, it is difficult to finish SaaS combinatorial testing in single machines. A cloud with large number of processors with distributed databases can be used to perform combinatorial testing. One simple way to perform combinatorial testing in a cloud environment is:

1) Partition the testing tasks;

2) Allocate these testing tasks to different processors in the cloud platform for test execution;

3) Collect results from those participating processors.

However, this is not optimal. While computing and storage resources have increased significantly, the number of combinations to be considered is still too high. Testing all of the combinations in a SaaS system with millions of components can consume all the resources of a cloud platform for millions of years. Two ways to improve this approach are both based on learning from previous test results:

Devise a mechanism to merge test results quickly, and detect any inconsistency in testing; Eliminate as many configurations as possible from future testing using existing testing results. With cloud computing, test results may arrive asynchronously and autonomously. This paper proposes a TaaS design that supports SaaS combinatorial testing works with Test Algebra (TA) [21], and Adaptive Reasoning (AR) [25]. TA and AR facilitate concurrent combinatorial testing.

TA: It uses existing test results to eliminate candidate configurations from testing without knowing how these results were obtained.

AR: It uses earlier test results to generate new test cases to detect faults in tenant applications.

Combinatorial testing technique can be used in SaaS application testing. Due to the large number of possible combinations, it is difficult to finish SaaS combinatorial testing in single machines. A cloud

with large number of processors with distributed databases can be used to perform combinatorial testing. One simple way to perform combinatorial testing in a cloud environment is:

1) Partition the testing tasks;

2) Allocate these testing tasks to different processors in the cloud platform for test execution;

3) Collect results from those participating processors.

However, this is not optimal. While computing and storage resources have increased significantly, the number of combinations to be considered is still too high. Testing all of the combinations in a SaaS system with millions of components can consume all the resources of a cloud platform for millions of years.

Two ways to improve this approach are both based on learning from previous test results:

- Devise a mechanism to merge test results quickly, and detect any inconsistency in testing;
- Eliminate as many configurations as possible from future testing using existing testing results.

4. PROPOSED SYSTEM

This paper proposes a TaaS design for SaaS combinatorial testing. Test Algebra (**TA**) and Adaptive Reasoning (**AR**) algorithm are used in the TaaS design. Several TaaS definitions are available [9], [19]. It often means that testing will be online, composable, Web-based, on demand, scalable, running in a virtualized and secure cloud environment with virtually unlimited computing, storage and networking. This paper proposes a TaaS definition from two perspectives: user's point of view and cloud internal point of view. From user's point of view, TaaS provides the following four services.

Test Case and Script Development: Users can develop, debug, and evaluate test cases/script online using automated tools in a collaborative manner. Test scripts may even be developed by customizing/composing existing components following the MTA approach. *Test Script Compilation and Deployment:* Test scripts can be compiled and deployed for execution in a cloud environment, and TaaS resource management can allocate and reclaim resources to meet the changing workload.

Test Script Execution: Test can be executed in parallel or in a distributed manner, and it can be triggered autonomously or on demand.

Test Result Evaluation: Cloud-based test database is built to support automated data saving, intelligent retrieval, concurrent transaction, parallel processing, and timely analysis of huge test results.

From cloud internal point of view, TaaS may have the following features common to most cloud operations.

Decentralized Operations: Testing tasks may be executed in a parallel or a distributed manner, migrated to dynamic allocated resources, and performed in a redundant manner, or embedded within other cloud operations.

Metadata-based Computing: Controller uses metadata to control test operations such as time, frequency, multitasking, redundancy, parallel execution. TaaS metadata may include information about test scripts, cases, environment, and results such as index, location, and organization.

Data-centric Testing: Big Test handles large sets of input data and produces large sets of test results. Techniques for Big Data storage, processing, and understanding are key to TaaS. For examples, test data can be saved in in-memory databases, classified by attributes (such as hot, warm, or cold), and analyzed in real-time.

Multi-tenancy Test Script Composition: Like tenant applications in a MTA SaaS platform, test scripts in a TaaS system may share the same test script base.

Automated Test Redundancy Management and Recovery: Testing tasks can be partitioned and sent to different processors for parallel and redundant processing. Test and test results can be recovered in case of failures in a processor or in a cluster due to automated redundancy management. Recovery can follow the metadata-based approach.

Automated Test Scalability: When the SUT (System Under Test) scales up at runtime in a cloud environment, TaaS also needs to scale up proportionally using common cloud scalability mechanisms such as 2-level scalability architecture and stateless service design [22].

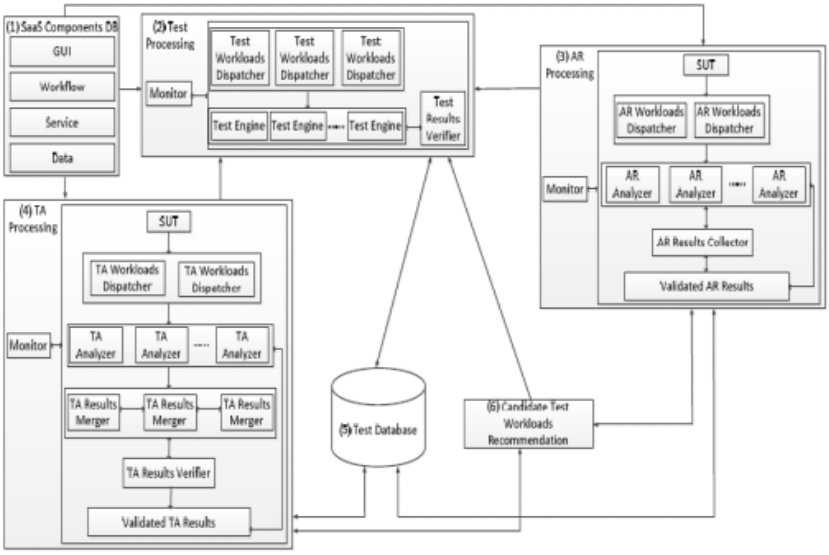

Fig 6: *Taas Design for Combinatorial Testing Using TA and AR*

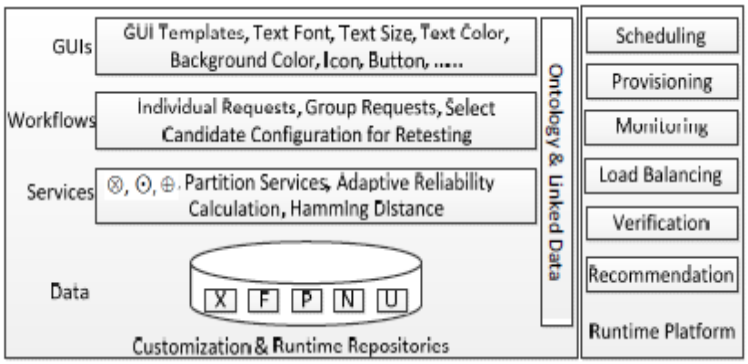

Fig 7: *TaaS Infrastructure*

1. SaaS Components DB: Each tenant application in SaaS has components from four layers: GUIs, workflows, services, and data.

2. Test Processing: It uses the following components to process SaaS combinatorial testing.

(a) Test Workloads Dispatcher*:* All testing workloads are sent to test dispatchers. Test dispatchers assign workloads to Test Engines according to the computation capacity of each Test Engine. The same workloads may be executed on different Test Engines for redundant testing.

(b) Test Engine*:* It runs different test cases to test the assigned workloads. Test results are sent to Test Results Verifier.

(c) Test Results Verifier*:* It verifies all returned test results. For the same configuration, it may have different returned test results from different Test Engines Test Result Verifiers finalizes the correct test result based on the confidence of each test result. Only those highly confident test results are saved in the Test Database and can be shared with others. If test results verifier cannot verify the returned test results, it requires Test Engines to retest these configurations.

(d) Monitor*:* It monitors the testing process. Test Workloads Dispatcher, Testing Workloads, and their related Test Engines are monitored. Each Test Engine is monitored during the testing process. Test Results Verifier is also monitored.

3. AR Processing: It is used to figure out faulty configurations from the candidate set rapidly based on the existing test results.

SUT: It is the candidate test set.

(a) AR Workloads Dispatcher: It works similarly as the Test Workloads Dispatcher of Test Processing. Different amount of candidate testing workloads are assigned to different AR Analyzers based on the computation capacity.

(b) AR Analyzer: It runs AR algorithm on candidate configurations based on the existing test results. The analyzed test results are sent to the collector. It also reanalyzes those returned incorrect test results that did not pass validation.

(c) AR Results Collector: It collects all test results from different AR Analyzers. Collector also sends those candidate configurations that cannot pass test results validation to their related AR analyzers.

(d) Validated AR Results: They save all validated AR results and send them to the shared test database. The saved validated results are shared to all AR analyzers.

(e) Monitor: It is similar as monitor of Test Processing. The process of AR Analysis is monitored.

(f) **TA Processing**: It analyzes test results by TA .Similar to AR Processing; TA also has SUT, test dispatcher, and monitor. Their functions are same as the corresponding parts in AR. The other parts of TA have their own features.

(i)TA Analyzer: It runs TA to analyze the test results of candidate test set based on the existing test

results. Test results of those candidate configurations related to existing X or F interactions can be

finalized.

(ii) TA Results Merger: It merges the returned from different TA analyzers by three defined

operations. The merged test results are sent to test result verifier.

(iii)TA Results Verifier: It verifies all returned test results. Usually test results with high confidence

are treated as correct test results. Those test results that cannot be verified are sent back to TA

analyzer for re-analyzing.

(iv) Validated TA Results: They save and share all validated test results. The validated test results are categorized according the number of components.

4. Test Database: It not only saves test results from Testing Processing, but also saves analyzed test results from AR and TA. Only validated test results can be saved in Test Database. All saved test results are shared and can be reused. Different from traditional databases, the saved test results are categorized by type and the number of components. For instance, 2-way and 3-way F configurations are saved in its own table respectively. Due to the large number of test results, only the roots of X, and F configurations are saved in test database. For example, configuration (a, b, c, d, and e) is F and configuration (a, b, c) is the faulty root, so only configuration (a, b, c) is saved in F data table. Test results of those configurations that contain configuration (a, b, c) are automatically considered as fault.

5. Candidate Test Workloads Recommendation: It is used to figure out those priority configurations for testing. Based on the existing test results, it recommends those potential faults in the candidate set. Those configurations in candidate set that have one or two hamming distance between existing faulty configurations are 22 recommended for TA and AR. TA, AR and Recommendation system communicate often. TA and AR send their analyzed test results to Recommendation system. Recommendation system sends related candidate configurations to them. Comparing TA and AR, the communication between Test Engine and Recommendation system is one-way direction. Only Recommendation system sends candidate configurations to Test Engine. The parent sets of faulty configurations found by AR are recommended to Test Engine for testing.

5. RESULTS

A group of simulations have been performed, and this section provides one SaaS example for testing. The SaaS has four layers, and each layer has five components, and each component has two options as the initial settings. When the current workloads are finished reaching to 20%, one new component is

added to each layer until each layer has ten components. The experiments are done for t-way configurations for $2 \leq t \leq 6$. The initial settings of infeasible, faulty, and irrelevant configurations are shown in Table 7.1.1. The number of candidate configurations from five components to ten components each layer are also shown in Table 1.

There are six attempts in total from five components to ten components of each layer. Figure 7.1.1 shows the number of VMs used in each attempt. When the number of components in each layer increases, the trend is that more VMs are required to process the workloads. The average computation time of each VM takes each attempt. Figure 7.1.2 shows the average computation time of each VM in each attempt. The computation time is counted in seconds. Similarly, when the workloads increase, more execution time of each VM spends. Based on the proposed TaaS design, when workloads increase, the current working mechanism can be extended. More VMs are added, including test engines, TA Analyzers, AR Analyzers. TaaS scalability issues involving redundancy and recovery, and data migration can be solved in the proposed design. The returned test results from each VM can be shared to other VMs through the current test results sharing mechanism.

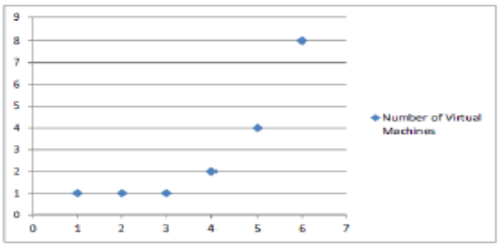

Fig 8: *The Number of virtual machines on each attempt*

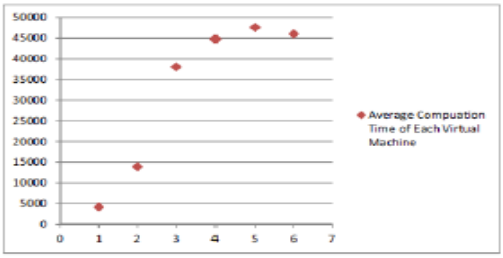

Fig 9: *Average computation time of each virtual machine of each attempt*

For example, consider a software system has two input programs, P1, P2. Each parameter contains 3 programs. Let the First input 1.e P1 contains 3 programs i.e Addition1.java Addition2.java Addition3.java which is treated as a0 a1 a2. The second input i.e P2 contains 3 programs i.e ReverseString1.java ReverseString2.java ReverseString3.java which is treated as b0 b1 b2. For this case there are a total of 3 * 3 = 9 combinations of input values. The combination of ordered pairs is as shown below. {a0, b0}, {a0, b1}, {a0, b2}, {a1, b0}, {a1, b1}, {a1, b2}, {a2, b0}, {a2, b1}, {a2, b2}. According to techniques of combinatorial testing techniques like In parameter order strategy the test vectors are shown below.

T1: a0 b0 T2: a0 b1 T3: a0 b2 T4: a1 b0 T5: a1 b1 T6: a1 b2

Table 1. *The Initial settings of Configuration*

Configuration	Size								
Type	Infeasible	Faulty	Irrelevant	5-Component	6-Component	7-Component	8-Component	9-Component	10-Component
2-way	5	15	20	760	1,104	1,512	1,984	2,520	3,120
3-way	50	5	200	9,120	16,192	26,208	39,680	57,120	79,040
4-way	500	0	2,000	77,520	170,016	327,600	575,360	942,480	1,462,240
5-way	5,000	0	20,000	496,128	1,360,128	3,144,960	6,444,032	12,063,744	21,056,256
6-way	50,000	1	200,000	2,480,640	8,614,144	24,111,360	57,996,288	124,658,688	245,656,320

6. CONCLUSION & FUTURE WORK

In this project a cloud application is implemented and tested without errors with the help of pair wise testing technique. This project talks about TaaS architecture and design. New issues introduced by cloud are discussed and three generations of TaaS are proposed. A TaaS framework has been proposed. TaaS as one type of SaaS can be used to test SaaS. This paper illustrates the process of using TaaS to test SaaS. Hence, it is concluded that a pairwise test generation strategy is proposed to provide the test cases for a cloud application. When used properly, pair wise testing is an important

technique that can help you better to test the cloud applications. Now a days cloud computing plays an important role. Many Number of cloud applications has been designed .Hence, the future of cloud computing seems very promising and so bright. Testing the cloud application is also a big task to remove bugs. So in future there may be a chance for applying different types of combinatorial testing techniques.

REFERENCES

[1]. Ahmed. ***Software Testing as a Service****, 1st edition.* Auerbach Publications, September, 2009.

[2]. X. Bai, M. Li, B. Chen, W.-T. Tsai, and J. Gao. ***Cloud Testing Tools***. In *Proceedings of IEEE 6th International Symposium on Service Oriented System Engineering (SOSE)*, pages 1–12, Irvine, CA, USA, 2011.

[3]. R. Buyya, R. Ranjan, and R. N. Calheiros. ***Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities***. *CoRR*, abs/0907.4878, 2009.

[4]. G. Candea, S. Bucur, and C. Zamfir. ***Automated software testing as a service***. In *Proceedings of SoCC*, pages 155– 160, 2010.

[5]. Y. Chen and X. Sun. ***Stas: A scalability testing and analysis system***. In *2006 IEEE International Conference on Cluster Computing*, pages 1–10, 2006.

[6]. L. Ciortea, C. Zamfir, S. Bucur, V. Chipounov, and G. Candea. ***Cloud9: a software testing service***. *Operating Systems Review*, 43(4):5–10, 2009.

[7]. D. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. ***The AETG System: An Approach to Testing Based on Combinatorial Design.*** *Journal of IEEE Transactions on Software Engineering*, 23:437–444, 1997.

[8]. J. Gao, X. Bai, and W. T. Tsai. ***Cloud Testing-Issues, Challenges, Needs and Practice***. In *Software Engineering:An International Journal (SEIJ), IGI Global*, volume 1, pages 9–23, 2011.

[9]. J. Gao, X. Bai, and W.-T. Tsai. ***SaaS : Performance and Scalability Evaluation in Cloud.*** In *Proceedings of The 6th IEEE International Symposium on Service Oriented System Engineering*, SOSE '11, 2011.

[10]. J. Gao, K. Manjula, P. Roopa, E. Sumalatha, X. Bai, W.- T. Tsai, and T. Uehara. ***A cloud-based taas infrastructure with tools for saas validation, performance and scalability evaluation.*** In *Proceedings of CloudCom*, pages 464–471, 2012.

[11]. J. Gao, P. Pattabhiraman, X. Bai, and W. T. Tsai. ***Saas performance and scalability evaluation in clouds***. *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, 0:61–71, 2011.

[12]. C. Kaner, J. Falk, and H. Q. Nguyen. ***Testing Computer Software,*** *2nd Edition*. Wiley, New York, NY, USA, 1999.

[13]. R. Mathew and R. Spraetz. ***Test automation on a saas platform***. In *Proceedings of 2009 International Conference on Software Testing Verification and Validation, (ICST 09)*, pages 317–325, 2009.

[14]. T. Muller and D. Friedenberg. Certified Tester Foundation Level Syllabus. *Journal of International Software Testing Qualifications Board*.

[15]. T. Parveen and S. R. Tilley. ***When to migrate software testing to the cloud?*** In *Proceedings of ICST Workshops*, pages 424–427, 2010.

[16]. B. P. Rimal, E. Choi, and I. Lumb. ***A Taxonomy and Survey of Cloud Computing Systems***. In *Proceedings of 5th International Joint Conference INC, IMS and IDCNCM*, pages 44–51, 2009.

[17]. L. M. Riungu, O. Taipale, and K. Smolander. ***Proceedings of research issues for software testing in the cloud.*** In *CloudCom*, pages 557–564, 2010.

[18]. L. M. Riungu, O. Taipale, and K. Smolander. ***Software testing as an online service: Observations from practice.*** In *Proceedings of ICST Workshops*, pages 418–423, 2010.

[19]. M. Sinnema and S. Deelstra. ***Classifying Variability Modeling Techniques.*** *Inf. Softw. Technol.*, 49(7):717–739, July 2007.

[20]. W.-T. Tsai, C. Colbourn, J. Luo, G. Qi, Q. Li, and X. Bai**.** ***Test Algebra for Combinatorial Testing***. In *Proceedings of the 2013 8th International Workshop on Automation of Software Test (AST)*, pages 19–25, 2013.

[21]. W.-T. Tsai, Y. Huang, X. Bai, and J. Gao. ***Scalable Architecture for SaaS.*** In *Proceedings of 15th IEEE International Symposium on Object Component Service-oriented Real-time Distributed Computing*, ISORC '12, Apr. 2012.

[22]. W.-T. Tsai, Y. Huang, and Q. Shao. ***Testing the Scalability of SaaS Applications***. In *Proceedings of IEEE InternationalConference on Service-Oriented Computing and Applications (SOCA)*, pages 1–4, Irvine, CA, USA, 2011.

[23]. W.-T. Tsai, Y. Huang, Q. Shao, and X. Bai. ***Data Partitioning and Redundancy Management for Robust Multi-Tenancy SaaS***. *International Journal of Software and Informatics(IJSI)*, 4(3):437–471, 2010.

[24]. W.-T. Tsai, Q. Li, C. J. Colbourn, and X. Bai**.** ***Adaptive Fault Detection for Testing Tenant Applications in Multi- Tenancy SaaS Systems.*** In *Proceedings of IEEE InternationalConference on Cloud Engineering (IC2E)*, March 2013.

[25]. W.-T. Tsai, J. Luo, G. Qi, and W. Wu. ***Concurrent Test Algebra Execution with Combinatorial Testing***. In *Proceedings of 8th IEEE International Symposium on Service-Oriented System Engineering (SOSE2014)*, 2014.

[26]. W.-T. Tsai, Q. Shao, and W. Li. ***OIC: Ontology-based Intelligent Customization Framework for SaaS***. In *Proceedings of International Conference on Service Oriented Computing and Applications(SOCA'10)*, Perth, Australia, Dec. 2010.

[27]. Wikipedia. Software Testing, 2013.

[28]. W. Wu, W.-T. Tsai, C. Jin, G. Qi, and J. Luo. ***Test-Algebra Execution in a Cloud Environment***. In *Proceedings of 8th IEEE International Symposium on Service-Oriented System Engineering (SOSE2014)*, 2014.

[29]. Y. ***Yang, C. G. Onita, J. S. Dhaliwal, and X. Zhang. Testqual***: Conceptualizing software testing as a service. In *Proceedings of AMCIS*, page 608, 2009.

[30]. L. Yu, W.-T. Tsai, X. Chen, L. Liu, Y. Zhao, L. Tang, and W. Zhao. ***Testing as a service over cloud***. In *Proceedings of SOSE*, pages 181–188, 2010.