Transcribe@Replay: A Web Requests Initiation Framework for Client-Side High Concurrency

Yiliang Wang¹,a, Yiping Rong²,b, Xueming Qiao¹,c, Chao Wang¹,d, Yao Tang¹,e, Dongjie Zhu³,f

¹State Grid Shangdong Electric Power Company
²State Grid Shangdong Electric Power Company WeiHai Power Supply Company
³Department of Computer Science and Technology Harbin Institute of Technology at Weihai

a whwyl@163.com, b ryp123@126.com, c 13561832200@163.com, d 49786192@qq.com, e tangyaonuli@163.com, f zhudongjie@163.com

Keywords: Performance Testing, Concurrency, HTTP, EPOLL

Abstract. As the E-Business prevails at an unprecedented rate, an urgent press is impelled for a testing tool aiming at the high client-side concurrency. In this paper, aiming at the client-side concurrent requests, a performance testing framework is put forward and a prototype is implemented as a motivating case study. First, an attempt is made and a scalable testing framework, namely Transcribe@Replay is proposed. It is composed of three components, e.g. Transcriber, Monitor, and a Replay. To emulate high concurrent requests in a scalable manner, the distributed architecture is adopted for the Replay infrastructure. Second, a prototype is designed as a motivating case study. The implementation of each component, including Transcriber, Monitor and Replay, are described in detail. And the optimization spaces are discussed in-depth, e.g. the transcription template file generation, tasks generation and dispatch for virtual users, persistent request for HTTP 1.0 and HTTP 1.1, and the communication protocol based on EPOLL, and etc. Third, an experiment is conducted and the request/response cycle is traced. Experimental results demonstrate the feasibility of the proposed Transcribe@Replay framework.

I Introduction

As Electronic-Business prevails in an unprecedented rate in recent years, an urgent press for a testing tool aiming at the client-side high concurrency is impelled. Taking TaoBao as an example, in time between dawn and 13 o’clock in the 11th of November 2015, the trade turnover exceeded 57,112 million RMB. Statistics showed that in busiest transaction hour, the client-side concurrency reached an average of 47,500 requests per second, with the peak concurrency of about 80,000 requests per second.

Taking the e-government affairs EPMBIS (Electric Power Marketing Business Information System) as another example, for which the amount of the users have reached 10 million for some specific provinces, the peak concurrency reaches about 40,000 per second. In fact, e-business and e-government appear to have taken the leading positions in internet [1]-[4].

High concurrency is of great importance to service availability for production system. Therefore, an urgent press for a performance testing framework aiming at high client-side concurrency is impelled. Therefore, for commercial website, such as TaoBao, some measures must be taken in advance before the 11th of November, and some planning repair must be taken for e-government affairs such as EPMBIS as well.

In this paper, aiming at the prompt high concurrent requests a performance testing framework is proposed and a prototype is implemented as a case study. The testing framework is named after Transcribe@Replay. It is composed of three components, e.g. Transcriber, Monitor, and a Replay. To initiate high concurrency requests, the distributed architecture is adopted for Replay. In this regard, our primary contributions are two folds.
First, an attempt is made and a scalable testing framework, namely Transcribe@Replay aiming at the client-side high concurrency is put forward. It is composed of three components, e.g. Transcriber, Monitor, and a Replay. To emulate high concurrent web requests in a scalable manner, the distributed architecture is adopted for the Replay infrastructure.

Second, a prototype is designed as a motivating case study. The implementation of each component, including Transcriber, Monitor, and Replay are described in detail. The optimization spaces are discussed as well, e.g. the transcription template file generation, tasks generation and dispatch for virtual users, persistent request for HTTP 1.0 and HTTP 1.1, the communication protocol based on EPOLL between Monitor and Driver, the optimization of Driver, and etc. An experiment is conducted by tracing browser’s request/response cycle, demonstrating the feasibility of the proposed Transcribe@Replay framework.

This paper is organized as follows. Section II presents the Transcribe@Replay framework. Section III describes the detailed design of the prototype, e.g. the Transcriber, the Monitor, and the Replay. Section IV presents the experimental results. Section V concludes the paper.

II Framework of Transcribe@Replay

Either to commercial web sites or to e-government affairs, the prompt high concurrency of client-side requests is a severe threat to service availability. This impels an urgent press for a performance testing framework aiming at the high concurrency of client requests. Aiming at the high client-side concurrency, a scalable performance testing framework is proposed in this paper. The intuition is that by transcribing a representative business service for an operator, e.g. the payment in TaoBao or the electric power accounting and marketing in EPBMIS, http packages are captured and a transcribe file is generated. Then the large amount of requests is emulated by replays. The overall framework of Transcribe@Replay is depicted in Fig. 1. It is composed of three components, e.g. Transcribe, Monitor, and a Replay infrastructure.

Transcribe is implemented by capturing the packages of representative business services. Important actions of request and response are extracted. In particular, time information is resolved and a stamp is supplemented. This is of significance to faithful replay, i.e. emulating the thinking time, keying time, and menu time of virtual users. The important operations of request and response are extracted and a timestamp is supplemented, then a transcribe file is generated in PKL format.

Monitor is the control node that communicates with the distributed Replay infrastructure, monitors and manages the each node in the infrastructure.

The scalability of the Transcribe@Replay framework lies in the Replay infrastructure. To emulate the large amount of virtual users, a distributed architecture is adopted. Each node in the Replay infrastructure is named a Driver by us. The number of Drivers in the Replay infrastructure and the

![Figure 1. The Transcribe@Replay performance testing framework. It is composed of Transcribe, Monitor, and the Replay infrastructure. The distributed architecture of Replay infrastructure acquires scalability. Each replay node in the Replay infrastructure is named a Driver. The number and the configuration of Driver can be specified according to the requirement a performance test, e.g. the concurrency of the virtual users. This embodies the scalability of the Transcribe@Replay framework.](image-url)
configuration of each Driver are scalable and can be configured according to the concurrency of performance testing. To emulate a large number of virtual users, each Driver can be configured as a high end server.

III Motivating Case Study

To validate the feasibility of the proposed Transcribe@Replay framework, a prototype is implemented. The architecture of the prototype is depicted in Fig. 2. It includes three components, the Transcribe, Monitor, and the Driver. Transcribe is in charge of transcribing representative business service and a template transcribe file is generated.

The transcribe file is submitted to the Monitor and used as a template to generate large amount of tasks for virtual users according to a configuration file. Then the tasks are dispatched to the distributed Replay infrastructure in terms of load balance of Drivers. After that, the Monitor starts the tasks in each Driver and the large amount of virtual users are emulated by replaying the assigned tasks. The Monitor monitors the status of each Driver, accumulates the emulation results of virtual users and controls the execution of Driver. The detailed implementation of Transcribe, Monitor and Driver are described as follows.

A. Transcribe

The Transcribe component includes several modules, e.g. configuration, subscribe, transcribe template generation, and etc.

The transcribe module is in charge of transcribing the operations of an operator for a particular service, e.g. the payment in TaoBao or the power rate accounting and marketing in EPBMIS. It is accomplished by http package capturing according to a configuration file.

And a raw pcap file is generated [5]. To generate the template transcription file, two steps are conducted. First, the important operations of request and response are resolved. In present work, the GET and POST methods in HTTP protocol are extracted. Second, the timing is of significance to faithful replay [6]. So the timing information of each request and response is resolved via TCP protocol and a stamp is supplemented with each operation. After that the important request and response operations extracted together with the timestamp, are used to generate a template transcription file, which is saved in PKL format. Accordingly, the thinking time, keying time, and menu time of virtual users are faithfully emulated.

B. Monitor

Monitor is the control node that monitors and manages the distributed Replay infrastructure. This component includes several modules, e.g. configuration, tasks generation, tasks dispatch, and an attendant communication module to monitor and control Drivers in the distributed Replay infrastructure.
Tasks generation for virtual users and tasks dispatch are the responsibilities of Monitor. A python program, namely Link.py, is specifically designed for generation of tasks for virtual users. It binds the corresponding account information, e.g. the username and password with each virtual user. And a task is generated for each virtual user. After that, the tasks are dispatched to the Replay infrastructure in terms of load balance for all Drivers. In the present version, it is assumed that the configuration of each Driver in the Replay infrastructure is the same.

After that, the Monitor starts the tasks in each Driver. And the Driver emulates large amount of virtual users by replaying the assigned tasks. The Monitor monitors the status of each driver, accumulates the emulation results of all virtual users in the driver and controls the execution of Driver.

C. Driver and Distributed Replay Infrastructure

The scalability of the Transcribe@Replay framework lies in the Replay infrastructure. To emulate ten-thousands of order of magnitude prompt concurrency, the distributed architecture is adopted. Each node in the Replay infrastructure is named a Driver. The number of Drivers in the Replay infrastructure and the configuration of each Driver are scalable. The configuration of Driver depends upon the concurrency of a performance testing, e.g. the peak simultaneous requests per seconds. In fact, each Driver can be configured as a high-end server. The software and hardware optimizations of Driver are of importance to client-side concurrency.

1) Persistent Connections

HTTP is the dominant protocol used by browsers to surf Internet. The utilized version of HTTP is 1.0 or 1.1. HTTP 1.0 opened a new TCP connection for each object in a Web page. As the establishment of a TCP connection requires a 3-way handshake, this overhead is of significance to high concurrency of performance testing [7]. HTTP version 1.1 proposed persistent connections, which are kept open once an object is delivered to the browser. The connections can be reused thus avoiding the overhead of 3-way handshake [8]. Although experiment results indicate that HTTP version 1.0 and 1.1 do not manifest significant difference of traffic behaviors [9], the persistent connection is of significance to high concurrency of a performance testing tool.

Under HTTP1.0, there is no official specification of how to keep persistent connection operates. To initiate a persistent connection, an additional header is added, e.g. Connection: Keep-Alive. Then the server receives this request and generates a response, server also adds the same header to the response, e.g. Connection: Keep-Alive. Following this, the connection between the Driver and the target website is not dropped but kept open instead. When the virtual user, e.g. a process or a thread in a Driver, initiates another request, connection is reused. The connection continues until either the Driver or the target website decides the conversation is over and one of them drops it. This greatly saves the overhead of 3-way handshake.

Under HTTP 1.1 the official Keep-Alive method is delved in a different way. All connections are kept alive unless stated otherwise with the following header, e.g. Connection: close. This means that the Keep-Alive header no longer takes effect and the HTTP 1.1 is a persistent request by default. In addition, for present prototype, the HTTPS protocol is supported, although not very perfectly. The excellent HTTPS support is left for future work. Besides, other protocol support, such as the HTTP-MPLEX, is left for future work [10].

2) Optimization Spaces

For the Driver, there are ample spaces can be optimized. For example, in Driver each thread is created to emulate a virtual user and the size of thread stack can be shrunk, e.g. to 2M, to increase the number of virtual users created. The TCP/IP protocol driver is modified to directly transmit data to user address space bypassing the kernel address space, and etc.

3) Communication Protocol

As stated in previous section, after the tasks are generated by Monitor, they are dispatched to all of the Drivers in Replay infrastructure. Then the tasks in each Driver are started by Monitor, and the large amount of virtual users is emulated by replay in Drivers.

In the Replay infrastructure, ten-thousands of order of magnitude of virtual users can be emulated. The Monitor monitors the status of each Driver, controls the execution of Driver and accumulates the
replay results of virtual users. Consequently, the communication between Monitor and Replay is of significance to high concurrency of client request. In the present version, a communication protocol based on EPOLL is specifically designed. Due to space constraints, the detailed communication protocol description is omitted.

IV. Experimental Results

In the proposed Transcribe@Replay framework, there is no specific requirement for configuration of the Transcribe node. The performance requirement of Monitor is a necessity. The number of Drivers in the Replay infrastructure and the configuration of each Driver are of significance to high concurrency of the performance test. For a Driver, the configuration of CPU, memory, Disk and Network is of significance to high performance testing. In this experiment, only one Driver is used. A Thinkpad W-series high performance server is used as Driver. The 28G DDR3/1600 memory and a Samsung 850 pro of 256G SSD are used. In addition, it is customized in the aspects such as the thread stack of OS, and the TCP/IP driver etc.

In this work, the www.google.com is used as the target web site, and the request/response cycle of browser is traced. The request/response cycle has seven timing components: wait, start, request, response, gap, domcontentload, and load. Wait denotes the time since start of page navigation to when the request starts. Start is the time from the request created to when it starts to be sent. Request is known as time to first byte, which is defined as the time when the request is started to be sent to the first response. Response is the time from first byte to completion of receiving the response. Gap is the time between response completion to the page load event. DOMContentLoaded is the time of the DOMContentLoaded event, while Load is that of the page Load event. In the motivating case study, the replay of a Driver node works in silent mode. So, the events such as DOMContentLoaded and Load are meaningless and thus omitted. Experimental result is depicted in Table II.

Experimental results show that client-side concurrency reaches ten-thousands of order of magnitude with a peak concurrency of about 50,000. This demonstrates the feasibility of the proposed framework.

V. Conclusions

In recent years as e-business prevails at an unprecedented rate, an urgent press is impelled for a testing tool aiming at high client-side concurrency. Taking TaoBao as an example, in busiest transaction hour of the 11th November 2015 client-side concurrency reached an average of 47,500 requests per second with a peak concurrency of about 80,000 per second. Taking the e-government

<table>
<thead>
<tr>
<th>Event</th>
<th>Percentage</th>
<th>Duration</th>
<th>Percentage</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>wait</td>
<td>3.15</td>
<td>0.77s</td>
<td>60.15</td>
<td>2.00s</td>
</tr>
<tr>
<td>request</td>
<td>6.06%</td>
<td>0.12s</td>
<td>5.48%</td>
<td>0.03s</td>
</tr>
<tr>
<td>response</td>
<td>4.01%</td>
<td>0.20s</td>
<td>4.06%</td>
<td>0.07s</td>
</tr>
<tr>
<td>gap</td>
<td>2.01%</td>
<td>0.03s</td>
<td>2.06%</td>
<td>0.01s</td>
</tr>
<tr>
<td>start</td>
<td>2.02%</td>
<td>0.01s</td>
<td>2.01%</td>
<td>0.01s</td>
</tr>
<tr>
<td>request</td>
<td>6.06%</td>
<td>0.12s</td>
<td>5.48%</td>
<td>0.03s</td>
</tr>
<tr>
<td>response</td>
<td>4.01%</td>
<td>0.20s</td>
<td>4.06%</td>
<td>0.07s</td>
</tr>
<tr>
<td>gap</td>
<td>2.01%</td>
<td>0.03s</td>
<td>2.06%</td>
<td>0.01s</td>
</tr>
</tbody>
</table>

In the Transcribe@Replay motivating case study, the replay of a Driver node works in silent mode. Accordingly, the events such as DOMContentLoaded and Load are meaningless and omitted.
affair EPMBIS - Electric Power Marketing Business Information System - as another example, the peak concurrency reached about 40,000 per second.

Aiming at the client-side high concurrency, a performance testing framework is put forward and a prototype is implemented as a motivating case study. First, a scalable testing framework, namely *Transcribe@Replay*, is proposed. It is composed of three components, e.g. *Transcriber*, *Monitor*, and a *Replay*. To achieve scalability, the *Replay* infrastructure adopts the distributed architecture. Second, a prototype is designed as a motivating case study. The design and implementation of each component, including *Transcriber*, *Monitor* and *Replay*, are described in detail. And the optimization spaces are discussed as well. The experiment is conducted by tracing the browser’s request/response cycle. Experimental results demonstrate the feasibility of the *Transcribe@Replay* framework.

**Acknowledgment**

Thanks to the students of HIT and the staff of Industrial Technology Research Institute of Heilongjiang Province. This work is supported by project under Grant no. SGSDWH00YXJS1500229.

**References**


