WS-TaaS: A Testing as a Service Platform for Web Service Load Testing

Minzhi Yan, Hailong Sun, Xu Wang, Xudong Liu
School of Computer Science and Engineering
Beihang University
Beijing, China
{yanmz, sunhl, wangxu, liuxd}@act.buaa.edu.cn

***Abstract*—Web services are widely known as the building blocks of typical service oriented applications. The performance of such an application system is mainly dependent on that of component web services. Thus the effective load testing of web services is of great importance to understand and improve the performance of a service oriented system. However, existing Web Service load testing tools ignore the real characteristics of the practical running environment of a web service, which leads to inaccurate test results. In this work, we present WS-TaaS, a load testing platform for web services, which enables load testing process to be as close as possible to the real running scenarios. In this way, we aim at providing testers with more accurate performance testing results than existing tools. WS-TaaS is developed on the basis of our existing Cloud PaaS platform: Service4All. First, we briefly introduce the functionalities and main components of Service4All. Second, we provide detailed analysis of the requirements of Web Service load testing and present the conceptual architecture and design of key components. Third, we present the implementation details of WS-TaaS on the basis of Service4All. Finally, we perform a set of experiments based on the testing of real web services, and the experiments 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.**

***Keywords*-web services; cloud computing; load testing; testing as a service**

I. Introduction

Most Internet applications usually face constantly changing user requests. Under-estimating user demand may lead to disastrous system failure. For example, in the Chinese New Year travel rush of 2012, the online ticketing system(www.12306.cn) crashed soon after it was launched in January. The crashing was caused by 1.66 million daily transactions and over 1 billion visits during one peak period. In this aspect, load testing[1] is considered to be an effective method to identify the maximum operating capacity of an application as well as any bottlenecks and determine which factor is causing degradation.

In recent years, service-oriented computing[2] has been regarded as a novel way to develop applications and web services are usually adopted as the building blocks. The performance of such an application system is mainly dependent on that of component web services. Thus the effective load testing of web services is of great importance to understand and improve the performance of a service oriented system. 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; (3) Automating the load testing process.

Existing web service load testing tools, such as SoapUI[3], Apache JMeter[4] and IBM Rational Performance Tester[5], support testers to create and execute load testing tasks on a single node or in a LAN environment. However, in practice users coming from different places may access to a web service concurrently. The single-node or small LAN based load testing cannot simulate the real characteristics of massive user requests due to the limitation of node resources. Indeed, as the increasing of concurrent threads in a test node, more and more threads will be put into the waiting queue, thus these threads actually are not executing the test task simultaneously. Hence, it's costly for traditional test methods to test the target web service from geographically distributed nodes to simulate the practical user requests. 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. In a word, existing tools do not meet the challenges of Web Service load testing.

With the rapid development of cloud computing, resource capacity and many system functions are provided as services, like SaaS (Software as a Service), PaaS (Platform as a Service) and IaaS (Infrastructure as a Service) and so on. Among them, TaaS (Testing as a Service)[6][7] 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 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[8], and has the ability to provide test environment for large-scale concurrent testing. Third, it is also an outstanding advantage that cloud

testing serves testers with the best convenience of simulating geographically distributed requests. Furthermore, cloud testing providers always supply efficient test task management and scheduling mechanism, 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.

However, to the best of our knowledge, there is no work providing a cloud service for Web Service load testing yet. This is probably due to the complexity of hosting web service middlewares and testing tools. In this work, we propose WS-TaaS, a TaaS platform for Web Service load testing, which is preliminarily introduced in [9]. WS-TaaS is built on the basis of our PaaS platform: Service4All, a cloud platform for service oriented software development[10][11]. 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 a web service. It provides an efficient and accurate test service for Web Service load testing and can substantially reduce the test cost, complexity and time. Major contributions of this work are listed as follows:

(1) We clearly identify the requirements of Web Service load testing and the challenges it faces.

(2) We give the architecture and key component design of a cloud TaaS platform for Web Service load testing. Especially we present preliminary algorithms for test node selection and test task scheduling.

(3) We describe the implementation details of WS-TaaS based on Service4All and present the experiment results with the testing of 30 real web services.

The remainder of this paper is organized as follows. Section II provides introduction to related works. The main functionalities of Service4All is introduced in Section III. Section IV describes the requirements of building WS-TaaS and the design details. The implementation of WS-TaaS is presented in Section V. Then, Section VI presents the performance evaluation results of WS-TaaS with two experiments. Finally, we conclude this work and propose the direction for future work.

II. Related Work

A. Web Service Testing

At present, many Web Service testing technologies, including online testing, ranking, automated test case generation, monitoring and simulation etc, become available. Bai *et al*.[12] propose the process and algorithms of automatic test case generation based on WSDL (Web Services Description Language). Test data is 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 [13], and define a test ontology model (TOM) 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[4] supports load testing functional behavior and measuring performance of web services on single node or a LAN. SoapUI[3] is also a Web Service testing tool which supplies the capability of WS 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 testing as a service, and makes the test process more convenient and accurate.

B. Cloud Testing

As described in Section I, in recent years cloud computing technology have been introduced to software testing, and TaaS is one of the results. There are many industrial and academic research efforts towards this direction.

In academia, there are many research results on cloud testing. Taksyuki Banzai[14] introduced a software cloud testing environment D-Cloud for distributed systems, using cloud computing technology and virtual machines with fault injection facility. Lian Yu[15] 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 Ganon[16] presented a method which 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[17] offers e-commerce & B2B sites load testing and reporting as an online service. SOASTA's CloudTest[18] supports functional and performance cloud testing services for today's web and mobile application. All the products above cannot support Web Service load testing except the cloud-based test tool TestMaker[19]. 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 as we know, only one cloud-based test product supports Web Service load testing, but it cannot provide testing as a service and is too complex to use.

III. Introduction to 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[11]. It consists of three main portions: ServiceXchange(www.servicexchange.cn), ServiceFoundry and Service-Oriented AppEngine(SAE).

A. ServiceXchange

It 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 provide web service discovery services for external users. Recently we have integrated it with Service4All platform.

B. ServiceFoundry

ServiceFoundry is an online development environment for service-oriented software which supports two programming models: Service compostition based on BPMN(Business Process Modeling Notion) and Mashup. It provides developers with BPIDE(Business Process Integrated Development Environment) 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 the functions.

C. Service-Oriented AppEngine

SAE provides an online runtime and evolution environment. When an application is deployed into SAE, it 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, service bus and application server etc. The VM images for runtime middlewares are called Software Appliance(SA), they should be generated before application deployment step, and this provision process is finished by SAE automatically.

Service4All is now deployed over iVIC(http://www.ivic.org.cn/ivic/) with 100 VMs running Windows XP SP3, locating in Shahe Campus of Beihang University. It provides a powerful platform for the development, deployment and runtime for service-oriented softwares.

IV. Design of WS-TaaS

A. Overview

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 above aspects into account, we conclude that the following features should be guaranteed while 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. (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. 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.
- *Massive Concurrency and Sufficient Bandwidth*. As in Web Service load testing process the load span can be very wide, so WS-TaaS have to serve massive concurrent load testing. Meanwhile, the bandwidth need to be sufficient accordingly.

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 1 shows the conceptual architecture of a cloud-based WS load testing system, including the four main components above, which we explain as follows:

- *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.

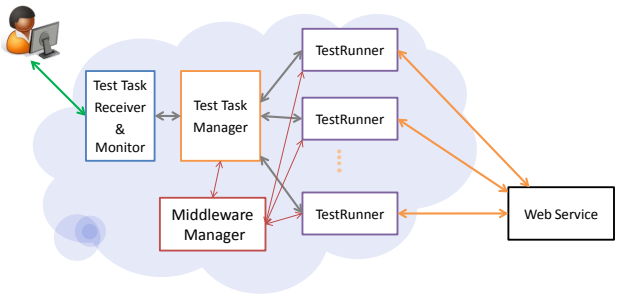

Fig. 1. Conceptual Architecture of a WS Cloud Testing System

- *Test Task Manager*. TTM manages the queue of test tasks and dispatchs them to test nodes in the light of testers' intention, then gathers and trims the test results.
- *TestRunner*. TestRunners are deployed on each test node and play the role of web service invoker, and they can also analyse the validity of web service invocation results.
- *Middleware Manager*. It is needed to manage all the TestRunners and provide available TestRunners for TTM with elasticity.

3) Workflow: The workflow of WS-TaaS is divided into five steps, as shown in Figure 2.

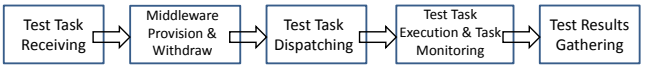

Fig. 2. Workflow of WS-TaaS

Firstly, a tester inputs the test configuration of a test task in TTRM and submit it to TTM. Secondly, TTM selects test nodes from available nodes according to their performance, and asks Middleware Manager to provision TestRunners on them. If one TestRunner has been idle for a fixed time span and presently there is no test task designated to it, then Middleware Manager decides to withdraw it and shut it immediately. Thirdly, TTM decides the numbers of concurrent request of each node and divides the test task into subtasks with a specific dispatching strategy, and then sends them to the chosen test nodes. Then, after dispatching the test task, TTM notifies all the participanting TestRunners to start invoking the target service simultaneously. During test task execution, TTRM sends query requests to TTM at a fixed period to obtain the execution state. And 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 process will be continued until the test task finishes.

B. Preliminary Algorithm Design

Here we present two preliminary algorithms for test node selection and test task scheduling & dispatching.

1) Test Node Selection: Upon receiving a test task, TTM chooses the needed test nodes according to the tester submitted number of concurrent requests N as follows:

(a) If test node list is given by the tester, then the needed nodes would be set as the N_d user-specified nodes which meet:

$$\sum_{i=1}^{N_d}\{O_i - \sum_{j=1}^{S}\mu_j C_{ij}\} \geq N \tag{1}$$

Here O_i refers to the upper bound of the original concurrent test threads number of node i. RFC 793[20] recommends 2 as the value of delay variance factor in computing R-TO(Retransmission Timeout). Similarly, if we let t_1 be the ART(Average Response Time) which is calculated by non-concurrently invoking the 20 most used web services in ServiceXchange from node i, respectively. And let t_2 be the average value of the 20 ART while concurrently invoking each of those 20 web services with R requests. Obviously, as R grows, t_2 will increase correspondingly. We set the value of O_i as R when $t_2 = 2t_1$. In Service4All, every node may host several types of SAs at the same time and each SA will consume a certain amount of node resources. Thus we need to take this factor into account when calculating O_i. Suppose there are S types of other SA hosted in each node. And C_{ij} stands for the number of the $\boldsymbol{j}^{th}$ type of SA on node i, correspondingly, μ_j refers to the impact factor of the $\boldsymbol{j}^{th}$ type of SA to O_i which is decided by testing the O_i difference when one of the $\boldsymbol{j}^{th}$ SA is added.

If these specified nodes cannot satisfy Inequality (1), the tester will be asked to revise the test node list or concurrent request number N to make it valid.

(b) If the tester does not specify any test node, then the test nodes are all selected from the available test node list according to their available concurrent test thread number(ACTTN). N_a is the minimum number that the first N_a test nodes in the sorted node list satisfy the Inequality (2).

$$\sum_{i=1}^{N_a}\{O_i - \sum_{j=1}^{S}\mu_j C_{ij}\} \geq N \tag{2}$$

2) Test Task Scheduling & Dispatching: 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 ARUR (Average Resource Utilization Ratio) in a time span T_S as Equation (3), which stands for the ratio of concurrent thread capacity used during T_S to the total available concurrent thread capacity:

$$ARUR = \frac{\sum_{i=1}^{n} t_i N_i}{T_s \sum_{i=1}^{Sum} O_i} \tag{3}$$

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 O_i refer to the total number of available test nodes and the concurrent request limit of each node, respectively.

Figure 3 shows a scheduling example. In this example, we assume that test task 1 is scheduled to node A, B, C, D and its concurrent number is 200. Task 2 is specified to node C, D, E, F and the concurrent number is 220. 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 3(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 be executed concurrently). However, if the non-shared test nodes A, B take subtasks as their best, then task 2 can be dispatched in the same way and executed simultaneously with task 1, as shown in Figure 3(b). If we assume that the executing time of task 1 and task 2 are both 10 seconds, then the ARUR of Equally Dividing(ED) in 10 seconds will be 45.5% and that of the latter schedule method will be 95.5%.

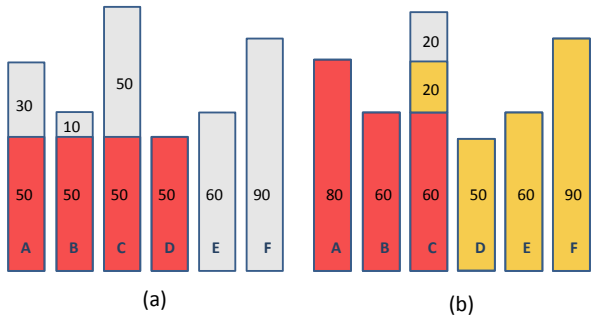

Fig. 3. A Scheduling Example

In this example, we can see that by dispatching subtasks to non-shared test nodes preferentially and letting all the chosen nodes exert their best efforts, the parallel level of test tasks can be enhanced significantly and better ARUR could be gained. Therefore, we design the following algorithm NSF (Non-shared First) to schedule test tasks periodically as shown in Algorithm 1. In NSF, for each test task, shared and non-shared node lists are both sorted by ACTTN, then subtasks of a test task are preferentially dispatched to non-shared nodes and then to the shared nodes.

C. Test Mode

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

1) 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.

2) Step Test: In a step test, the tester need 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.

3) 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 is 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 under a load, then we can take this load amount as the load limit of the target web service. This determination is made by Test Task Manager 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.

V. Implementation

A. System Architecture

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 4 are newly designed ones, including Web Service LoadTester in SaaS layer, Test Task Manager (TTM) and a new type of Software Appliance TestEngine in PaaS layer. SA Manager and Local Agent deployed in every node are also extended to support the registration and management of TestEngine.

Algorithm 1 Non-shared First

1: Divide all the specified test nodes of each test task received during Δt(Δt is very small compared to the task execution time, e.g., 500ms) into two groups: shared nodes and non-shared nodes.
2: For every test task, repeat the following steps(3-29).
3: Sort the specified test nodes in descending order of their ACTTN into two arrays, one for the shared nodes and the other one for the non-shared nodes. The corresponding upper bounds of concurrent requests 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: **while** $N > 0$ **do**
7: **if** $i < O.length$ **then**
8: **if** $N > O[i]$ **then**
9: $N = N - O[i]$;
10: $D_o[i] = O[i]$;
11: **else**
12: $D_o[i] = N$;
13: $O[i] = O[i] - D_o[i]$;
14: $N = 0$;
15: **end if**
16: $i = i + 1$;
17: **else**
18: **if** $N > S[j]$ **then**
19: $N = N - S[j]$;
20: $D_s[j] = S[j]$;
21: **else**
22: $D_s[j] = N$;
23: $S[j] = S[j] - D_s[j]$;
24: $N = 0$;
25: **end if**
26: $j = j + 1$;
27: **end if**
28: **end while**
29: return D;

B. Key Modules

1) Web Service LoadTester: LoadTester is the implementation of Test Task Receiver & Monitor mentioned in Section III which is responsible for receiving load test tasks from testers and displaying the test results for them.

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 WS-DL address or the service ID in Service4All platform, testers

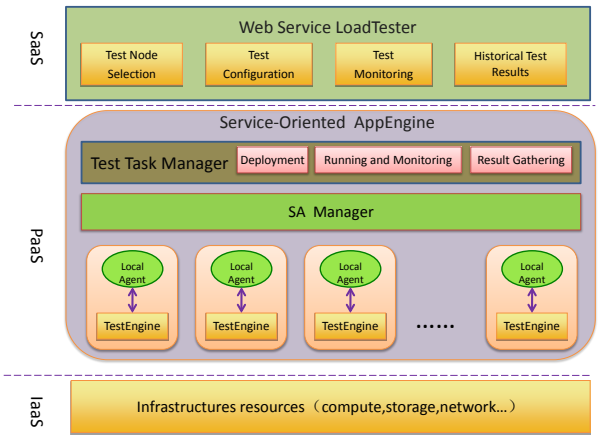

Fig. 4. System Architecture of WS-TaaS

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 etc. 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.

2) Test Task Manager: Test Task Manager is the core module in WS-TaaS, 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 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 is determined on the basis of NSF scheduling strategy. After sending test deployment requests to all the participanting 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, e.g., in a maximal test, it needs to analyse the data to determine whether the last deployed test number is the load limit of the target service.

3) TestEngine: TestEngine is the specific SA in WS-TaaS, and it is used to invoke web services.

When receiving test request, containing test message for invoking target service, concurrent number etc. TestEngine prepares corresponding number of test threads and reply Test Task Manager with the 'ready' response. When ordered to start the test process, TestEngine activates all the ready threads to invoke the target service simultaneously, it also compares the invoke result with the expected result to ascertain if the test executed in a thread is successful or not, then organizes and stores these information for the test state query from TTM.

4) SA Manager: Middleware Manager is implemented as SA Manager. The functions of SA Manager are extended to manages all types of Software Appliance in Service4All, including web services container, BPMNEngine and TestEngine etc. It is in charge of the registration, querying of SA and decides the deployment and undeployment of all types of SAs according to diverse provision and withdrawal strategy.

5) Local Agent: Local Agent 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 etc. Additionally, Local Agents distributed on each test node detect and collect the performance information of these nodes, including network traffic, CPU and memory usage etc. for comparison of node performance.

C. 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.

1) Web Service Selecting: Tom searches and subscribes a web service into ServiceFoundry from ServiceXchange by clicking the circled icon which is shown in Figure 5, then this web service can be reused to build a new service-oriented application.

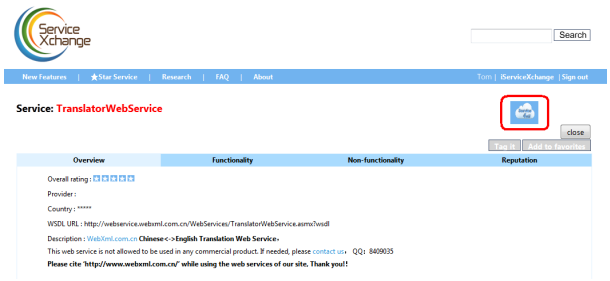

Fig. 5. Snapshot of Web Service Subscribing Page

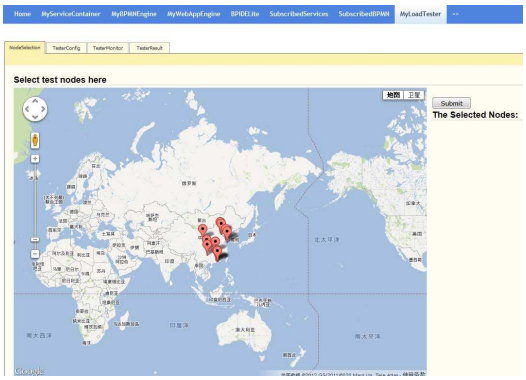

Fig. 6. Test Node Selecting View

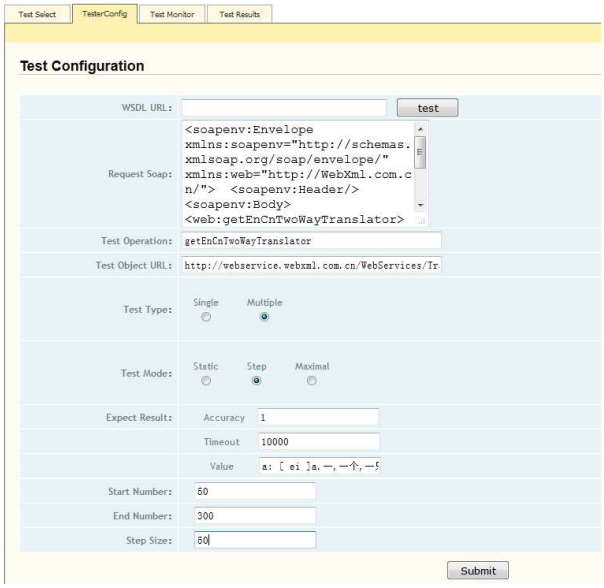

Fig. 7. Test Configuration View

2) Test Configuration: Before using this subscribed service, Tom would like to check its load capacity. Then he goes to LoadTester which locates in ServiceFoundry to select test nodes and input the other test configuration information in Test Node Selecting View and Test Configuration View, respectively, as shown in Figure 6 and Figure 7.

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 Monitoring View in ServiceFoundry, which is displayed in Figure 8. Figure 8(a) shows the number of successful and failed requests of each test step, and Figure 8(b) expresses the ART curve.

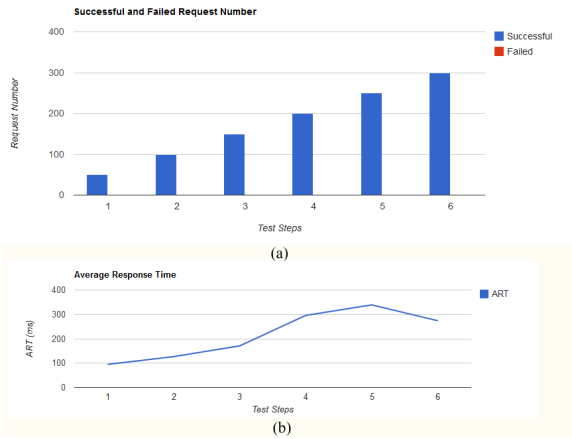

Fig. 8. Test Monitoring View

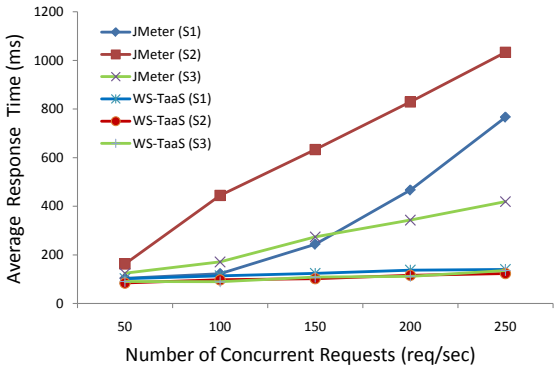

Fig. 9. ART Comparison Under Diverse Load

VI. Experiments

A. Traditional VS WS-TaaS

In this experiment, we choose 30 services as the target services to compare the performance of traditional single node web service load testing approach(JMeter) with that of WS-TaaS.

Figure 9 displays the results of 3 services(S1,S2 and S3) out of the 30, which show that along with the increase of the concurrent request number, the ART(Average Response Time) 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 about 6 times the size of the WS-TaaS ART for the same service. As the hardware and network environment are exactly the same, 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. And the ART behavior for the other services are the same.

Figure 10 shows the error ratio comparison of JMeter and WS-TaaS when testing an email address validation web service: ValidateEmail (WSDL Address) . In WS-TaaS, we define the load limit of a web service as the load amount under which more than 10 percents of the invoking requests are

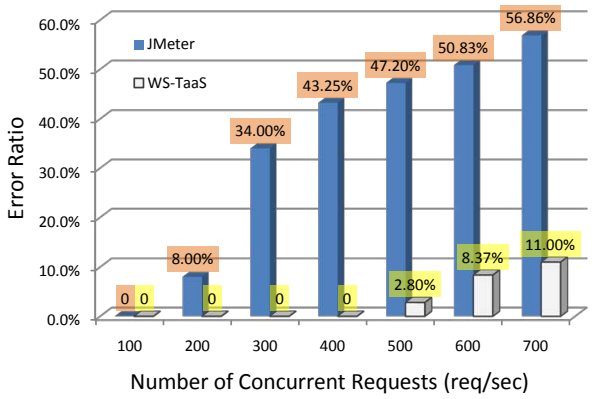

Fig. 10. Error Ratio Comparison

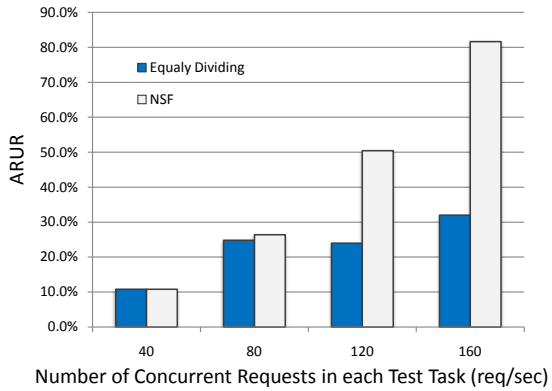

Fig. 11. ARUR performance With Different Scheduling Strategy

failed. If we also use this determining principle in JMeter, the load limit of this web service will be 210 which is confirmed by further test. However, it can be seen that the load limit given by WS-TaaS is more than 600, and it is about 3 times the size of the former. The great difference between these two results tells 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. The error ratio comparison results for the other services have similar performance.

B. Test Task Scheduling

This experiment is devised to compare the ARUR of WS-TaaS in 20 seconds with two scheduling strategies: NSF and Equally Dividing(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 11 we can see that the ARUR of the two strategies are nearly the same under lower load(40 req/sec) because the shared node can still handle the three tasks simultaneously with ED. But as the 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.

VII. Conclusion and Future Work

Traditional testing approaches cannot satisfy the load testing requirements of web services, and cloud testing technology can provide a novel and efficient solution for this problem. However, the related work on TaaS platform for WS load testing can barely be seen. Therefore, in the present paper, we identify the requirements of Web Service load testing and propose a testing platform WS-TaaS. Preliminary algorithms for test node selection and test task scheduling are presented, as well as the implementation details of WS-TaaS based on Service4All. Additionally, two experiments with 30 real web services are performed to illustrate the efficiency and accuracy of load testing using WS-TaaS.

We have implemented a prototype of WS-TaaS, but there are still some issues need to resolve. Currently, we are mainly focusing on the following problems: (1) We have applied 959 slices on the world-wide platform: PlanetLab[21], and is starting to build a TaaS platform on it to preferably realize the characteristic of geographical distribution authentically. (2) For simulating various kinds of performance conditions of user computing node, we need to study on designing some mechanisms to simulate different level of memory occupancy, network congestion and CPU usage.

Acknowledgment

This work was supported by China 863 program (No. 2012AA011203), National Natural Science Foundation of China (No. 61103031), Specialized Research Fund for the Doctoral Program of Higher Education (No. 20111102120016), the State Key Lab for Software Development Environment (No. SKLSDE-2012ZX-12), the Fundamental Research Funds for the Central Universities (No. YWF-12-LXGY-023), and CPSF 2011M500218.

References

[1] Load Testing. http://en.wikipedia.org/wiki/Load_testing
[2] M. P. Papazoglou, P. Traverso, S. Dustdar and F. Leymann, *Service-Oriented Computing: State of the Art and Research Challenges,* in IEEE Computer. vol. 40 (11), 2007, pp. 64-71.
[3] SoapUI, http://www.soapui.org/
[4] D. Nevedrov, Using JMeter to Performance Test Web Services, http://loadstorm.com/files/Using-JMeter-to-Performance-Test-Web-Services.pdf, 2006.
[5] Rational Performance Tester, http://www-01.ibm.com/software/awdtools/tester/performance/
[6] HP Testing as a Service. http://www8.hp.com/us/en/software/software-product.html?compURI=tcm:245-936967
[7] Testing as a Service. http://www.tieto.com/what-we-offer/it-services/testing-as-a-service
[8] L. M. Riungu, O. Taipale, K. Smolander, *Research Issues for Software Testing in the Cloud*, 2^{nd} IEEE International Conference on Cloud Computing Technology and Science, Nov 2010, pp. 557-564.
[9] M. Yan, H. Sun, X. Wang and X. Liu, *Building a TaaS Platform for Web Service Load Testing*, IEEE International Conference on Cluster Computing 2012, Sep 2012.
[10] H. Sun, X. Wang, C. Zhou, Z. Huang and X. Liu, *Early Experience of Building a Cloud Platform for Service Oriented Software Development*, IEEE International Conference on Cluster Computing 2010, Sep 2010.
[11] Service4All Project Overview, http://www.ow2.org/view/ActivitiesDashboard/Service4All
[12] X. Bai, W. Dong, W. T. Tsai, Y. Chen, *WSDL-Based Automatic Test Case Generation for Web Services Testing*, Proc. of IEEE International Workshop on Service-Oriented System Engineering, 2005, pp. 207-212.
[13] X. Bai, S. Lee, W. T. Tsai and Y. Chen, *Ontology-Based Test Modeling and Partition Testing of Web Services*, The 6th International Conference on Web Services (ICWS), China, 2008, pp. 465-472.
[14] T. Banzai, H. Koizumi, R. Kanbayashi, T. Imada, H. Kimura, T. Hanawa, and M. Sato, *D-Cloud: Design of a software testing environment for reliable distributed systems using cloud computing technology*, 10^{th} International Conference on Cluster, Cloud and Grid Computing, May 2010. pp. 631-636.
[15] L. Yu, W. Tsai, X. Chen, L. Liu, Y. Zhao, L. Tang and W. Zhao, *Testing as a Service over Cloud*, 5^{th} IEEE International Symposium on Service Oriented System Engineering, Jun 2010, pp. 181-188.
[16] Z. Ganon, I. E. Zilbershtein, *Cloud-based Performance Testing of Network Management Systems*, IEEE International Workshop on Computer-Aided Modeling Analysis and Design of Communication Links and Networks, Jun 2009. pp. 1-6.
[17] Load Impact, http://loadimpact.com.
[18] SOASTA, http://www.soasta.com/.
[19] TestMaker, http://www.pushtotest.com/cloudtesting
[20] J. B. Postel, *RFC 793: Transmission Control Protocol*, September 1981.
[21] PlanetLab, http://www.planet-lab.org/