

Chapter 2

Combinatorial Testing in Cloud Computing

Abstract This chapter gives an introduction to combinatorial testing. In particular, it describes the applications and challenges of combinatorial testing in cloud environment and briefly introduces solutions to address challenges. It also discusses and compares existing faulty location analysis solutions of combinatorial testing.

2.1 Combinatorial Testing in Cloud Computing

Cloud computing plays an important role today, as a new computing infrastructure to enable rapid delivery of computing resources as a utility in a dynamic, scalable, and visualized manner. Software-as-a-Service (SaaS), as a part of cloud computing among Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS), is a new software delivery model designed for Internet-based services. One single code is designed to run for different tenants. SaaS provides frequent upgrades to minimize customer disruption and enhance satisfaction. For maintenance, fixing one problem for one tenant also fixes it for all other tenants.

SaaS supports *customization*: Tenant applications are formed by composing components in the SaaS database [1, 10, 11] such as GUI, workflow, service, and data components. SaaS supports *multi-tenancy architecture (MTA)*: One code base is used to develop multiple tenant applications, so that each tenant application is a customization of the base code [9]. SaaS often also supports *scalability*, as it can supply additional computing resources when the workload is heavy.

Once tenant applications are composed, they need to be tested. However, a SaaS system can have millions of components and hundreds of thousands of tenant applications. New tenant applications are added continuously, while other tenant applications are running on the SaaS platform. New tenant applications can cause new components to be added to the SaaS system.

Combinatorial testing [3] is a popular testing technique to test a component-based application. It tests interactions among components in the configuration, assuming that each component has been tested individually. A *t-way interaction* is one that involves *t* components, and *t-way coverage* in a test suite means that every *t*-way interaction appears in at least one test configuration. Traditional combinatorial testing

techniques focus on tests to detect the presence of faults, but fault location is an active research area. Each configuration needs to be tested, as each configuration represents a tenant application. Traditional combinatorial testing methods, such as *AETG* [4], can reveal the existence of faults by using few test cases to support t -way coverage for $t \geq 2$. But knowing the existence of a fault does not indicate which t -way interactions are faulty. When the problem size is small, an engineer can identify faults by knowing which test configurations contain a fault. However, when the problem is large, it is difficult or even impossible to identify faults if the test suite only ensures t -way coverage.

2.2 Improvements of Combinatorial Testing in Cloud Environment

The movement to Big Data and cloud computing can make hundreds of thousands of processors available. Potentially, a large number of processors with distributed databases can be used to perform large-scale combinatorial testing. Indeed, these provide significant computing power that was not available before; for example, they support concurrent and asynchronous computing mechanisms such as MapReduce, automated redundancy and recovery management, automated resource provisioning, and automated migration for scalability [8]. 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 the processors.

However, this is not efficient. 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. 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 necessitates a new testing framework. This book proposes a new algebraic system, Test Algebra (TA) [12], to facilitate concurrent combinatorial testing. The key feature of TA is that the algebraic rules follow the combinatorial structure and thus can track the test results obtained. The TA can then be used to determine whether a tenant application is faulty and which interactions need to be tested. The TA is an algebraic system in which elements and operations are formally defined. Each

element represents a unique component in the SaaS system, and a set of components represents a tenant application. Assuming each component has been tested by developers, testing a tenant application is equivalent to ensuring that there is no t -way interaction faults for $t \geq 2$ among the elements in a set.

The TA uses the principle that if a t -way interaction is faulty, every $(t + 1)$ -way interaction that contains the t -way interaction as a subset is necessarily faulty. The TA provides guidance for the testing process based on test results so far. Each new test result may indicate if additional tests are needed to test a specific configuration. The TA is an algebraic system, primarily intended to track the test results without knowing how these results were obtained. Specifically, it does not record the execution sequence of previously executed test cases. Because of this, it is possible to allocate different configurations to different processors for execution in parallel or in any order, and the test results are merged following the TA rules. The execution order and the merge order do not affect the merged results if the merging follows the TA operation rules.

2.3 Faulty Location Analysis in Combinatorial Testing

There are some solutions of fault location analysis in combinatorial testing. Four of them are picked for demonstration in the following paragraphs.

2.3.1 *Fault Localization Based on Failure-Inducing Combinations*

A spectrum-based approach to fault localization leverages the notion of an inducing combination with two novelties [6].

1. Traditional fault localization solutions only focus on the identification of inducing combinations. The proposed solution is the first effort to perform code-based fault localization based on combinatorial testing.
2. Existing spectrum-based approaches do not deal with the problem of test generation rather than assume the existence of a large number of tests that are generated randomly and/or by other methods. The proposed solution uses a systematic manner to generate a small group of tests from an inducing combination. The faults can be quickly located by analyzing the execution traces of these tests.

The idea of proposed test generation solution comes from nearest neighbor that faulty statements are likely to appear in the execution trace of a failed test but not in the execution trace of a passed test that is as similar to this failed test as possible. As the core member, one test in the group consists of the inducing combination and produces a failed test execution. As the derived member from the core member, other tests in the group execute a trace that is similar to the trace of the core member, but

produce a different outcome, such as a pass execution. The spectrums of the core member and each derived member are compared to a ranking of statements in terms of their likelihood to be faulty.

There are two steps of the proposed solution.

1. *Test Generation*: It generates a group of tests. This group consists of one failed test, which is referred to as the core member, and at most t passed tests, which are referred to as the derived members. Each derived member is expected to produce a similar trace as the core member.
2. *Rank Generation*: It compares the spectrum of the core member to the spectrum of each derived member and then produces a ranking of statements in terms of their likelihood of being faulty.

The quality of top-ranked suspicious combinations affects the effectiveness of proposed solution. A suspicious combination is ranked based on the suspiciousness value of each component in the subject program. However, the top-ranked suspicious combination obtained in this way may not be a truly inducing combination.

If the generated core member does not fail, it has to pick a failed test from the initial test set as the core member. The picked failed test contains this top-ranked combination, but may not minimize the suspiciousness of its environment. This may reduce the probability for the derived members to pass.

According to the core member, the corresponding derived members are generated. The derived members pass tests that have a similar trace to the core member. If a derived member fails, it is discarded. If all the derived members fail, it has to pick a passed test from the initial test set that is similar to the core member as possible.

2.3.2 Identifying Failure-Inducing Combinations in a Combinatorial Test Set

An approach was proposed to identify failure-inducing combinations that have caused some tests to fail [5]. Given a t -way test set, it first identifies and ranks a set of suspicious combinations, which are candidates that are likely to be failure-inducing combinations. Next, it generates a set of new tests, which can be executed to refine the ranking of suspicious combinations in the next iteration. This process can be repeated until a stopping condition is satisfied.

While the proposed approach focuses on analyzing t -way combinations, it guarantees to identify inducing combinations involving no more than t parameters to be suspicious combinations. Let c be an inducing combination; it considers the following two cases.

- Case (1): c is a t -way combination. As the initial test set is a t -way test set, there is at least one test that contains c , and all test cases containing c must fail, since c is inducing. Therefore, c is identified to be a suspicious combination by the proposed approach.

- Case (2): The size of c is less than t . All t -way combinations containing c are inducing combinations and are identified to be suspicious combinations. Hence, the reduction step identifies c as a suspicious combination.

When an inducing combination involves more than t parameters, it may not appear in the initial t -way test set, and the proposed algorithm does not identify it to be a suspicious combination. The proposed approach is by nature heuristic. On the one hand, suspicious combinations that are ranked top by the proposed approach may not be truly inducing. On the other hand, truly inducing combinations may not be ranked top by the proposed approach.

The proposed approach is different from these previous techniques in that it tries to identify inducing combinations in a combinatorial test set, instead of a single failed test. On one hand, a test set contains more information than a single test. On the other hand, doing so makes it possible to identify inducing combinations that appear in different tests. Moreover, the assumption made by FIC and FIC_BS may not hold for many applications, as changing a value in a test introduces many new combinations, and assuming that all of them are non-inducing is overoptimistic.

2.3.3 *Faulty Interaction Identification via Constraint Solving and Optimization*

The proposed solution is an automated test result analysis technique for faulty combinatorial interactions (FCIs) based on pseudo-Boolean constraint solving and optimization techniques [14]. It uses the test results of combinatorial test suite as input and does not generate additional test cases.

The proposed approach can identify all possible solutions for the combinatorial test suite. The number of possible solutions can be used as a criterion to measure the precision of solutions. A solution is a set of suspicious FCIs. The FCI identification problem as a constraint satisfaction problem (CSP) or satisfiability (SAT) problem is formulated. It solves the formulated CSP or SAT problem to obtain corresponding FCI solutions. There are some variables in a CSP. Each of them can take values from a certain domain, and there are also some constraints. To solve a CSP, it needs to find a suitable value in the domain for each variable, such that all the constraints hold. A failing test case should match the FCI vector, and a passing test case should not match it.

This paper first assumes that there is only one failing test case in the test suite. It identifies the value combination (i.e., FCI) in this test case that can be represented by a vector like this: $\langle x_1, x_2, \dots, x_k \rangle$, where each x_i can be 0 or a value in the set D_i . When $x_j = 0$, it means that the j th attribute does not appear in the FCI. After solving the CSP, it gets the values of the variables x_i ($1 \leq i \leq k$). Then, deleting the 0s, it gets the FCI.

Since there may be many FCI solutions of the constraint satisfaction problem, it is desirable to find the optimal one according to some criterion. The minimization

of FCI size is formulated as a pseudo-Boolean optimization (PBO) problem. In this case, it needs to maximize the total number of zero values of all FCIs. It follows the objective function: Minimize $\sum_i \sum_j P_{i,j,0}$ to meet passing and failing constraints.

The test suite may not be sufficient to provide enough information about the failure all the time. When the test suite is insufficient, there may be a number of possible solutions for the generated constraints. In this case, the localized FCIs may not be unique. The more possible solutions are obtained, the lower precision of each possible solution is, and the lower accuracy of FCI localization is. The proposed approach provides evidence of the trade-off between reducing the size of the test suite and enhancing its fault localization ability.

2.3.4 *Characterizing Failure-Causing Parameter Interactions by Adaptive Testing*

Zhang introduced a new fault characterization method called faulty interaction characterization (FIC) and its binary search alternative FIC_BS to locate one faulty interaction in a single failing test case, based on the following four assumptions [15].

- Assumption 1: The outcome of executing a test case on the software under test (SUT) is either pass or fail.
- Assumption 2: Test cases matching a faulty interaction must fail.
- Assumption 3: All parameters are independent.
- Assumption 4: All faulty interactions that appear during FIC are faulty interactions of Vs.

This paper provides a trade-off strategy of locating multiple faulty interactions in one test case. It supposes there is a set of parameters U. It uses each failing test case as a seed test case Vs. Then, it adaptively generates and executes some new test cases by modifying parameter values of Vs in U to locate the faulty interactions of the seed test case. All faulty interactions containing any parameters in U will be deactivated, and the rest faulty interactions are still active. For locating a t-way faulty interaction in the seed test case, the number of adaptive test cases is at most k (for FIC) or $t(\lceil \log_2 k \rceil + 1) + 1$ (for FIC_BS), where k is the number of parameters.

Comparing to most existing methods, the proposed solution has stronger or equivalent ability of locating faulty interactions and needs smaller number of adaptive test cases for locating randomly generated faulty interactions. The proposed solution is based on the assumptions. However, those assumptions do not always hold in many practical applications. The proposed solution tries to weaken the assumptions.

- Weakening Assumption 1: In some applications, the SUT fails with multiple failure types. A solution is to use the appearance of a specific failure type instead of pass and fail.

Table 2.1 Comparisons between proposed solution and existing solutions

	Proposed AR+TA	Fault localization based on failure-inducing combinations	Identifying failure-inducing combinations in a combinatorial test set	Faulty interaction identification via constraint solving and optimization	Characterizing failure-causing parameter interactions by adaptive testing
Contribution	(1) Compare existing test results to identify the faulty roots by AR (2) Propagate the identified faulty roots to eliminate potential faults	(1) Code-based fault localization (2) Systematic manner to generate a small group of tests from an inducing combination	Identify inducing combinations in a combinatorial test set, instead of a single failed test	Identify all possible solutions for the combinatorial test suite. The number of possible solutions can be used as a criterion to measure the precision of solutions	A new fault characterization method called faulty interaction characterization (FIC) and its binary search alternative FIC_BS to locate one faulty interaction in a single failing test case
Weakness	The integrated test analysis framework depends on the existing test results. AREfficiency depends on the number of passing test results.	The quality of top-ranked suspicious combinations affects the effectiveness of proposed solution.	It is nature heuristic. On the one hand, suspicious combinations that are ranked top by the proposed approach may not be truly inducing. On the other hand, truly inducing combinations may not be ranked top by the proposed approach.	The test suite may not be sufficient to provide enough information about the failure all the time.	Based on four assumptions
Step	(1) AR analysis on existing test results (2) TA analysis	(1) Test generation (2) Rank generation	(1) Suspicious combination identification and ranking (2) Test generation	(1) Formulate FCIs as CSP or SAT problem (2) Solve the formulated CSP or SAT problem	(1) Use each faulty test case as a seed (2) Generate and execute new test cases to locate faulty interactions in seed
Feature	Asynchronous, decentralized, concurrent	Code-based, suspicious combinations' ranking	Suspicious combinations' ranking	Optimization, CSP/SAT problem solving	Binary search alternative FIC_BS

- Weakening Assumption 2: If this assumption does not hold and a test case matching one or more faulty interactions passes by coincidence, then several irrelevant parameters will be decided as fixed parameters of the faulty interaction under locating, so that FIC and FIC_BS can locate a superinteraction of the real faulty interaction.
- Weakening Assumption 3: Many applications have dependent parameters. A solution is to label all invalid test cases with pass. Similar to weakening Assumption 2, several irrelevant parameters will be decided as fixed parameters. So FIC and FIC_BS can locate a superinteraction of the real faulty interaction in this condition.
- Weakening Assumption 4: The concept of safe values is introduced. A safe value for parameter v_i is a value $s_{safe,i} \in \{1, 2, \dots, s_i\}$ such that no-faulty interaction of the SUT contains the fixed parameter v_i with $v_i = s_{safe,i}$.

2.3.5 Comparisons of Existing Faulty Location Analysis Solutions

The faulty location analysis is an interesting and meaningful research topic in combinatorial testing. Existing solutions focus on different features of faults and explore the potential faulty locations. Table 2.1 shows the comparisons between proposed solution and existing solutions.

2.4 Related Work

Test results are used to isolate the faulty combinations that cause the software under test to fail. Effective classification can increase efficiency [7]: The faulty combinations in scenarios where failures are not commonly observed are classified. Test augmentation and feature selection can be used to enhance classification.

ACTS (Advanced Combinatorial Testing System), a combinatorial test generation research tool, supports t-way combinatorial test generation with several advanced features such as mixed-strength test generation and constraint handling [2, 13].

Most of existing CT methods use different strategies to generate test cases and only identify faulty configurations, but do not exploit the faulty root of each configuration. Those existing solutions of faulty location analysis have different assumptions and constraints. Our methods do not rely on whether random, anti-random, combinatorial interaction, or another type of combination-based test suite generation is used. We focus on the task of large-scale distributed testing, analyzing, merging, and maintaining test results in order to reduce the amount of testing needed.

References

1. X. Bai, M. Li, B. Chen, W.-T. Tsai, J. Gao, Cloud testing tools, in *Proceedings of IEEE 6th International Symposium on Service Oriented System Engineering (SOSE)* (Irvine, CA, USA, 2011), pp. 1–12
2. M. Borazjany, L. Yu, Y. Lei, R. Kacker, R. Kuhn, Combinatorial testing of acts: a case study, in *Proceedings of 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST)* (2012), pp. 591–600
3. R.C. Bryce, Y. Lei, D.R. Kuhn, R. Kacker, Combinatorial testing, in *Handbook of Software Engineering Research and Productivity Technologies* (2010)
4. D. Cohen, S.R. Dalal, M.L. Fredman, G.C. Patton, The AETG system: an approach to testing based on combinatorial design. *J. IEEE Trans. Softw. Eng.* **23**, 437–444 (1997)
5. L.S.G. Ghandehari, Y. Lei, T. Xie, R. Kuhn, R. Kacker, Identifying failure-inducing combinations in a combinatorial test set, in *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, ICST '12* (2012), pp. 370–379
6. L.S.G. Ghandehari, Y. Lei, D.C. Kung, R. Kacker, D.R. Kuhn, Fault localization based on failure-inducing combinations, in *IEEE 24th International Symposium on Software Reliability Engineering, ISSRE 2013*, November 4–7 (Pasadena, CA, USA, 2013)
7. K. Shakya, T. Xie, N. Li, Y. Lei, R. Kacker, R. Kuhn, Isolating failure-inducing combinations in combinatorial testing using test augmentation and classification, in *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, ICST '12* (IEEE Computer Society, Washington, DC, USA, 2012), pp. 620–623. ISBN 978-0-7695-4670-4
8. W. Tsai, G. Qi, Z. Zhu, Scalable saas indexing algorithms with automated redundancy and recovery management. *Int. J. Softw. Inf.* **7**(1), 63–84 (2013a)
9. W.-T. Tsai, Y. Huang, Q. Shao, X. Bai, Data partitioning and redundancy management for robust multi-tenancy SaaS. *Int. J. Softw. Inf. (IJSI)* **4**(3), 437–471 (2010a)
10. W.-T. Tsai, Q. Shao, 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, 2010b)
11. W.-T. Tsai, Y. Huang, Q. Shao, Testing the scalability of SaaS applications, in *Proceedings of IEEE International Conference on Service-Oriented Computing and Applications (SOCA)* (Irvine, CA, USA, 2011), pp. 1–4
12. W.-T. Tsai, C. Colbourn, J. Luo, G. Qi, Q. Li, X. Bai, Test algebra for combinatorial testing, in *Proceedings of the 2013 8th International Workshop on Automation of Software Test (AST)* (2013b), pp. 19–25
13. L. Yu, Y. Lei, R. Kacker, D. Kuhn, ACTS: a combinatorial test generation tool, in *Proceedings of 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation (ICST)* (2013), pp. 370–375
14. J. Zhang, F. Ma, Z. Zhang, Faulty interaction identification via constraint solving and optimization, in *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing, SAT'12* (Springer, New York, 2012), pp. 186–199
15. Z. Zhang, J. Zhang, Characterizing failure-causing parameter interactions by adaptive testing, in *Proceedings of the 2011 International Symposium on Software Testing and Analysis, ISSA '11* (ACM, New York, NY, USA, 2011), pp. 331–341

Combinatorial Testing in Cloud Computing

Tsai, W.-T.; Qi, G.

2017, X, 128 p. 55 illus., 37 illus. in color., Softcover

ISBN: 978-981-10-4480-9