# Dynamic Random Testing of Web Services: A Methodology and Evaluation

Chang-ai Sun, *Senior Member, IEEE,* Hepeng Dai, Zhiqiang Zhang, Guan Wang, Dave Towey, *Member, IEEE,* Kai-Yuan Cai, *Member, IEEE,* and Tsong Yueh Chen, *Member, IEEE,*

**Abstract**—In recent years, Service Oriented Architecture (SOA) has been increasingly adopted to develop distributed applications in the context of Internet. To develop reliable SOA-based applications, an important issue is how to ensure the quality of web services. In this paper, we propose a dynamic random testing (DRT) technique for web services, which is an improvement of the widely-practiced random testing (RT). We examine key issues when adapting DRT to the context of SOA, including a framework, guidelines for parameter settings, and a prototype for such an adaptation. Empirical studies are reported where DRT is used to test three real-life web services and mutation analysis is employed to measure the effectiveness. Our experimental results show that, compared with two baseline techniques, namely RT and Random Partition Testing (RPT), DRT demonstrates higher fault-detection effectiveness with a lower test case selection overhead, and the theoretical guidelines of parameter setting for DRT are validated to be effective. The proposed DRT and the prototype provide an effective and efficient approach to testing web services.

**Index Terms**—Software Testing, Random Testing, Dynamic Random Testing, Web Service, Service Oriented Architecture.

## 1 Introduction

SERVICE oriented architecture (SOA) [2] defines a loosely coupled, standard-based, service-oriented application development paradigm in the context of Internet. Within SOA, three key roles are defined, namely service providers which develop and own services, service requestors which consume or invoke services, and service registry which registers services from service providers and return services to service requestors. In this context, applications are built upon services that expose functionalities by publishing their interfaces in appropriate repositories, and abstracting entirely from the underlying implementation. Published interfaces may be searched by other services or users and then invoked. web services are the realization of SOA based on open standards and infrastructures [3]. Ensuring the reliability of SOA-based applications becomes crucial particularly when such applications implement important business processes.

Software testing is a practical method to guarantee quality and reliability of software. However, some features of SOA pose challenges for testing web services [4], [5]. For instance, service requestors often cannot access the source code of web services which are published and owned by another organization, and as a consequence white-box testing techniques become inapplicable. Therefore, testers naturally turn to black-box testing techniques.

Random Testing (RT) [6] is one of the most widely-practiced black-box testing techniques. In RT, test cases are randomly selected from the input domain (which refers to the set of all possible inputs of the software under test). Therefore, RT is easy to use. Nevertheless, RT may be inefficient in some situations because it does not make use of any information about software under test (SUT) and test history. In recent years, many efforts have been made to improve to RT in different ways [7]–[10]. For example, adaptive random testing (ART) [8] is proposed to improve RT in order to make more diverse distribution of test cases in the input domain.

Different from RT, partition testing (PT) aims to generate test cases in a more "systematic" way, hopefully using fewer test cases reveal more faults. Firstly, the input domain of SUT divided into disjoint partitions. Then test cases are selected from each and every partition. In PT, each partition is expected to have a certain degree of homogeneity, that is, test cases in same partition should have similar software execution behavior. Ideally, a partition should be homogeneous in fault detection. In other words, if one input is fault-revealing/non-fault-revealing, all other inputs in the same partition shall be fault-revealing/non-fault-revealing too.

RT and PT are based on different intuitions, and they have their own advantages and disadvantages. It is likely that they can be complementary to each other in detecting different faults. Therefore, it is intuitively appealing to investigate the integration of RT and PT. Cai et al. [7] have proposed the so-called random partition testing (RPT). In RPT, the input domain is first divided into $m$ partitions $s_1, s_2, \ldots, s_m$, and each $s_i$ is allocated a probability $p_i$. A partition $s_i$ is randomly selected according to the testing profile $\{\langle s_1, p_1\rangle, \langle s_2, p_2\rangle, \ldots, \langle s_m, p_m\rangle\}$, where $p_1 + p_2 + \cdots + p_m = 1$. A concrete test case is then randomly

*A preliminary version of this paper was presented at the 36th Annual IEEE Computer Software and Applications Conference (COMPSAC 2012) [1].*
*C.-A. Sun, H. Dai, Z. Zhang and G. Wang are with the School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China. E-mail: casun@ustb.edu.cn.*
*D. Towey is with the School of Computing Science, University of Nottingham Ningbo China, Ningbo 315100, China. E-mail: dave.towey@nottingham.edu.cn*
*K.-Y. Cai is with the School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China. E-mail: kycai@buaa.edu.cn.*
*T.Y. Chen is with the Department of Computer Science and Software Engineering, Swinburne University of Technology, Hawthorn VIC 3122, Australia. Email: tychen@swin.edu.au.*

selected from the chosen $s_i$.

In traditional RPT testing, the partitions and corresponding testing profile remain constant during software testing, which may not be a good strategy. Independent researchers from various areas have individually made the same observation that the fault-revealing inputs tend to cluster into "continuous regions" [11], [12], that is, the similarity of execution behavior among neighboring software inputs. Following the idea of software cybernetics, Cai et al. proposed dynamic random testing (DRT) [7], which is to improve RT and RPT. Different from the original RPT, where the values of $p_i$ are fixed and kept static along the testing process, DRT attempts to dynamically change the values of $p_i$: If a test case from a partition $s_i$ reveals a fault, the corresponding $p_i$ will be increased by a constant $\epsilon$; otherwise, decreased by the $\epsilon$.

In practice, web services have been somehow tested at the side of service providers. Those simple or easy-to-test faults are removed and the remaining faults are normally hard to detect. To pursue a higher reliability of web services, simple RT strategy may not be an appropriate candidate technique, especially when the scale of web services is huge or there are some stubborn faults. Some studies show that DRT can improve RT in term of fault detection effectiveness [13], [14], [15].

In this paper, we present a dynamic random testing (DRT) for web services, which is an enhanced version of RT and in the meanwhile an adaptation of DRT to the context of SOA. We further examine key issues of such an adaption, and conduct empirical studies to evaluate the feasibility and effectiveness of the proposed DRT. Experimental results show that DRT evidently outperforms RT. The contributions of this work include:

- We develop an effective and efficient testing technique for web services, including a DRT framework which addresses key issues to testing web services and a prototype which partially automates the framework.
- We evaluate the performance of DRT through a series of empirical studies on three real web services. It has shown that DRT has significantly higher fault-detection effectiveness than random testing and random partition testing, while its test case selection overhead is lower.
- We provide the guidelines of parameter settings of DRT supported by theoretical analysis, and the guidelines are also validated by the empirical studies.

The remaining of the paper is organized as follows. Section 2 introduces the underlying concepts of DRT, web services and mutation analysis. Section 3 presents a framework DRT for web services, guidelines for parameter settings in DRT, and a prototype which partially automates the DRT. Section 4 describes an empirical study where the proposed DRT is used to test three real-life web services, the results of which are summarized in Section 5. Section 6 discusses related work and Section 7 concludes the paper.

## 2 BACKGROUND

In this section, we present the underlying concepts of DRT, web services, and mutation analysis.

### 2.1 Dynamic Random Testing(DRT)

DRT combines RT and partitioning testing [31] in order to enjoy the benefits of both testing techniques. Assume the test suite *TS* is classified into $m$ partitions denoted as $s_1, s_2, \ldots, s_m$, and suppose that a test case from $s_i$ ($i = 1, 2, \ldots, m$) is selected and executed. If this test case reveals a fault, $\forall j = 1, 2, \ldots, m$ and $j \neq i$, we set

$$p'_j = \begin{cases} p_j - \dfrac{\epsilon}{m-1} & \text{if } p_j \geq \dfrac{\epsilon}{m-1} \\ 0 & \text{if } p_j < \dfrac{\epsilon}{m-1} \end{cases}, \tag{1}$$

and then

$$p'_i = 1 - \sum_{\substack{j=1 \\ j \neq i}}^{m} p'_j. \tag{2}$$

Otherwise, if the test case does not reveal a fault, we set

$$p'_i = \begin{cases} p_i - \epsilon & \text{if } p_i \geq \epsilon \\ 0 & \text{if } p_i < \epsilon \end{cases}, \tag{3}$$

and then for $\forall j = 1, 2, \ldots, m$ and $j \neq i$, we set

$$p'_j = \begin{cases} p_j + \dfrac{\epsilon}{m-1} & \text{if } p_i \geq \epsilon \\ p_j + \dfrac{p'_i}{m-1} & \text{if } p_i < \epsilon \end{cases}. \tag{4}$$

The detailed algorithm of DRT is given in Algorithm 1. In DRT, the first test case is selected from a partition that is randomly selected according to the initial probability profile $\{p_1, p_2, \ldots, p_m\}$ (Lines 2 and 3 in Algorithm 1). After the execution of a test case, the testing profile $\{\langle s_1, p_1 \rangle, \langle s_2, p_2 \rangle, \ldots, \langle s_m, p_m \rangle\}$ will be updated through changing the values of $p_i$: If a fault is revealed, Formulae 1 and 2 will be used; otherwise, Formulas 4 and 3 will be used. The updated testing profile will be used to guide the random selection of the next test case (Lines 8). Such a process is repeated until the termination condition is satisfied (refer to Line 1). The termination condition here can be either "testing resource has been exhausted", or "a certain number of test cases have been executed", or "a certain number of faults have been detected", etc.

From Formulae 1 to 3, we can see that an update of testing profile involves $m$ simple calculations, that is, it requires a constant time. In addition, the selection of $s_i$, the selection and execution of a test case all need a constant time. Simply speaking, the execution time for one iteration in DRT is constant. Therefore, the time complexity for DRT to select $n$ test cases is $O(n)$.

**Algorithm 1** DRT

**Input:** $\varepsilon, p_1, p_2, \ldots, p_m$

```
while termination condition is not satisfied
    Select a partition s_i according to the testing profile {⟨s_1,p_1⟩, ⟨s_2,p_2⟩, ..., ⟨s_m,p_m⟩}
    Select a test case t from s_i
    Test the software using t
    if a fault is revealed by t
        Update p_j (j = 1,2,...,m and j ≠ i) and p_i according to Formulae 1 and 2.
    else
        Update p_j (j = 1,2,...,m and j ≠ i) and p_i according to Formulas 3 and 4.
    end_if
end_while
```

### 2.2 Web Services

A web service is a platform-independent, loosely coupled, self-contained programmable web-enabled application that can be described, published, discovered, coordinated and configured using XML artifacts for the purpose of developing distributed interoperable applications [2]. A web service consists of description which is usually specified in WSDL, and implementation which can be written in any programming language. web services expose their functionalities by publishing interfaces and at run-time are usually deployed in a service container. In order to invoke a web service, one needs to analyze the input message of its WSDL and assign the expected values.

Web service is a basic component of SOA software. The adoption of SOA, in addition to changing the architecture of a system, brings changes in the process of building the system and using it, and this has effects on testing too [5].

- *lack of observability of the service code and structure:* users and service registry only get interfaces of services, which prevents white-box testing approaches.
- *dynamicity and adaptiveness:* For SOA, testers may not be able to determine the component invoked, or the set of possible targets.
- *lack of control:* For traditional software testing, testers can control components under test. However, only the services provider has control over the service.
- *lack of trust:* A service provider may lie or provide incorrect/inaccutrate description of a service's functional and non-functional behavior, which makes the testing task be difficult.

Obviously, SOA testing is more difficult than testing traditional software. RT is a widely used software testing technology, but because of the characteristics of the RT itself, it may not be efficient in web services.

### 2.3 Mutation analysis

Mutation analysis [16] is widely used to assess the adequacy of test suites and the effectiveness of testing techniques. It applies some mutation operators to seed various faults into the program under test, and thus generates a set of variants, namely mutants. If a test case causes a mutant to show a behavior different from the program under test, we say that this test case "kills" the mutant and thus detects the fault injected into the mutant. We normally use the mutation score (MS) to measure how thoroughly a test suite "kills" the mutants, which is defined as

$$MS(p, ts) = \frac{N_k}{N_m - N_e} \tag{5}$$

where $p$ refers to the program being mutated, $ts$ refers to test suite under evaluation, $N_k$ refers to the number of killed mutants, $N_m$ refers to the total number of mutants, and $N_e$ refers to the number of equivalent mutants. An equivalent mutant refers to one whose behaviors are always the same as those of $p$. It has been pointed out that compared with manually seeded faults, the automatically generated mutants are more similar to the real-life faults, and the mutant score is a good indicator for the effectiveness of a testing technique [17]. In this paper, we will use mutation analysis technique to evaluate the effectiveness of the proposed DRT for web services.

## 3 Dynamic Random Testing for Web Services

We describe a framework of applying DRT to web services, discuss guidelines for parameter settings in DRT, and present a prototype which partially automates the DRT for web services.

### 3.1 Framework

Considering the principle of DRT and the features of web services, we propose a framework of DRT for web services, as illustrated in Figure 1. Components in the box are relevant to DRT, and web services under test are located out of the box. The interactions between DRT components and web services are depicted in the framework. We next discuss components in the framework individually.

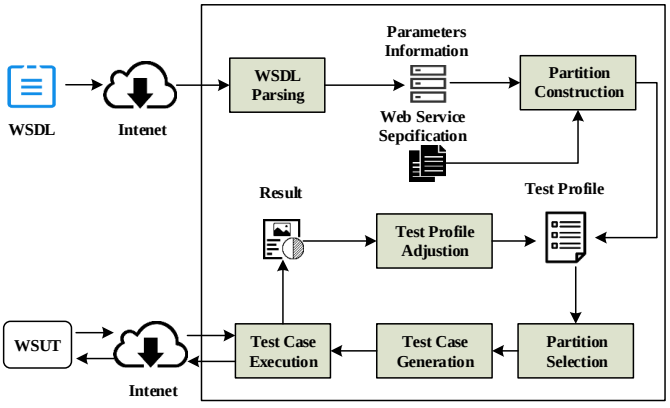



Fig. 1. The framework of DRT for web services

1 *WSDL Parsing*. web services are composed of services and relevant WSDL document. By parsing the WSDL document, we can get input information for each operation in the services, including the number of parameters, parameter names, parameter types, and additional requirements related to the input parameters.

2 *Partition Construction*. Partition testing refers to a class of testing techniques [18]. Although existing partitioning schema may vary from statement, dataflow, branch, path, functionality to risk, we prefer partitions at the specification level to ones at the program level, because DRT is a kind of black-box testing technique combining random testing and partition testing. The principles on achieving convenient and effective partitions have been discussed in some literature [18]–[21]. According to the specification of web service under test (WSUT) and parameters parsed, we can choose an appropriate method to partition the input domain of WSUT. After partitioning, the testers can assign probability distribution partitions as the initial testing profile. There are different ways to set the initial testing profile. One is to make a uniform probability distribution; the other is to set larger probabilities for those core partitions while smaller probabilities for less important ones.
3 *Partition Selection*. When the DRT starts, the component is responsible for selecting a partition randomly according to the testing profile.
4 *Test Case Generation*. Suppose the partition $s_i$ is selected. We can randomly and independently generate a test case that belongs to $s_i$. Since we have parsed the WSDL document, generating test cases can be automated and not particularly difficult.
5 *Test Case Execution*. The component receives test case generated by Test Case Generation. During the testing, the component is responsible for converting test cases into input messages, invoking web services through the SOAP protocol, and intercepting the test results (namely output messages).
6 *Test Profile Adjustment*. After one test is completed, the component decides whether the test passes or fails by comparing the actual output with the expected output. With the evaluation result, Probability Distribution Adjustment accordingly adjusts the probability distribution. In some situation, it may be impossible to decide whether one test succeeds or not (namely the oracle problem), then metamorphic testing [22] may provide an aid to make decisions.

In general, test case generation in DRT for web services still follows the principle of random testing; on the other hand, the selection of next test case (corresponding to a partition) is in accordance with the probability distribution. In this way, DRT takes both advantages of random testing (easiness) and partition testing (effectiveness). In addition, some components in the framework of DRT for web services can be automated, including Partition Selection, Test Case Generation, Test Case Execution, Test Profile Adjustment. In order to make DRT for web services more efficient and practical, we developed a prototype to be described at section 3.3.

### 3.2 Guidelines for parameter settings

Our previous work [1] found that the performance of DRT strategy may be affected by the parameter $\epsilon$ of DRT strategy and the number of partitions. We further explore their impacts through a theoretical analysis. In order to be mathematically tractable, we make the following assumptions:

1 The failure rate $\theta_i$ of each partition $s_i$ ($i = 1, 2, \ldots, m$, and $m > 1$) is unknown, but can be estimated.
2 The failure rate $\theta_i$ ($i = 1, 2, \ldots, m$, and $m > 1$) remains unchanged during the testing process, that is, faults are not removed after detection of failure.
3 Test cases are selected with replacement, that is, same test cases may be selected more than once.

The main idea of DRT strategy is to increase the selecting probabilities of the partitions that have larger failure rates. Parameter $\epsilon$ is used to update the testing profile during the testing process, and the number of partitions also affects the speed of updating the testing profile according to Formulae 1 and 3. Therefore, under a certain number of partitions, we are interested in investigating the settings of $\epsilon$ to achieve better performance of DRT strategy.

We assume that the maximum of all failure rates is $\theta_M$, that is, $\theta_M \geq \theta_i$ for all $i \in \{1, 2, \ldots, m\}$. $s_M$ denotes the partition that has a failure rate $\theta_M$. Let $p_i^n$ denote the probability of after executing the $n^{th}$ test case from partition $s_i$. As the testing proceeds, the probability $p_M$ of partition $s_M$ being selected is expected to be increased, namely

$$p_M^{n+1} > p_M^n \tag{6}$$

At the beginning, the testing profile is $\{\langle s_1, p_1^0\rangle, \langle s_2, p_2^0\rangle, \ldots, \langle s_m, p_m^0\rangle\}$. After $n^{th}$ test cases have been executed, the testing profile has updated to $\{\langle s_1, p_1^n\rangle, \langle s_2, p_2^n\rangle, \ldots, \langle s_m, p_m^n\rangle\}$. Note that $p_i^n$ is increased/decreased by the parameter $\epsilon$ during the testing process, and the value of $\epsilon$ is relatively small. For instance, in [13] and [15], the value of $\epsilon$ is set 0.05. On the other hand, the initial $p_i^0$ is larger than $\epsilon$, and the adjustment of $p_i$ is relatively small according to Formulae 1 to 4, which rarely leads to the situation $p_i < \epsilon/(m-1)$ or $p_i < \epsilon$ ($i = 1, 2, \ldots, m$), so we do not consider this.

Without loss of generality, we explore the relation between $p_i^{n+1}$ and $p_i^n$. We calculate the conditional probability $p(i|\delta)$ of following four cases (denoted as $\delta_1, \delta_2, \delta_3$, and $\delta_4$, respectively).

Case 1 ($\delta_1$): $t_n \notin s_i$ **and a fault is detected by** $t_n$, then $p(i|\delta_1)$ is calculated as follows according to Formula 1.

$$p(i|\delta_1) = \sum_{i \neq j} \theta_j (p_i^n - \frac{\epsilon}{m-1}) \tag{7}$$

Case 2 ($\delta_2$): $t_n \in s_i$ **and a fault is detected by** $t_n$, then $p(i|\delta_2)$ is calculated as follows according to Formula 2.

$$p(i|\delta_2) = \theta_i (p_i^n + \epsilon) \tag{8}$$

Case 3 ($\delta_3$): $t_n \in s_i$ **and no fault is detected by** $t_n$, then $p(i|\delta_3)$ is calculated as follows according to Formula 3.

$$p(i|\delta_3) = (1 - \theta_i)(p_i^n - \epsilon) \tag{9}$$

Case 4 ($\delta_4$): $t_n \notin s_i$ **and no fault is detected by** $t_n$, then $p(i|\delta_4)$ is calculated as follows according to Formula 4.

$$p(i|\delta_4) = \sum_{i \neq j} (1 - \theta_j)(p_i^n + \frac{\epsilon}{m-1}) \tag{10}$$

Therefore, we have $p_i^{n+1}$ for all cases together is as follows:

$$\begin{aligned} p_i^{n+1} =& p_i^n\theta_i(p_i^n+\epsilon) + p_i^n(1-\theta_i)(p_i^n-\epsilon) \\ &+ \sum_{j\neq i} p_j^n\theta_j(p_i^n - \frac{\epsilon}{m-1}) \\ &+ \sum_{j\neq i} p_j^n(1-\theta_j)(p_i^n + \frac{\epsilon}{m-1}) \\ =& p_i^n + Y_i^n \end{aligned} \tag{11}$$

where

$$\begin{aligned} Y_i^n =& \frac{\epsilon}{m-1}(2p_i^n\theta_i m - p_i^n m - 2p_i^n\theta_i + 1) \\ &- \frac{2\epsilon}{m-1}\sum_{j\neq i} p_j^n\theta_j \end{aligned} \tag{12}$$

From Formula 12, we have:

$$\begin{aligned} Y_M^n - Y_i^n =& \frac{2\epsilon}{m-1}(m(p_M^n\theta_M - p_i^n\theta_i) - \\ & \frac{m(p_M^n - p_i^n)}{2}) \end{aligned} \tag{13}$$

Before presenting the final guidelines, we give the following lemma first.

***Lemma 1.*** If $(p_i^n - p_i^m)\epsilon > 2(p_i^n\theta_i - p_M n\theta_M)((m+\epsilon)/m)$, then $p_M^{n+1} > p_M^n$.

*Proof:* Since $0 < \epsilon < 1$, and $(m+\epsilon)/m > 1$, then $p_i^n - p_M^n > (p_i^n - p_M^n)\epsilon$, and $2(p_i^n\theta_i - p_M^n\theta_M) < 2(p_i^n\theta_i - p_M^n\theta_M)((m+\epsilon)/m)$. Hence, we have $p_i^n - p_M^n > 2(p_i^n\theta_i - p_M^n\theta_M)((m+\epsilon)/m)$. Furthermore, the following formulae is true:

$$p_i^n - p_M^n > 2(p_i^n\theta_i - p_M^n\theta_M) \tag{14}$$

According to Formula 14, we have:

$$(p_M^n\theta_M - p_i^n\theta_i) - \frac{p_M^n - p_i^n}{2} > 0 \tag{15}$$

Since $0 < \epsilon < 1$, and $m > 1$, hence we have:

$$\frac{2\epsilon}{m-1}(m(p_M^n\theta_M - p_i^n\theta_i) - \frac{m(p_M^n - p_i^n)}{2}) > 0 \tag{16}$$

According to Formulae 16 and 13, we have $Y_M^n - Y_i^n > 0$. Directly, $\sum_{i=1}^m Y_i^n = 0$, because $\sum_{i=1}^m p_i^{n+1} = 1$, and $\sum_{i=1}^m p_i^n = 1$. Then, we have $Y_M^n > 0$. According to Formula 11, we have $p_M^{n+1} = p_M^n + Y_M^n$. Since $Y_M^n > 0$, obviously, $p_M^{n+1} > p_M^n$. □

Accordingly, we now present the following theorem that states a sufficient condition to achieve $p_M^{n+1} > p_M^n$.

***Theorem 1.*** For all failure rates $\theta_i, i \in \{1, 2, \ldots, m\}$, and $\theta_M$ is the maximum of $\theta_i$, the following condition is sufficient to guarantee that $p_M^{n+1} > p_M^n$

$$\frac{1}{2\theta_M} < \frac{m+\epsilon}{m\epsilon} < \frac{1}{2\theta_\Delta} \tag{17}$$

where $\theta_i \leq \theta_\Delta < \theta_M$, and $\theta_i \neq \theta_M$.

*proof:* In order to guarantee $p_M^{n+1} > p_M^n$, we consider the following three kinds of cases:

**Case 1 ($p_i^n = p_M^n$):** Since $0 < m$, $0 < \epsilon < 1$, and $\theta_i < \theta_M$, then $\epsilon(p_i^n - p_M^n) > 2(p_i^n\theta_i - p_M^n\theta_M^n)(m+\epsilon)/m$ is satisfied. According to Lemma 1, we have $p_M^{n+1} > p_M^n$.

**Case 2 ($p_i^n > p_M^n$):** Since $1/2\theta_\Delta > (m+\epsilon)/m\epsilon$, and $1/2\theta_i \geq 1/2\theta_\Delta$, then $(p_i^n - p_M^n)\epsilon > 2(p_i^n - p_M^n)\theta_i(m+\epsilon)/m$. Because $\theta_M > \theta_i$, we have $2(p_i^n - p_M^n)\theta_i(m+\epsilon)/m > 2(p_i^n\theta_i - p_M^n\theta_M)(m+\epsilon)/m$. Then $(p_i^n - p_M^n)\epsilon > 2(p_i^n\theta_i - p_M^n\theta_M)(m+\epsilon)/m$. According to Lemma 1, we have $p_M^{n+1} > p_M^n$.

**Case 3 ($p_i^n < p_M^n$):** Since $1/2\theta_M < (m+\epsilon)/m\epsilon$, then $(p_M^n - p_i^n)\epsilon < 2(p_M^n - p_i^n)\theta_M(m+\epsilon)/m$. Because $\theta_i < \theta_M$, we have $2(p_M^n - p_i^n)\theta_M(m+\epsilon)/m < 2(p_M^n\theta_M - p_i^n\theta_i)(m+\epsilon)/m$. Furthermore, we have $(p_M^n - p_i^n)\epsilon < 2(p_M^n\theta_M - p_i^n\theta_i)(m+\epsilon)/m$, and thus $(p_i^n - p_M^n)\epsilon > 2(p_i^n\theta_i - p_M^n\theta_M)(m+\epsilon)/m$. According to Lemma 1, we have $p_M^{n+1} > p_M^n$. □

To sum up, there is always an interval:

$$\frac{m+\epsilon}{m\epsilon} \in (\frac{1}{2\theta_M}, \frac{1}{2\theta_\Delta}) \tag{18}$$

where $\theta_i \leq \theta_\Delta < \theta_M$, and $\theta_i \neq \theta_M$, which can guarantee $p_M^{n+1} > p_M^n$. Furthermore, we can have:

$$\frac{m+\epsilon}{m\epsilon} = \frac{1}{2\theta_M} + (\frac{1}{2\theta_\Delta} - \frac{1}{2\theta_M}) * h, (0 < h < 1) \tag{19}$$

### 3.3 Prototype

Figure 2 shows a snapshot of the prototype which partially automates DRT for web services. To start, testers need to input the address of web service being tested (namely the URL of WSDL), and press Parse button to analyze input formats and output formats. Next, an operation is selected from the operation list (in the bottom left). As to the partitions and test suites, the prototype provides two options. One is to automatically generate partitions; the other is to upload the predefined partitions and test suites. If the former is selected, the initial testing profile will be automatically set; otherwise, this task is left for testers. Before executing tests (the Test button), testers are required to set limits on the number of tests (Test Repetition Limit). During the testing, if a failure is detected without exceeding the limits, the prototype suspends the testing and asks for testers instruction. Testers can choose to remove defects and continue tests or stop tests. When tests are completed, the test report is summarized in a file which may be used for measuring the effectiveness of DRT.

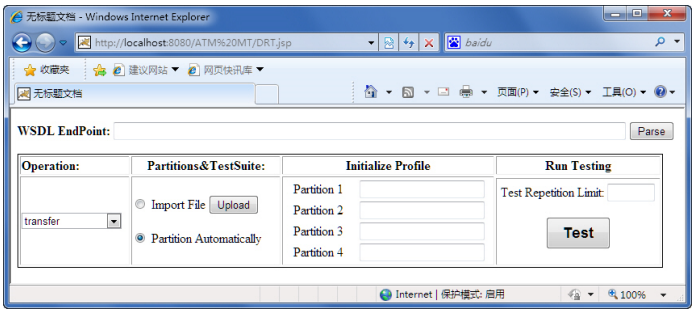



Fig. 2. The snapshot of the prototype interfaces

## 4 Empirical Study

We have conducted a series of empirical studies to evaluate the performance of DRT.

### 4.1 Research questions

In our experiments, we focus on the following three research questions:

RQ1 How effective is DRT in detecting faults of web services?
The fault-detection effectiveness is one key criterion to evaluate the performance of a testing technique. In this study, three common real-life web services are selected as subject programs, and mutation analysis is used to evaluate the effectiveness.

RQ2 How does the number of partitions and the parameter of DRT affect the failure detection efficiency of DRT?
In our earlier work [1], we found that the parameter of DRT have a significant effect on DRT efficiency, and the size of the parameter may be related to the number of partitions. The relationship between the parameter of DRT strategy and the number of partitions is obtained through theoretical analysis, which is verified through empirical studies.

RQ3 What is the actual test case generation overhead for DRT strategy.
In Section 2.1, we have theoretically demonstrated that DRT only require linear time for generating test case. We also wish to empirical evaluate the test case generation overhead of it in detecting web services faults.

### 4.2 Subject Web Services

In our study, we selected three real-life web services, namely `Aviation Consignment Management Service (ACMS)`, `China Unicom billing service (CUBS)`, `Parking billing service (PBS)`, as the subject program of the experiments. Mutation analysis help us generate 1563 mutants for all objects. After identifying equivalent mutants, we only include those mutants being hard to detect for the evaluation. Further to say, every those mutants cannot be detected with less than 20 randomly generated test cases. Table 1 summarizes the basic information of the object web services. In following section, we give a detailed description of each web services.

TABLE 1
Objects web services

| Web Services | LOC | Number of mutants |
|---|---|---|
| ACMS | 116 | 2 |
| CUBS | 131 | 11 |
| PBS | 129 | 4 |

#### 4.2.1 Aviation Consignment Management Service (ACMC)

The ACMS helps customs check the weight of free luggage and the cost of Overweight luggage. According to destinations, flights are divided into international and domestic flights. Each aircraft offers three kinds of cabins for passengers to choose: economy, business and first class. Passengers in different classes can enjoy different weight of luggage for free. The detailed billing rules are summarized in Table 2, where $price_0$ means economy class fare. When the flight is international, the weight of free luggage is different according to whether the passenger is a student or not. If the passenger is a student, the weight of luggage is 30, and if not it is 20.

#### 4.2.2 China Unicom Billing Service (CUBS)

CUBS provide an interface in which customers can know how much fee needed to pay according to cell-phone plans, cell-phone calls, and date usage. The detailed cell-phone plans are summarized in Tables 3,4,5.

TABLE 4
Cell-phone plan B

| | cell-phone plan (CNY) | 46 | 66 | 96 | 126 | 156 | 186 |
|---|---|---|---|---|---|---|---|
| contains | free calls (min) | 120 | 200 | 450 | 680 | 920 | 1180 |
| | free date (MB) | 40 | 60 | 80 | 100 | 120 | 150 |
| | free incoming calls | domestic(including video calls) | | | | | |
| overuse | incoming calls (CNY/min) | 0.25 | 0.20 | 0.15 | | | |
| | date (CNY/KB) | 0.0003 | | | | | |
| | video calls (CNY/min) | 0.60 | | | | | |

TABLE 5
Cell-phone plan C

| | cell-phone plan (CNY) | 46 | 66 | 96 |
|---|---|---|---|---|
| contains | free calls (min) | 260 | 380 | 550 |
| | free date (MB) | 40 | 60 | 80 |
| | free date (MB) | domestic (including video calls) | | |
| overuse | incoming calls (CNY/min) | 0.25 | 0.20 | 0.15 |
| | date (CNY/KB) | 0.0003 | | |
| | video calls (CNY/min) | 0.60 | | |

#### 4.2.3 Parking Billing Service (PBS)

Consider a parking billing service, which accepts the parking details of each vehicle, including type of vehicle, ~~type of car,~~ day of week, discount coupon, and hours of parking. This service first rounds up the parking duration to the next full hour, and than calculates the parking fee for a vehicle according to the hourly rates in Table 6. If a discount is presented, a 50% discount off the parking fee will be given.

To facilitate better parking management, at the time of parking, customers can optionally provide an estimation of parking duration in terms of three different time ranges, namely $(0.0, 2.0]$, $(2.0, 4.0]$, and $(4.0, 24.0]$. Suppose a customer providers an estimation. If the estimated hours of parking and the actual hours of parking fall into the same time range, then the customer will receive a 40% discount. If the estimated hours and the actual hours are in different time range, a 20% markup will be added. A customer may also choose to either use a discount coupon, or provide an estimation of parking duration, but not both. Obviously, a customer may also choose to neither provide an estimation, nor use a discount coupon. Note that no vehicles are allowed to park across two consecutive days on a continuous basis.

TABLE 2
The detailed billing rules of ACMC

| | Domestic Flights | | | International Flights | | |
|---|---|---|---|---|---|---|
| | first class | business class | economy class | first class | business class | economy class |
| Take along (kg) | 5 | 5 | 5 | 7 | 7 | 7 |
| weight for free (kg) | 40 | 30 | 20 | 40 | 30 | 20/30 |
| Overweight billing (kg) | $price_0 * 1.5\%$ | | | $price_0 * 1.5\%$ | | |

TABLE 3
Cell-phone plan A

| cell-phone plan (CNY) | | 46 | 66 | 96 | 126 | 156 | 186 | 226 | 286 | 386 | 586 | 886 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| contains | free calls (min) | 50 | 50 | 240 | 320 | 420 | 510 | 700 | 900 | 1250 | 1950 | 3000 |
| | free date (MB) | 150 | 300 | 300 | 400 | 500 | 650 | 750 | 950 | 1300 | 2000 | 3000 |
| | free incoming calls | domestic (including video calls) | | | | | | | | | | |
| overuse | incoming calls (CNY/min) | 0.25 | 0.20 | 0.15 | | | | | | | | |
| | date (CNY/KB) | 0.0003 | | | | | | | | | | |
| | video calls (CNY/min) | 0.60 | | | | | | | | | | |

TABLE 6
Hourly parking rates

| Actual hours | Hourly parking rates | | | | | |
|---|---|---|---|---|---|---|
| | Weekday | | | Saturday and Sunday | | |
| | Motorcycle | Car:2-door coupe | Car:others | Motorcycle | Car:2-door coupe | Car:others |
| $(0.0, 2.0]$ | 4.00$ | 4.50$ | 5.00$ | 5.00$ | 6.00$ | 7.00$ |
| $(2.0, 4.0]$ | 5.00$ | 5.50$ | 6.00$ | 6.50$ | 7.50$ | 8.50$ |
| $(4.0, 24.0]$ | 6.00$ | 6.50$ | 7.00$ | 8.00$ | 9.00$ | 10.00$ |

## 4.3 Variables

### 4.3.1 Independent Variables

The independent variable in our study is the testing technique. The DRT technique must be chosen for this variable. In addition, we selected RPT and RT as the baseline techniques for comparison.

### 4.3.2 Dependent Variables

The dependent variable for RQ1 is the metric for evaluating the fault-detection effectiveness. There exist quite a few effectiveness metrics, such as the P-measure (the probability of at least one fault being detected by a test suite), the E-measure (the expected number of faults being detected by a test suite), the F-measure (the expected number of test cases to detect the first fault), and the T-measure (the expected number of test cases to detect all faults). Among them, the F-measure and T-measure are the most appropriate metrics for measuring the fault-detection effectiveness of DRT testing techniques. In our study, we use $F_{method}$, $T_{method}$ to represent the F-measure and the T-measure of a testing method, where $method$ can be DRT, RPT, and RT.

As shown in Algorithm 1, the testing process may not be terminated after the detection of the first fault. In addition, the fault detection information can lead to different probability profile adjustment mechanisms. Therefore, it is also important to see what would happen after the first fault is revealed. In our study, we introduce a new metric F2-measure, which refers to how many additional test cases are required to reveal the second fault after the detection of the first fault. Similarly, we use $F2_{method}$ to represent the F2-measure of a testing method.

For RQ3, an obvious metric is the time required on detecting faults. In this study, corresponding to the T-measure, we used the $T\text{-}time$ to measure the time for detecting all faults, respectively. Similarly, $F\text{-}time$, $F2\text{-}time$, is used to denote these time metrics.

For each of the above four metrics, a smaller value intuitively implies a better performance.

## 4.4 Experimental Settings

### 4.4.1 Partitioning

In our study, we made use of the decision table [23], [24], a typical PT technique, to conduct the partitions. A decision table is based on a simple principle: sets of responses for sets of conditions. It is used to present a large quantity of complex information in a simple, straightforward manner. Usually, a decision table is composed of the constrain part and action part. The constraint part specifies valid domains to input parameters and additional constraints. there constraints are the pre-conditions of a contract and have to be resolvable to true or false. The action part specifies valid system responses with respect to constrains ~~sets~~, called *rules*. The action part represents the post-conditions of a contract. Thus, every rule of the decision table defines a *contract*. In our study, every *contract* corresponds to a partition.

After constructing decision table, a test case can be generated by a rule. To investigate the performance of three testing techniques under various scenarios, we develop two partition schemas shown in Table 7 :

- **Scheme 1**: every *contract* in the decision table corresponds to a partition.
- **Scheme 2**: After constructing the decision table, we group those *contract* with the same responses, then

every *contract* in the decision table corresponds to a partition.

TABLE 7
Two partition schemas

| web services | Scheme 1 | Scheme 2 |
|---|---|---|
| ACMS | 24 | 7 |
| CUBS | 20 | 3 |
| PBS | 18 | 3 |

#### 4.4.2 Initial Test Profile

Since the test cases are randomly generated during the test processing, it is a conservative and feasible method to use the equal probability distribution as the initial testing profile. On the other hand, testers can also be based on past experience to use a unequal probability distribution as the initial testing profile.

#### 4.4.3 Constants

In the experiments, we explore the relationship between the number of partitions and parameter of DRT strategy. Therefore, we design a set of parameter values, $\epsilon \in \{1.0E\text{-}05, 5.0E\text{-}05, 1.0E\text{-}04, 5.0E\text{-}04, 1.0E\text{-}03, 5.0E\text{-}03, 1.0E\text{-}02, 5.0E\text{-}02, 1.0E\text{-}01, 2E\text{-}01, 3E\text{-}01, 4E\text{-}01, 5E\text{-}01\}$. Note that $\epsilon = 5E\text{-}01$ is already a big one. Consider this Scenario: In PBS, when the test is carried out under the partition scheme 2, if we set $\epsilon = 7.5E\text{-}01$ and equal probability distribution as testing profile, that is, $p_i = 1/3$. Suppose that the first test case belong to $c_1$ was executed and did not reveal any faults, then the value of $p_1$ would be 0 according to Formula 3. Obviously, it is not unreasonable. The value of $\epsilon$ should not be too big.

### 4.5 Experimental Environment

Our experiments were conducted on a virtual machine running the Ubuntu 11.06 64-bit operating system. In this system, there were two CPUs and a memory of 2GB. The test scripts were generated using Java. In the experiments, we repeatedly run the testing using each technique for 30 times with 30 seeds to guarantee the statistically reliable mean values of the metrics (F-measure, F2-measure, T-measure, F-time, F2-time, T-time).

### 4.6 Threats To Validity

#### 4.6.1 Internal Validity

The threat to internal validity is related to the implementations of the testing techniques, which involved a moderate amount of programming work. The code was also cross-checked by different individuals. We are confident that all techniques were correctly implemented.

#### 4.6.2 External Validity

One obvious threat to external validity is that we only considered three object web services. However, they are real-life web services. In addition, 17 distinct faults were used to evaluate the performance. Though we have tried to improve the generality by applying different granularities in partitioning and 13 kinds of parameters, we cannot say whether or not similar results would be observed on other types of web services.

#### 4.6.3 Construct Validity

The metrics used in our study are simple in concept and straightforward to apply, hence the threat to construct validity is little.

#### 4.6.4 Conclusion Validity

We have run a sufficient number of trials to guarantee the statistical reliability of our experimental results. In addition, as to be discussed in Section 5, statistical tests were conducted to verify the significance of our results.

## 5 Experimental Results

### 5.1 RQ1: Fault detection effectiveness

The experimental results of F-measure, F2-measure and T-measure are summarized in Tables 8 to 10. The distributions of F-measure, F2-measure and T-measure on each object program are displayed by the boxplots in Figures 3 to 5. In the boxplot, the upper and lower bounds of the box represent the third and first quartiles of a metric, respectively, and the middle line denote the median value. The upper and lower whiskers respectively indicate the largest and smallest data within the range of $\pm 1.5 \times IQR$, where $IQR$ is the interquartile range. The outliers outside $IQR$ are denoted by hollow circles. The solid circle represents the mean value of a metric.

From Tables 8 to 10 and Figures 3 to 5, we can observe that in general, DRT was the best performer in terms of F-measure, F2-measure, and T-measure, followed by RPT. We further conducted statistical testing to verify the significance of this observation. We used the Holm-Bonferroni method [25] (with p-value being 0.05) to determine which pair of testing techniques have significant difference. The statistical testing results are shown in Tables 11, 12 and 13. Each cell in the tables gives the number of scenarios where the technique on the top row performed better than that on the left column. If the difference is significant, the number will be displayed in bold. For example, the bold 76 in the top right corner in Table 13 indicates that out of 78 scenarios (13 parameters $\times$ 2 partition schemas $\times$ 3 object web services), DRT had smaller T-measure than RT for 76 scenarios, and the fault-detection capabilities of these two techniques were significantly different.

TABLE 11
Number of scenarios where the technique on top row has smaller F-measure than that on left column

| | RT | RPT | DRT |
|---|---|---|---|
| RT | — | **4** | **71** |
| RPT | **2** | — | **64** |
| DRT | **7** | **14** | — |

TABLE 12
Number of scenarios where the technique on top row has smaller F2-measure than that on left column

| | RT | RPT | DRT |
|---|---|---|---|
| RT | — | **5** | **78** |
| RPT | **1** | — | **74** |
| DRT | **0** | **4** | — |

TABLE 8
The results of ACMS

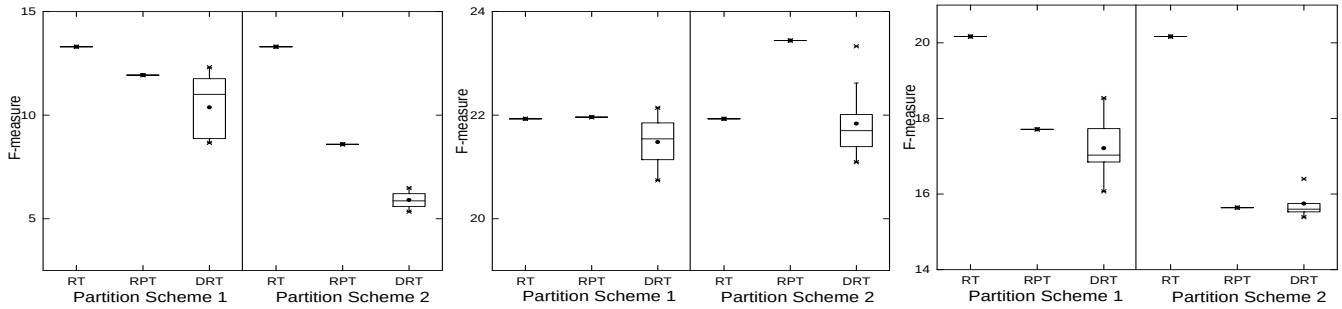
| Strategy | | Partition Schema 1 | | | | | | Partition Schema 2 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | F | $SD_F$ | F2 | $SD_{F2}$ | T | $SD_T$ | F | $SD_F$ | F2 | $SD_{F2}$ | T | $SD_T$ |
| RT | | 13.30 | 10.34 | 28.50 | 23.95 | 41.80 | 27.13 | 13.30 | 10.34 | 28.50 | 23.95 | 41.80 | 27.13 |
| RPT | | 11.93 | 11.24 | 24.36 | 22.56 | 35.99 | 25.02 | 8.59 | 8.02 | 22.47 | 19.97 | 27.06 | 18.73 |
| DRT | 1.0E-5 | 12.31 | 11.89 | 24.47 | 24.20 | 36.78 | 26.64 | 5.80 | 6.25 | 20.27 | 17.93 | 26.08 | 19.03 |
| | 5.0E-5 | 11.96 | 11.62 | 22.52 | 21.72 | 34.48 | 24.69 | 6.00 | 6.90 | 21.53 | 19.18 | 27.54 | 20.15 |
| | 1.0E-4 | 11.76 | 11.95 | 23.84 | 23.24 | 35.60 | 26.03 | 6.48 | 7.84 | 20.54 | 18.87 | 27.01 | 20.04 |
| | 5.0E-4 | 11.46 | 10.58 | 22.73 | 22.21 | 34.19 | 23.64 | 6.21 | 7.26 | 21.45 | 19.66 | 27.66 | 20.32 |
| | 1.0E-3 | 12.02 | 11.60 | 22.19 | 21.88 | 34.21 | 24.42 | 6.32 | 7.46 | 21.10 | 19.69 | 27.43 | 20.80 |
| | 5.0E-3 | 11.00 | 9.82 | 20.64 | 19.88 | 31.64 | 21.27 | 6.14 | 6.85 | 20.53 | 18.69 | 26.68 | 19.63 |
| | 1.0E-2 | 11.01 | 9.66 | 18.32 | 15.45 | 29.33 | 18.12 | 5.59 | 5.99 | 20.35 | 18.36 | 25.94 | 19.26 |
| | 5.0E-2 | 8.87 | 6.60 | 13.45 | 10.56 | 22.31 | 11.53 | 6.26 | 7.09 | 20.85 | 17.71 | 26.34 | 18.53 |
| | 1.0E-1 | 9.26 | 6.95 | 12.95 | 9.81 | 22.21 | 11.12 | 5.86 | 5.95 | 21.83 | 17.85 | 26.73 | 18.34 |
| | 2.0E-1 | 9.00 | 6.67 | 13.21 | 9.50 | 22.21 | 10.61 | 5.82 | 6.81 | 22.21 | 18.59 | 26.63 | 18.97 |
| | 3.0E-1 | 8.66 | 6.54 | 13.69 | 9.59 | 21.55 | 10.50 | 5.34 | 6.34 | 22.11 | 18.34 | 26.50 | 18.75 |
| | 4.0E-1 | 8.80 | 6.61 | 13.38 | 9.80 | 22.18 | 10.82 | 5.45 | 6.17 | 22.11 | 17.50 | 26.63 | 18.00 |
| | 5.0E-1 | 8.76 | 6.41 | 13.57 | 10.29 | 22.33 | 11.10 | 5.51 | 5.42 | 22.22 | 17.68 | 26.66 | 18.12 |

TABLE 9
The results of CUBS

| Strategy | | Partition Schema 1 | | | | | | Partition Schema 2 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | F | $SD_F$ | F2 | $SD_{F2}$ | T | $SD_T$ | F | $SD_F$ | F2 | $SD_{F2}$ | T | $SD_T$ |
| RT | | 21.93 | 20.37 | 38.17 | 38.17 | 4203.07 | 3219.40 | 21.93 | 20.37 | 38.17 | 38.17 | 4203.07 | 3219.40 |
| RPT | | 21.96 | 20.36 | 28.74 | 27.56 | 2590.38 | 1768.49 | 23.44 | 23.40 | 26.60 | 24.81 | 4190.39 | 2466.12 |
| DRT | 1.0E-5 | 20.83 | 21.16 | 26.81 | 28.42 | 2521.96 | 1866.25 | 21.88 | 20.52 | 26.81 | 26.52 | 4166.31 | 2708.14 |
| | 5.0E-5 | 21.14 | 20.89 | 25.50 | 27.22 | 2523.77 | 1808.06 | 21.39 | 20.49 | 26.01 | 26.23 | 4188.00 | 2798.73 |
| | 1.0E-4 | 21.64 | 20.66 | 26.82 | 26.59 | 2497.56 | 1774.01 | 21.60 | 20.64 | 27.02 | 27.24 | 4077.07 | 2668.71 |
| | 5.0E-4 | 21.29 | 21.22 | 28.37 | 30.68 | 2538.89 | 1882.89 | 22.09 | 22.20 | 25.23 | 26.04 | 4073.15 | 2518.34 |
| | 1.0E-3 | 20.74 | 19.64 | 26.60 | 28.85 | 2449.44 | 1615.91 | 21.95 | 21.01 | 25.35 | 25.45 | 4224.83 | 2582.71 |
| | 5.0E-3 | 20.95 | 19.68 | 25.96 | 27.18 | 2487.14 | 1778.65 | 21.09 | 21.89 | 24.80 | 24.45 | 4045.44 | 2329.75 |
| | 1.0E-2 | 21.54 | 20.94 | 26.42 | 27.05 | 2579.27 | 1809.45 | 21.39 | 19.16 | 25.49 | 26.54 | 3991.68 | 2377.37 |
| | 5.0E-2 | 22.07 | 21.08 | 25.37 | 27.13 | 2662.38 | 1931.18 | 21.46 | 21.95 | 25.56 | 25.70 | 4005.80 | 2557.51 |
| | 1.0E-1 | 21.34 | 21.39 | 27.14 | 28.41 | 2539.91 | 1752.16 | 22.62 | 22.79 | 25.63 | 25.92 | 4016.92 | 2371.21 |
| | 2.0E-1 | 21.75 | 20.10 | 25.39 | 26.44 | 2515.45 | 1780.26 | 23.33 | 21.67 | 26.15 | 25.70 | 4002.59 | 2554.38 |
| | 3.0E-1 | 21.94 | 21.36 | 25.93 | 25.58 | 2531.10 | 1860.04 | 22.01 | 21.24 | 26.56 | 27.54 | 4068.50 | 2448.81 |
| | 4.0E-1 | 22.14 | 21.45 | 26.39 | 27.45 | 2486.92 | 1732.76 | 21.34 | 22.77 | 27.37 | 29.88 | 4215.41 | 2991.54 |
| | 5.0E-1 | 21.85 | 19.91 | 26.33 | 28.81 | 2490.95 | 1703.39 | 21.70 | 22.09 | 25.62 | 26.15 | 4097.77 | 2598.52 |

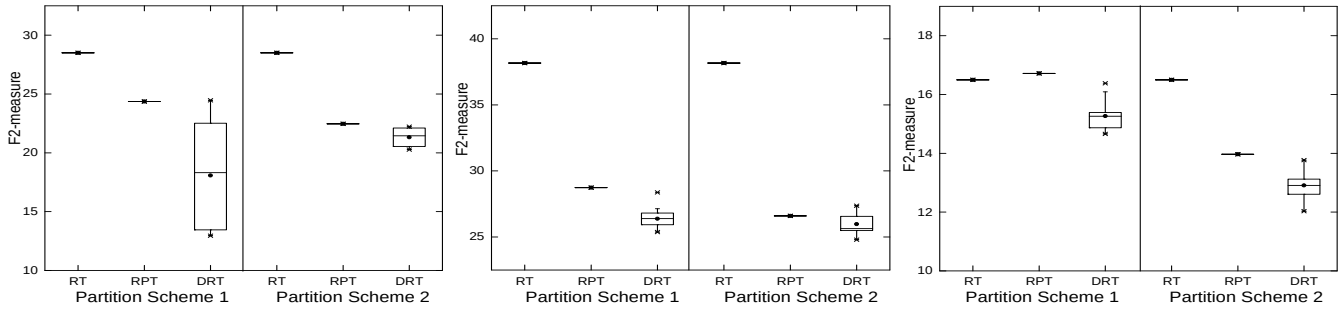
TABLE 10
The results of PBS

| Strategy | | Partition Schema 1 | | | | | | Partition Schema 2 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | F | $SD_F$ | F2 | $SD_{F2}$ | T | $SD_T$ | F | $SD_F$ | F2 | $SD_{F2}$ | T | $SD_T$ |
| RT | | 20.17 | 16.32 | 16.50 | 13.20 | 252.80 | 191.54 | 20.17 | 16.32 | 16.50 | 13.20 | 252.80 | 191.54 |
| RPT | | 17.71 | 16.78 | 16.72 | 25.12 | 178.91 | 147.96 | 15.64 | 13.40 | 13.97 | 22.24 | 175.00 | 126.88 |
| DRT | 1.0E-5 | 16.93 | 17.55 | 16.38 | 23.38 | 169.63 | 136.31 | 15.68 | 13.62 | 12.91 | 21.87 | 173.13 | 133.79 |
| | 5.0E-5 | 17.03 | 19.15 | 14.77 | 22.93 | 174.06 | 141.08 | 16.40 | 14.88 | 11.43 | 19.18 | 172.53 | 135.13 |
| | 1.0E-4 | 16.45 | 15.03 | 16.09 | 24.73 | 167.90 | 137.66 | 15.73 | 14.11 | 12.73 | 19.52 | 172.39 | 131.70 |
| | 5.0E-4 | 16.85 | 17.13 | 15.39 | 22.63 | 174.46 | 142.88 | 16.23 | 14.74 | 13.12 | 21.00 | 171.36 | 127.36 |
| | 1.0E-3 | 16.07 | 16.99 | 14.66 | 21.23 | 177.75 | 149.88 | 15.55 | 13.66 | 13.46 | 21.85 | 174.54 | 125.99 |
| | 5.0E-3 | 17.61 | 18.54 | 15.26 | 23.70 | 179.77 | 157.27 | 15.39 | 15.05 | 12.91 | 21.63 | 169.50 | 123.41 |
| | 1.0E-2 | 17.08 | 19.09 | 15.36 | 27.05 | 173.48 | 139.33 | 15.53 | 14.79 | 12.03 | 22.23 | 168.55 | 127.52 |
| | 5.0E-2 | 17.98 | 17.27 | 15.37 | 23.91 | 174.42 | 142.11 | 16.35 | 14.43 | 13.12 | 22.20 | 180.89 | 133.50 |
| | 1.0E-1 | 17.73 | 18.47 | 15.80 | 22.54 | 176.90 | 141.45 | 15.75 | 13.12 | 12.39 | 20.52 | 173.45 | 132.78 |
| | 2.0E-1 | 17.83 | 18.24 | 14.87 | 23.25 | 178.63 | 140.00 | 15.39 | 14.23 | 13.77 | 21.71 | 176.68 | 140.26 |
| | 3.0E-1 | 16.95 | 17.13 | 14.96 | 21.82 | 175.69 | 143.24 | 15.60 | 14.83 | 12.81 | 20.88 | 172.45 | 128.59 |
| | 4.0E-1 | 16.74 | 17.07 | 14.67 | 23.10 | 178.32 | 137.52 | 15.59 | 13.94 | 13.36 | 20.81 | 171.18 | 125.94 |
| | 5.0E-1 | 18.54 | 18.56 | 14.88 | 23.07 | 178.21 | 138.59 | 15.53 | 14.44 | 13.09 | 21.77 | 172.31 | 128.52 |



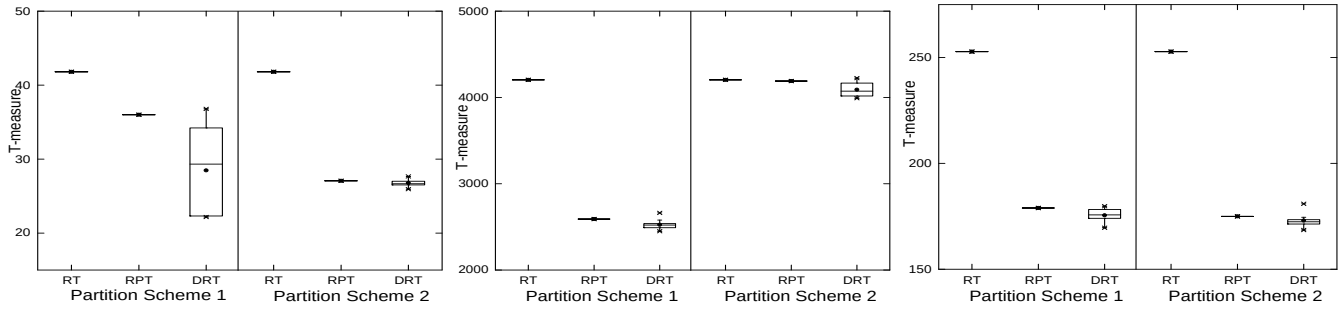

(a) ACMS (b) CUBS (c) PBS

Fig. 3. Boxplots of F-measures on each object program




(a) ACMS (b) CUBS (c) PBS

Fig. 4. Boxplots of F2-measures on each object program




(a) ACMS (b) CUBS (c) PBS

Fig. 5. Boxplots of T-measures on each object program

TABLE 13
Number of scenarios where the technique on top row has smaller T-measure than that on left column

| | RT | RPT | DRT |
|---|---|---|---|
| RT | — | **6** | **76** |
| RPT | **0** | — | **69** |
| DRT | **2** | **9** | — |

Tables 11, 12 and 13 clearly show that the effectiveness difference between each pair of testing techniques was always significantly different.

### 5.2 RQ2: Relationship between partition number and $\epsilon$

In 3.2, we have analyzed the relationship between partition numbers and parameter. This section we validate that our theoretical analysis is reasonable to instruct testers set parameter of DRT strategy.

In order to study how the value of $h$ affects the test efficiency of DRT technology, 5 parameters were set $h \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. According to Formula 14, the values of $\epsilon$ in each scenario for all objective web services show in Table 14.

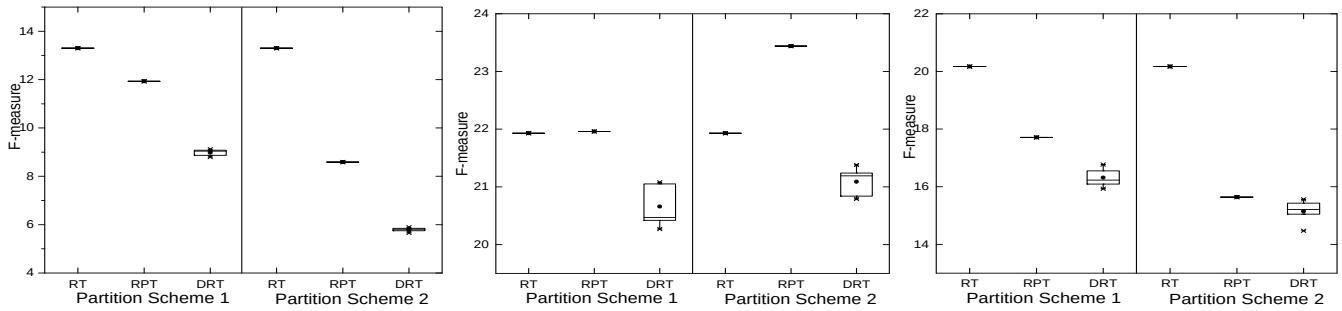
We used three web services to validate our theoretical analysis. Before starting our test, the failure rates of each partition are obtained by running test cases which belong to corresponding partition until revealed a fault and computing the test cases used. We develop a series of experiments with parameters in accordance with Table 14, and the distributions of F-measure, F2-measure and T-measure on each object program are displayed by the Figures 6 to 8. Except for the parameters of DRT strategy, other experimental Settings are the same as 5.1.

From Figures 6 to 8 and Table 14, we have the following observations:

- From Figures 6 to 8, it is obvious that the boxs of

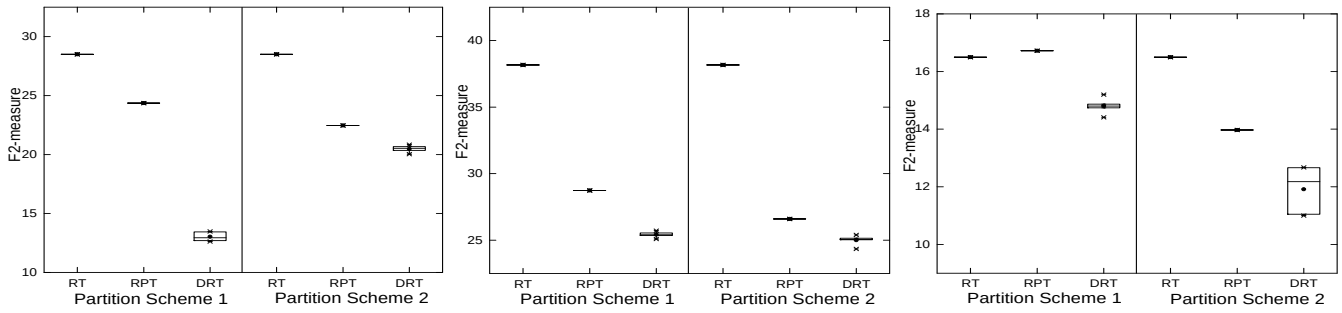
TABLE 14
The theoretical value Of DRT parameters

| web services | partition schema | $\theta_M$ | $\theta_\Delta$ | $h$ | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | 0.1 | 0.3 | 0.5 | 0.7 | 0.9 |
| ACMS | 1 | 6.452E-2 | 5.452E-2 | 1.274E-1 | 1.229E-1 | 1.188E-1 | 1.149E-1 | 1.113E-1 |
| | 2 | 4.718E-3 | 2.797E-3 | 8.841E-3 | 7.833E-3 | 7.031E-3 | 6.378E-3 | 5.836E-3 |
| CUBS | 1 | 2.332E-2 | 1.112E-2 | 4.218E-3 | 3.515E-3 | 3.016E-3 | 2.642E-2 | 2.349E-2 |
| | 2 | 6.987E-3 | 1.397E-3 | 1.001E-2 | 6.364E-3 | 4.664E-3 | 3.680E-3 | 3.040E-3 |
| PBS | 1 | 4.001E-3 | 3.827E-3 | 7.969E-3 | 7.897E-3 | 7.827E-3 | 7.758E-3 | 7.691E-3 |
| | 2 | 2.516E-3 | 1.492E-3 | 1.001E-2 | 6.364E-3 | 4.664E-3 | 3.680E-3 | 3.040E-3 |



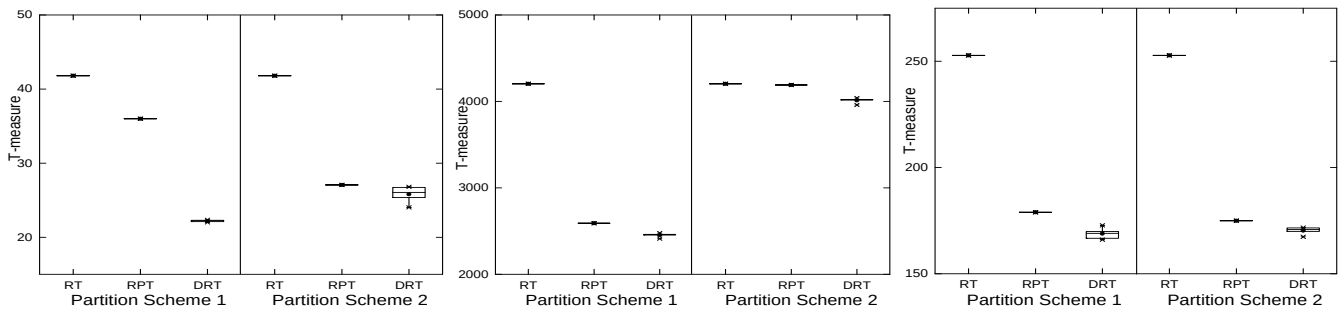

(a) ACMS (b) CUBS (c) PBS

Fig. 6. Boxplots of F-measure on each object program with theoretical parameter




(a) ACMS (b) CUBS (c) PBS

Fig. 7. Boxplots of F2-measure on each object program with theoretical parameter




(a) ACMS (b) CUBS (c) PBS

Fig. 8. Boxplots of T-measure on each object program with theoretical parameter

DRT strategy with different theoretical parameters are very narrow, that is, no matter the value of $h(0 < h < 1)$ is, the ability of DRT's detecting faults is very approximate.

- It is clear that DRT strategy with theoretical parameter has better effectiveness than RT and RPT. Moreover, even in scenarios where the DRT strategy performs worst, its fault detection efficiency is still better than RT and RPT.

On the other hand, when the value of $\epsilon$ in DRT is very small(such as $1.0E$-$06, 1.0E$-$07$), even many test cases were executed then the testing profile is almost constant. In this scenario, DRT may be as efficient as RPT. On the contrary, if the value of parameter in DRT is very big(such as $0.7, 0.8$), it has high probability that when a test case were selected from a partition $s_i$, and it can not reveal any faults, the value of $p_i$ is $0$. Therefore, the value of parameter in DRT should not be too large or too small. We want to explore the different performance of DRT with parameter in theoretical interval and out of theoretical interval. Results in Figures 9 to 11, show that in most of scenarios, DRT with theoretical parameter have a higher efficiency, and the theoretical interval is generally narrow. Further, our results show that the efficiency of DRT with different theoretical values is similar. Conservatively, we can set $h = 0.5$ for a concrete software under testing.

### 5.3 RQ3: Selection Overhead

Tables 15 to 17 summarize the experimental results of $F$-$time$,$F2$-$time$, and $T$-$time$, respectively. The results of distribution on every web services are showed in Figures 12 to 14.

TABLE 15
F-time, F2-time and T-time on web service ACMS (in ms)

| Strategy | | Partition Schema 1 | | | Partition Schema 2 | | |
|---|---|---|---|---|---|---|---|
| | | F-time | F2-time | T-time | F-time | F2-time | T-time |
| RT | | 5.04 | 5.41 | 28.84 | 5.04 | 5.41 | 28.84 |
| RPT | | 4.89 | 5.84 | 30.23 | 3.43 | 4.94 | 25.50 |
| DRT | 1.0E-5 | 5.06 | 5.87 | 30.93 | 2.32 | 5.63 | 24.78 |
| | 5.0E-5 | 4.92 | 5.40 | 29.00 | 2.40 | 5.18 | 26.16 |
| | 1.0E-4 | 4.83 | 5.72 | 29.94 | 2.59 | 5.48 | 25.66 |
| | 5.0E-4 | 4.71 | 5.46 | 28.75 | 2.48 | 5.23 | 26.28 |
| | 1.0E-3 | 4.94 | 5.33 | 28.77 | 2.53 | 5.10 | 26.06 |
| | 5.0E-3 | 4.52 | 4.95 | 26.61 | 2.46 | 4.75 | 25.35 |
| | 1.0E-2 | 4.53 | 4.40 | 24.67 | 2.24 | 4.21 | 24.64 |
| | 5.0E-2 | 3.65 | 3.23 | 18.76 | 2.20 | 3.09 | 25.02 |
| | 1.0E-1 | 3.81 | 3.11 | 18.68 | 1.96 | 2.98 | 25.39 |
| | 2.0E-1 | 3.70 | 3.17 | 18.68 | 1.77 | 3.04 | 25.30 |
| | 3.0E-1 | 3.56 | 3.09 | 18.12 | 1.76 | 2.96 | 25.18 |
| | 4.0E-1 | 3.62 | 3.21 | 18.65 | 1.81 | 3.08 | 25.30 |
| | 5.0E-1 | 3.60 | 3.26 | 18.78 | 1.78 | 3.12 | 25.33 |

Similarly, we use the Holm-Bonferroni method to check the different between each pair of testing strategies in terms of $F$-$time$, $F2$-$time$, and $T$-$time$, showing in Tables 18 to 20.

We can observe that DRT only slightly outperformed RPT from Tables 18 to 20, but DRT significantly better than RT especially in term of $T$-$time$.

In short, DRT strategy was considered as the best testing technique across all six metrics, and RPT was marginally outperformed RT.

TABLE 16
F-time, F2-time and T-time on web service CUBS (in ms)

| Strategy | | Partition Schema 1 | | | Partition Schema 2 | | |
|---|---|---|---|---|---|---|---|
| | | F-time | F2-time | T-time | F-time | F2-time | T-time |
| RT | | 13.78 | 19.07 | 2067.46 | 13.78 | 19.07 | 2067.46 |
| RPT | | 13.84 | 14.34 | 1331.54 | 13.72 | 13.74 | 2036.33 |
| DRT | 1.0E-5 | 13.36 | 15.51 | 1527.19 | 13.53 | 14.06 | 2046.32 |
| | 5.0E-5 | 14.03 | 15.53 | 1433.49 | 13.48 | 13.9 | 2107.06 |
| | 1.0E-4 | 12.79 | 13.61 | 1327.54 | 13.88 | 14.02 | 1988.05 |
| | 5.0E-4 | 13.23 | 14.74 | 1385.40 | 14.80 | 13.93 | 2013.44 |
| | 1.0E-3 | 12.52 | 13.97 | 1380.07 | 13.43 | 14.71 | 1926.19 |
| | 5.0E-3 | 13.16 | 14.47 | 1365.75 | 13.87 | 13.32 | 2027.36 |
| | 1.0E-2 | 13.82 | 14.44 | 1329.04 | 14.63 | 13.45 | 1975.64 |
| | 5.0E-2 | 13.66 | 15.53 | 1250.44 | 14.87 | 13.11 | 1918.87 |
| | 1.0E-1 | 13.15 | 14.11 | 1230.99 | 14.25 | 13.38 | 1983.51 |
| | 2.0E-1 | 13.38 | 14.72 | 1279.54 | 13.75 | 13.24 | 2053.69 |
| | 3.0E-1 | 14.12 | 13.52 | 1237.68 | 15.34 | 13.19 | 1962.61 |
| | 4.0E-1 | 14.02 | 13.71 | 1219.80 | 13.84 | 14.11 | 1966.21 |
| | 5.0E-1 | 13.80 | 15.17 | 1200.34 | 14.67 | 13.62 | 2073.06 |

TABLE 17
F-time, F2-time and T-time on web service PBS (in ms)

| Strategy | | Partition Schema 1 | | | Partition Schema 2 | | |
|---|---|---|---|---|---|---|---|
| | | F-time | F2-time | T-time | F-time | F2-time | T-time |
| RT | | 12.39 | 6.77 | 129.82 | 12.39 | 6.77 | 129.82 |
| RPT | | 11.32 | 9.63 | 90.46 | 9.28 | 7.22 | 89.14 |
| DRT | 1.0E-5 | 11.46 | 9.46 | 86.19 | 8.54 | 7.29 | 88.47 |
| | 5.0E-5 | 11.46 | 10.46 | 88.74 | 8.50 | 6.30 | 88.16 |
| | 1.0E-4 | 10.47 | 8.47 | 85.17 | 7.50 | 7.29 | 86.05 |
| | 5.0E-4 | 9.49 | 9.49 | 88.74 | 9.52 | 7.28 | 87.56 |
| | 1.0E-3 | 10.48 | 8.48 | 90.27 | 8.51 | 7.27 | 89.19 |
| | 5.0E-3 | 12.42 | 8.42 | 91.29 | 9.50 | 6.29 | 92.43 |
| | 1.0E-2 | 11.69 | 9.40 | 88.23 | 9.54 | 7.28 | 88.63 |
| | 5.0E-2 | 10.59 | 9.42 | 88.74 | 10.55 | 7.28 | 86.61 |
| | 1.0E-1 | 10.67 | 8.39 | 89.76 | 8.54 | 6.30 | 86.13 |
| | 2.0E-1 | 10.62 | 8.41 | 90.78 | 8.53 | 7.26 | 90.28 |
| | 3.0E-1 | 11.65 | 8.40 | 89.25 | 9.55 | 6.28 | 88.12 |
| | 4.0E-1 | 12.68 | 8.39 | 85.86 | 9.49 | 7.29 | 87.47 |
| | 5.0E-1 | 11.72 | 8.44 | 91.07 | 9.52 | 6.33 | 88.05 |

TABLE 18
Number of scenarios where the technique on top row has smaller F-time than that on left column

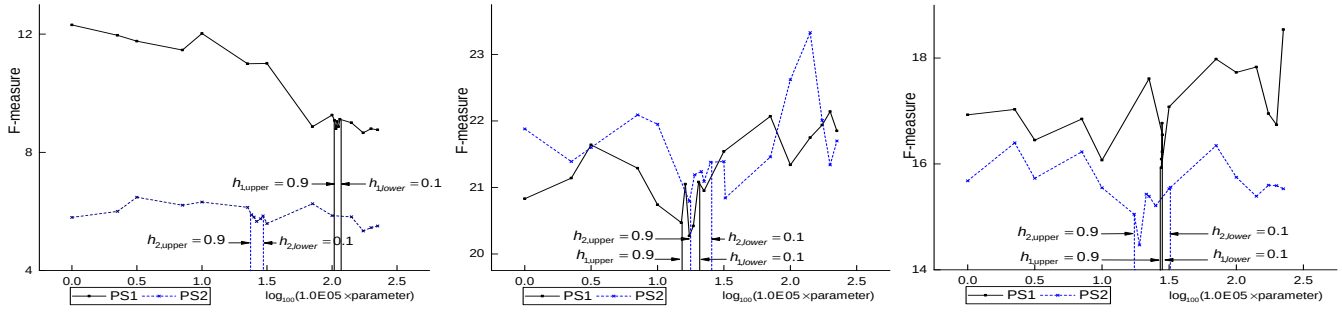
| | RT | RPT | DRT |
|---|---|---|---|
| RT | — | **5** | **62** |
| RPT | **1** | — | **47** |
| DRT | **16** | **31** | — |

TABLE 19
Number of scenarios where the technique on top row has smaller F2-time than that on left column

| | RT | RPT | DRT |
|---|---|---|---|
| RT | — | **3** | **52** |
| RPT | **3** | — | **48** |
| DRT | **26** | **30** | — |

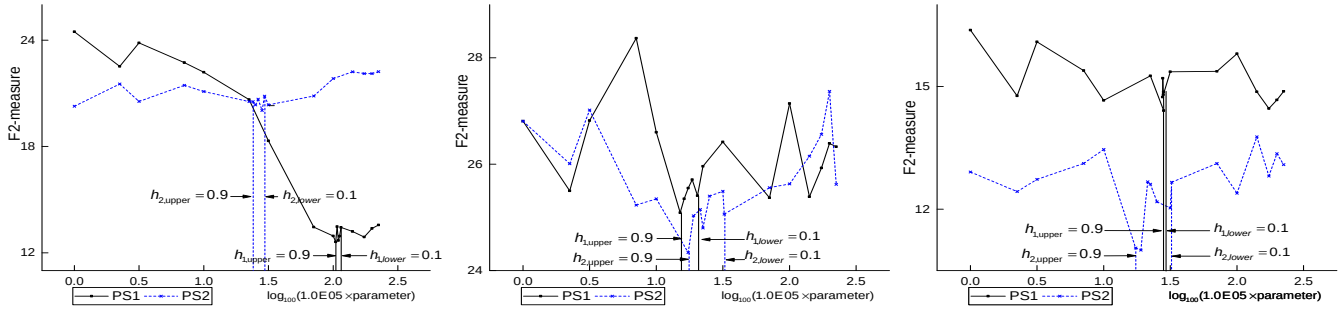
TABLE 20
Number of scenarios where the technique on top row has smaller T-time than that on left column

| | RT | RPT | DRT |
|---|---|---|---|
| RT | — | **5** | **73** |
| RPT | **1** | — | **58** |
| DRT | **3** | **20** | — |



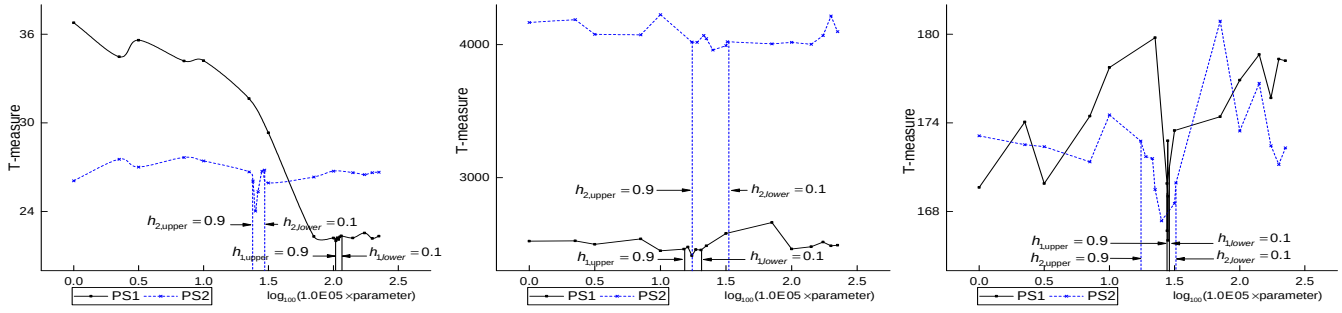

(a) ACMS (b) CUBS (c) PBS

Fig. 9. Boxplots of F-measure on each object program with theoretical parameter



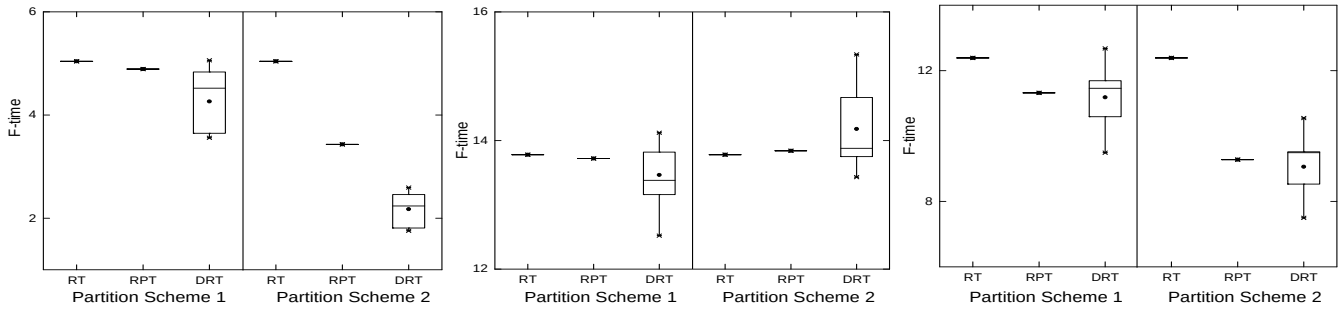

(a) ACMS (b) CUBS (c) PBS

Fig. 10. Boxplots of F2-measure on each object program with theoretical parameter



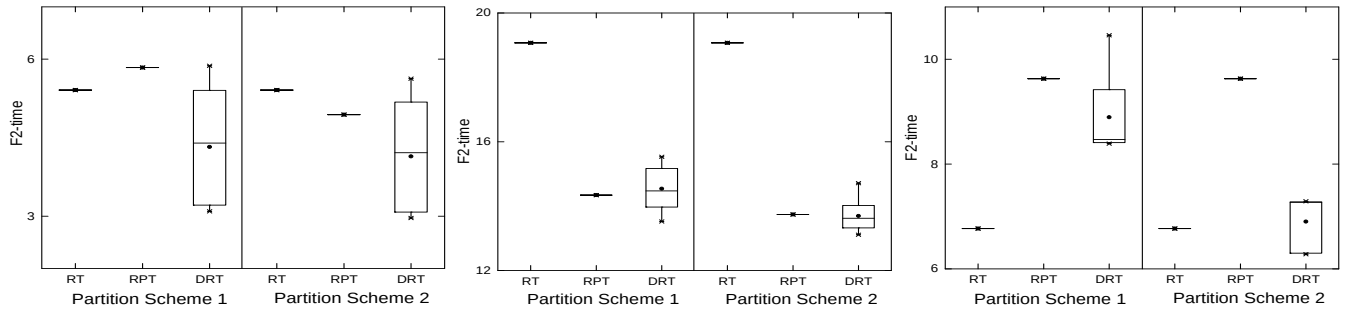

(a) ACMS (b) CUBS (c) PBS

Fig. 11. Boxplots of T-measure on each object program with theoretical parameter



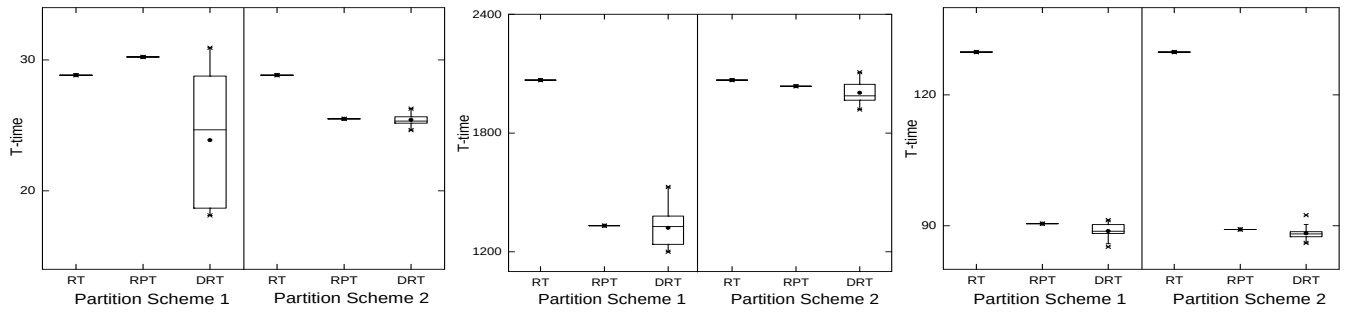

(a) ACMS (b) CUBS (c) PBS

Fig. 12. Boxplots of F-time on each object program




(a) ACMS (b) CUBS (c) PBS

Fig. 13. Boxplots of F2-time on each object program




(a) ACMS (b) CUBS (c) PBS

Fig. 14. Boxplots of T-time on each object program

## 6 Related Work

In this section, we describe related work in two dimensions, one is related to testing techniques for web services, and the other is related to improvements on RT and PT.

### 6.1 Testing Techniques For Web Services

In recent years, many efforts have been made test web services. Test case generation that includes generating and selecting test cases, is core procedure to test web services. Model-based [26] and specification-based techniques are two common test case generation methods. Before service providers put services on Internet, testers can use model-based technique to verify the behaviors of WSUT meet their requirements. In this technique, test data are generated from a data model that is a specification of the inputs to the software, and can be built before or parallel to the software development process. There are quite a few verification methods using models such as theorem proving, model-checking and Petri-Nets can also be used as model-based testing approaches. Sinha et al. [27] generated test cases using theorem proving, where they used of existing test generation methods based on Extended Finite State Machine(EFSM) specification. Majdi et al. [28] generate test data using model checking. Dong et al. [29] combined Petri-Nets (HPN) with constraint-based test case generation to acquire high fault coverage. On the other hand, the specification of web service is only information that users can receive. Therefore, specification-based testing becomes a natural choice. In traditional web service, the specification of web service is named WSDl, and provides information about available operations and its parameters. Many methods proposed for WSDL-based test data generation are based on the XML schema data type. Hanna and Munro proposed a framework that can be used to test the robustness quality attribute of a web service [30]. This framework is based on analyzing the web service Description Language (WSDL) document of web services to identify what faults could affect the robustness attribute and then test cases were designed to detect those faults.

As for selecting test cases, it is pity that there are not many works involving to select test cases. Simple RT may not be a good strategy, because it does not use any information of WSUT and testing process. Our approach introduce software cybernetics into partition testing. In DRT, the strategy of selecting a partition is according to testing profile, and the testing profile is update during the test process. The main advantage of DRT is that partitions with bigger failure rates have higher selected probability.

Besides, there are some frameworks to illustrate how the test task is completed. Zhu and Zhang [31] proposed a framework of collaborative testing. In this framework, test tasks are completed by collaborating various test services. A test service is a service assigned to perform various test task. This framework focus on complete test task. However, our framework proposed in section 3.1 aims to find more faults of WSUT. In our framework, the result of current test case execution feedback to the control system in order that the next test case selected has a bigger chance to reveal faults of WSUT.

Most existing testing techniques for web services assume that for each test case, the computed output is verifiable. However, the assumption is not always true in practice and thus these testing techniques may be inapplicable in some

situations. To address the outstanding oracle problem with testing web services, a metamorphic testing technique was proposed, which not only alleviates the oracle problem, but also presents a feasible and efficient choice for testing web services [22].

The proposed DRT for web services has the following advantages: (1) it is easier to use; (2) it is more efficient because it enhances partition testing with dynamic testing profiles updates and the resulting overhead is very limited. DRT is applicable whenever partition testing is applicable.

### 6.2 Improvement On RT And PT

Based on the observation that failure causing inputs tend to cluster into contiguous regions within the input domain [11], [12], many works have been done to improve RT. Adaptive random testing [32] is a family of advanced random testing techniques aimed to improve the failure detection effectiveness by evenly spread test cases executed. The most known ART is the Fixed-Size Candidate Set ART technique (FSCS-ART), which is the most widely studies. Others ART algorithms have been developed and their effectiveness are validated by simulation experiments [8], [33], [34].

Adaptive testing (AT) [19] is based same observation, which outperformed both RT and RPT. However, it requires very long execution time in practice. To alleviate this problem, Cai et al. [7] proposed DRT, which uses testing information dynamically adjusted testing profile. There are several aspects that affect the test efficiency of DRT such as testing profile, parameters etc. Yang et al. [14] proposed A-DRT, which adjusts parameters during testing process. Li et al. [15] developed O-DRT technique, which will changed testing profile to a theoretically optimal one when the predefined criterion is satisfied. Lv et al. [13] studied the relationship of parameters in DRT, but he did not give the relationship between parameters and the number of partitions.

## 7 Conclusion

We have presented a dynamic random testing technique for web services to address the challenges of testing SOA-based applications. The proposed technique employs random testing to generate test cases, and ~~while~~ selects test cases for execution from different partitions in accordance with the testing profile of partitions which is dynamically updated in response to the test date collected online. In this way, the proposed test technique takes benefits of both random testing and partition testing.

We proposed a framework which examines key issues when applying DRT to testing web services, and developed a prototype to make the method practical and effective. In order to guide testers flexibly set parameter of DRT strategy according to concrete testing objects, we use the theoretical analysis method to get the relationship between number of partitions, parameter of DRT and the failure rates of partitions. Validating the feasibility and effectiveness of our approach and the reasonable theoretical analysis, three real web services are used to experimental objects. The results of empirical study show that in general DRT shows better performance than RT and RPT, and DRT always delivers an outstanding and stable performance in particular when the parameters lie in our theoretical interval. The latter means that our theoretical analysis can provided a truly useful guidance when DRT is used for different situations.

In our future work, we plan to conduct experiments on more web services to further validate the effectiveness and identify the limitations of our method.

## Acknowledgment

This research is supported by the National Natural Science Foundation of China under Grant No. 61370061, the Beijing Natural Science Foundation (Grant No. 4162040), the Aeronautical Science Foundation of China (Grant No. 2016ZD74004), and the Fundamental Research Funds for the Central Universities under Grant No. FRF-GF-17-B29.

## References

[1] C.-A. Sun, G. Wang, K.-Y. Cai, and T. Y. Chen, "Towards dynamic random testing for web services," in *Proceedings of the 36th IEEE International Computer Software and Applications Conference (COMPSAC'12)*, 2012, pp. 164–169.

[2] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: a research roadmap," *International Journal of Cooperative Information Systems*, vol. 17, no. 02, pp. 223–255, 2008.

[3] C.-A. Sun, E. el Khoury, and M. Aiello, "Transaction management in service-oriented systems: Requirements and a proposal," *IEEE Transactions on Services Computing*, vol. 4, no. 2, pp. 167–180, 2011.

[4] C. Bartolini, A. Bertolino, S. Elbaum, and E. Marchetti, "Whitening soa testing," in *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2009, pp. 161–170.

[5] G. Canfora and M. D. Penta, "Service-oriented architectures testing: A survey," *Software Engineering*, vol. 5413, pp. 78–105, 2009.

[6] R. Hamlet, "Random testing," in *Encyclopedia of Software Engineering*. John Wiley and Sons, Inc., 2002.

[7] K.-Y. Cai, H. Hu, C. Jiang, and F. Ye, "Random testing with dynamically updated test profile," in *Proceedings of the 20th International Symposium On Software Reliability Engineering (ISSRE'09)*, 2009, pp. 1–2.

[8] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. Tse, "Adaptive random testing: The art of test case diversity," *Journal of Systems and Software*, vol. 83, no. 1, pp. 60–66, 2010.

[9] F.-C. Kuo, K. Y. Sim, C.-A. Sun, S. F. Tang, and Z. Zhou, "Enhanced random testing for programs with high dimensional input domains," in *Proceedings of the 19th International Conference on Software Engineering and Knowledge Engineering (SEKE'07)*, 2007, pp. 135–140.

[10] K.-Y. Cai, B. Gu, H. Hu, and Y.-C. Li, "Adaptive software testing with fixed-memory feedback," *Journal of Systems and Software*, vol. 80, no. 8, pp. 1328–1348, 2007.

[11] P. E. Ammann and J. C. Knight, "Data diversity: An approach to software fault tolerance," *IEEE Transactions on Computers*, vol. 37, no. 4, pp. 418–425, 1988.

[12] G. B. Finelli, "NASA software failure characterization experiments," *Reliability Engineering and System Safety*, vol. 32, no. 1, pp. 155–169, 1991.

[13] J. Lv, H. Hu, and K.-Y. Cai, "A sufficient condition for parameters estimation in dynamic random testing," in *Proceedings of the 3rd IEEE International Workshop on Software Test Automationthe, Co-located with the 35th IEEE Annual International Computer Software and Applications Conference*, 2011, pp. 19–24.

[14] Z. Yang, B. Yin, J. Lv, K.-Y. Cai, S. S. Yau, and J. Yu, "Dynamic random testing with parameter adjustment," in *Proceedings of the 6th IEEE International Workshop on Software Test Automationthe, Co-located with the 38th IEEE Annual International Computer Software and Applications Conference*, 2014, pp. 37–42.

[15] Y. Li, B. B. Yin, J. Lv, and K.-Y. Cai, "Approach for test profile optimization in dynamic random testing," in *Proceedings of the 39th IEEE Annual International Computer Software and Applications Conference (COMPSAC'15)*, 2015, pp. 466–471.
[16] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34–41, 1978.
[17] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?" in *Proceedings of the 27th International Conference on Software Engineering (ICSE'05)*, 2005, pp. 402–411.
[18] E. J. Weyuker and B. Jeng, "Analyzing partition testing strategies," *IEEE Transactions on Software Engineering*, vol. 17, no. 7, pp. 703–711, 1991.
[19] K.-Y. Cai, T. Jing, and C.-G. Bai, "Partition testing with dynamic partitioning," in *Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05)*, 2005, pp. 113–116.
[20] T. Y. Chen and Y.-T. Yu, "On the relationship between partition and random testing," *IEEE Transactions on Software Engineering*, vol. 20, no. 12, pp. 977–980, 1994.
[21] ——, "On the expected number of failures detected by subdomain testing and random testing," *IEEE Transactions on Software Engineering*, vol. 22, no. 2, pp. 109–119, 1996.
[22] C.-A. Sun, G. Wang, B. Mu, H. Liu, Z. Wang, and T. Y. Chen, "Metamorphic testing for web services: Framework and a case study," in *Proceedings of the 2011 IEEE International Conference on Web Services (ICWS'11)*, 2011, pp. 283–290.
[23] D. Gettys, "If you write documentation, then try a decision table," *IEEE Transactions on Professional Communication*, no. 4, pp. 61–64, 1986.
[24] Y. Tao and H. Chongzhao, "Entropy based attribute reduction approach for incomplete decision table," in *Proceedings of the 20th International Conference on Information Fusion (Fusion'17)*, 2017, pp. 1–8.
[25] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian Journal of Statistics*, vol. 6, pp. 65–70, 1979.
[26] S. R. Dalal, A. Jain, N. Karunanithi, J. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz, "Model-based testing in practice," in *Proceedings of the 21st International Conference on Software Engineering (ICSE'99)*, 1999, pp. 285–294.
[27] A. Sinha and A. Paradkar, "Model-based functional conformance testing of web services operating on persistent data," in *Proceedings of the 2006 Workshop on Testing, Analysis, and Verification of Web Services and Applications, Co-located with the ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'06)*, 2006, pp. 17–22.
[28] M. Ghannoudi and W. Chainbi, "Formal verification for web service composition: A model-checking approach," in *Proceedings of the International Symposium on Networks, Computers and Communications (ISNCC'15)*, 2015, pp. 1–6.
[29] W. L. Dong and Y. Hang, "Web service testing method based on fault-coverage," in *Proceedings of the Intanational Workshop on Models for Enterprise Computing (IWMEC'06), Co-located with the 10th IEEE International Enterprise Distributed Object Conference (EDOC'06)*, 2006, pp. 43–43.
[30] S. Hanna and M. Munro, "An approach for wsdl-based automated robustness testing of web services," in *Information Systems Development*, 2009, pp. 1093–1104.
[31] H. Zhu and Y. Zhang, "Collaborative testing of web services," *IEEE Transactions on Services Computing*, vol. 5, no. 1, pp. 116–130, 2012.
[32] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. H. Tse, "Adaptive random testing: The ART of test case diversity," *Journal of Systems and Software*, vol. 83, no. 1, pp. 60–66, 2010.
[33] T. Y. Chen, F.-C. Kuo, and C.-A. Sun, "Impact of the compactness of failure regions on the performance of adaptive random testing." *Chinese Journal of Software*, vol. 17, no. 12, pp. 2438–2449, 2006.
[34] A. C. Barus, T. Y. Chen, F.-C. Kuo, H. Liu, R. Merkel, and G. Rothermel, "A cost-effective random testing method for programs with non-numeric inputs," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3509–3523, 2016.