CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (2014)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3246

Delivering Web service load testing as a service with a global cloud

Minzhi Yan, Hailong Sun*,†, Xudong Liu, Ting Deng and Xu Wang

School of Computer Science and Engineering, Beihang University, Beijing, China

SUMMARY

In this paper, we present WS-TaaS, a Web services load testing platform built on a global platform PlanetLab. WS-TaaS enables load testing process to be simple, transparent, and as close as possible to the real running scenarios of the target services. First, we briefly introduce the base of WS-TaaS, Service4All. Second, we provide detailed analysis of the requirements of Web service load testing and present its conceptual architecture as well as algorithm design for improving resource utilization. Third, we present the implementation details of WS-TaaS. Finally, we perform the evaluation of WS-TaaS with a set of experiments based on the testing of real Web services, and the results illustrate that WS-TaaS can efficiently facilitate the whole process of Web service load testing. Especially, comparing with existing testing tools, WS-TaaS can obtain more effective and accurate test results. Copyright © 2014 John Wiley & Sons, Ltd.

Received 21 September 2013; Revised 29 January 2014; Accepted 2 February 2014

KEY WORDS: Web services; Service4All; cloud computing; load testing; testing as a service; PlanetLab

1. INTRODUCTION

Web services are currently the most promising service-oriented computing (SOC) based technology [1], and the SOC paradigm usually adopts Web services as the building blocks of rapid, low-cost, interoperable, and evolvable distributed applications [2]. The performance of these applications is mainly dependent on that of the component Web services. The user requests of most of the Internet accessible applications, including many service-oriented applications, are constantly changing. Accurate measurement of the operating capacities under different load level of user requests can help the developer to improve the application. Underestimating user demand may lead to system failure or out of service. For instance, in the Chinese 'Double-11' Festival of 2013, which is a shopping spree or the 'Black Friday' for China, many users of www.taobao.com and www.tmall.com got into trouble with paying their orders smoothly, and thus, they had to try several times. It was caused by the 171 millions orders generated within that day. In this aspect, load testing [3] for these Internet accessible applications is regarded as an effective way to identify their maximum operating capacity as well as any bottlenecks and finding out which factor is causing degradation. Therefore, effective Web service load testing is very important for evaluating the performance of the component Web services and thus the service-oriented applications. The load testing results can significantly help service provider/developer with improving the performance of these applications. Here, we identify three challenges that Web service load testing faces as follows: (1) simulation of real characteristics of massive user requests, including real concurrency, diversity of geographical location, and system configuration; (2) flexible and elastic provisioning of testing environments, consisting of underlying resources and runtime middleware; and (3) automating the load testing process.

For finishing Web service load testing effectively, service provider/developer need to choose a proper testing tool first. Several testing tools can be found to meet this requirement, such as LoadUI

*Correspondence to: Hailong Sun, School of Computer Science and Engineering, Beihang University, Beijing, China.
†E-mail: sunhl@act.buaa.edu.cn

[4], Apache JMeter [5], and IBM Rational Performance Tester [6]. These tools support testers to create and execute load testing tasks on a single node or in a LAN environment. However, in real running scenarios, the user requests pointing to a Web service concurrently may come from different locations rather than a LAN environments; thus, a good testing tool should provide geographically distributed test nodes for a test task. It is difficult for traditional test methods to test the target Web service from geographically distributed nodes. As the existing tools are based on single node or small LAN, they cannot simulate the real characteristics of massive user requests due to the limitation of node resources, either. Actually, as the increasing of concurrent testing threads in one test node, more and more threads will be put into the waiting queue; thus, these threads actually are not executing the test task simultaneously. In addition, traditional test methods require testers to install test tools, build, and configure test environment manually, which will increase the cost of load testing and impact the efficiency as well. There is another cloud-based Web service load testing tool TestMaker [7] that supports running massive concurrency load test on Amazon EC2. However, TestMaker still needs installation and complex test configuration. In brief, all these existing tools cannot meet the challenges of Web service load testing well.

In recent years, cloud computing has emerged as an important solution to ease the computing needs with low cost. In the meantime, many IT resources and system functions are provided as services, like the cloud computing stack: software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). Similarly, testing as a service (TaaS) [8–10] is proposed to provide software testers with cost-effective testing services. One of the services provided by TaaS is the hosting of hardware and software tools for software testing, which means that TaaS is usually built on the top of IaaS and PaaS. There are several advantages that can be obtained by incorporating cloud computing into software testing. First, the testing environment, including middleware and underlying infrastructure, can be provisioned automatically in accordance with tester requirements, and testers do not have to concern about where or how the test environment is built. Second, cloud testing supports great elasticity, which is inherited from cloud computing technology [11], and has the ability to provide test environment for large-scale concurrent testing. Third, it is also an outstanding advantage that cloud testing service built on a global cloud platform serves testers with significant convenience of simulating geographically distributed requests with its globally distributed data centers. Furthermore, cloud testing providers always supply efficient test task management and scheduling mechanisms, as well as test results analysis. Last but not least, cloud testing solution can observably reduce the test cost as pay-as-you-go payment model is adopted.

In our previous work [12, 13], we propose WS-TaaS that is a TaaS platform for Web service load testing. WS-TaaS is built on the basis of our PaaS platform, Service4All, a cloud platform for service-oriented software development [14, 15]. With WS-TaaS, testers can deploy, execute, and monitor test tasks and obtain test results through just simple configurations. Test environment provisioning is transparent to testers, and the environment can simulate the real runtime scenario of Web services. Meanwhile, efficient test task scheduling mechanisms are designed for improving resource utilization. WS-TaaS provides an efficient and accurate test service for Web service load testing and can substantially reduce the test cost, complexity and time. In [13], we (1) clearly identified the requirements of Web service load testing and the challenges it faces, (2) gave the architecture and key component design of a cloud TaaS platform for Web service load testing, (3) presented preliminary algorithms for test node selection and test task scheduling, and (4) described the implementation details of WS-TaaS and present the experiment results run with WS-TaaS, which is built on the datacenter in our lab.

This paper is an extended version of [13], by including new contributions not found in [13] as follows:

(1) We present a formal model of test task scheduling problem when tester do not specify any test nodes. We show that the problem is an NP-hard optimization problem and provide an approximation algorithm, which is a full polynomial time approximation scheme (FPTAS) (Section 4.2.1).

DOI: 10.1002/cpe

(2) We give a revised heuristic algorithm based on non-shared first (NSF) in [13] for test task scheduling when testers specify test nodes for the test tasks. The algorithm can gain better geographical distribution diversity of invocation requests (Section 4.2.2).
(3) We transplant WS-TaaS to a global platform PlanetLab (Section 5.6) to simulate the geographically distribution of multiple requests invoking a Web service concurrently and run enhanced experiments with this global version of WS-TaaS to illustrate the efficiency and accuracy of WS-TaaS (Section 7).

The remainder of this paper is organized as follows. The main functionalities of Service4All is introduced in Section 2. Section 3 describes the requirements of building WS-TaaS and the architecture design. In Section 4, we present the algorithms for node selection and test task scheduling. The architecture of WS-TaaS is presented in Section 5. In Section 6, a use case of WS-TaaS is provided to show how it works. Then, Section 7 presents the performance evaluation results of WS-TaaS with three experiments. Section 8 provides the related works. Finally, we conclude this work and propose the direction for future work.

2. SERVICE4ALL

In this section, we present the working basis of WS-TaaS, our previous work Service4All, a cloud platform for service-oriented software development. Service4All is now an open source project at OW2 [15], and the latest version, Service4All 2.0, was released on March 26, 2013. Figure 1 shows the system architecture of Service4All. Service4All consists of three main portions: ServiceXchange, ServiceFoundry, and Service-Oriented AppEngine (SAE).

ServiceXchange is a platform to aggregate Web services, mine the relationships of them, and provide service-oriented software developers with service resources. More than 20,000 Web services are collected here. Developers can subscribe these Web services and reuse them to build complex service-oriented applications. Originally, ServiceXchange was a stand-alone platform that provided Web service discovery services for external users. Now, we have integrated it with Service4All platform.

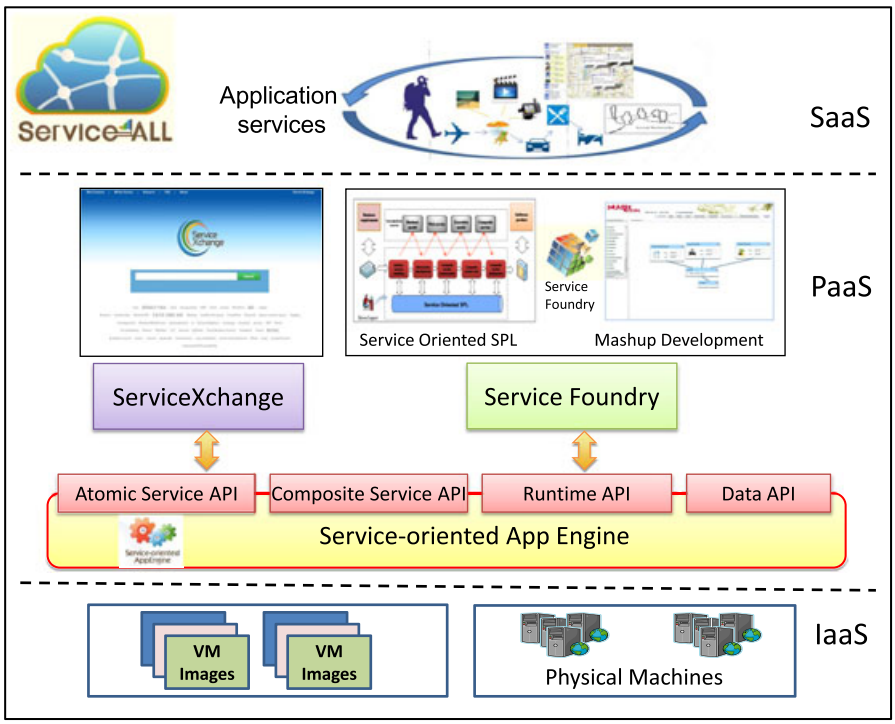

Figure 1. System architecture of Service4All.

DOI: 10.1002/cpe

ServiceFoundry is an online development environment for service-oriented software, which supports two programming models: service composition based on Business Process Modeling Notion (BPMN) and Mashup, respectively. It provides developers with Business Process Integrated Development Environment (BPIDE) and a Mashup Editor Mashroom for supporting these two programming models. After the developing process, developers can deploy the atomic Web services, composite services, or Mashup applications into SAE to test their functions.

Service-Oriented AppEngine provides an online runtime and evolution environment. When an application is deployed into SAE, SAE will dynamically set up a runtime environment for it, then the application can be run and monitored. The middleware that Service4All supports includes atomic Web service container, composite service engine, and service bus and application server. In Service4All, the VM images for runtime middlewares are called software appliance (SA), which should be generated before application deployment step, and this provision process is finished by SAE automatically.

With Service4All, developer can reuse the service resources in ServiceXchange and compose them with the new services developed in Service Foundry to build all kinds of service-oriented applications and then deploy them into SAE for serving the end users.

Service4All is now deployed over iVIC,[†] which is located in Beihang University, and PlanetLab with 100 VM/Slices running Debian 6.0 and Fedora 8. And the open source version, Service4All2.0, is also deployed at Aliyun Cloud Platform [16].

3. OVERVIEW OF WS-TAAS

In this section, we give an overview of WS-TaaS by analyzing the requirements of WS-TaaS, presenting the design of conceptual architecture and explaining the testing workflow in WS-TaaS.

3.1. Requirements of WS-TaaS

To design a cloud-based Web service load testing environment, we need to analyze its requirements and take advantage of the strong points of cloud testing. Taking earlier aspects into account, we conclude that the following features should be guaranteed when building WS-TaaS: transparency, elasticity, geographical distribution, massive concurrency, and sufficient bandwidth.

- *Transparency.* The transparency in WS-TaaS is divided into two aspects: (1) hardware transparency, testers have no need to know exactly where the test nodes are deployed and (2) middleware transparency, when the hardware environment is ready, the testing middlewares should be prepared automatically without tester involvement.
- *Elasticity.* All the test capabilities should scale up and down automatically commensurate with the test demand [17]. In WS-TaaS, the required resources of every test task should be estimated in advance to provision more or withdraw the extra ones.
- *Geographical distribution*. To simulate the real runtime scenario of a Web service, WS-TaaS is required to provide geographically distributed test nodes to simulate multiple users from different locations all over the world.
- *Massive concurrency and sufficient bandwidth*. As in Web service load testing process the load span can be very wide, so WS-TaaS have to support massive concurrent load testing. Meanwhile, the bandwidth need to be sufficient accordingly.

3.2. Conceptual architecture

In a cloud-based load testing environment for Web service, we argue that these four components are needed: Test Task Receiver & Monitor (TTRM), Test Task Manager (TTM), Middleware Manager, and TestRunner. Figure 2 shows the conceptual architecture of a cloud-based Web service load testing system, including the four main components earlier, which we explain as follows:

[†]http://www.ivic.org.cn/ivic/

DOI: 10.1002/cpe

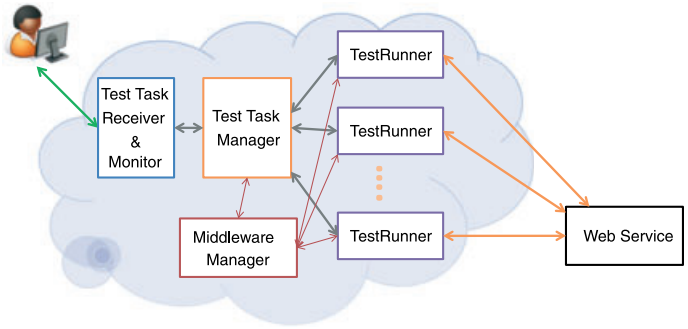

Figure 2. Conceptual architecture of a Web service cloud testing system.

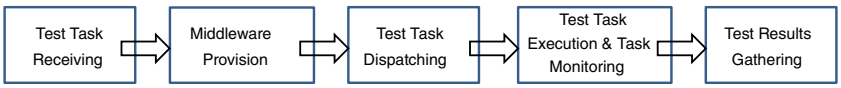

Figure 3. Workflow of WS-TaaS.

- *Test Task Receiver & Monitor*. TTRM is in charge of supplying testers with friendly guide to input test configuration information and submitting test tasks. The testing process can also be monitored here.
- *Test Task Manager*. TTM manages the queue of test tasks and dispatchs them to test nodes in the light of testers' intention and then gathers and merges the test results.
- *TestRunner*. TestRunners are deployed on all test nodes and play the role of Web service invoker. They can also analyze the validity of Web service invocation results.
- *Middleware Manager*. Middleware Manager manages all the TestRunners and provide available TestRunners for TTM with elasticity.

3.3. *Testing workflow*

The workflow of WS-TaaS is divided into five steps, as shown in Figure 3.

(1) TTRM receives configuration of a test task submitted by the tester and transmit it to TTM. (2) TTM selects test nodes from available nodes in terms of their performance and asks Middleware Manager to provision TestRunners on the test nodes selected (please see Section 4.1 for details). (3) TTM decides the numbers of concurrent request of each node selected and divides the test task into subtasks with specific scheduling and dispatching strategies (the details of the strategies are shown in Section 4.2) and then dispatches them to the test nodes selected. (4) After dispatching the test task, TTM notifies all the participating TestRunners to start invoking the target service simultaneously. During test task execution, TTRM periodically sends query requests to TTM at a fixed time interval to obtain the execution state. (5) In the meantime, TTM also periodically queries the test results from all the involved TestRunners. On receiving a query response, the intermediate test results will be displayed to users. The query and gathering process will be continued until the test task finishes.

3.4. *Test mode*

For different kinds of test needs, WS-TaaS provides three test modes as follows.

- *Static test.* Static test is provided to test the performance of a Web service under user-specified load. In such a test, the test task is just deployed and executed once.
- *Step test.* In a step test, the tester needs to input the start number, step size, and the end number of concurrent requests. Then, the test task is deployed and executes step by step with different numbers of concurrent requests, which increase by the step size from the start number till it meets the end one. Step test can tell the tester how the Web service would offer usability in a specific load span.

DOI: 10.1002/cpe

- *Maximal test.* Maximal test is used to determine the load limit of a Web service. Like a step test, the start number and step size are needed, but the end number is not needed. So the strategy for judging whether the current load is the load limit is required. At present, we briefly define judgement principle as follows: If more than 10% of the concurrent requests are failed (the invoking result is inconsistent with the expected result or the invoking process timed out) under a load, then we define this failure and take this load amount as the load limit of the target Web service. This determination is made by TTM with the calculation of failure percentage. Once the load limit is determined, this test task will be stopped and the final report is generated for the tester.

4. ALGORITHM DESIGN

In this section, we present algorithms for test node selection and test task scheduling, respectively. In this work, a test node is a PM/VM (a slice of PlanetLab in this paper) which is located in a specific site of the world with a certain concurrent ability. All these algorithms are developed under the following two situations. (1) Some testers may concern about the performance of invoking target services from some specific locations. In this situation, they would specify the needed test nodes for each of the tasks when they submit test tasks to TTRM. (2) However, for the other testers, they do not have any test node preference and want to obtain the average performance when invoking the target services from as many test nodes as possible. These testers would not specify test nodes for test tasks.

The main requirements of regular computational services are high performance CPU and memory, for example, MapReduce focus on the parallel processing of large-scale datasets. However, for a Web service load test task, a test node needs to launch several threads to invoke the remote Web service through network. The data produced by each test thread are usually not too large to store, but these threads need high bandwidth and CPU performance to invoke the target service smoothly. We now know the key factors that decide the concurrent ability of a test node, but how to present this ability? Here, we provide the definition of a basic indicator of the concurrent ability of a test node as follows:

Available concurrent test thread number (ACTTN): The ACTTN of each test node is the concurrent ability of this node which means how many test threads at most can be launched concurrently on this node to invoke target services smoothly. It is calculated by minus the concurrent ability consumed by other SAs from the original concurrent test thread number of the node. Details for calculating the ACTTN of node i is given as follows.

$$ACTTN_i = O_i - \sum_{j=1}^{S} \mu_j C_{ij}, \tag{1}$$

where

- O_i refers to the upper bound of the original concurrent test threads number of node i. Let t_1 be the average response time (ART) which is calculated by non-concurrently (with just 1 test thread) invoking the 20 most used Web services in ServiceXchange from node i. And let t_2 be the ART while concurrently invoking each of those 20 Web services with R requests. R is initially set as 2 and increased step by step while $t_2 < 2t_1$, in which process t_2 would increase as R increases and finally reach $2t_1$. We set the value of O_i as R when $t_2 = 2t_1$ according to the value of delay variance factor in computing retransmission timeout in RFC 793 [18] because when retransmission is detected, we can decide that a network congestion is happening;
- S are the number of types of other SA hosted in each node;
- C_{ij} stands for the number of the $\mathbf{j}^{th}$ type of SA on node i. In Service4All, every node may host several types of SAs at the same time and each type of SA will consume a certain amount of node resources. Thus, we need to take this factor into account when calculating the ACTTN;
- μ_j is the impact factor of the $\mathbf{j}^{th}$ type of SA to the ACTTN which is defined as the average reduction of O_i when one of the $\mathbf{j}^{th}$ SA is added on all the test nodes. For instance, if we

DOI: 10.1002/cpe

add a BPMNEngine (with a composite service executing on it) to each node and the average reduction of O_i is d, then $\mu_{\text{BPMNEngine}}$ is set as d.

According to the definition of ACTTN and the calculation of O_i in Equation (1), when the concurrent invoking thread number of a node exceed ACTTN, a network congestion would be triggered on this node and affects the test results. And in high concurrency situation, the test results obtained in this node would also be significantly affected by high CPU utilization. Thus, ACTTN is an indicator of the ability of the single node.

4.1. Test node selection

Upon receiving a test task, TTM chooses the needed test nodes according to the number of concurrent requests N and the preferred test node list submitted by the tester and then notifies Middleware Manager to provide TestRunners on them. How test nodes are selected is shown as follows.

4.1.1. With node assignment. If test node list of a test task is given by the tester, the needed nodes would be set as the N_d user-specified nodes the sum of whose ACTTN meets:

$$\sum_{i=1}^{N_d} ACTTN_i \geqslant N, \tag{2}$$

where N is the number of concurrent requests. Intuitively, if the specified nodes satisfy inequality (2), they can support at least N concurrent test thread at the same time. Otherwise, the tester will be asked to revise the test node list or concurrent request number N to make it valid.

4.1.2. Without node assignment. If the tester does not specify any test node, then the test nodes are all selected from the available test node list in terms of their ACTTN. And the sum of these ACTTN should satisfy inequality (3).

$$\sum_{i=1}^{N_a} ACTTN_i \geqslant N, \tag{3}$$

Here, N_a is the number of available test nodes in WS-TaaS.

4.2. Test task scheduling

For gaining high resource utilization ratio and throughput, we need to design an efficient task scheduling and dispatching algorithm to improve the parallel level of test tasks. We define Average Resource Utilization Ratio (ARUR) in a time span T_S as Equation (4), which stands for the ratio of concurrent thread capacity used during T_s to the total available concurrent thread capacity:

$$ARUR = \frac{\sum_{i=1}^{n} t_i N_i}{T_s \sum_{i=1}^{Sum} ACTTN_i} \tag{4}$$

Here, t_i and N_i stand for the executing time and concurrent request number of test task i, respectively, n is the number of finished tasks in T_s, sum and $ACTTN_i$ refer to the total number of available test nodes and the ACTTN of each node, respectively.

4.2.1. Without node assignment. For the test tasks that are not specified to any test node, all these tasks can be divided and assigned to any available test node. Denote by $a_1, a_2, \ldots, a_n$ the n available test nodes' ACTTN, because the tasks to be scheduled are not specified to any test node, they can be assigned to any set of nodes as long as the sum of their concurrent request are less than the sum of ACTTN of these nodes. As a result, we can treat all available nodes as a 'Single' node with ACTTN $B = \sum_{i=1}^{n} a_i$. Our scheduling problem then can be stated as follows:

Test task scheduling problem (TTSP): Given a set of test tasks $T = \{t_1, t_2, \ldots, t_m\}$ to be scheduled, with specified numbers of concurrent requests $c(t_i) \in Z^+$, and a super 'Single' test node with ACTTN $B = \sum_{i=1}^{n} a_i$, find a subset T' of T such that $\Sigma_{t_i \in T'} c(t_i) \leqslant B$ and is maximized.

DOI: 10.1002/cpe

We next give an algorithm for the TTSP problem. Unfortunately, we do not know if TTSP has an polynomial time algorithm because it is an NP optimization problem, that is, its corresponding decision problem is a NP-hard problem [19]. The decision problem of TTSP is given a set of test tasks $T = \{t_1, t_2, \ldots, t_m\}$ with specified numbers of concurrent requests $c(t_i) \in Z^+$, a super 'Single' test node with ACTTN B and a bound C, does there exists a subset T' of T such that $\Sigma_{t_i \in T'} c(t_i) \leqslant B$ and $\Sigma_{t_i \in T'} c(t_i) \geqslant C$. We next show that TTSP is an NP optimization problem, by proving that its decision problem is NP-hard.

Theorem 1
TTSP is an NP optimization problem.

Proof
It suffices to show that the decision problem of TTSP is NP-hard. We do it by reduction from the Knapsack problem [20]. Given a set U where each $u \in U$is associated with a size $s(u) \in Z^+$ and a profit $v(u) \in Z^+$, and a positive integers K and L, the decision problem of knapsack problem is to decide if there exists a subset $U' \subseteq U$ such that $\Sigma_{u \in U'} s(u) \leqslant K$and $\Sigma_{u \in U'} v(u) \geqslant L$. It is known that the decision problem of Knapsack problem is NP-complete even when $s(u) = v(u)$ for all $u \in U$ and $B = K = \Sigma_{u \in U} s(u)$ [21]. So we only need to construct a reduction from this special case of Knapsck problem to TTSP.

Given a fix set U, function s and v, and positives B and K earlier where $s = u$, we define $T = U, K = B, L = C$, and for each $t \in T$, $c(t) = s(t) = v(t)$. It is easy to see that if there exists a subset $U' \subseteq U$ such that $\Sigma_{u \in U'} s(u) \leqslant K$and $\Sigma_{u \in U'} v(u) = \Sigma_{u \in U'} s(u) \geqslant L$ if and only if there exists a subset T' of T such that $\Sigma_{t_i \in T'} c(t_i) \leqslant B$ and $\Sigma_{t_i \in T'} c(t_i) \geqslant C$. Thus, the decision problem of TTSP is NP-hard and hence TTSP is a NP optimization problem. □

Theorem 1 shows that TTSP does not admit a polynomial time algorithm; however, the good news is that it has an FPTAS that enables us to find a solution $T' \subseteq T$ such that the sum $\Sigma_{t \in T'} c(t) \leqslant B$ and $\Sigma_{t \in T'} c(t)$ is at least $(1 - \varepsilon) \cdot OPT$ in time bounded by a polynomial in n and $1/\varepsilon$, where OPT is the optimal solution of TTSP.

It is known that the Knapsack problem has an FPTAS [20]. Given a fix set U of objects, with specified sizes $s(u)$ and profits $v(u)$ for each object $u \in U$ and positives B, the Knapsack problem is to find a subset of objects whose total size is bounded by B and total profit is maximized. Obviously, we can show that the reduction give in the proof of Theorem 1 is also an approximation factor preserving reduction from the special case of Knapsack problem (i.e., for each $u \in U, s(u) = v(u)$) to the TTSP problem. Thus, this FPTAS, which is shown in Algorithm 1, can be adopted here for solving this optimization problem.

In Algorithm 1, clearly $A(1, p)$ is known for every $p \epsilon \{1, \ldots, nP'\}$ (lines 8 to 14), and other values $A(i, p)$ can be recurrently calculated with Dynamic Programming Algorithm (lines 15 to 23). After obtaining the result of this algorithm T', the test tasks in T' can then be successively dispatched to these n test nodes which compose the super 'Single' node, and each of these n nodes take subtask from a test task in proportion to their ACTTN.

Vijay [19] proved that this FPTAS can output a set T' which meets

$$profit(T') \geqslant (1 - \varepsilon) \cdot OPT. \tag{5}$$

Here, OPT denotes the optimal profit for this problem.

The running time of this FPTAS is $O(n \cdot nP') = O(n^2 \lfloor \frac{P}{K} \rfloor) = O(n^2 \lfloor \frac{n}{\varepsilon} \rfloor)$, which is polynomial in n and $1/\varepsilon$. In conclusion, we can obtain any good scheduling solution close enough to the optimal one with a polynomial time cost, say $\varepsilon = 0.05$, the running time is $O(20n^3)$.

4.2.2. With node assignment. For the test tasks with specified test nodes, they cannot be divided and scheduled to the nodes out of the corresponding specified ones. With these constraints, the problem of maximizing the resource utilization ratio of WS-TaaS become even more complicated than TTSP, and we define this problem as follows:

Test task scheduling problem (with node assignment) (TTSP-NA): Given a set of test tasks $T = \{t_1, t_2, \ldots, t_m\}$ to be scheduled, with specified numbers of concurrent requests $c(t_i) \in Z^+$,

DOI: 10.1002/cpe

Algorithm 1 FPTAS for TTSP

Input ε, $\{ profit(t) \mid t \epsilon T \}$, $\{ c(t) \mid t \epsilon T \}$; // $\varepsilon > 0$, $profit(t_i) = c(t_i)$
$P = max_{t \epsilon T}\ profit(t)$;
$K = \frac{\varepsilon P}{n}$;
For each test task t_i, define $profit'(t_i) = \lfloor \frac{profit(t_i)}{K} \rfloor$;
$P' = max_{t \epsilon T}\ profit'(t)$;
For each $i \epsilon \{1, \ldots, n\}$ and $p \epsilon \{1, \ldots, nP'\}$, let $T_{i,p}$ denote a subset of $\{t_1, \ldots, t_i\}$ whose total profit is p and whose total number of concurrent requests is minimized;
Let $A(i, p)$ denote the total number of concurrent requests of task set $T_{i,p}$;
for $(p = 1; p \leqslant nP'; p++)$ **do**
  if $p == c(t_1)$ **then**
    $A(1, p) = c(t_1)$;
  else
    $A(1, p) = \infty$;
  end if
end for
for $(i = 1; i < n; i++)$ **do**
  for $(p = 1; p \leqslant nP'; p++)$ **do**
    if $profit'(t_{i+1}) < p$ **then**
      $A(i + 1, p) = min\{A(i, p), c(t_{i+1}) + A(i, p - profit'(t_{i+1}))\}$;
    else
      $A(i + 1, p) = A(i, p)$;
    end if
  end for
end for
Find the most profitable set $T_{n,p}$ which meets $A(n, p) \leqslant B$, say T'; // $B = \sum_{i=1}^{n} a_i$
Output T';

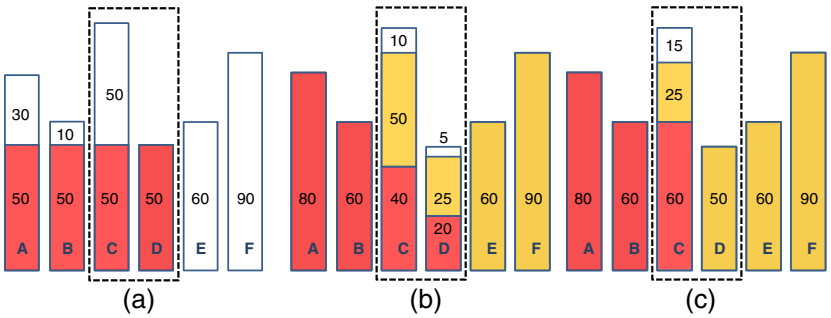

Figure 4. A scheduling example.

and a set of test nodes $W = \{w_1, w_2, \ldots, w_n\}$ whose corresponding ACTTNs are $a_1, a_2, \ldots, a_n$, each test task t_i is specified to a subset W_i of W, find a subset $T'$of T such that $\Sigma_{t_i \in T'} c(t_i) \leqslant B$ and is maximized.

The FPTAS we adopted earlier is not suitable for TTSP-NA anymore. Fortunately, we can design a heuristic algorithm based on some trivial fact to gain an acceptable solution.

Here, we choose an example to explain the idea of the heuristic algorithm: As shown in Figure 4, we assume that test task 1 is specified to nodes A, B, C, D, and its concurrent number is 200. Task 2 is specified to nodes C, D, E, F, and the concurrent number is 225. As the diverse capabilities, these nodes have different concurrent upper bounds as follows: 80, 60, 100, 50, 60, 90. If we divide task 1 equally to those four nodes as shown in Figure 4(a), then task 2 cannot be dispatched to C, D, E, F because the left capability is not enough (the subtasks of the same test task in WS-TaaS must

DOI: 10.1002/cpe

be executed concurrently). However, if the non-shared test nodes A, B take subtasks as their best, and the shared nodes (C and D, circled with doted rectangle) take subtasks in proportion to their ACTTN, then task 2 can be dispatched in the same way and executed simultaneously with task 1, as shown in Figure 4(b). If we assume that the executing time of task 1 and task 2 are both 10 s, then the ARUR of equally dividing (ED) in 10 s will be 45.5% and that of the latter schedule method will be 96.6%.

In this example, we can see that by dispatching subtasks to non-shared test nodes preferentially and letting these nodes exert their best efforts, then dispatch the left subtasks to the shared nodes in proportion to their ACTTN, the parallel level of test tasks can be enhanced significantly and better ARUR could be gained. Therefore, we design the following algorithm NSF^+ based on NSF [13] to schedule test tasks periodically as shown in Algorithm 2. In NSF, after delivering subtasks to non-shared nodes, subtasks are dispatched to the shared nodes in descending order of their ACTTNs, and each of them takes the subtasks as their best. Thus, in some case, part of the shared nodes

Algorithm 2 Non-shared First Plus

1: Divide all the specified test nodes of each test task received during Δt into two groups: shared nodes and non-shared nodes.
2: For every test task, repeat the following steps(3-35).
3: The ACTTN for each node are kept in S(for the shared ones) and O(for the non-shared ones), concurrent request numbers dispatched to each node would be set into D_s and D_o accordingly.
4: $i = 0,\ j = 0$;
5: $D_o = \mathbf{0},\ D_s = \mathbf{0}$;
6: $p = 0$;
7: **while** $N > 0$ **do**
8: **if** $i < O.length$ **then**
9: **if** $N > O[i]$ **then**
10: $N = N - O[i]$;
11: $D_o[i] = O[i]$;
12: $O[i] = 0$;
13: **else**
14: $D_o[i] = N$;
15: $O[i] = O[i] - D_o[i]$;
16: $N = 0$;
17: **end if**
18: $i = i + 1$;
19: **else**
20: **if** $p = 0$ **then**
21: $p = \frac{N}{\sum_{k=0}^{S.length-1} S[k]}$;
22: **end if**
23: **if** $j < S.length - 1$; **then**
24: $N = N - \lceil S[j] \cdot p \rceil$;
25: $D_s[j] = \lceil S[j] \cdot p \rceil$;
26: $S[j] = S[j] - \lceil S[j] \cdot p \rceil$;
27: **else**
28: $D_s[j] = N$;
29: $S[j] = S[j] - D_s[j]$;
30: $N = 0$;
31: **end if**
32: $j = j + 1$;
33: **end if**
34: **end while**
35: return D;

DOI: 10.1002/cpe

may not obtain any subtask (Like node D in Figure 4(c), which does not obtain subtask from Task 1 in the example); thus, the location diversity of the test nodes is not maximized. In NSF^+, for each test task, subtasks of a test task are preferentially dispatched to non-shared nodes (lines 8 to 18) and then to the shared nodes in proportion to their ACTTN (lines 19 to 33). With this modification, all the shared nodes can be used for a test task, and test data can be obtained from more sites in the world; therefore, better geographical distribution diversity of invocation requests can be obtained.

5. ARCHITECTURE OF WS-TAAS

WS-TaaS is a typical three-layer structure of cloud, which consists of SaaS, PaaS, and IaaS layer. Based on Service4All, we develop three new modules and extend two existing ones in WS-TaaS. Three modules in Figure 5 are newly designed ones, including Web Service LoadTester in SaaS layer, TTM, and a new type of SA TestEngine in PaaS layer. SA Manager and Local Agent deployed in every node are also extended to support the registration and management of TestEngine.

5.1. Web Service LoadTester

LoadTester is the implementation of Test Task Receiver & Monitor mentioned in Section III (Figure 2). It is responsible for receiving load test tasks from testers and displaying the test results for them, which are explained in steps 1 and 4 of Figure 3.

LoadTester provides testers with the possibility of selecting diversely located test nodes. Testers can freely select the needed nodes on the map in the test node selecting view. In test configuration view, after inputting the target Web service Web Services Description Language (WSDL) address or the service ID in Service4All platform, testers are guided to input the test parameters; then, the test message for invoking the target service is generated automatically. LoadTester also receives test configuration information like test strategy and concurrent request number. When a test task is submitted to WS-TaaS from LoadTester, it periodically queries the running task, organizes the intermediate data as charts for the monitoring requirement of testers. When a load test task is finished, tester can also save its test results and review them in LoadTester whenever they need.

5.2. Test Task Manager

Test Task Manager is the core module in WS-TaaS and nearly takes part into every step in Figure 3. It is in charge of test task dispatching, test results gathering, and preliminary trim.

Test Task Manager receives test task from LoadTester and then starts the test process according to the test mode described in the test task. For example, static test just needs to be deployed once, while step test needs different times of deployment in the light of test configuration. Before deploying

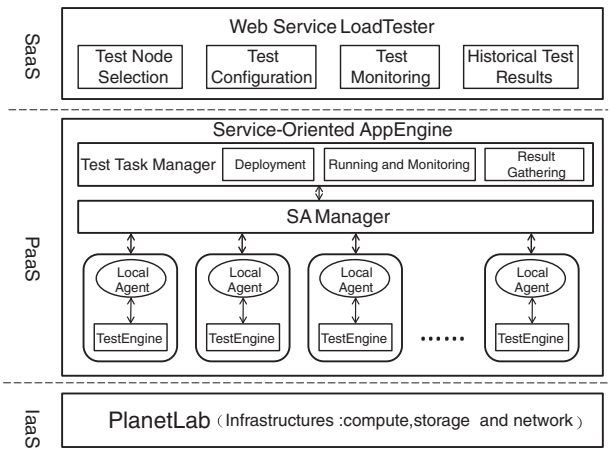

Figure 5. System architecture of WS-TaaS.

DOI: 10.1002/cpe

test tasks to test nodes, TTM asks SA Manager for available TestEngine list. Once the concurrent number is set in a deployment, the corresponding concurrent request numbers of each TestEngine are determined on the basis of the scheduling algorithms described in Section 4.2. After sending test deployment requests to all the participating TestEngines and receiving their deployment responses, TTM sends the command to start testing, and periodically queries and stores test execution state from these TestEngines. Upon receiving query request for test task state from LoadTester, TTM replies with the latest stored state. It also executes analysis on the intermediate data, for example, in a maximal test, it needs to analyze the data to determine whether the last deployed test number is the load limit of the target service.

5.3. TestEngine

TestEngine is a specific SA in WS-TaaS, which plays the role of the TestRunner in Figure 2, and is used to invoke Web services and check the test results.

When receiving test request, containing test message for invoking target service, concurrent number, and so on, TestEngine prepares corresponding number of test threads and replies TTM with the 'ready' response. When ordered to start the test process, TestEngine activates all the ready threads to invoke the target service simultaneously as mentioned in step 4 of Figure 3, it also compares the invoke result with the expected result to ascertain if the test executed in a thread is successful or not, and then organizes and stores these information for the test state query from TTM (step 5 in Figure 3).

5.4. SA Manager

Middleware Manager in Figure 2 is implemented as SA Manager. The functions of SA Manager are extended to manages all types of SA in Service4All, including Web services container, BPMNEngine, and TestEngine. It is in charge of the registration, querying of SA and decides the deployment (step 2 in Figure 3) and undeployment of all types of SAs according to diverse provision and withdrawal strategy.

5.5. Local Agent

Local Agent, which is also a part of Middleware Manager, assists SA Manager with managing SAs in a computing node. Just like SA Manager, we also extend it to support the operation of TestEngine deployment, undeployment, registration, and so on. For example, when SA Manager is asked to provision a TestEngine on a specific node, it notifies the Local Agent on that node to deploy a TestEngine. Additionally, Local Agents distributed on each test node detect and collect the performance information of these nodes, including network traffic, CPU, and memory usage for comparison of node performance.

5.6. Transplant

PlanetLab [22] is a global research network that supports the development of new network services. Since the beginning of 2003, more than 1000 researchers at top academic institutions and industrial research labs have used PlanetLab to develop new technologies for distributed storage, network mapping, peer-to-peer systems, and so on. PlanetLab currently consists of 1172 nodes at 552 sites. We have registered a Planetlab account and applied for 959 slices.

In our previous work [12, 13], WS-TaaS was built on iViC, which is located in our lab, running Windows XP SP3. Currently, we have transplanted WS-TaaS to PlanetLab and select 50 stable slices locating in USA, Spain, Argentina, Sweden, and so on as test nodes. As most of our source codes are written in Java and JSP, it is convenient to transplant and deploy WS-TaaS quickly on PlanetLab slices running Fedora 8. In the transplant process, we just need to make some tiny modifications to the sources codes, for example, modifying the codes related to file paths and the manner of startup.

DOI: 10.1002/cpe

6. USE CASE

Here, we describe the functions of WS-TaaS with a use case, in which a tester Tom uses WS-TaaS to test the load capacity of a third-party Web service.

6.1. Web service selecting

Tom searches and subscribes a Web service that translates English words into Chinese words from ServiceXchange by clicking the circled icon, which is shown in Figure 6; then, this Web service can be reused to build a new service-oriented application.

6.2. Test configuration

Before using this subscribed service, Tom would like to check its load capacity. Then, he goes to LoadTester that is located in ServiceFoundry to select test nodes from the available PlanetLab nodes

Figure 6. Web service subscribing page.

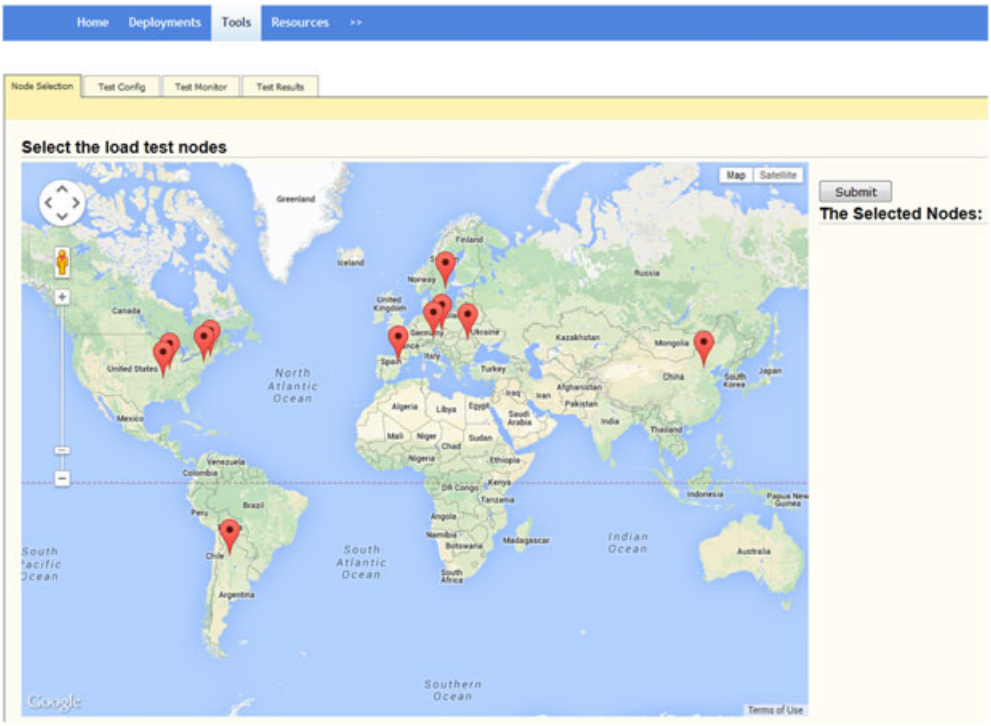

Figure 7. Test node selecting view.

DOI: 10.1002/cpe

and input the other test configuration information in test node selecting view and test configuration view, respectively, as shown in Figures 7 and 8.

6.3. Test execution and monitoring

After the submission of the test task, TTM and SA Manager set up a test environment for this task in SAE automatically. While the task is under execution, Tom can monitor the execution state from test

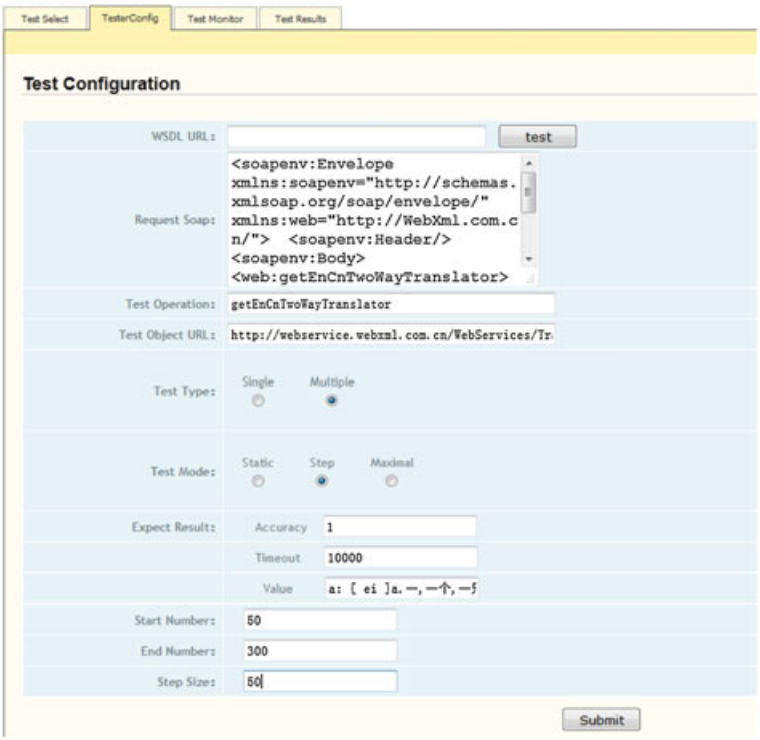

Figure 8. Test configuration view.

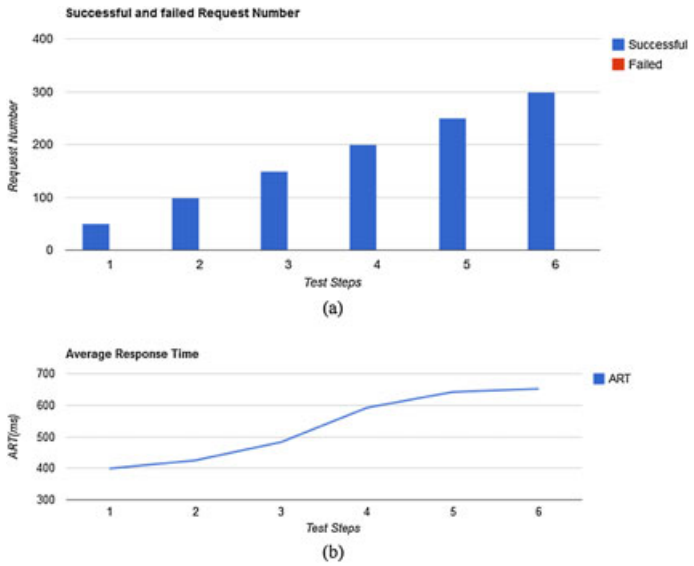

Figure 9. Test monitoring view.

DOI: 10.1002/cpe

monitoring view in ServiceFoundry, which is displayed in Figure 9. Figure 9(a) shows the number of successful and failed requests of each test step, and Figure 9(b) expresses the ART curve.

7. PERFORMANCE EVALUATION OF WS-TAAS

The experiments in this section are mainly run with the global version of WS-TaaS deployed on 50 PlanetLab slices and some test nodes in our lab, which is located in Beijing, China, are also used.

7.1. Location diversity

In this experiment, we hire three different services to evaluate their performance when invoked from New Heaven (North America), Barcelona (Europe), Buenos Aires (South America), and Beijing (Asia), respectively. As shown in Figure 10(a), the ART of the invoke requests to the same Web service from different test nodes have various behavior. And some test nodes (Buenos Aires) may perform the best on one Web service (Service 1) while having the worst performance on another service (Service 2 or 3). These results illustrate that invoking requests from diversely located nodes perform variously. However, the results obtained from three different nodes in our Lab shown in Figure 10(b) tell us that invoking a service from the nodes of the same LAN can obtain similar performance because they locate in the same site and choose the same way while accessing the target Web service. Therefore, it is very helpful to simulate the real invoking environment for a Web service with PlanetLab, which can provide world-wide located nodes.

7.2. Traditional VS WS-TaaS

In this experiment, we choose three services (S1, S2, S3) as the target services to compare the performance of traditional single node Web service load testing approach (JMeter) with that of WS-TaaS. The node for running JMeter (with 100 Mb/s bandwidth, 4 GB memory, and 4 cores, 2.8 GHz CPU) is located in our lab, and we choose 10 PlanetLab nodes(slices) to serve as WS-TaaS test nodes. The bandwidths of these PlanetLab nodes are all 100 Mb/s, but the other characteristics (memory and CPU) of these nodes are not as good as those of the 'JMeter' node.

Figure 11 displays the results of these services, which show that along with the increase of the concurrent request number, the ART of JMeter grow quickly while those of WS-TaaS grow at very slow rates for S1, S2, and S3. Under the load of 250 concurrent requests, the ART of JMeter for S2 is more than 600 ms higher than the WS-TaaS ART for the same service. We could make the conclusion that the ART difference under heavy load mainly results from the limited bandwidth and computing capability of the sole node, which leads to the queuing of multiple threads. Nevertheless, instead of queuing, test threads in WS-TaaS can be scheduled more efficiently.

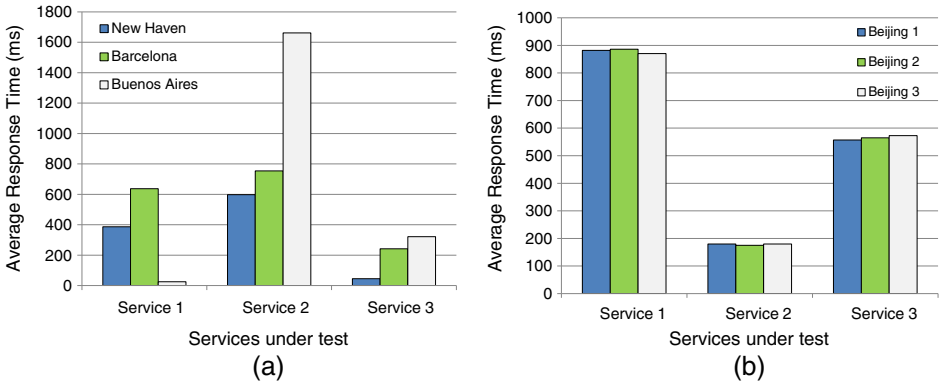

Figure 10. ART comparison with diversely located test nodes.

DOI: 10.1002/cpe

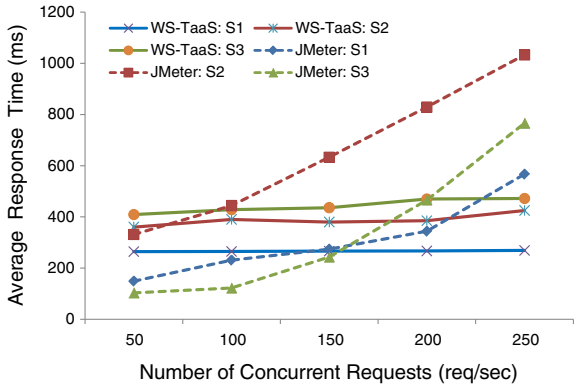

Figure 11. ART comparison under diverse load.

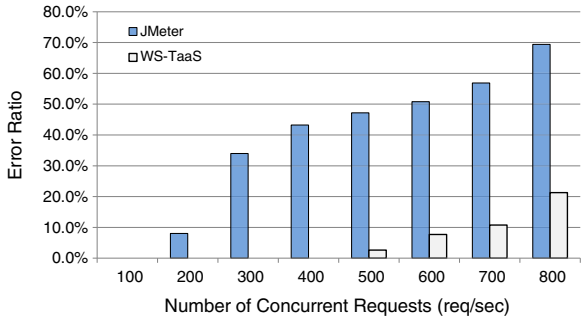

Figure 12. Error ratio comparison.

Figure 12 shows the error ratio comparison of JMeter and WS-TaaS when testing an email address validation Web service: ValidateEmail (*WSDL*: http://www.webxml.com.cn/WebServices/ValidateEmailWebService.asmx?wsdl, *Operation*: ValidateEmailAddress, *Input*: An email address to be validated, *Output*: A number varies from 0 to 7 presenting different types of validation results). In WS-TaaS, we define the load limit of a Web service as the load amount when more than 10 % of the invoking requests are failed, which is also defined failure of this service. If we also use this determining principle in JMeter, the load limit of this Web service will be around 210, which is confirmed by further test. However, it can be seen that the load limit given by WS-TaaS is close to 700, and it is more than three times the size of the former. The great difference between these two results tells us that the limited capacity of a single node can distinctly affect the accuracy of test result. And with WS-TaaS, a more accurate result can be obtained.

7.3. Evaluation of the scheduling algorithms

7.3.1. FPTAS for TTSP. In this experiment, we compare the performance of Dynamic Programming (DP), FPTAS for TTSP, and a Greedy Algorithm. For this Greedy Algorithm, as $profit(t) = c(t)$ for each task, instead of sorting the tasks in decreasing order of profit per unit, we sort them in decreasing order of profit and then insert them into the sack until the left space is not enough for any task. Table I shows four examples of scheduling results of these algorithms, where '1' in the second column means that the corresponding task is selected while '0' means not. We randomly simulate 10–20 tasks whose sizes (number of concurrent requests) are between 50 and 200, and the upper bound of WS-TaaS is set as 1000, and the ε in FPTAS is set as 0.05. Dynamic Programming can find the optimal solution, and in most situation FPTAS performs as good as DP (cases 1, 3, 4), and in the other situations still outperforms the Greedy Algorithm (case 2). Greedy Algorithm has more efficient time cost which is $O(nlog_2 n)$, but the performance is not guaranteed, for example, in case 3 it performs as good as the other two algorithms, but produces 0.8% accuracy loss in case 4.

DOI: 10.1002/cpe

Table I. Examples for comparison of DP, FPTAS, and Greedy Algorithm.

Sequence	Selected tasks	Profit	Accuracy loss (%)
	DP: (1,1,1,1,0,0,1,1,1,1,0,0,1,0,0)	1000	
1	FPTAS: (1,1,1,1,0,0,1,1,1,1,0,0,1,0,0)	1000	0
	Greedy: (1,1,0,1,0,1,1,0,1,0,0,0,1,1,0)	998	0.2
	DP: (1,0,1,1,1,1,0,1,1,0,1,1)	1000	
2	FPTAS: (1,1,0,1,0,0,1,1,0,1,1,1)	999	0.1
	Greedy: (1,0,1,1,0,0,0,1,1,1,1,1)	997	0.3
	DP: (1,1,1,1,1,1,0,1,1,1,1,1)	978	
3	FPTAS: (1,1,1,1,1,1,0,1,1,1,1,1)	978	0
	Greedy: (1,1,1,1,1,1,0,1,1,1,1,1)	978	0
	DP: (1,1,1,0,1,1,1,1,1,1,0,1,1,0,0)	1000	
4	FPTAS: (1,1,1,0,1,1,1,1,1,0,1,0,1,0,1)	1000	0
	Greedy: (1,0,0,1,1,0,0,0,1,1,1,1,1,0,1)	992	0.8

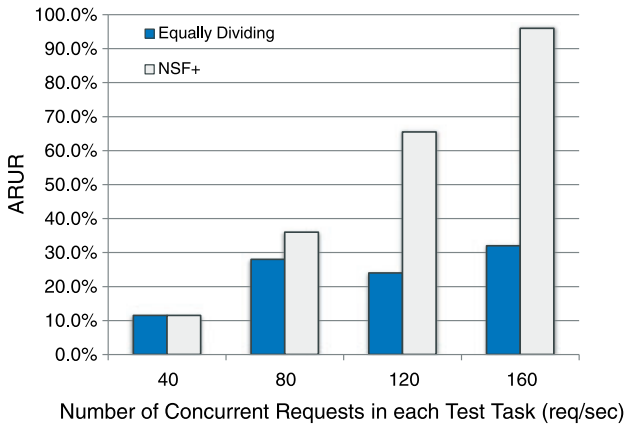

Figure 13. ARUR performance with different scheduling strategies.

After simulating 1000 scheduling cases, the average accuracy loss of Greedy Algorithm, which we obtain, is 0.32% and that of FPTAS is 0.038%, which is significantly smaller than the former one. The earlier results illustrate that FPTAS performs very well in both time cost and profit lost.

7.3.2. NSF^+. This experiment is devised to compare the ARUR of WS-TaaS in 100 s with two scheduling strategies: NSF^+ and ED. We adopted three test tasks (submitted in the same Δt) and 10 test nodes (the concurrent thread limit of each is 50) here, each task is specified to four test nodes, three out of which are non-shared nodes, and only one node is shared by these three tasks.

From Figure 13, we can see that the ARUR of the two strategies are nearly the same under lower load(40 req/s) because the shared node can still handle the three tasks simultaneously with ED. But as the load increase, it can just process 2 or 1 task once with ED, and the ARUR is significantly lower than that of NSF^+. It illustrates that NSF^+ has better performance on improving the resource utilization.

8. RELATED WORK

This paper is an extension of our previous work [13], it is also related to the prior work on identifying the challenges Web service load testing faces, presenting the architecture and component design of WS-TaaS and the implementation details. The following changes and enhancements from [13] are proposed in this paper: (1) revising the NSF heuristic algorithm to gain better geographical distribution diversity of invocation requests, (2) the computing complexity analysis of the TTSP under the situation of 'without node assignment' which is proved NP-hard, and thus adopting an

DOI: 10.1002/cpe

approximation algorithm to solve this problem, and (3) enhanced experiments run on a global platform Planetlab to verify the efficiency of the proposed algorithms and the performance of WS-TaaS.

Other related works are categorized into two types: Web service testing and cloud testing.

8.1. Web service testing

At present, many Web service testing technologies, including online testing, ranking, automated test case generation, monitoring, and simulation, become available. Siblini *et al.* [23] proposes a technique based on mutation analysis to validate the operation of Web services. The technique applies mutation operators to the WSDL document in order to generate mutated Web service interfaces which are used to test the Web service. Xiaoying *et al.* [24] propose the process and algorithms of automatic test case generation based on WSDL. Test data are generated by analyzing the message data types according to standard XML schema syntax. Operation flows are generated based on the operation dependency analysis. They also propose an ontology-based approach for Web service testing [25], and define a test ontology model to specify the test concepts, relationships, and semantics. Automatic test case generation based on WSDL can certainly increase test productivity, but its flexibility is poor when compared to manual test case generation. Moreover, test data and operations are generated mostly by syntactical analysis such as data type analysis. Hence, the parameters generated in the test case can be meaningless, and the veracity of the test result cannot be guaranteed. In our work, we combine automatic generation and manual intervention together: after automatic analysis of the WSDL file, tester can select the operation and input test parameters manually, then a new test case will be generated automatically with these inputs.

The Apache JMeter [5] supports load testing functional behavior and measuring performance of Web services on single node or a LAN. LoadUI [4] is also a Web service testing tool that supplies the capability of Web service load testing, it allows testers to easily create, and execute automated load tests. These two test tools both need to be downloaded and installed in a tester's computer, and the test processes can only be executed on one node or a few nodes, so the load amount is limited. However, our work provides Web service load TaaS and makes the test process more convenient and accurate.

8.2. Cloud testing

In recent years, cloud computing technologies have been introduced to software testing, and TaaS is one of the results. There are many industrial and academic research efforts toward this direction.

In academia, there are many research results on cloud testing. Taksyuki *et al.* [26] introduced a software cloud testing environment D-Cloud for distributed systems, using cloud computing technology and virtual machines with fault injection facility. Lian *et al.* [27] proposed a TaaS model and implemented a VM-based cloud testing environment. In this work, the task scheduling, monitoring, and resource management problems were discussed. Zohar *et al.* [28] presented a method that leverages commercial cloud computing services for conducting large-scale tests of network management systems. All of these works do not pay attention to Web service load testing in cloud.

In industrial area, many cloud testing products have been released. Load Impact [29] offers e-commerce & B2B sites load testing and reporting as an online service. SOASTA's CloudTest [30] supports functional and performance cloud testing services for today's Web and mobile application. All the products earlier cannot support Web service load testing except the cloud-based test tool TestMaker [7]. However, TestMaker is still a desktop software and not provided as a testing service. It needs installation, and the test configuration process is troublesome and difficult for testers. The test cloud need to be configured manually if a tester wanted to run a test task in Amazon EC2. That is, he has to define the number of required test nodes and lots of other information, which fails to reflect the important feature of elasticity in cloud. However, in our testing environment, testers only need to configure a few information while all the other works can be finished automatically.

In summary, traditional Web service load testing approaches ignore the real characteristics of the practical running environment of a Web service. Moreover, they are not user-friendly enough because the installation and configuration process can be troublesome for testers. Furthermore, as far

DOI: 10.1002/cpe

as we know, few cloud-based test product can support Web service load testing, and all the existing cloud-based test products cannot provide Taas.

9. CONCLUSION AND FUTURE WORK

Cloud testing technology can provide a novel and efficient solution over traditional approaches for satisfying the load testing requirements of Web services. Therefore, in this paper, we identify the requirements of Web service load testing and propose a testing platform WS-TaaS built on PlanetLab. Test node selection algorithm, approximation algorithm, and heuristic algorithm for test task scheduling are presented, as well as the implementation details of WS-TaaS based on Service4All. Additionally, three experiments run on the global version of WS-TaaS are performed to illustrate the efficiency and accuracy of load testing using WS-TaaS.

To provide WS-TaaS as an open test environment, we still need to solve the following problems. (1) Security: as WS-TaaS can launch massive concurrent requests for a target service, it is very easy for a malicious people to start a DDoS attack to a specific service with WS-TaaS. Thus, monitoring and controlling mechanisms for these malicious users should be developed to guarantee the security of the target services. (2) Communication efficiency: WS-TaaS is now built on PlanetLab, which is a world-wide platform; the communication cost is significantly higher than that of the LAN version. Therefore, it is necessary to design an approach to evaluate the running time of each test task and set appropriate polling intervals for them to reduce the communication cost.

ACKNOWLEDGEMENTS

This work was supported by the National Natural Science Foundation of China (nos. 61370057 and 61103031), China 863 program (no. 2012AA011203), A Foundation for the Author of National Excellent Doctoral Dissertation of PR China (no. 201159), Beijing Nova Program (no. 2011022), and Specialized Research Fund for the Doctoral Program of Higher Education (no. 20111102120016).

REFERENCES

1. Sanjiva W, Francisco C, Frank L, Tony S, Donald FF. *Web Services Platform Architecture: Soap, Wsdl, Ws-Policy, Ws-Addressing, Ws-Bpel, Ws-Reliable Messaging and More*. Prentice Hall PTR: NJ, USA, 2005.
2. Michael PP, Paolo T, Schahram D, Frank L. Service-oriented computing: state of the art and research challenges. *IEEE Computer* 2007; **40**(11):64–71. DOI: 10.1109/MC.2007.400.
3. *Load testing*. (Available from: http://en.wikipedia.org/wiki/Load_testing) [Accessed on 21 September 2013].
4. LoadUI. (Available from: http://www.loadui.org/) [Accessed on 21 September 2013].
5. Dmitri N. *Using JMeter to performance test Web services*. (Available from: http://loadstorm.com/files/Using-JMeter-to-Performance-Test-Web-Services.pdf) [Accessed on 21 September 2013], 2006.
6. Rational Performance Tester. (Available from: http://www-01.ibm.com/software/awdtools/tester/performance/) [Accessed on 21 September 2013].
7. TestMaker. (Available from: http://www.pushtotest.com/cloudtesting) [Accessed on 21 September 2013].
8. HP Testing as a Service. (Available from: http://www8.hp.com/us/en/software/software-product.html) [Accessed on 21 September 2013].
9. Testing as a Service. (Available from: http://www.tieto.com/what-we-offer/it-services/testing-as-a-service) [Accessed on 10 July 2013].
10. Jerry G, Xiaoying B, Wei TT. Cloud testing: issues, challenges, needs and practice. *Software Engineering: An International Journal* 2011; **1**(1):9–23.
11. Leah MR, Ossi T, Kari S. Research issues for software testing in the cloud. *2nd IEEE International Conference on Cloud Computing Technology and Science* November 2010, IEEE Computer Society Press: Los Alamitos, CA, 2010; 557–564.
12. Minzhi Y, Hailong S, Xu W, Xudong L. Building a TaaS platform for web service load testing. *IEEE International Conference on Cluster Computing 2012* September 2012, IEEE Computer Society Press: Los Alamitos, CA, 2012; 576–579.
13. Minzhi Y, Hailong S, Xu W, Xudong L. WS-TaaS: a testing as a service platform for Web Service load testing. *Proceedings of the 18th IEEE International Conference on Parallel and Distributed Systems* December 2012, IEEE Computer Society Press: Los Alamitos, CA, 2012; 456–463.
14. Hailong S, Xu W, Chao Z, Zicheng H, Xudong L. Early experience of building a cloud platform for service oriented software development. *Proceedings of IEEE International Conference on Cluster Computing 2010* Septemper 2010, IEEE Computer Society Press: Piscataway, NJ, 2010.

DOI: 10.1002/cpe

15. Service4All Project Overview. (Available from: http://www.ow2.org/view/ActivitiesDashboard/Service4All) [Accessed on 21 September 2013].
16. Service4All. (Available from: http://www.service4all.org.cn/) [Accessed on 21 September 2013].
17. Kyong HK, Anton B, Rajkumar B. Power-aware provisioning of virtual machines for real-time services. *Concurrency and Computation: Practice and Experience* 2011; **23**(13):1492–1505. DOI: 10.1002/cpe.1712.
18. Jon P. RFC 793: transmission control protocol. *DARPA Internet Program Protocol Specification* September 1981, Information Sciences Institute University of Southern California: Marina del Rey, California, 1981.
19. Vijay VV. *Approximation Algorithms*. Springer: New York, 2001.
20. Hans K, Ulrich P, David P. *Knapsack Problems*. Springer: Heidelberg, 2003.
21. Michael RG, David SJ. *Computers and Intractability: A Guide to the Theory of Np-Completeness*. W. H. Freeman & Co.: New York, 1990.
22. PlanetLab. (Available from: http://www.planet-lab.org/) [Accessed on 21 September 2013].
23. Reda S, Nashat M. Testing web services. *The 3rd ACS/IEEE International Conference on Computer Systems and Applications,* January 2005, IEEE Computer Society Press: Los Alamitos, CA, 2005; 763–770.
24. Xiaoying B, Wenli D, Wei TT, Yilong C. WSDL-based automatic test case generation for Web services testing. *Proceeding of IEEE International Workshop on Service-Oriented System Engineering(SOSE) 2005,* October 2005, IEEE Computer Society Press: Los Alamitos, CA, 2005; 207–212.
25. Xiaoying B, Shufang L, Wei TT, Yilong C. Ontology-based test modeling and partition testing of Web services. *IEEE International Conference on Web Services(ICWS 2008),* September 2008, IEEE Computer Society Press: Los Alamitos, CA, 2008; 465–472.
26. Takayuki B, Hitoshi K, Ryo K, Takayuki I, Toshihiro H, Mitsuhisa S. D-Cloud: design of a software testing environment for reliable distributed systems using cloud computing technology. *The* 10^{th} *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing,* May 2010, IEEE Computer Society Press: Los Alamitos, CA, 2010; 631–636.
27. Lian Y, Wei TT, Xiangji C, Linqing L, Yan Z, Liangjie T, Wei Z. Testing as a service over cloud. *The* 5^{th} *IEEE International Symposium on Service Oriented System Engineering,* June 2010, IEEE Computer Society Press: Los Alamitos, CA, 2010; 181–188.
28. Zohar G, Itai EZ. Cloud-based performance testing of network management systems. *IEEE 14th International Workshop on Computer-Aided Modeling Analysis and Design of Communication Links and Networks,* June 2009, IEEE Computer Society Press: Los Alamitos, CA, 2009; 1–6.
29. *Load impact*. (Available from: http://loadimpact.com) [Accessed on 21 September 2013].
30. SOASTA. (Available from: http://www.soasta.com) [Accessed on 21 September 2013].

DOI: 10.1002/cpe