W.T Tsai, X. Bai, Y. Chen, X. Zhou, "Web Service Group Testing with Windowing Mechanisms," IEEE International Workshop on Service-Oriented System Engineering (SOSE), Beijing October 2005, 213-218.

Web Service Group Testing with Windowing Mechanisms

Wei-Tek Tsai, Xiaoying Bai*, Yinong Chen, Xinyu Zhou
Computer Science & Engineering Department, Arizona State University, USA
**Department of Computer Science and Engineering, Tsinghua University, China*

Abstract

ASTRAR provides a framework for testing Web Services (WS) using the group testing technique. This paper extends the basic two-phase testing process and introduces the windowing mechanism to further improve testing efficiency. Rather than testing a large number of WS simultaneously, WS are divided into subsets called windows and testing is exercised window by window. Testing results are analyzed for different strategies such as using all of the historical data, using the most recent windows, and using the current window only. Based on the results, test cases are ranked according to their potency to detect faults; and oracles and the confidence level of each oracle are established for individual test cases at run-time. In addition, different strategies are proposed to determine the optimal window size at run-time. By incorporating the windowing mechanism, the two-phase training and volume testing process becomes a continuous learning process and the basic group testing process becomes more adaptive to dynamically changing environment.

Keywords: Web Services, group testing, verification, ranking

1. Introduction

Service-Oriented Architecture (SOA) and its implementation Web Services (WS) received significant attention as major computer companies, such as IBM, Microsoft, Oracle, SAP, Sun Microsystems, and etc., are adopting this new approach to develop software and systems. SOA advocates run-time system integration of loosely coupled services across heterogeneous platforms in a distributed environment [11]. SOA improves the flexibility of system development. However, trustworthiness becomes a serious problem and appropriate tradeoffs have to be paid during WS testing [2][4].

Traditionally, a software system is developed a single organization. Requirements are centrally managed and quality-control policies are uniformly enforced. Different teams working on different subsystems trust each other and their deliveries are consistent and can be easily integrated. In SOA approach, systems are generated by integrating services from different providers published on the Internet. In most cases, services are self-contained and distributed components that are maintained by independent vendors [11]. System developers look up services through third-party service brokers who maintain the directory of registration information of the available services. It is hard for different parties to trust each other unless certain agreement has been achieved regarding system quality and security. Therefore, trustworthiness has been a major problem that hampers WS from wide applications in industry.

To address the challenge, testing is critical to establish the trustworthiness among different parties [1][2][4][5][6][7][8][9][10]. However, SOA differs from traditional software architecture in various ways and imposes new challenges to traditional testing techniques [2]. The new problems include:

- Collaborative Testing: Cooperation and collaboration among different testing activities and stakeholders including service provider, service requestors, and service brokers.
- Specification-Based Testing: SOA proposes a fully specification-based process. WS define a XML-based protocol stack to facilitate service inter-communication and inter-operation. Specifications in, such as WSDL, OWL-S, WSFL, etc., describe the basic information of service features. Hence, test cases have to be automatically generated based on the specifications.
- Run-time Testing: All activities regarding service publishing, discovering, matching, composition, binding, execution, verification, and monitoring are implemented at run-time. To fit into the dynamic process, testing has to be generated, exercised, compared, and analyzed at run-time.
- Different implementations of the same specification: For the same specification of a service requirement, many alternative implementations could be available online. Effective algorithms are needed to rank and select the best WS.

To solve these problems, the ASTRAR (Adaptive

Service Testing and Ranking with Automated oracle generation and test case Ranking) framework is developed to comprehensively test and evaluate WS [5] [6][7][8][12]. It includes the activities of specification-based test case generation, collaborative testing, group testing, fault detection, reliability evaluation, oracle establishment, test case ranking, and WS ranking. To improve the efficiency, testing is divided into two phases: a training phase and a volume testing phase. This ASTRAR framework has been found to be effectively in reducing the number of tests needed to evaluate a large number of WS rapidly. The larger the WS sample size, the more saving can be achieved.

However, several issues exist in the basic ASTRAR testing process. First, the volume testing is still expensive if both of the number of test cases and the number of services are large. Second, in an open environment, the service providers can certainly improve the quality of their services using the published test cases. Thus, as new WS arrive, those early potent test cases may start to lose their potency because the new WS will have passed these known test cases before submission. Third, test case potency and test case's oracle confidence are conflict with each other. A mechanism is necessary to balance the tradeoffs. To address these issues, the paper proposes a windowing approach to further improve the test effectiveness of ASTRAR process.

This paper extends the basic ASTRAR framework by introducing a windowing mechanism in the volume testing phase. The set of WS under test in this phase is divided into small groups called windows. At the end of each window, testing results are collected to re-calculate the ranking of test cases and the confidence levels of oracles. Different strategies can be incorporated to evaluate the potency of a test case, i.e., the probability of detecting a fault, such as based on all historical data (this corresponds to the basic algorithm without windowing), based on the current window only, or based on most recent *n* windows. Test cases are re-organized with new ranking before starting the next window. By constantly monitoring test results, the windowing mechanism enables dynamic adjustment of the test strategies, such as the group test hierarchy, service rule-out strategy, and window size. The basic ASTRAR volume testing process is improved by the continuous learning process. This paper also presents different strategies for selecting and dynamically adapting service rule-out algorithms and window size. A case study is performed to illustrate the approach and the impacts of window sizes on test cost.

Section 2 introduces the background of group testing and ASTRAR framework. Section 3 presents the windowing approach. Section 4 depicts the case study and discusses the impacts of window size on test cost. Section 5 draws the conclusion.

2. Background

2.1 Progressive Group Testing of Web Service

Motivated by the Blood Group Testing techniques [3], Tsai et al. proposed the progressive WS Group Testing [5][6][7]. Test cases are organized into a layered structure based on their potency to detect failures. WS are tested layer by layer through the hierarchy. At the beginning, a fast testing process is applied to immediately eliminate the unlikely-to-win candidates as many as possible. As going up of the layers, the sophistication of testing increases progressively and the final winners will be tested rigorously. At each layer, testing results are collected and evaluated against the oracle. Multiple ranking and voting mechanisms can be applied to rule out WS progressively. In this way, it can greatly save test cost by reducing the number of WS under test at each level. The generic group testing process is as follows:

1. Organize the test cases hierarchically according to their ranks;
2. Starting from layer 1, for each layer:
   a) Apply all the test case in the current layer to all surviving WS and validate the results against pre-defined oracles;
   b) Calculate test results, WS are scored and those scored worst are ruled out.

2.2 ASTRAR Framework

ASTRAR extends the progressive group testing technique into a comprehensive WS testing and evaluation framework [8][12]. Essentially, ASTRAR ranks test cases and applies the highly ranked test cases first. Thus, more failures can be detected in fewer test runs.

In ASTRAR, test cases are ranked according to their *potency*, that is, the probability of detecting failures. Suppose during a test cycle, a test case is exercised on S WS under group test, and H of them failed, then the potency P of the test case is:

$$P = H/S$$

The higher the potency of a test case is, the higher its rank is. The purpose is to apply the test cases with the highest probability to detect failures first, and thus to reduce test cost by ruling out more failed WS earlier.

One of the difficult problems in WS runtime testing is to construct an oracle that can determine objectively and automatically if a failure has occurred. ASTRAR automatically generate test oracles using the majority voting mechanism in group testing proposed in [6]. *Confidence level* is defined for each test oracle, i.e., the probability that the oracle is correct. Taking the formula above, the confident level can be defined to be

the percentage of succeeded test runs:

$$C = (S- H)/S = 1-P$$

In ASTRAR [8], the group testing technique is further developed into a two-phase process, which significantly reduces the group testing overhead: the training phase and the volume testing phase. *Phase 1* (Training phase) is to establish the test oracle (the expected output of the test input) and the ranking of test cases. *Phase 2* (Volume testing phase) continues to test the remaining WS and any newly arrived WS, based on the profiles and history (test case potency, test oracle) obtained in the training phase. In this phase, it will first create the test hierarchy of group testing based on the potency of each test case obtained from the training phase. Then, the group testing technique is performed on all the remaining services layer-by-layer throughout the test hierarchy. At each layer, it will determine the correctness of a service output against the test oracle, eliminate the WS that have an unacceptable level of failure rate or reliability, update the confidence level of test oracles, and update the potency of test cases.

3. WS Group Testing with Windowing

Due to the open platform of WS and free competition of offering WS over internet, for any given WS specification, there may exist thousands of WS designed according to the specification. Feeding hundreds of test cases to thousands of WS one after another is both time and resource consuming. On the other hand, while it is desirable that a test case has a high potency P, it is also necessary to have a high confidence level C of the oracle associated with the test case. However, the relation $P = 1 - C$ makes these two factors conflict: higher test case potency implies a lower oracle confidence level. It is necessary to achieve a balance between these two factors.

3.1 Equal-Sized Windowing Mechanism

To achieve this balance between the test case potency and oracle confidence, a windowing mechanism is introduced in the ASTRAR framework, which can further reduce the test cost. The windowing mechanism consists of following steps:

1. Break down the WS under test (called sample set) into equal-sized windows, which are simply sets of WS. The window size is the number of WS in the set. The last window may have a different size.
2. Apply group testing on each window.
3. Compute test case potency and oracle confidence level at the end of each window and use the statistic data to guide the selection of test case in the next window.

This process learns from the test results of the previous windows to optimize the new test strategies dynamically and hence can better balance the tradeoffs and improve the test effectiveness. Notice that the windowing mechanism is applied to the volume testing phase (phase 2) only without changing the initial training phase during which an initial set of WS is randomly selected, group tested, and the test cases are ranked at the end of the training phase. Although the training phase can be considered as the first window of the new process with windowing, it limits the training size to the window size.

The major difference between the basic ASTRAR volume testing and this new windowing mechanism is that, in the former process, test case ranks and the confidence level are continuously updated after each test and it takes the entire history into consideration. In the latter scheme, the updates are done at the end of each window, resulting significant cost saving, and only the last window or last a few windows' test results are taking into consideration, giving a heavier weight to the latest testing results. This can prevent the WS developers to do "design-for-test-cases" work that make sure their WS pass the well-known test cases.

In fact, the basic ASTRAR volume testing can be considered a special case of the process with windowing when (1) the window size = 1 and (2) All previous windows are taken into consideration in evaluating test case potency and oracle confidence.

3.2 Evaluation of Test Case Potency

The windowing approach enables the re-evaluation of the test case potency at the end of the each window. Let

1) $H_{tc,i}$ be the number of failed services by test case tc_i in window i;
2) $S_{tc,i}$ be the total number of tested services by the test case tc_i in window i;
3) $P_{tc,i}$ be the potency of a test case tc_i in window i;
4) w_i be the weight of the window i, where $i \geq 1, w_{i+1} \geq w_i$; and
5) $PP_{tc,i}$ be the accumulated potency of a test case tc_i after window i.

We have:

$$P_{tc,i} = H_{tc,i} / S_{tc,i},$$

and $PP_{tc,n} = F(w_1 P_{tc,1}, w_2 P_{tc,2}, ... w_n P_{tc,n})$.

A simple definition of F is to use the weighted average:

$$PP_{tc,n} = \frac{\sum_{i=1,n} w_i P_{tc,i}}{\sum_{i=1,n} w_i}$$

Different strategies may be applied to select the weight. For example, we can simply take

$w_i = i$, and then we have $$PP_{tc,n} = \frac{\sum_{i=1,n} iP_{tc,i}}{\sum_{i=1,n} i} \quad (1)$$

A more stringent strategy is to define F as the decay function since the influence of test case potency in previous windows on current test case ranking will decay over time. Hence, we have

$w_i = e^{-(n-i)}$,

and $$PP_{tc,n} = \frac{\sum_{i=1,n} (P_{tc,i} * e^{-(n-i)})}{\sum_{i=1,n} e^{-(n-i)}} \quad (2)$$

Table 1gives a simplified example of the dynamic adjustment process. Suppose 3 test cases tc_1, tc_2 and tc_3 are used to test 170 services which are divided into a training set of size 20 and three windows of size 50. The weight of the window is the sequence number of the window, that is, $w_i = i$. During the training phase, the potency of tc_1, tc_2 and tc_3 are tested 50%, 25% and 20%, and ranked 1, 2, 3 respectively. After each window test, the potencies of test cases are recalculated using the previous two functions PP in (1) as the weighted average and PP in (2) as the decay function. By constant adjustment after each window, the cost of each window test decreases. Compared with the basic ASTRAR testing approach, the adjusted windowing approach can save test runs by $(370-320)/380 \approx 16\%$.

Table 1. Example test cases potency calculation

	Training			Window 1					Window 2					Window 3				
	H	S	P	H	S	P	PP(1)	PP(2)	H	S	P	PP(1)	PP(2)	H	S	P	PP(1)	PP(2)
tc1	10	20	50%	10	50	20%	20%	20%	5	20	25%	23%	24%	5	20	25%	24%	25%
tc2	4	20	20%	15	30	50%	50%	50%	20	50	40%	43%	43%	20	50	40%	42%	41%
tc3	5	20	25%	10	40	25%	25%	25%	10	30	33%	30%	31%	10	30	33%	32%	32%

In practice, the size of sample services and the number of test cases are much larger. The proposed mechanisms can greatly reduce test runs.

3.3 Fixed and Adaptive Window Size

In general, two approaches exist to address the sizing problem: the fixed window approach and the adaptive window approach. Fixed window is easier to implement and manage while adaptive window can be better reactive to the dynamically changing operational environment.

Different strategies exist to select a proper window size for the purpose of saving cost. If the window size is very large, the windowing technique would loose its power; on the contrary, if the window size is too small, then it goes back to the basic ASTRAR volume testing without windowing. Hence, the major issue for fixed window algorithm is to find an appropriate window size that could lead least testing runs.

Rather than pre-defined window size, adaptive window size strategy is to dynamically determine the proper window size by monitoring run-time system behavior. A key issue of the adaptive approach is to decide when to adapt and how to adapt. One solution is based on the arrival rate of services, that is, based on TIME rather than the number of services. If the arriving rate is high, we can reduce the window size; otherwise, we can enlarge the window size.

Another solution is based on the change of the potency of test cases. A Ranking Difference (RD) is introduced as follows to show the fluctuation of potency changes.

$$RD_{i,i-1} = \sqrt{\frac{1}{n} \sum_{j=1,n} (PP_{tc_j,i} - PP_{tc_j,i-1})^2}$$

where,

1) $RD_{i,i-1}$ is the ranking difference of two successive windows *i* and *i-1*.
2) $RP_{tc_j,i}$ and $RP_{tc_j,i-1}$ are the accumulated potency of a test case tc_j at two successive windows *i* and *i-1*.

Figure 1 shows that the test cases Ranking Difference decreases when testing is exercised within a window.

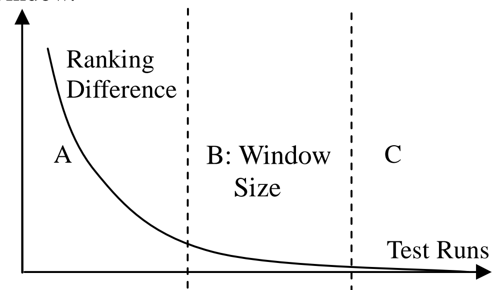

Figure 1. Test cases Ranking Difference decreases when performing testing within the window

In range A, the RD is large, and test cases ranking changes dramatically, indicating that test case ranking

are not solid enough to sufficiently decide the correctness of test cases potency. Window size could not be set into this range. In range C, the Ranking Difference decreases to 0, which means the test case ranking would not change even if more testing runs are exercised. Testing runs in this range are useless and wasteful so the window has to be closed before landing into range C. Hence, the window size should be set into range B. Two thresholds are set: the upper bound and the lower bound. The adjustment of the window size will be triggered once the RD drops outside the boundaries. When the RD is greater than the upper bound, the window size should be decreased; and if the RD is less than the lower bound, the window size should be increased.

3.4 Reliable Oracle Establishment

In the ASTRAR training phase where group testing is performed, an oracle is established using the majority voting mechanism. Due to the limited size of the training set, an incorrect oracle may be established. The windowing approach allows for the re-examination of a test oracle after each window of test and the re-calculation of its confidence level based on the latest test results. A log is created for each test case which records its oracles and the associated confidence level after each window test as well as accumulated reliability based on the entire testing history. Continue the definition in section 3.2, let

1) $C_{tc,i}$ be the confidence level of an oracle of test case *tc* in window *i* test; and
2) $CC_{tc,i}$ be the accumulated confidence of test case tc_i after window *i* test.

We have:

$$C_{tc,i} = (S_{tc,i} - H_{tc,i}) / S_{tc,i} \ ,$$

and $CC_{tc,n} = F(C_{tc,1}, C_{tc,2}, ...C_{tc,n})$.

Different functions can be used to calculate *C* based on historical data, such as taking the minimum, maximum or average value of historical data. The confidence level of an oracle should not be lower than 50% since it is defined by the majority principle. To ensure the reliability of the oracle, a threshold can be set to monitor the change of oracle confidence level during each window test, and policies can be defined to specify the conditions that an oracle needs to be revalidated. For example, we take the average function to calculate the accumulated confidence level of a test case, that is,

$$CC_{tc,n} = \frac{\sum_{i=1,n} C_{tc,i}}{n}$$

Taking the example in Table 1, Table 2 gives the confidence levels $C_{tc,i}$ and accumulated confidence levels $CC_{tc,i}$ of the oracle of each test case at each window test. Suppose the threshold, *t*, is set to 70% and the policy says: "for a test case *tc*, if the accumulated confidence level of its oracle drops below the threshold for continuous 3 windows, that is,

$$\exists i, i \geq 1, \ CC_{tc,i} \leq t, CC_{tc,i+1} \leq t, \text{and } CC_{tc,i+2} \leq t \ ,$$

the majority voting mechanism should be triggered to re-calculate the oracle."

According to the policy, the oracle of tc_2 needs to be re-calculated after the three window tests, and hence, tc_2 will go through the majority voting process to re-establish its oracle.

Table 2. Example Confidence Level

	tc_1		tc_2		tc_3	
	$C_{tc_1,i}$	$CC_{tc_1,i}$	$C_{tc_2,i}$	$CC_{tc_2,i}$	$C_{tc_3,i}$	$CC_{tc_3,i}$
Training	50%	50%	80%	80%	75%	75%
Window 1	80%	80%	50%	50%	75%	75%
Window 2	75%	72.5%	60%	55%	67%	71%
Window 3	75%	76.7%	60%	56.7%	67%	69.7%

4. Case Study and Experiment Results

To comparatively study the improvement based on windowing schemes, we use the same BBS (Best Buy Stock) example as defined in the basic ASTRAR paper [8]. For the same specification (on making decision of what stock is the best-buy stock) 60 different BBS WS were implemented. Some of the implementations contain faults. Thirty-two test cases were generated. In this case study, we will compare the test cost of basic ASTRAR approach, fixed windowing approach, and adaptive windowing approach.

A decay function is used to compute the potency of test cases. To simplify the process, we compute the most recent three windows, i.e.,

$$PP_{tc,n} = \frac{P_{tc,n-2} * e^{-2} + P_{tc,n-1} * e^{-1} + P_{tc,n}}{e^{-2} + e^{-1} + 1} \ ,$$

In the adaptive windowing approach, the upper threshold is set to 1 and the lower is 0.5, that is, to keep 0.5 < RD < 1. In this experiment, we set the training set size to 10. Table 3 shows the cost (in total test runs) of

the three approaches. It can be observed from the statistical results that the adaptive windowing approach is the most efficient one in terms of cost saving.

Figure 2 shows the cost saving efficiency for different approaches: the upper line denotes the basic ASTRAR algorithm, the middle curve denotes the fixed window scheme, and the bottom line denotes the adaptive windowing scheme.

Table 3. BBS experiment results

	Basic ASTRAR	Fixed Window						Adaptive Window
Cost	1125	Size	1	2	3	4	5	1051
		Cost	1073	1089	1107	1118	1139	
		Size	10	15	20	30	40	
		Cost	1167	1191	1195	1198	1194	

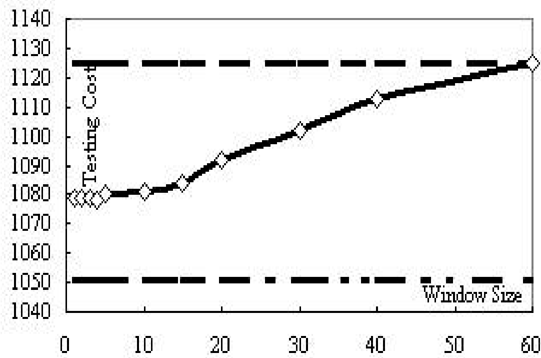

Figure 2. Cost comparison of the three schemes

5. Conclusion

This paper extends the generic ASTRAR framework with the windowing mechanism and discusses the strategies for dynamic ranking and test hierarchy re-organization of the group testing at each window, reliable test oracle establishment, and fixed/adaptive window size selection. Experiment results show that adaptive windowing algorithm is the most efficient mechanism to save testing runs. The WS specification-independent heuristic mechanism in the adaptive windowing algorithm could also find the most efficient window size at run time.

Reference

[1] P.A. Bonatti, P. Festa, "On Optimal Service Selection", *Proc.of the International World Wide Web Conference (WWW)*, 2005, pp.530-538.

[2] B. De, "Web Services - Challenges and Solutions", WIPRO white paper, 2003, http://www.wipro.com.

[3] D. Z. Du and F. Hwang, *Combinatorial Group Testing and Its Applications*, World Scientific, 2nd edition, 2000.

[4] N. Milanovic and M. Malek, "Verifying Correctness of Web Services Composition", *Proc. of the 11th Infofest*, Budva, Montenegro 2004.

[5] W. T. Tsai, X. Wei, Y. Chen, B. Xiao, R. Paul, and H. Huang, "Developing and Assuring Trustworthy Web Services", *Proc. of the 7th International Symposium on Autonomous Decentralized Systems (ISADS)*, 2005, pp.43-50.

[6] W.T. Tsai, Y. Chen, D. Zhang, H. Huang, "Voting Multi-Dimensional Data with Deviations for Web Services under Group Testing," *Proc. of the 4th International Workshop on Assurance in Distributed Systems and Networks (ADSN)*, in conjunction with ICDCS-25, June 2005, pp. 65 - 71.

[7] W. T. Tsai, Y. Chen, Z. Cao, X. Bai, H. Huang, R. Paul, "Testing Web Services Using Progressive Group Testing", *Proc. of the Advanced Workshop on Content Computing*, Zhenjiang, China, November 2004, pp.314-322.

[8] W. T. Tsai, Yinong Chen, Ray Paul, Hai Huang, Xinyu Zhou, Xiao Wei, "Adaptive Testing, Oracle Generation, and Test Case Ranking for Web Services," *Proc. of the 29th Annual International Computer Software and Applications Conference (COMPSAC)*, Edinburgh, July 2005, pp 101-106.

[9] W. Xu, "Generating Test Cases for Web Services Using Data Perturbation", *Proc. of the Workshop on Testing, Analysis and Verification of Web Services*, Boston, MA, July 2004, pp.144-149

[10] J. Yang and M. P. Papazoglou, "Web Component: A Substrate for Web Service- Reuse and Composition", *Proc. of the 14th International Conference on Advanced Information Systems Engineering (CAiSE02)*, Toronto, Lecture Notes in Computer Science, Vol. 2348, pp 21-36. pp.21-36, Springer, 2002

[11] Web Services Conceptual Architecture (WSCA1.0), IBM Software Group, May 2001.

[12] ASTRAR, http://asusrl.eas.asu.edu/webstrar/