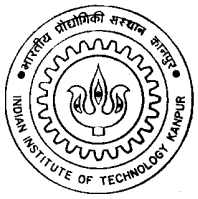
WebPerMoS: A Tool for Performance Profiling of Web Sites

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Master of Technology

by

Shashikant M. Kore



to the

Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur

April, 2003

Certificate

This is to certify that the work contained in the thesis entitled " *WebPerMoS: A Tool for Performance Profiling of Web Sites* ", by *Shashikant M. Kore*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

April, 2003

(Dr. Pankaj Jalote)
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

Abstract

In recent years, response time and availability of a web service have become an area of concern for website owners. User discontentment is quite high if the site does not meet users' expectation of reasonable response time. Hence it is utmost important to find out what the users' experience at the website is. The users' experience includes response time for the web pages and availability, ability to access a web page successfully, of the site. For a website, the average response time is not an effective metric of performance, as the users of website may be spread across globe and response time experienced by users in one region may be entirely different from users in other region.

We define a new metric called, Performance Profile which takes into account geographical location while reporting response time and availability of a website. We have developed a system to monitor Performance Profile of website from remote locations. The central node of the system, Reporting Centre(RC), coordinates the activity of the testing where as the Test Delegate, resides on actual end-users' desktop PC, monitors the site at regular interval and sends results to RC. The Delegate is capable of executing multi-step transactions on the website, imitating a real-world user interacting with website with forms. The test results are used to generate graphs indicating websites behaviour over the period of time in different geographical regions. The graphs can be analyzed to locate the break points of the website.

Acknowledgements

I take this opportunity to express my gratitude toward my supervisor, Dr. Pankaj Jalote. He provided a motivating and enthusiastic atmosphere during the many discussions we had. He has changed my thought process radically with his critical approach to the problem. Working with him was, indeed, a great learning experience! He will always be constant source of inspiration for me.

Thanks are due to all the faculty members of Department of Computer Science and Engineering who introduced to the breadth and depth of various areas in most enjoyable manner. I am thankful to Shukla ji, Tiwari ji, Chandan, Nadeem and other staff for providing with all the facilities needed.

I thank Atul ji for giving me helping hand numerous times - sometimes at the most awkward hour of the day. Kedar, my first friend in IITK, has been my elder brother, teacher and friend - all at once. I thank Kedar for teaching me to have a child-like curiosity. My classmates, seniors and juniors filled the ambience with vibrant energy. I sincerely thank them for providing me with all the memorable moments in IITK.

Throughout my thesis, I have extensively used Open Source Software. Thanks are due to the Open Source Community for their good quality software. I thank Anshul, Dwipal, Milind, Pavan and Sumit for helping me to test the system.

There are people in this world who shower me with love, encourage my each endeavour and bear my stubbornness all the time. I consider, thanking them may mount to disrespect of their affection for me.

Contents

1 Introduction **1**

1.1 Website Testing 1

1.2 Present Work 5

1.3 Related Work And Commercial Services 7

1.4 Organization of the Thesis 9

2 System Architecture and Design **10**

2.1 Test System Objectives 10

2.2 Test System Architecture 11

2.3 Design Issues 13

2.3.1 Modelling Communication between RC and Delegate 13

2.3.2 Result Reporting by Delegates 14

2.3.3 Unit of Workload 15

2.3.4 Adding/Deleting Delegates 16

2.3.5 Specifying Web Sites and Transactions 17

2.3.6 Assigning a Job to Delegate 18

2.3.7 Sampling Frequency for a Test 18

2.3.8 System Privileges of Delegate 19

2.4 Communication between Delegate and RC 19

2.5 Design of Delegate 21

2.6 Design of the RC 22

3 Implementation **25**

3.1 Delegate using HTTP for Communcation 25

3.2 RC 28
3.3 Tools used for Development 30
3.4 Class Details of Delegate 30

4 An Example **36**
4.1 Experimental Setup 36
4.2 Various Results 36
4.3 Other Experiments 39
4.4 User Feedback 41

5 Conclusion and Future Work **44**

A Installation of the Complete System **46**
A.1 Setting Up RC 46
A.1.1 Installing RC Source Code 47
A.1.2 Configuring RC 48
A.2 Configuring Delegate Source 48
A.2.1 Changes in Source Code 48
A.2.2 Building Distribution 49

B Installation of Delegate **51**

C Source Code Organization **57**
C.1 RC 57
C.2 Delegate 58

D Format of a Job Profile **61**
D.1 One-step Transaction without parameters 61
D.2 Multi-Step Transaction with Parameters 64

E Format of a Transaction Result **66**

F Format of Configuration Files **71**
F.1 rcparam.conf - RC Parameters 71

F.2 userinfo.conf - Delegate User Info 72
F.3 proxy.conf - Proxy configuration 73

Bibliography **75**

List of Figures

2.1 Architecture of the System 12
2.2 The FSA for Communication between RC and Delegate 20
2.3 State Transition Diagram for Delegate 21
2.4 The Architecture for RC 23
2.5 The Architecture for RC from the perspective of Delegate 24

3.1 Time-line of a web page access 26
3.2 Class Diagram for the Delegate 31

4.1 Adding a New Job : Scheduling Parameters 37
4.2 Adding a New Job : Access Parameters 37
4.3 Response Time of Home Page in Various Regions 38
4.4 Global Response Time of Home Page 39
4.5 Availability of Home Page in Various Regions 40
4.6 Throughput of Home Page in Various Regions 40
4.7 Cumulative Response Time of 2-Step Transaction on Google 41
4.8 Pagewise Response Time of 2-Step Transaction on Google 42
4.9 Throughput of 2-Step Transaction on Google 42

B.1 Registration Form Shown to User when Delegate is Installed 56

Chapter 1

Introduction

1.1 Website Testing

In last few years, WWW has gained significant economic relevance. The earlier days of excitement are over and WWW is marching toward an era where its quality has become an important aspect. As websites change rapidly, compared to traditional softwares, the testing is needed on continuous basis to ensure quality of the website. Insufficient testing may result into loss of credibility and at times, financial losses!

The heterogeneous environment of the Internet throws up new challenge in the area of testing. The applications are developed using plenty of languages, and run on a variety of hardware/software platforms. There are a wide variety of servers and browsers with various versions of each, running on different operating systems with different versions of libraries, different connection speeds, multiple standards and protocols, etc. In addition, we have the extra components such as caching proxy server, firewall, load-balancer, different application and database servers. Due to all these dynamic factors, testing for web sites can become a major ongoing effort rather than a one-time activity.

Paul Gerrard[11] gives broad and non-exhaustive classifications of the website testing checklist.

Static Testing consists of analysis of code without executing it. The analysis can be done manually or using automated tools. Following are the tests conducted under static analysis.

- Content Checking : Check content of web pages for precision, comprehensiveness, spellings and accessibility.
- HTML Validation : Validate HTML code against W3C standards.
- Usability Testing : User experience should be as easy and fast as possible.

Structural Testing ensures all the components of the web page are glued together as per the requirement. There might be some broken links due to typos in writing URLs. Also, it is difficult to achieve exactly same appearance for a web page across different browsers. Validating the pages across different browsers makes sure that there is not much difference in the *look and feel* across browsers. It involves conducting following tests.

- Link checking : Check for broken links, orphan pages.
- Browser compatibility testing : Test for browser-specific tags and its impact on other browsers.
- Visual browser validation : Difference in rendering (colors, scaling, table layout, border colors) of web pages in different browsers.

Functional Testing is carried out to ensure correct working of all the components. The approach taken is bottom-up, which starts with the smaller units tested independently and then combined in incremental fashion to get the desired functionality. This is a comprehensive exercise and covers following tests.

- Page testing : Test all the objects executing in browser, client-side scripts.
- Active component testing : Server side scripting, active page objects which are executed by form-based user interaction. This normally covers the business logic communicating with database.

- Transaction testing : A transaction consists of a set of steps together offering a logical functionality. Only ensuring correct behavior of individual step is not sufficient. In such cases, the integration of complete end-to-end functionality must be checked using selected cases.
- Application system testing : Cover complete testing of application with real-world scenario to confirm all features meet the requirement. Various issues to be addressed here are "stateless" web, application not controlling browser, issues with cookies, loss of connection.
- Client configuration testing : End-user can have a variety of combination of Operating System, Network connection, Internet Service Provider, Browser. Exhaustive testing for all combination is an uphill task. Check for some popular combinations.
- Internationalization : In multilingual site, verify all messages and output are translated correctly and exact functionality is carried out.
- Regression testing : Frequent changes to website by adding, changing and removing functionality. Ensure that earlier test cases give desired output.
- Soak testing : Testing system's dexterity to endure continuous running of variety of transactions which may reveal problems such as obscure memory leaks, locking or race condition.
- Java test : Java is used widely on client and server in the form of applets, java server pages, servlets, EJB, swing.
- Section 508 testing : Section 508 requires that US Federal Agencies' electronic information is accessible to people with disabilities.

Performance Testing is often a non-issue in normal software because the software runs in a controlled environment which will be stable over the period of time. But that is not true with website as it runs in essentially heterogeneous and uncontrolled environment. It is prudent to ensure that the system performs predictably under various conditions.

- Load Testing : Determine website performance at predefined level. Transaction volumes are unpredictable, user behavior and variables uncontrollable, system variables such as network, hardware, software, hosting, load-balancer, firewall make things complicated. Load testing helps to uncover performance and uptime problems.

- Stress Testing : Determine web site's breaking point or unacceptable performance point. Test system functionality while system is under peak load.

- Regression Testing : Changes in content, design and layout of web pages may affect response time. Check whether response time is within acceptable limits.

- Performance Monitoring : Ensure website is available and performing optimally. Test page/transaction availability from multiple locations.

Non-Functional Testing addresses issues which might crop up due to change in requirements, design and other aspects. The security concerns are high in web-based applications. All such issues are addressed in non-functional testing.

- Scalability testing : Testing system's architecture and its capability to meet future requirement. Areas of scalability are hardware, application software, database and middleware.

- Security testing : For website using firewall, encryption, user authentication, monetary transactions, access to database with sensitive data, digital certificates.

- Reliability and Recovery testing : Analyze behavior of the system under selected failure conditions. Simulate failure scenarios and confirm the system will handle them successfully.

- Server log analysis : Logs can be used to track website usage, to find errors encountered.

Large-Scale Integration is concerned about coupling existing application with the new website. All the interfaces are identified and verified for the correctness of functionality.

- Legacy System Integration : Some applications need to be integrated with legacy system. Physical interfaces must be known in such cases.
- End-to-End functionality : Demonstrate, the system meets the objectives. This is associated with use acceptance.

While the "To-Do" list of testing quite huge, the resources are often limited. Hence the prioritization of testing a becomes a necessity.

1.2 Present Work

The key expectations of a user visiting website are the usability and performance, not necessarily in that order. Usability is more of an art than science. The two key performance parameters for a user visiting a site are availability and response time. The response time of a web page is the time interval between sending the request for the page and receiving the page completely. The availability of a service is the probability to complete an activity successfully. Performance is something which is affected by myriad factors, controllable and uncontrollable included. As *On Internet, your competitor is just a click away*, and users are not willing to wait, response time has become a key factor in the success of a site. Clearly if the site is not available or if response is very slow, users' experience worsens, resulting in loss of users. The importance of availability is highlighted by *Scott McNealy* of *Sun Microsystem* who once said, *"We are paying people for uptime. The only thing that really matters is uptime, uptime, uptime, uptime and uptime"*.

The response time at end-users heavily depends upon factors such as connectivity to Internet, time of the day, Internet Service Provider, etc. Geographical location also plays an important role because ISPs and connectivity will change with country.

And so does the distance (in terms of number of hops) of web server from the end-user. Hence the response time of a site is different as perceived from different geographical locations.

We define **Performance Profile** of a site/service as performance of the site over time, as seen in different geographical regions.

The users of websites are spread across the globe. The users' experience in one region may be entirely different than other region. It might happen that the bad performance in one region is compensated with very good performance in other region, resulting into a situation where average performance is within acceptable limits. This situation does not reveal the problem faced in certain areas. Hence, it is not the average performance, but the Performance Profile that is important to the site owner/webmaster.

This leads us to believe that it will be helpful, if we provide a tool which can give exact behavior (response time and availability) of the website by conducting some field tests. This information, along with the information on load from different regions (which only a server can collect) can be invaluable in deciding the location to mirror or replicate a service.

In this thesis, we describe a distributed testing tool that we have developed for monitoring the global performance of a site. The tool has a centralized site where request for performance monitoring can be given and from where the global performance profile is obtained. However, to determine the performance profile, the tool uses delegates running in different regions. The number and location of delegates is decided by the tool owner. The delegates work under the control of the central site, perform the testing they are asked to and send the results to the central site. The central site compiles data from the different Delegates to create the global performance profile of the site being tested.

1.3 Related Work And Commercial Services

There are other approaches for measuring user-perceived response time. One approach is to instrument the web pages with client-side scripting. [22] When a link to an instrumented web page is followed from a web page, which itself is an instrumented web page, the script starts a timer. When the web page is downloaded completely, the script notes the time. The difference between the two timestamps gives the response time of the web page. This response time is, then, communicated to the website. This approach gives response time only for some instrumented pages and that too with the prerequisite that the page is accessed from another instrumented page. The response time for each of the embedded object such as images, flash animations, style sheets, etc. cannot be calculated.

Another approach is to collect and analyze the packet trace from the server to determine the server performance.[9]From the network trace, the communication between client and server is reconstructed, and heuristics are used for determining response time. This approach needs a huge amount of disk and CPU resources, and is impractical for a very busy site. This method only gives response time and not availability of the service. It does not work if web pages have embedded components from different servers.

Few of the services and tools available which monitor performance of websites[13].

- ActiveWatch [1] is a hosted monitoring solution that measures performance and availability of Web sites from outside the firewall. It is the service that offers a complete view of end-user performance from multiple locations around the world while providing alerting, diagnostic and reporting tools to identify and fix problems.

- FarSight [7] is a hosted service that lets you monitor your Web site in real time, using performance data gathered from points on a global network. The service provides both URL and transaction monitoring from outside firewall, from a secure network. FarSight provides performance data for every object

on a Web page, for every test and from every node. FarSight saves that data which can be used for historical analysis to identify the cause of performance degradation and errors.

- eValid [6] Test Services enable companies and web site managers to monitor E-commerce service validity, availability, internal quality, and performance – as perceived by typical users with realistic test scripts. eValid Test Services are based on use of versions of the eValid Test Engine combined with custom-developed test scripts and custom-developed support utilities. The "user perspective" testing assures site's quality in the context in which customers use it.

- Keynote [23] Web Site Perspective Consumer Edition provides Web site performance data of sites accessed by "at home" consumers working remotely over 56 Kb modem, DSL and Cable Modem connections. This service provides information about how Web sites perform for lower-bandwidth users, the performance of secure pages across phone lines. Keynote measurement computers connect to the Internet once per hour and report data back through a variety of reporting options including e-mail or raw data via FTP. Web Site Perspective Business Edition enables customers to analyze full-page measurement data, from worldwide locations over T-3 and other high-speed business connections.

- Gomez Performance Network [12] measures Web page and transaction performance in real-time, monitoring site availability and responsiveness. From over 50 locations spanning more than a dozen Internet backbones, secure service provides the data which helps to identify factors that impact Web site performance.

- Porivo's peerReview [20] is a service that measures the end-to-end performance of Web applications from the customers' perspective. Using the global testing network - real PCs spread across the globe - peerReview provides an end-to-end evaluation of the speed of your Web site. Porivo is now a part of Gomez.

Most of the services use points of Internet backbone to monitor the sites, testing a site from a T1 (1544 kbps) or higher (T2=6312 kbps, T3=44736 kbps) link. This gives highly unrealistic information This gives highly unrealistic information particularly for users connected by slower links. Also, as they are commercial services, very little information about their methodology, is available. Some of these also seem to need manual scripting.

1.4 Organization of the Thesis

This thesis is organized into various chapters, each throwing light on some aspect of the tool. In Chapter 2, we discuss the system architecture and design employed. Chapter 3 is about the implementation details of the tool. In Chapter 4, various experiments conducted and their results are discussed. We discuss the conclusion and pointers to future work in Chapter 5.

Appendix A gives detailed instructions to deploy the system. Appendix B is the installation manual for the Delegate. The source code organization is given in Appendix C. This will be helpful in future if there is a need to change some source. Appendix D and Appendix E give specification of a job profile and a transaction result, respectively. Appendix F discusses the configuration files used by the Delegate.

Chapter 2

System Architecture and Design

To recall, our goal is to build an easy-to-install and easy-to-use system for testing the Performance Profile of web-service as seen by the users in different geographical regions. In other words, we want to actually measure the performance of the site in different specified regions over a period of time and then show, how the performance in different regions changes with time. We also want to measure the availability as viewed in these different regions and report the times when the service was unavailable.

2.1 Test System Objectives

The basic objective of the testing tool is to monitor the performance profile of a website from some specified regions. For performance, we focus primarily on availability and response time. Some of the key objectives of the system are :

- The end-user perceived performance of the website from different regions of the world should be directly measured from those regions. In order that the results are not biased by a particular site, there should be possibility of redundancy in each region. That is, the tool should allows performance to be measured from multiple sites in a region to determine the performance of the web site in that region.

- All the test should be controlled from a centralized location. This means that request for monitoring a site, along with the specification of the regions from which it will be monitored should be given at one site. Furthermore the profiling results should be available from this site itself.
- The complete system should be configurable, easy-to-use and easy-to-install and the testing process should be automatic with minimal manual intervention.

With above goals in mind, we discuss the system architecture and design.

2.2 Test System Architecture

Since the performance has to be measured from different regions, we clearly will need agents of our system operating in these regions. We call these agents as Test Delegates, which collect all the user-level performance information. To provide a central point where the entire performance profile across different regions can be determined and displayed, we have a special site called Reporting Centre (RC). The RC coordinates the entire testing activity, as well as distribution of of Delegates. All Delegates communicate only with RC. In other words, the system has a star architecture with RC at the centre. The nodes are the Delegates which interact with website being tested.

The high-level architecture of the system, describing major components and interfaces, is given in Figure 2.2.

The **Test Delegate** (here onward referred as Delegate) is the test agent which resides on the PC of an end-user. The Delegate gets an environment which is very close to that of an user of any website. The Delegate can be considered as a mini-browser without the GUI. A *Region* is a predefined geographical zone. For example, continent can be a region and so can be a country. The Delegate belongs to an unique region. The Delegates belonging to a region together define performance of web site for that region. We have chosen continent as region with India as special

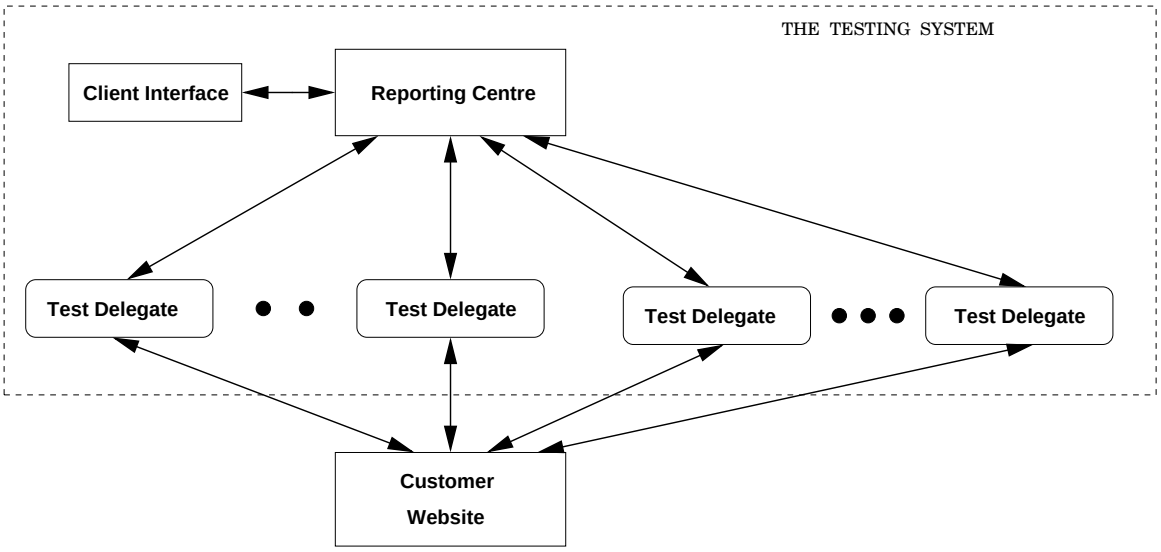


Figure 2.1: Architecture of the System
The direction of the array indicates the active access by the component

region. Functionalities of the Delegate include *Registration*, *Communicating* with RC and *Scheduling* the jobs.

The Delegate registers itself with Reporting Centre, described below, when it runs for the first time. Whenever the Delegate is active or running, send Alive message to Reporting Centre at regular interval. When Reporting Centre assigns some job to the Delegate, it gets the profile for that job from Reporting Centre. Delegate schedules the jobs as per the requirement, conducts the test at scheduled time and notes down various results for the job. All the results for the executed test are uploaded to Reporting Centre.

Reporting Cetre is a central node from which the activities of the system are co-ordinated. The important responsibilities of Reporting Centre (here onward referred as RC) including Adding a new job, Communicating with Delegates and Generating results.

When a new job is submitted for monitoring to RC, all the required parameters are specified. The Delegate when runs for the first time, registers with RC along with the registration information. According to the need of Job, RC identifies a Delegate which meets the requirement of the job and assigns the job to a particular Delegate. RC collects test results uploaded by Delegate. The test results are aggregated to produce various results and statistics about the website.

Client Interface is an interface of RC through which the user of the system interacts. The user can submit new jobs specifying details and requirements of the job. The user can get the reports generated as per the parameters specified.

Customer Website is the website whose performance is to be monitored. Henceforth, the word website refers to Customer Website, unless otherwise stated.

2.3 Design Issues

In this section, we discuss some of the alternatives possible in detailed design. The various approaches are explored, compared and finally the one which suits our needs is chosen.

2.3.1 Modelling Communication between RC and Delegate

The communication between RC and Delegate can be modeled in two ways. In the first approach, the Delegate sends Alive message at a regular interval. Whenever a test is to be conducted, RC sends the profile of the test to the Delegate as a reply to Alive message from the Delegate, the Delegate conducts the test and uploads the results immediately. The Delegate does not store any of the test or result information with it. Thus, RC schedules various jobs which are currently active. In a second approach, which generally will have a higher interval of communication, the Delegate sends the Alive message at regular interval. If there is some job to be assigned to the Delegate, the profile is given to the Delegate. This profile also has

the information about duration in which tests should be conducted. The onus of scheduling the jobs is on Delegate.

The advantage of first approach is simple Delegate design. But it also means more work on the part of RC. Also, RC can be almost sure that the test is conducted by Delegate. The first approach has lot of communication overhead as it might download same profile again and again. In second approach, once a job is assigned to Delegate, RC assumes that it will conduct all the tests. The Delegate may go offline for some time, which translates to lesser number of test for a job than required. We have chosen the second approach because of lesser communication and simple design of RC and Delegate.

2.3.2 Result Reporting by Delegates

When Delegate conducts the tests, it records the results of the test. All these measurements are collectively taken as result for that test. These results are, then, uploaded to RC. In the first approach, Response time is measured for each page of the transaction and each component of the page. For each page and its components, individual size(in bytes) and response time(in millisecond) is recorded. If an error condition occurs for some component, it is noted down. All the measurements are uploaded as raw data. The results are not interpreted by the Delegate. In the second approach, only high-level details are measured. Response time is measured for each page of transaction, but component response time is not taken into consideration. They are assumed to be part of the page. The size of the page is recorded inclusive of all the components. Also, throughput for the page is calculated by considering the size of the page and response time.

While uploading the result, the first approach needs more bandwidth than the second one. But first approach gives the flexibility of interpreting the results. With second approach, there is possible information loss as component results are not measured. We have chosen the first approach.

If there is any major error occurred while executing any step of the transaction, that test is abandoned and its results is said to be an error in that step. The failed test indicates unavailability of the service. All the test results are reported to RC in a specific format. It starts with high-level details such Delegate ID, Job ID, time at which test was conducted, result of the test (either success or failure) and response time for complete transaction. Detailed results of each step of transaction follow this.

After the Delegates start monitoring the site(s), the results for those tests trickle to RC over time. The results from Delegates belonging to one region are grouped together so that region-wise results are available. To calculate response time, the results in error are not taken into consideration. *The average response time from all the Delegates at particular hour of day* is taken as the response time of the transaction at that hour in that region. Availability of the transaction from a region at particular hour is given as *the number of successful tests conducted by total number of tests conducted in that hour.* Throughput means Bytes received per Second from the site. For each transaction, we know the size of data (in Bytes) received and time taken for completion of transaction. The results in error are discarded. Now, we have throughput for each successful transaction. Now similar to method used to calculate region-wise response time, throughput for each-region at each hour of the day can be found.

2.3.3 Unit of Workload

A node running a Delegate may not want the resources being used above certain threshold. There can be disparity in the contribution to the testing by Delegates. To differentiate the resource allocation by different Delegate, an unit is needed which will measure the workload. The following options were examined.

Each transaction has certain number of steps, each step being an access to a web page. The transactions have a scheduling period. If the scheduling period of transaction is 30 minutes, the transaction will be executed after every 30 minutes.

From these two parameters, we define a unit, Pages Per Hour or PPH. The idea is Delegate has to access certain number of pages below a threshold. Consider a transaction with 2 steps is executed at the interval of 30 minutes. Hence the workload for this transaction is 4 PPH.

The web page accessed translates to use of bandwidth. If we know the size of the web pages, the workload can be assigned depending on bandwidth consumption. For this, the unit of workload will be Kilo Bytes Per Hour or KBPH. In this case, the idea is Delegate has to access pages such that the cumulative data fetched is below a predetermined threshold. If a transaction consists of 2 pages (sizes of which are 40 KB and 60 KB), and scheduling period for the transaction is 30 minutes, then the workload for this transaction is [(40 + 60)*2] = 200 KBPH.
The first approach is simple, but the granularity is coarse. The second approach is very precise, offers fine granularity; but complicated. Also, the sizes of the pages may change with time. The RC uses first approach of PPH for assigning workload as it is more intuitive.

All the Delegates may not give same amount of service for testing purpose. To assort such Delegates, we have defined three levels of rights. With *User* rights, least amount of resources can be used for testing. The RC has fixed the workload given to such Delegate at 10 PPH. *Administrator* rights are higher than *User* and workload is set to 20 PPH. *Dedicated* is the highest right and workload is set to 30 PPH. While registration of the Delegate, the user chooses the right level.

2.3.4 Adding/Deleting Delegates

As the test system is to monitor the global performance profile, it is to be expected that new Delegates may join the system and old may leave. Hence, such a system must permit adding/deleting of Delegates. The procedure for adding new Delegate is a simple one. When the user installs the Delegate, user is shown a form. The user enters information such as Name, Email Address, Region to which he belongs, City, Operating System, Link to Internet and Proxy configuration, if any. The unique

identity of the Delegate will be a parameter called DelID, which is generated by the Delegate itself. Now, the Delegate registers itself with the RC providing all this information. Now the Delegate is added to the system and participate in the testing process. In this way, a Delegate is added to the system dynamically. Thus RC can start with no Delegates in the system and will get the Delegates on the fly. Currently, for deleting a Delegate the configuration file containing the names of the Delegates have to be edited. Note that if a Delegate goes offline, it will not send Alive messages, hence it is not treated as active and jobs are not assigned to it.

We have defined region to be a continent i.e. in our system the regions are Asia, Europe, Africa, North America, etc. But this can be changed to, say country, with some modification. The regions, shown in the form to user while installing the Delegate, can be changed to the desired values. The regions now reflect the new values and results will be generated according to it. Thus adding and deleting region is easy task.

2.3.5 Specifying Web Sites and Transactions

A transaction is sequence of one or more steps to be followed on web site to complete a logical activity. For example, visiting home page of site is a one-step transaction, visiting home page and then searching the site with the form provided is a two-step transaction. This enables Delegate monitor a transaction which involves interaction of end-users with web site using forms. Each job is identified with JobID generated by RC. Each job has set of scheduling parameters and access parameters. Scheduling parameters specify the duration in which the test should be conducted along with number of tests to conducted in that duration. The access parameters specify which web pages should be accessed.

The customer will specify the parameters for the job using a form-based interface at RC. The customer will specify the scheduling parameters for the job and then, for each step of transaction more details such as URL of the page, HTTP Method, parameters for the page, if any, are specified.

2.3.6 Assigning a Job to Delegate

The Delegate sends an Alive message to RC at regular interval indicating it is online. Whenever there is a Job at RC for monitoring, RC checks if the this Job can be assigned to the Delegate. If yes, RC gives Job ID to Delegate indicating it has been assigned to the Delegate. Now the Delegate asks RC for the profile for that job and RC replies with the profile for the Job. This profile is in a predefined format, which both - Delegate and RC - understand. Delegate adds this job to its queue of jobs. The scheduling of the job starts on the day and hour mentioned in the profile. All the time parameters are synchronized to GMT which takes care of different timezones. From the scheduling parameters, Delegates find the interval at which the test has to be executed. Delegate maintains a timer for each job which helps scheduling the job at correct interval. When the scheduling period expires, Delegate removes that job from its queue.

2.3.7 Sampling Frequency for a Test

Sometimes, the apparent failures of website are instantaneous. Such errors come because of transitory problem with local network. The test conducted may hit such failure and conclude that the page is in error and abandon the transaction. Such situation is definitely undesirable as it gives false alarms. Such momentary error conditions may vanish if we execute the same test again. Sampling Frequency is the number of tests to be conducted to report the performance of the site at that time. We have taken this parameter to be 2. This means, whenever a test is to conducted, actually two tests are conducted by the Delegate within a short interval of time and the final result is the average of results of the two tests conducted. If both the tests are successful, then component average of response time is taken as final result. If failure is occurred only for one test, then that result is discarded the other result is take as final. The result of test is failure only if both the tests fail. If both tests fail that surely indicates that it is not an instantaneous failure, but the service is actually unavailable. Only the final result is uploaded to RC. Calculating average of two results is described in 3.4. The catch in this method is, the amount of work done gets doubled. Though the workload assigned to a Delegate is N PPH, it is

actually does the work of *2N* PPH.

2.3.8 System Privileges of Delegate

The Delegate should run as long as the PC is ON. The Delegate should automatically start whenever PC boots up. We need this feature because the user does not have to worry about starting the Delegate manually every time PC boots up. Also, we want Delegate to run on as many operating systems as possible. Such program is called *daemon* in Linux/Unix and *service* in Windows. As the start-up scripts for daemon have root/superuser permissions, the Delegate also runs with those privileges.

2.4 Communication between Delegate and RC

The protocol for the communication between Delegate and RC has been modeled by a set of Finite State Automata (FSA), with one FSA representing one part of the system. [17] The FSA for the RC and a Delegate in the system is shown in 2.2.

The state of an FSA represents the state of the protocol at that particular node. A transition in an FSA consists of receiving one or more messages, optionally, performing some computation and sending zero or more messages. These FSAs are non-deterministic. Transition in FSA is marked by a condition on the conditions on the messages received (the *"numerator"*) that cause the transition, and the message the FSA sends (the *"denominator"*).

The *initial* state is denoted by q.

Reporting Centre is waiting in state w for the messages from Delegate. The Delegate may send ADD or ALIVE message. When the Delegate runs for the first time, it sends ADD message indicating it wishes to register itself. If the Delegate has already been registered with RC, it sends ALIVE message informing that it is active.

When RC receives ADD message, it registers the Delegate with itself and sends YES message indicating successful registration. Now, the Delegate does not have to

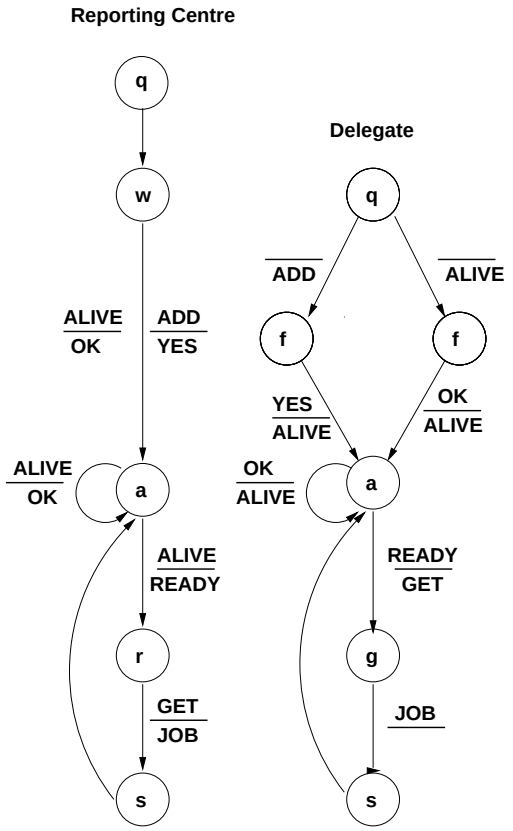


Figure 2.2: The FSA for Communication between RC and Delegate

send ADD message again in any session. The reply to ALIVE message is OK which means the Delegate will be considered for assigning some workload.

When RC has some workload to be assigned to the Delegate, it sends READY message as reply to ALIVE. Delegate then sends GET message asking for the job profile. After receiving GET message, RC sends the JOB message, which includes the job profile of the workload for Delegate. The Delegate, after receiving JOB, is in state *s*. In this state, the Delegate schedules and executes the tests for jobs assigned to it.

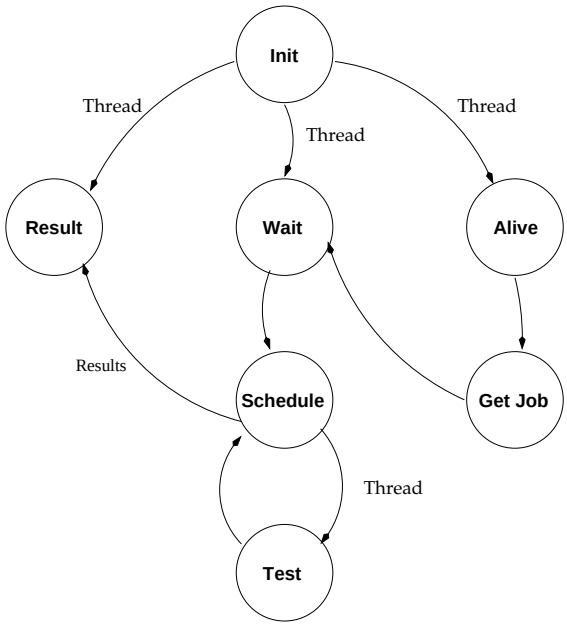


Figure 2.3: State Transition Diagram for Delegate

The Delegate and RC, both, work in an infinite loop which does not end. The FSA stops only when the whole system is dismantled. Similar to *Adding a Delegate*, there is no state machine for *Deleting a Delegate* from the system. Timeouts are used to differentiate between Alive and Offline Delegate.

2.5 Design of Delegate

The Delegate has to do multiple tasks at the same time. It needs to contact RC when it might be conducting tests for some job. Hence, the Delegate has to be designed as a multi-threaded application. Some threads might work independent of others whereas some threads need assistance of others to complete their task.

When the Delegate starts, it is in *Init* state. In this state, the Delegate reads all the configuration parameters associated with it. If the Delegate has not been registered with RC, the Registration procedure is completed. Unless Registration is

done, the Delegate cannot proceed.
The Delegate comes out of *Init* and three different controls of execution begin. Each thread performs different function. The *Alive* state represents the thread which sends Alive message to RC at regular interval. If RC replies with *Add Job* message, the Delegate then downloads the profile of the new job assigned. All the profiles to be monitored are stored together.
The *Result* thread takes care of the results of various tests conducted by the Delegate. The results of transactions are stored locally in a file. The results are uploaded to RC at regular interval. The uploaded results are, then, discarded from local store to avoid multiple uploading. If there are no results to be uploaded, this thread goes in wait mode for a stipulated interval before checking for the new results.

The *Wait* state indicates waiting for some jobs to be monitored. When the Delegate receives some job for monitoring, it goes into *Schedule* state. The Delegate schedules various jobs as per the parameters given in profile. When the time comes to execute a test, a new thread *Test* is started to execute the transaction. This thread conducts the test and collects the results. When the test completes, it returns the results for the test conducted and the thread *Test* is destroyed. The *Schedule* submits the results for this transaction to *Result* for further processing.

2.6 Design of the RC

The layered architecture for RC is shown in 2.4. The figure shows the components accessed by the user.

Authentication is the common access layer. All the components are accessible only after authentication. The user has to provide login name and password, which is verified against the values in database. If authentication is successful, access to other module is allowed.

Submit Job is an interface to submit a new job. User has to submit various information about the job such as the duration of job, hours of the day during which test should be executed, number of tests in a day, number of delegates to be assigned this job and number of steps in the transaction. For each step of transaction, URL,

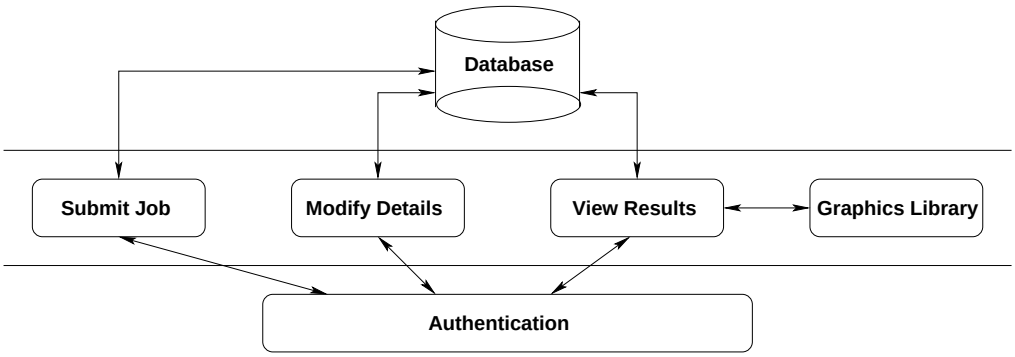


Figure 2.4: The Architecture for RC

port number, HTTP Method and version, parameters if any are read from user. Finally, user gives the locational preference for the delegate.

The user can use *Modify Details* to update registration details such as password, contact info, etc.

View Results This module shows various results for the jobs submitted. The results are generated in graphical format (line plots) using the values from database. The various results that can be seen are

- Response Time for transactions for a single day or for a period. The transaction results can further be analyzed with details of each step.
- Response Time in each region and cumulative. Same as above except the location consideration.
- Availability of Transaction - region wise as well as cumulative.
- Throughput for Transaction for a given time interval.

The results are shown by generating graphs. For generating graphs, *Graphics Library* is used. This is a third-party library.

The Architecture of RC as seen by Delegate is shown in Figure 2.5.

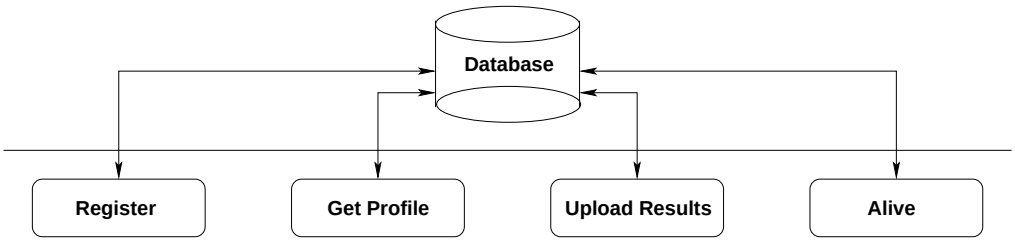


Figure 2.5: The Architecture for RC from the perspective of Delegate

When Delegate runs for the first time, it completes registration process using *Register* interface of RC. The registration process is done only once. When Delegate is running, it sends Alive message to RC at a regular interval. These messages are sent to RC using *Alive* module of RC. When RC has some job to be assigned to the Delegate, RC replies the Alive message with the Job ID. The Delegate then contacts *Get Profile* unit to download the profile of the job. The Delegate can submit the results of various tests conducted to *Upload Results*.

Chapter 3

Implementation

In this chapter, we give a brief description of the software developed to implement the approach proposed in previous chapter. We have implemented the Delegate in Java language. Java was chosen so that Delegate becomes platform independent. Also, writing multi-threaded application is easier with Java due to its thread support. The RC has been implemented using PHP scripting language and MySQL as database. The client interface is also written in PHP. However, as it is straightforward, we do not discuss it any further.

3.1 Delegate using HTTP for Communcation

The Delegate is a mini-browser that emulates the actions of an user of the website. The Delegate uses only HTTP for communication, which means communication with RC is also based on HTTP. The most basic function Delegate is to access a web page located at an URL. The steps involved in accessing a web page are shown by a time-line of access in Figure 3.1. These steps are described below.

Establish the connection with server or proxy depending on the configuration. The time interval (T_2-T_1) gives the *Connect Time*. HTTP request is sent to the server. The time interval (T_3-T_2) is the *Request Latency*. When the server receives the request, it processes the request and starts sending the response. T_4 indicates

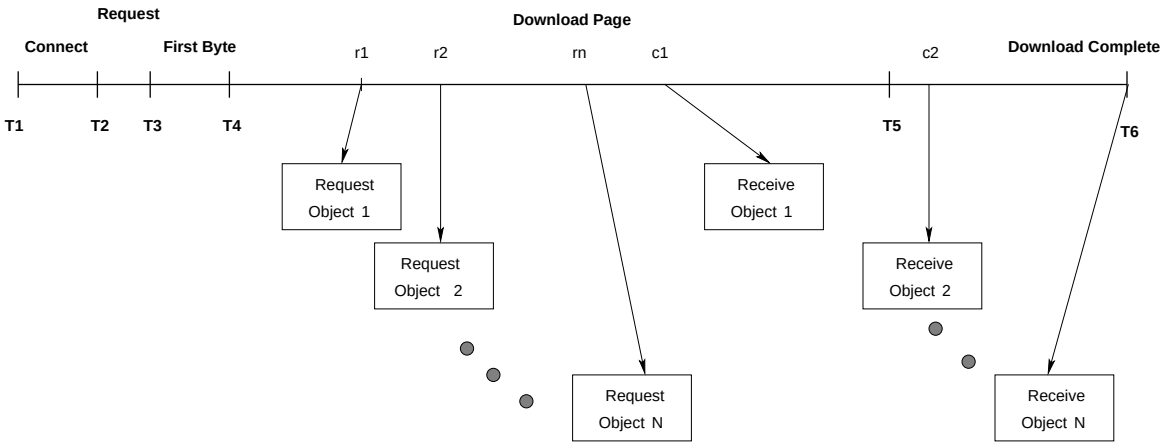


Figure 3.1: Time-line of a web page access

the point in time when the first byte from the server is received by the client. (T_4-T_1) gives the *First Byte Latency* - time required to receive first byte of data from server. *First Byte Latency* is slightly misleading as the data is read from socket in chunks of bytes, which means at T_4, first chunk of response has been received from the server.

The web page may have embedded objects in it. The components of the web page are retrieved and fetched concurrently. The various components of the web page viz. images, flash animations, style sheets are identified with the help of HTML tags. All the components are fetched without waiting for the web page to finish. Whenever a new object is found in the web page, request for that object is sent to the server. The request for objects are sent at r_1, r_2,..., r_n. The objects are received completely at times c_1, c_2,..., c_n for requests r_1, r_2,.., r_n respectively. The response times for these object will be (c_1-r_1), (c_2-r_2) and so on.

At time T_5, the HTML code of the page has been downloaded completely. Hence, (T_5-T_1) gives the download time for HTML code of page. At time T_6, the last component of the page is received completely. This is not necessarily the last component for which request has been sent. Thus at T_6, the page is downloaded along with

all the components of the page. The interval (T_6 - T_1) gives the complete response time for the page.
The explanation given is simplistic one. The actual process might be much more complicated that this.

Here is the list of features supported by the Delegate. As explained in Section 3.1, the Delegate emulates a browser. Here is the list of features supported by the Delegate.

- There is a limit on maximum number of threads that can concurrently run. At a time, only a limited number of components can be brought concurrently by establishing connection with server. We have set this limit to 8, same as that of Mozilla.

- The Delegate supports only HTTP Version 1.0. But the sites using HTTP Version 1.1 can still be accessed.

- The web site may have some functionality in which user interacts with website using form-based interface. The action of such user can be emulated by Delegate. The Delegate can access web pages with HTTP GET as well as POST method. Parameters can be sent along with GET or POST request giving a feel of user interaction with website.

- The Delegate can interpret few of the HTTP response codes. The error conditions, indicated by status code 4XX and 5XX are identified by the Delegate. Delegate only understands 4XX (Client Error) and 5XX (Server Error). The finer granularity in these errors are not interpreted by the Delegate.

- Redirection with HTTP status code 3XX (in which the new URL is specified with *Location* header) is supported.

- The Delegate can run behind an HTTP proxy.

- The Delegate cannot access the secure web pages i.e. HTTPS/SSL is not supported by the Delegate.

- The Delegate cannot execute transactions which use cookies for maintaining the state.
- Some web pages which use frames cannot be brought correctly by the Delegate.
- The components URLs which are generated dynamically cannot be identified by the Delegate. For example, JavaScript is used to generate the URL of the banner-ads dynamically. Also, pop-up windows can be created by these scripts. Delegate can only find the components with static analysis.

These are not very serious limitations. Frames are not considered a good design. Sites using HTTP Version 1.1 can be accessed using HTTP 1.0. And hooks are provided in the Delegate for further improvement in features.

3.2 RC

RC is a collection of PHP scripts, most of which use the database. Here is a list of main scripts along with their functionality.

- *register.php* : Registers the new Delegate in the System, updates the database record with the registration information and sends OK message to the Delegate.
- *alive.php* : Accepts Alive messages from the Delegate and Assigns a new job to the Delegate, if any.
- *resupload.php* : Receives the results of transaction from Delegate. The results are sent by using HTTP POST Method in which the data part is the result of transaction. The results in plain text are parsed for different values, such as transaction result, step-wise result and detailed results. All these results are entered in database along with the Delegate ID, Job ID and time stamp of transaction.
- *getjob.php* : Sends the job profile for the given Job ID.

- *newtra.php* : This is an interface to submit a new job for monitoring. All the scheduling and access parameters for the transaction are specified with the forms presented.

- *getres.php* : For given parameters, it shows the results with various graphs. The input parameters for this script are the duration for which results have to be generated, type of result (response time, availability or throughput). This script uses other scripts (described below) to generate graphs of each type for each region. All the Delegates in one region define the performance of the site in that region.

- *restimeday.php, restimedur.php* : Generate the *Response Time* graphs for a day and for given duration, respectively. The response time results for a given Job ID and duration are fetched from database. Only the results which are not in error are taken into consideration. X-axis is the time axis, where as Y-axis is the response time in milliseconds. Any point on the graph indicates average response time from all Delegates at that time.

- *avday.php, avdur.php* : Generate the *Availability* graphs for a day and for given duration, respectively. All the test results for given duration are selected from database. Now at each hour of the day, the total number tests and total number of failed tests are counted. From these two parameters, the availability is the total number of successful tests over all the test. Thus the availability varies from 0.0 to 1.0.

- *thr.php* : Generate the *Throughput* graphs for a job. The database has detailed information about each step of transaction. The tests with error are not taken into consideration. For each successful test, the throughput can be found from total bytes received and response time. Now the procedure similar to the above discussed response time calculation is followed to plot the points. Y-axis indicates bytes received per second in the graph.

The scripts get executed whenever they are invoked actively by Delegate or the user. The states shown in Figure 2.2 correspond to a script here. The model is

process-based, but actually these processes do not exist. The intended functionality is accessed actively.

3.3 Tools used for Development

HTML has tags [5] for embedding various components e.g. images are specified with IMG tag, style sheets with LINK, java applets with APPLET and other components such as flash object are specified with OBJECT or EMBED tag. The Delegate uses JavaCC HTML Parser [15] to locate these embedded objects of the web page. Apache Ant, a Java-based build tool, has been used for building the code. It is similar to Make on Unix platform. Java Service Wrapper [14] is used to run the Delegate as Windows service.
RC uses JpGraph [18] library to generate the graphs in PHP.

3.4 Class Details of Delegate

The Delegate has been modeled with Object-Oriented design. The class diagram is shown in Figure 3.2. The various classes are described below.

The *Profile* class takes care of the profiles of various jobs assigned to Delegate. All the profiles are stored in a text file. The attribute *ProfileFileName* is the file name in which profile will be stored. The operation *GetJobList()* reads all the profiles from file and returns the list. Given the JobID, *DowloadProfile()* gets the profile of the job from RC and appends the downloaded profile to the list. Given the Job *GetJobDetails()* returns various scheduling and access parameters of the Job, if it exists. All the jobs have the scheduling interval associated with them. *PurgeExpired()* removes the job - of which scheduling period has expired - from the file.

Whenever the Delegate is running, it informs RC about itself by sending Alive message. This is an indication that Delegate is running. *SendAlive* takes care of

Figure 3.2: Class Diagram for the Delegate

sending these messages. The attribute *Period* specifies the interval between two alive messages.

The various parameters which are used globally, are with class *Global*. Various classes need to interact with RC for downloading profile, uploading result, sending alive message, etc. This class provides all such parameters for accessing RC with *RCParameters*. If the Delegate is being used inside a firewall, the configuration parameters for the proxy are available with *ProxyConfig* which includes, proxy URL and authorization. This class reads all these parameters from a configuration file. Various classes access these attributes with *get...()* method.

Each Delegate has zero or more jobs assigned to it. Each job has certain scheduling parameters such as frequency of executing the test, start and end hours of the day. The *Scheduler* has to execute the test as per the requirement of the job. The attribute *JobList* has the list of the jobs which have been assigned to Delegate. *Schedule()* picks up the job from the list which needs to be executed, starts the job, collects the result when it is done and finally submits the result. The scheduler, then, waits till the time of dispatch for next job. The operation *RemoveExpired()* is invoked when alloted test duration of a job expires.

The results of the tests conducted are managed by *Result*. The results of the tests executed are stored in a text file. *ResultFileName* identifies the file. *SubmitTrResult()* is invoked with result of transaction as parameter. The result is written to the file in a specific format. *GetResult()* returns the result which is yet to be uploaded to RC. If there is no such result, then an empty result is returned. *MarkUploaded()* is called with parameter which identifies the uploaded result uniquely. That result is marked as "uploaded". The result file will have entries which are uploaded and some yet to be uploaded. *PurgeUploaded()* removes all the uploaded results from the file.

TransactionResult has the result to be uploaded to RC. *UploadResult* handles the activity of uploading these results. The result is fetched by invoking *GetResult()* of

Result. This result is uploaded to RC by *UploadTrResult()* method. In reply, RC gives a message to indicate success. If this message is received from RC then that result is marked uploaded with *MarkUploaded()* of *Result*. When there is no result to be uploaded, *UploadTrResult* sleeps for certain interval, before probing again.

A *Transaction* represents a session in which one or more web pages are accessed by the Delegate one after the other. Accessing each web page is a step in transaction. The transaction may have one or more steps. The details of which web pages to access, along with other parameters are with *TransactionDetails*. The results of the transaction will be stored in *TransactionResult*.
The operation *StartTransaction()* initiates the transaction. Each step is executed and results are collected. If there is failure at any step, rest of the steps are not executed. *GetResult()* returns the result for this transaction.

TransactionDetails identifies various parameters associated with a *Transaction*. The Transaction is identified uniquely with *JobID*. Each transaction has one or more steps, given by *NumberOfSteps*. For each step, there is will be detailed parameters. Such details are completely available with *ListPageDetails*.

Each transaction has an associated result, *TransactionResult*, which is record of the web site's behavior. *JobID* identifies the to which job this result belongs. *NumberOfSteps* gives the number of steps in the transaction. *EEResponse* is the end-to-end response time for the transaction. It is the time elapsed from the beginning of first step to the end of last step, if the transaction is successful. Otherwise it is the time from the beginning of first step to the end of step where failure occurred. *ListPageResults* gives the finer result for each of the step in transaction. The success (or failure) of the transaction is denoted with *TrSuccess*.
SetFailure() sets the result flag as failure along with the reason. *DelegeTrFile()* takes care of deleting the files which are created while executing the test. Given another transaction results for the same job, *GetAvg()* tries to come up with an average of the current result and result from the parameter. In case both of the results represent successful completion of the transaction, *GetAvg()* constructs a

step-by-step result object comparing components of each result. If any of the result indicates failure, *GetAvg()* gives the optimistic result from the two.

Each step of the *Transaction* is a *Page*. This is a finer granularity of *Transaction*. Each *Page* has its details and results, similar to *Transaction*, given by *PageDetails* and *PageResults*.
ConstructReq() creates a HTTP request for the page to be brought. All the HTTP header parameters such as Method, Version, URL, etc. The parameters of request (needed for HTTP POST method) and proxy authorization are included in the header, if applicable. *GetPage* sends the request string received from *ConstructReq()* to the server and receives the reply. *GetHeader()* extracts the response header from received data. It has some useful information of which status of request is most important. It specifies whether request can be fulfilled. If not, it specifies the reason.
Each web page may contain some objects such as images, applets, style sheets, etc. These are specified with tags in HTML. While the content of the web page is being received, it is inspected for the various object of the web page. All such objects are concurrently accessed with *Component*.

PageDetails has various attributes needed for accessing the web page - a step in a multi-step transaction. The attributes are *URL*, the canonical URL of the web page, HTTP Method - *Method* (either of GET or POST). In case of POST or parametrized GET method, the parameters are specified by *PageParam* and the corresponding values with *DataSet*.
SetPageParam sets the *PageParam* and *DataSet* with the values received from parameter.

For each page, various results are noted with *PageResults*. *CompleteSize* is the size of the page, including all the components, in bytes. *Delay* is the response time, in milliseconds, for the page - measured from start of fetching the page to end of brining last component. *PageSuccess* is the flag to indicate the success in accessing the page.

ComponentDetails is similar to *PageDetails*, but with few attributes and operations. *URL* specifies the component URL and *Port*. The HTTP Method is assumed to be GET.

The attributes of *ComponentResults* are akin to that of *PageResults*. *ComponentSize* gives the size of the component in bytes, *Delay* gives the time, in milliseconds, required to download the component and *CmpSuccess* indicates success in fetching the component.

The request to the server for any page or component has a pre-defined format. This format contains various pre-defined fields. *HttpHeadGlobal* has information about all such parameters. *Method* has values GET or POST, *HTTPVersion* can be either of 1.0 or 1.1. Also some other header fields for redirection, proxy authentication are attributed in this class.

Chapter 4

An Example

In earlier chapters we discussed the design and implementation of the complete system. To demonstrate the proper functioning of the system, we conducted few experiments. In this chapter we briefly describe one such experiment and its results.

4.1 Experimental Setup

The RC was set up on *http://www.cse.iitk.ac.in/* i.e. CSE server at IIT Kanpur. The administrator added user *shashi* of the system. The Delegates were installed on one PC in India and three in USA. Thus there were Delegates from two regions (India and North America) participating in the experiment. The links to Internet were T1 and Cable Modem. The PCs had Linux and Windows operating system. In a way, good mix of all parameters was available in the Delegate environment.

We chose to monitor home page of Infosys, *http://www.infosys.com/*, which is one of the major IT companies from India. The new transaction was added with the help of web-based interface shown in Figure 4.1 and 4.2. This a one-step transaction.

4.2 Various Results

For this Job, various results can be generated. Response Time is the total time required to fetch a page. Availability of the service indicates whether the transaction

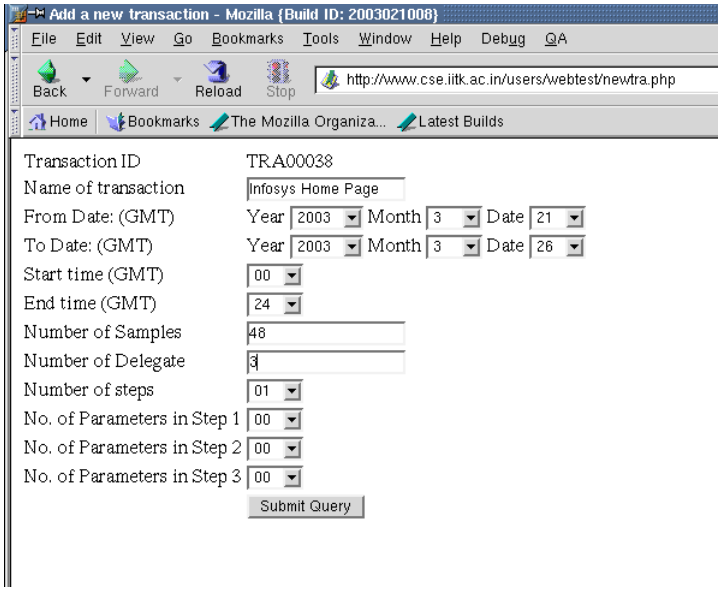


Figure 4.1: Adding a New Job : Scheduling Parameters

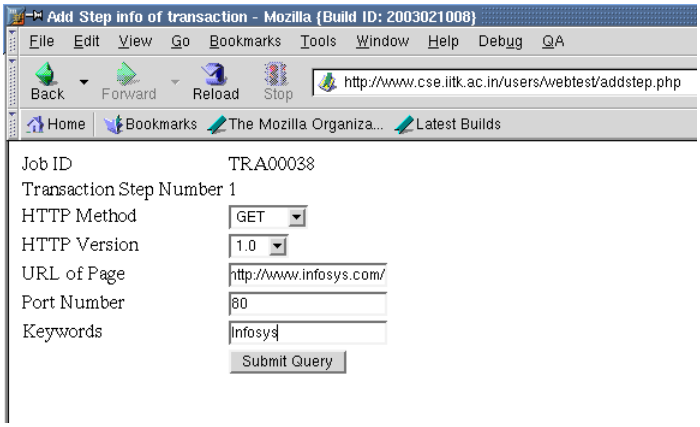


Figure 4.2: Adding a New Job : Access Parameters

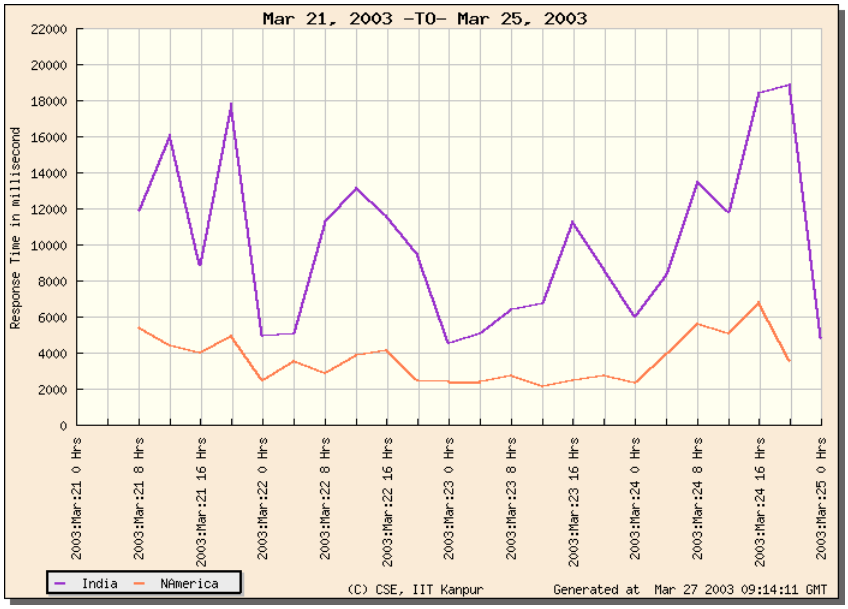


Figure 4.3: Response Time of Home Page in Various Regions

can be executed successfully. Any failure indicates the non-availability of the service. Throughput is bytes received per second for the transaction. All these results can be generated for a day or for given duration. The day or duration, for which results are to be seen, can be controlled with form-based interface. This interface makes it possible to view results for any duration during the testing period. Also, these results are generated by taking into account all the Delegate or Delegates segregated region-wise. *Please note that all the time/date parameters on X-axis are in GMT.*

Figure 4.3 shows the response time of the page in India and North America. The duration of result is 0000 Hrs on 21^{st} March, 2003 to 0000 Hrs on 25^{th} March, 2003. The response time experienced in North America is significantly lesser than in India. But the curves closely follow the same pattern. It indicates, some change parameter from server. The plot for North America shows better response time because of the good connectivity to Internet. Figure 4.4 shows the global average response time of the site. But this graph does not give any clue about region-wise response time, which makes it difficult to figure out which users are facing the problems.

Figure 4.5 shows the availability of the site in different region. Only once the

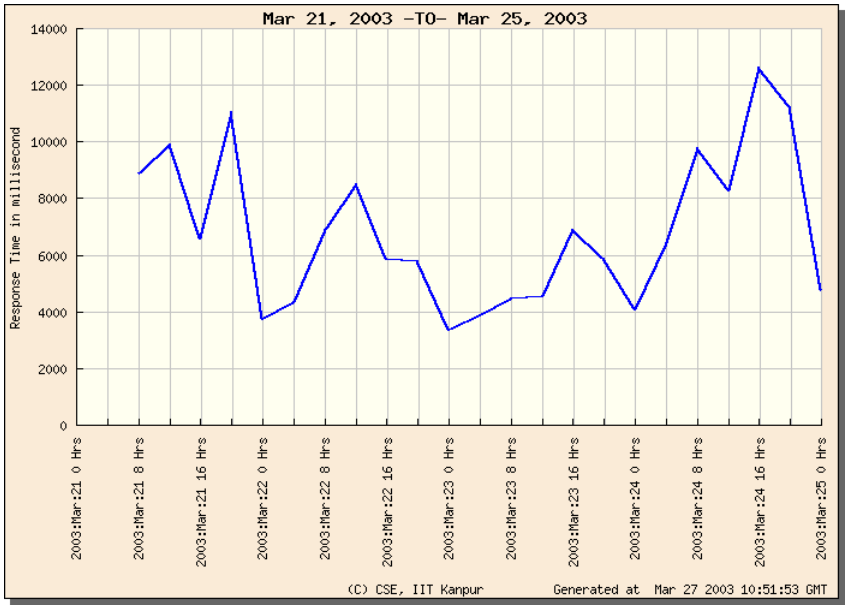


Figure 4.4: Global Response Time of Home Page

availability has fallen to 0.78 for North America. The service is available all the time in India.

Figure 4.6 shows the throughput experienced. Throughput is a derived attribute and it depends on response time.

4.3 Other Experiments

The Delegate is capable of handling multi-step transactions on the web site. To demonstrate this, we give another example. We selected Google News, *http://news.google.com/*, for 2-step transaction. First step accesses *http://news.google.com/*. Second step performs a search on this site, by accessing *http://news.google.com/news* for some string. The HTTP GET method is used for searching.

Figure 4.7 shows the response time of a two-step transaction on Google. The results shown are of the tests conducted by Delegates in India. Figure 4.8 shows the page-wise break of response time of a two-step transaction on Google. This is

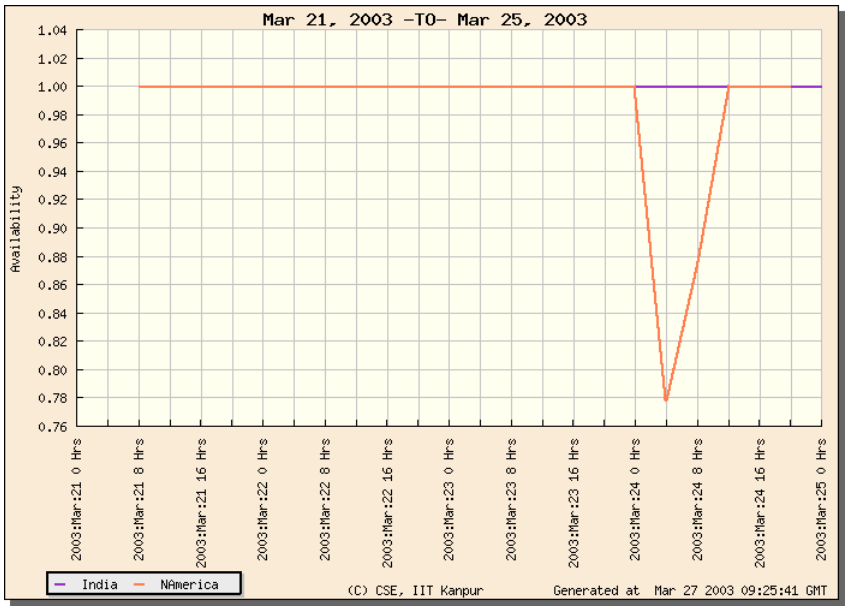


Figure 4.5: Availability of Home Page in Various Regions

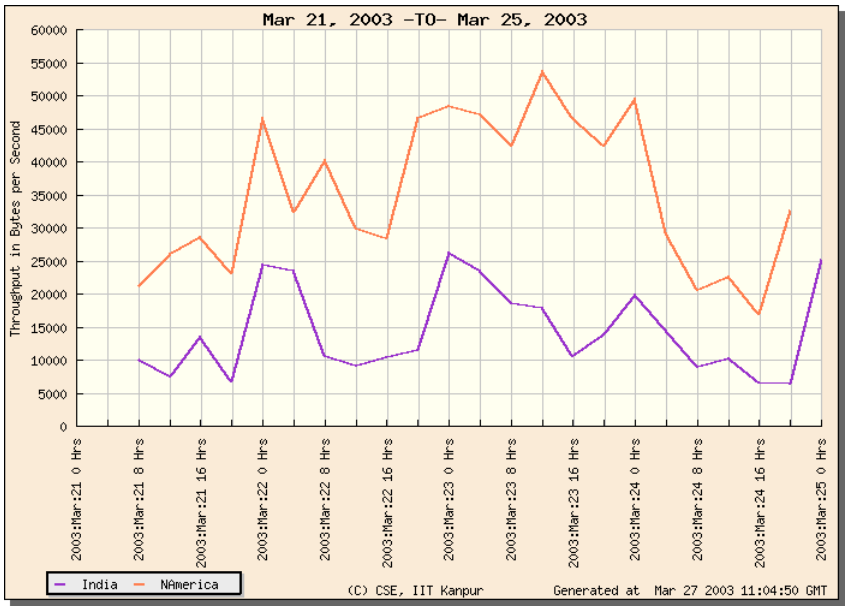


Figure 4.6: Throughput of Home Page in Various Regions

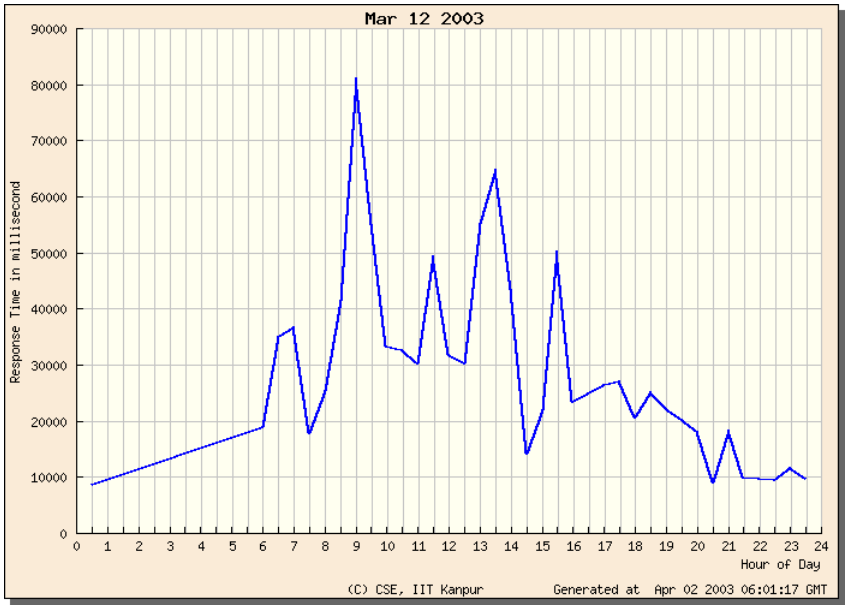


Figure 4.7: Cumulative Response Time of 2-Step Transaction on Google

a detailed result of the graph shown in Figure 4.7. The graph suggests that second step takes lesser time than first one. This is because the number of images on the second page is significantly lesser than first page, reducing the page size. Figure 4.9 shows page-wise throughput of transaction on Google. Almost all the time, the throughput of second step is lesser than first one. The reason for this is, first step is a static page, whereas second step needs some time for database access and generating result dynamically.

4.4 User Feedback

We asked the users to comment on the Delegate. Dwipal wrote, *" The installation was pretty easy for the tar file you sent which didn't require root, and I did not notice any issues with the system performance. This may be because mines is a very fast machine (Xeon 2.2 GHz with 1 GB RAM). But the process seemed to be sleeping most of the time, so it wouldn't have affected a slower system also."*
Please note that we had sent a customized version of the Delegate to him, which did

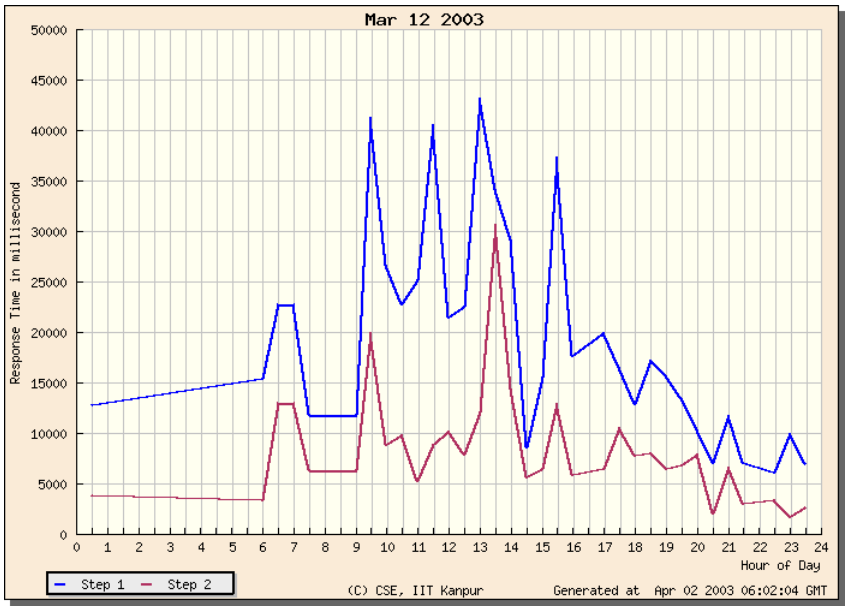


Figure 4.8: Pagewise Response Time of 2-Step Transaction on Google

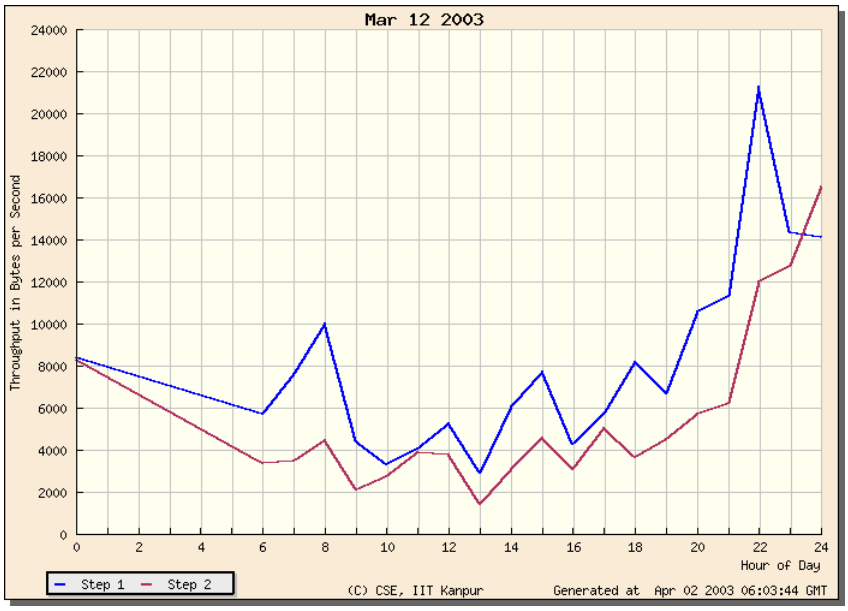


Figure 4.9: Throughput of 2-Step Transaction on Google

not require root permission. Such versions have inherent limitations. For example, user has to manually start and stop the Delegate, which is definitely should be avoided. We suggest that the standard version of the Delegate should be installed to avoid such tricky situations.

Anshul said, *"Installation procedure is simple and has been described in a very good way. I didn't face any problem in installation. It is easy to start and stop the software. There was no significant or observable decrease in speed after I ran the software on my system. I can suggest you one thing on the top of my head. Instead of you asking a person to send log files like "permos.log" or "trans.res", your software should be able to send it automatically in case of error or when asked for. "*
(We used to ask users for the log files to locate some anomalies. But as the software matures, there should not be any need of this.)

Chapter 5

Conclusion and Future Work

In recent years, performance of a web service has become a matter of concern for web site owners. As a web sites are inherently global, the service provider would like to know global performance of the site. The two key parameters of performance are response time and availability of the service. To know performance in different regions, actual tests in those regions have to be conducted. The currently available techniques report unified performance of the site which does not take geographical region into consideration.

We define a new performance criterion for the web site, called *Performance Profile*, which essentially captures the cross-section of users' locations. In this thesis, we outlined a distributed test system for global performance profiling of a web site. The system consists a centralized site, called RC, and agents of the system, called Delegates, running in various regions of the world. The RC is the node which coordinates the complete testing activity. It is responsible for accepting new jobs, assigning them to the Delegates, collecting the results. The RC compiles data from the different Delegates to create the global performance profile of the site being tested. The result graphs can be generated for duration specified with a form-based interface. Delegates are installed on actual end-users' desktop PC located in various regions of the globe. The Delegates work under the control of the central site, perform the testing they are asked to and send the results to the RC. As the tests are conducted in real-world environment, the results given by this tool are a close

representation of actual end-users' experience at the site.

This tool is capable of executing multi-step transactions. This enables testing of multi-step transactions involving user interaction through forms. The new transactions can be added with the help of simple web-based interface at RC. Once the job is submitted, the job is assigned to Delegates and Delegates start executing the tests. As the complete procedure is automatic, it offers greater degree of freedom and accuracy. The complete system is simple-to-use and simple-to-install. Using the RC for adding job and viewing results is also intuitive. A single interface for viewing different results from all the locations is forte of the system.

There is some room for improvement in the Delegate. More features can be added to Delegate to make it more powerful. The Delegate can be extended to work with HTTP Version 1.1. The sites handling sensitive information use Secure HTTP. The Delegate will be able execute transactions involving HTTPS if the required code is plugged. The Delegate can be made to handle cookies. Currently, the Delegate supports only *Add a New Job* feature. This protocol can be extended to *Remove a Job* and *Modify Parameters of a Job*. On the similar lines, *Shutdown* message can be added to indicate shutdown of complete system. Limited content checking can be implemented by checking for keywords on the page.

Currently, the detailed results for the components are not available. The data is available but it needs processing. For each component, the various delays are measured. It would be interesting to study all these parameters to deduce the bottlenecks. A pie-chart can be used to view the errors encountered while accessing the site. The algorithm for selecting a Delegate to assign a job is quite simple. It can be enhanced to offer better results by taking into account more parameters of the Job and Delegate. Similar to region, the type of link to Internet can be take into account to generate another set of results.

Appendix A

Installation of the Complete System

As discussed earlier, the system is made up of two parts - RC and the Delegate. Here we give a detailed instructions about setting up the complete system. Given the source code of RC and Delegate, we discuss at length the procedure to install the system. The steps for installation are

- Set up the RC
- Configure and Distribute the Delegate
- Ask the Delegate software to be installed on PCs.

Now the system is ready to add new users who can run experiments.

A.1 Setting Up RC

RC has been developed using PHP. MySQL has been used a back-end database. The prerequisite for the installing RC are as follows.

- zlib
- libpng
- GD Graphics Library [10]

Above three libraries are required for producing graphical output with PHP.

- MySQL [19] as the database.
- PHP [21] installation with MySQL and GD Library enabled.

A.1.1 Installing RC Source Code

All the PHP file with name starting with *jpgraph* indicate the JpGraph Library for displaying the results. These files should be kept in a directory named **jpgraph**. The source code tree will look like

```
<base_dir>
    |
    |--index.php
    |
    |--login.php
    |
    |--register.php
    |
    |--....
    |
    |--jpgraph
         |
         |--jpgraph_line.php
         |
         |--jpgraph_bar.php
         |
         |--....
         |
```

Typically, one need to get an area on the web server to host the RC. For example, if you have area with directory name "webtest" on http://www.someserver.com/ then, the URL for the <base_dir> becomes http://www.someserver.com/webtest/

A.1.2 Configuring RC

Most of the PHP pages access database. Some code needs to be changed to reflect the specific database parameters. These pages have following two statements.

```
$dbconn = mysql_connect("server_name", "user_name", "password" );
$result = mysql_select_db("database_name", $dbconn) or die("failure");
```

Here *server_ name* is the name of the host on which database server (MySQL) is running. *user_ name* and *password* are the MySQL user name and password respectively. *database_ name* is the database to used for storing all the data. These parameters need to be changed to reflect the local configurations.

Once the database is created and all the PHP source files are modified to reflect the parameters of the database, various tables need to be created. The operations of creating tables and basic set up is done with *tabc.php*. Executing this script will ensure all the basic operations are done and RC is ready for use.

While conducting experiments, we installed RC installed on RedHat Linux with Apache as Web server.

A.2 Configuring Delegate Source

The Delegate has been developed in Java. The Delegate has to be configured for the particular RC. Some changes need to be done in the source code of the Delegate before creating binaries for distribution.

A.2.1 Changes in Source Code

ConfigDelegate.java creates *rcparam.conf* file in which the parameters to access RC are stored. *rcparam.conf* gets created when the Delegate runs for the first time. In this file, various URLs of RC are specified. Delegate uses these URLs to communicate with RC. Following lines need to be changed.

```
out.write("UpResURL = http://www.yourserver.com/path/to/resupload.php ");
```

This line prints the location of interface for uploading the results to RC. This line has to be changed to reflect the correct path. Similar changes have to made for the parameters *AliveURL*, *JobDLURL* and *RegURL*.

A.2.2 Building Distribution

From the given source, the distribution is created as per the explanation given below. The prerequisite for building distribution is Ant[2].

- *Delegate for Linux*

```
a. Get the latest version of PerMoS System from
http://www.cse.iitk.ac.in/users/webtest/

b. Untar the distribution
        $tar zxvf PerMoSDelegate-<version>.tar.gz

c. Change Directory
        $cd PerMoSDelegate

 The "source" directory contains the Java source code for the Delegate,
"com" directory contains library classes and "docs" contains help files.

d. Create the Delegate jar file with
$ant

e. Now create the distribution
$./maketar.sh WebPerMoS-<version>.jar
where WebPerMoS-<version>.jar is the newly created jar file which
is in the "jar" directory.
This will create PerMoS.tar.gz which can now be distributed.
```

- *Delegate for Windows*

a. Get the latest version of PerMoS System from
http://www.cse.iitk.ac.in/users/webtest/

b. Unzip the distribution.

The "source" directory contains Java source code for the Delegate,
"form" contains Java source code for the GUI part of the Delegate,
"com" contains library classes and "docs" contains help files.

c. Create jar file by executing "ant" command in that directory.

d. Copy the above jar file to "Distribution\lib" directory.

e. Modify following parameter in "Distribution\conf\wrapper.conf" directory

wrapper.java.classpath.2=../lib/WebPerMoS-<version>.jar

Change the last component with the proper jar file name.

f. Zip the contents of "Distribution" directory to create PerMoS.zip file
which is ready for distribution.

Appendix B

Installation of Delegate

```
Installation - Linux
====================

Prerequisite
*************

a. RedHat/Mandrake distribution of Linux
b. root permission
c. JVM pre-installed
d. PATH variable set to JAVA installation
        You can check this with ‘which java‘. The output should something
like /usr/path/to/jdk_installation/bin/java

e. chkconfig in the PATH.
        Make sure chkconfig is available by trying ‘which chkconfig‘ as above.
In case of an error, put chkconfig in PATH. Most probably it will be in /sbin.

Steps:
*****
```

a. Get the latest version of PerMoS Delegate from
http://www.cse.iitk.ac.in/users/webtest/

b. Untar the distribution

```
$tar zxvf PerMoS-<version>.tar.gz
```

c. Change Directory

```
$cd PerMoS
```

d. Change User to root

```
$su
```

e. Run installation script

```
#./install.sh
```

f. The software will get installed in /use/local directory. And the daemon will start. See section on "Submitting Details" for further info.

Uninstallation - Linux
==========

a. Change User to root

```
$su
```

b. Run the uninstallation script

```
#./uninstall.sh
```

Installation - Windows
==========

Prerequisite

****

a. Windows 2000 or clones. (We've tested with Windows 2000 Professional)

b. Administrator permission

c. JVM pre-installed.

Steps:

*****

a. Get the latest version of PerMoS Delegate from
http://www.cse.iitk.ac.in/users/webtest/

b. Unzip the file in C:\
It will create a directory named "PerMos" in C:\
(As You have come till here, most probably, you've done above steps.)

After unzip you should see the directory tree as

```
C:\
|
----PerMoS
        |
        |---bin
        |
        |---conf
        |
        |---lib
        |
        |---docs
        |
        |---logs
```

c. Change current working directory on the command prompt.

C:\cd PerMoS\lib

d. Fill the registration details.

C:PerMoS\lib>java -jar RegDetail-<version>.jar

where <version> is the version number of the release.

This will display a form.

Please read section 6 to get more details on it.

e. Change to bin directory with

C:\PerMoS\lib>cd ..\bin

f. Install the delegate with

C:\PerMoS\bin>install

This will install the software on you system, and the program is registered as windows service.

g. Start the service with

C:\PerMoS\bin>net start WebPerMoS

It will display

"The WebPerMoS service is starting....

The WebPerMoS service was started successfully."

h. You are done with installation.

Uninstallation - Windows

==========

a. Change to PerMoS\bin directory and uninstall it.

```
C:\PerMoS\bin>uninstall

You have successfully uninstalled the software.
```

b. Now you can safely delete the folder C:\PerMoS.

Submitting Details

When you install the software, a form will be displayed to you asking various information.
PLEASE NOTE: The personal information given by you will be kept confidential with us.
ALSO NOTE: The form does not do much of error checking. So, PLEASE, make sure that the information entered is correct.

Various fields of the form:

a. Name, Email, Region, City, Internet Link and Operating System self-explanatory. Go through the list of choices and choose the closest match.
b. Privilege: This field will decide the amount of work done by Test Delegate.
"User" indicates least privileges and it will test less number of sites.
"Admin" is a higher privilege and will be doing more work.
"Dedicated" means the work load will be maximum.
If you want to play safe, choose "User".

* HTTP Proxy: If you are accessing Internet through proxy, choose YES. otherwise NO.

* If you have chosen YES above, then provide with details of proxy server such

as URL, Port number, authentication.

If some site has to be accessed without proxy, specify it with No-Proxy. Only one site can be specified in "No Proxy".

Figure B.1 shows the screen-shot of the form displayed to User with indicative values.

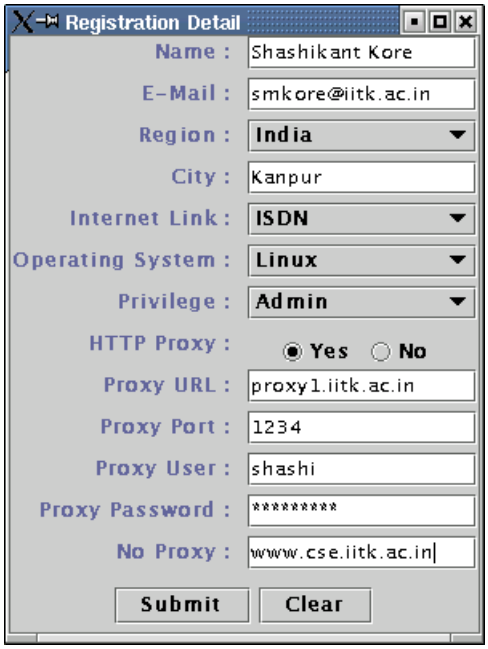


Figure B.1: Registration Form Shown to User when Delegate is Installed

Appendix C

Source Code Organization

This appendix gives all the details of all files used for implementation. This will enable anybody to modify the program.

C.1 RC

- **tabc.php** : Creates all the required tables in database.
- **register.php** : Registers the new Delegate in the System.
- **alive.php** : Accepts Alive messages from the Delegate and Assigns a new job to the Delegate, if any.
- **resupload.php** : Receives the results of transaction from Delegate.
- **getjob.php** : Sends the job profile for the given Job ID.
- **login.php** : Interface for customer login.
- **welcome.php** : Displays the menu for customer, if login is successful.
- **newtra.php, addstep.php** : Adds a new job for monitoring and alongwith the parameters.
- **result.php** : Shows the list of Jobs submitted by the customer.

- **jobres.php** : Shows various options for choosing a particular result.
- **getres.php** : For given parameters, it generates the result graphs.
- **restimeday.php, restimedur.php** : Generate the *Response Time* graphs for a day and for given duration, respectively.
- **avday.php, avdur.php** : Generate the *Availability* graphs for a day and for given duration, respectively.
- **thr.php** : Generate the *Throughput* graphs for a job.

C.2 Delegate

- **Global.java** : Takes care of globally used parameters and their values.
- **HttpHeadGlobal.java** : Has all the parameters required to send HTTP request.
- **Profile.java** : Manages the profiles of various jobs assigned to Delegate.
- **Scheduler.java** : Schedules and executes the transactions for various jobs.
- **Result.java** : Manages the transaction results for the tests conducted.
- **UploadResult.java** : Uploads the transaction results to RC.
- **Transaction.java** : Given a set of parameters, executes the transaction and collects the results.
- **JobInfo.java** : Lists only the scheduling parameters for a transaction.
- **JobDetails.java** : Lists all but scheduling parameters of the transaction.
- **TransactionResult.java** : Represents the result for complete transaction including all the steps.
- **Page.java** : Fetches a web page(step in a transaction) for given URL and other parameter.

- **PageDetail.java** : Represents the various parameters for a step in transaction.
- **PageResult.java** : Result of each page including all the components.
- **Component.java** : Fetches a component of web page (image, flash object, style sheets, etc.).
- **ComponentDetails.java** : Has the URL of the component to be brought.
- **ComponentResult.java** : Stores all the response time parameters for a component.
- **Base64.java** : Given a string, generates its Base-64 encoding.
- **ConfigDelegate.java** : Creates configuration files and register Delegate with RC.
- **ConfigProxy.java** : Creates proxy configuration file for the Delegate.
- **FormatUploadRes.java** : Results for a transaction are stored in a specific format before uploading.
- **FetchComponent.java** : Converts all the relative URLs of components to absolute ones and fetches unique components of the web page concurrently.
- **Form.java** : Displays a form to user when Delegate runs for the first time and reads the information for the user.
- **GetImgTag.java** : Writes contents of web page to a file and submits for analyzing to retrieve component URLs.
- **GetImgName.java** : Searches a web page and lists unique embedded objects in it. Components are identified with the help of JavaCC HTML Parser [15].
- **OwnSocket.java** : Wrapper around Java's Socket class for some extra functionality.

- **PendingJob.java** : List the Job IDs, profiles of which needs to be fetched from RC.
- **SendAlive.java** : Sends Alive message to RC at regular interval and interprets the reply from RC.
- **NumThread.java** : Synchronization class to ensure that number of concurrent threads do not cross a certain maximum limit.
- **MyMain.java** : Main program for the Delegate which initiates all the threads.

Appendix D

Format of a Job Profile

The profiles of the jobs assigned to Delegate are stored in a file **monitor.pro**. The information about each job is stored in a predefined format. When the duration of the job expires, the profile of the job is removed from this file. We illustrate the format with examples.

D.1 One-step Transaction without parameters

This the simplest job assigned to a Delegate. It is similar to visiting a page with specified URL. The format is given below.

```
#JPROFILE
JOBID TRA00099
TYPE TRA
FROM 2003-02-04
TO 2003-02-11
START 0
END 24
NUMSAMP 50
NUMSTEP 1
#JSTOP
#PAGE
```

```
STEP 1
HTTPM GET
HTTPV 1.0
URL http://www.iitk.ac.in/index.html
PORT 80
NUMSET 1
PARAM { NONE }
DATASET 1 { NONE }
KEYWORD 1 { NONE }
#ENDPAGE
#JEND
```

Now, we describe each attribute of this profile.
#JPROFILE is the start marker. It indicates start of a job profile. Similarly, **#JEND** is the end marker signaling end of the profile. Thus, each job profile is enclosed in a pair of **#JSTART** and **#JSTART**.
Each job includes scheduling and access parameters. Scheduling parameters specify the duration in which the test should be conducted along with its frequency. The access parameters specify the exact work to be done by the Delegate.
Each job in the system is identified uniquely with its ID. **JOBID TRA00099** gives the JobID for the job. In this case, the JobID is **TRA00099**. **TYPE TRA** says that the Job type is transaction. This parameters is not of much use in current context. It is included for future enhancements when there might be some different kind of jobs apart from transaction. The job has a stipulated time duration in which the tests should be carried out. **FROM 2003-02-04** and **TO 2003-02-11** specify the dates during which the tests should be conducted. The dates are in *YYYY-MM-DD* format. In this example, the tests should be conducted between from February 2nd to 11th (both inclusive), 2003. The next lines **START 0** and **END 24** specify the Hour of Day during which tests should be conducted. That means, on each day, the test should be conducted between 0000 Hrs and 2400 Hrs. The granularity provided is of hours. It is not possible to start the test at, say 0715 Hrs. *Also note that all the times mentioned are according to GMT.*

The next line **NUMSAMP 50** specifies the number of tests to be conducted on each day in the stipulated duration. In this case, 50 samples (or tests) are taken during 0000 Hrs to 2400 Hrs. This translates to one test per 28 minutes. The line **NUMSTEP 1** specifies the number of steps to be 1.
The marker #**JSTOP** indicates end of scheduling parameters of the Job.

The access parameters for each step are enclosed in the beginning marker, #**PAGE** and #**ENDPAGE**. **STEP 1** indicates that the the access information given is of Step 1 of transaction. **HTTPM GET** specifies the HTTP Method to be used for accessing the URL. Currently, the Delegate supports only GET and POST methods. Hence value of this parameter will be either of two. **HTTPV 1.0** gives the HTTP Version to be used. Currently, the Delegate supports only Version 1.0. **URL http://www.iitk.ac.in/index.html** specifies the URL of the page to be accessed in current step. URL is in canonical format. The HTTP Server normally runs on port 80. The Delegate contacts the server on that port. But in certain cases, it might be desirable to run the server on different port. Considering this situation in mind, the **PORT** parameter is used to specify the port to be used to connect to the server.

Some pages need to be accessed using parameters. For example, a search engine presents a text box for input. User enters the search string which goes in the URL. It is possible to access such pages by the Delegate. **PARAM** specifies the parameters string for the page. In there are no parameters to be sent, it is specified with **PARAM { NONE }**. The pages which are accessed with parameters, need the parameters as well as values of the parameters. Multiple sets of values can be specified in the profile. **NUMSET 1** gives the number of such sets available in profile. The minimum value of the parameter is 1. **DATASET 1 { NONE }** gives the values to be used for parameters specified **PARAM**. The value *NONE* indicates no parameters are used to access the page.

Limited content checking can be done by looking for some important strings on the web page. We call Keyword to such strings. These are specified using **KEYWORD**

parameters in the profile.

D.2 Multi-Step Transaction with Parameters

The example given in Section D.1 was an easier one. Here we give a more detailed example illustrating the capabilities of the Delegate.

```
#JPROFILE
JOBID TRA00097
TYPE TRA
FROM 2003-03-02
TO 2003-03-14
START 0
END 24
NUMSAMP 48
NUMSTEP 2
#JSTOP
#PAGE
STEP 1
HTTPM GET
HTTPV 1.0
URL http://www.google.com/
PORT 80
NUMSET 1
PARAM { NONE }
DATASET 1 { NONE }
KEYWORD 1 { NONE }
#ENDPAGE
#PAGE
STEP 2
HTTPM GET
HTTPV 1.0
```

```
URL http://www.google.com/search
PORT 80
NUMSET 4
PARAM { hl|ie|q| }
DATASET 1 { en|ISO-8859-1|aishwarya rai|}
KEYWORD 1 { Aishwarya|}
DATASET 2 { en|ISO-8859-1|war on iraq| }
KEYWORD 2 { iraq|}
DATASET 3 { en|ISO-8859-1|matrix movie|}
KEYWORD 3 { matrix|}
DATASET 4 { en|ISO-8859-1|Michelangelo| }
KEYWORD 4 { Art|}
#ENDPAGE
#JEND
```

This is a two step transaction as mentioned in the line **NUMSTEP 2**. The first step is similar to the example given Section D.1. The second step needs some explanation. The list of parameters is given in **PARAM { hl|ie|q| }**. This means the page has to be accessed with three parameters namely *hl*, *ie* and *q*. The symbol | (pipe) is used as delimiter for separating parameters. The values for these parameters are specified in subsequent lines. **DATASET 1 { en|ISO-8859-1|aishwarya rai|}** specifies the values to be used for the parameters specified with **PARAM**. Thus the actual URL used to access the page becomes *http://www.google.com/search?hl=en&ie=ISO-8859-1&q=aishwarya+rai*. If there are special characters in the parameters, they are encoded as specified in RFC 1738[4]. For this **DATASET**, the respective keywords are used for content checking i.e. **KEYWORD 1 { Aishwarya|}** specifies the keywords to be used.
Each test may use one of the 4 sets mentioned. The set to be used is chosen randomly.

Appendix E

Format of a Transaction Result

The results of various jobs are stored locally in a file **trans.res**. The results are stored in a particular format. Each results is in the file until it is uploaded. When a result is uploaded, it is marked as uploaded. At regular interval, all uploaded results are deleted from this file. Thus, this file has all the results to be uploaded on RC or uploaded recently on RC.

```
###JSTART
JID TRA00016
STM 2003-02-16 22:55:26
NUM 1
ETE 15788
ERR 0
FSN 0
###JSTOP
##PSTART
URL http://www.rediff.com/index.html
SIZ 97283
DEL 15783
CMP 10
ERR 0
FRS
```

```
##PEND
##CSTART
URL http://www.rediff.com/uim/common/trans.gif
CNT IMG
DEL 335 0 340 675
SIZ 43
HDS 375
#
URL http://www.rediff.com/uim/common/in_rediff_logo.gif
CNT IMG
DEL 340 0 350 690
SIZ 1234
HDS 378
#
URL http://www.rediff.com/uim/whatsthis-us.gif
CNT IMG
DEL 365 0 365 736
SIZ 1500
HDS 378
#
URL http://www.rediff.com/movies/2003/feb/15ano.jpg
CNT IMG
DEL 311 0 350 661
SIZ 2036
HDS 379
#
URL http://www.rediff.com/hindi/2003/feb/14head2.gif
CNT IMG
DEL 360 0 310 671
SIZ 297
HDS 377
```

```
#
URL http://www.rediff.com/hindi/2003/feb/13head2.gif
CNT IMG
DEL 405 0 290 696
SIZ 365
HDS 377
#
URL http://www.rediff.com/gujarati/2003/feb/15main.gif
CNT IMG
DEL 340 5 320 666
SIZ 347
HDS 378
#
URL http://www.rediff.com/uim/blue.gif
CNT IMG
DEL 291 0 270 561
SIZ 255
HDS 376
#
URL http://www.rediff.com/uim/mood-banner.gif
CNT IMG
DEL 301 0 340 641
SIZ 2768
HDS 378
#
URL http://www.rediff.com/index.html
CNT TXT
DEL 356 0 641 5168
SIZ 54498
HDS 315
#
```

```
##CEND
###JEND
```

The result of each transaction in enclosed by the beginning marker, ###**JSTART** and the end marker ###**JEND**. The beginning marker ###**JSTART** indicates the result is yet to be uploaded to RC. In case, the result is uploaded, the beginning marker is ###**JUPLD**#. The next line **JID TRA0018** gives the JobID. The start time of each transaction is noted down with **STM 2003-02-16 22:55:26**. The timestamp is in *YYYY-MM-DD HH:MM:SS* format. The given line specifies the test was conducted on *16th February, 2003* at *22 Hrs, 25 Minutes and 26 Seconds GMT*. **NUM 1** is the number of steps. Each transaction has an associated End-to-End response time (from start of the first step to the end of last step). It is given by **ETE 15788**. The value given is the number of milliseconds. The transaction may encounter an error during execution. **ERR 0** specifies that there was no error. In case of an error, the value is set to 1. If failure occurs, the step number of failed page is given by **FSN** parameter. The value 0 indicates no failure.

The high-level result of each step is contained in the marker ##**PSTART** and ##**PEND**. The line **URL http://www.rediff.com/index.html** specifies the URL for the step. **SIZ 97283** gives the size of the complete web page, along with all the components, in bytes. The complete download time or response time is given by **DEL 15783**, the value being in milliseconds. Each page has embedded components in it. **CMP 26** gives the number of components in the page. Similar to the transaction result **ERR 0** is the flag to indicate any error condition in the page. In case, **ERR** flag is set to 1, **FRS** gives the failure reason. The reason might a be a string such as TIMEOUT or HTTP Response Code such as 4XX or 5XX.

The detailed component results for each step, follow the page result. The component results are enclosed in ##**CSTART** and ##**CEND**.
URL http://www.rediff.com/uim/common/trans.gif gives the absolute URL of the component. **CNT IMG** gives the content-type of the component. It can be

TXT (text), IMG(image), AUD(audio), VID(video), etc. **DEL 335 0 340 675** specifies various delays in milliseconds. The delays specified are *Connect Delay, Send Request Delay, First Byte Latency and Complete Download time* in that order. **SIZ 43** is the size of the component in bytes. **HDS 375** is the HTTP header size (in bytes) for this component.

Note that there is a component with URL **http://www.rediff.com/index.html**, which same as that of the page. The HTML text of the page is also considered to be the component of the page. Hence there is an entry with same URL.

Only in case of an error in fetching the component, the **ERR** and **FRS** fields are included in the component result.

Appendix F

Format of Configuration Files

The Delegate uses various configuration files at boot-up. It needs to know about the RC server, the user information and proxy information, if any. Here we describe all the configuration files used by the Delegate.

F.1 rcparam.conf - RC Parameters

Delegate communicates with RC for various reasons. The various URLs of RC need to be known to Delegate. This information along with some other parameters is available in *rcparam.conf*. The format of the file is given below.

```
ResFile = trans.res
ProfileFile = monitor.pro
DelID =  f35b2159be
UpResURL = http://www.cse.iitk.ac.in/users/webtest/resupload.php
UpResPort = 80
UpResMethod = POST
UpResVers = 1.0
AliveURL = http://www.cse.iitk.ac.in/users/webtest/alive.php
AlivePort = 80
AliveMethod = GET
AliveVers = 1.0
```

```
JobDLURL = http://www.cse.iitk.ac.in/users/webtest/getjob.php
JobDLPort = 80
JobDLMethod = GET
JobDLVers = 1.0
RegURL = http://www.cse.iitk.ac.in/users/webtest/register.php
RegPort = 80
RegMethod = POST
RegVers = 1.0
```

The results of the transaction are stored in a text file locally, before uploading to RC. The name of the file is given by **ResFile = trans.res**. Similarly, **ProfileFile = monitor.pro** specifies the filename to store the profiles of the jobs. Each Delegate is identified uniquely by an ID. This is given by **DelID = f35b2159be**. The ID string is the time stamp in hex when the Delegate runs for the first time.

RC is accessed by Delegate for four purposes - Register Delegate, Send Alive message, Download job profile and Upload result. The parameters specified for each of the functionalities are URL, Server port, HTTP Method and HTTP Version to be used. For example,
UpResURL = http://www.cse.iitk.ac.in/users/webtest/resupload.php is the URL for uploading results. **UpResPort = 80** gives the server port, **UpResMethod = POST** is the HTTP Method and **UpResVers = 1.0** specifies the HTTP Version to be used.

Similarly, rest of the parameters are interpreted. The lines starting with an # are considered as comments.

F.2 userinfo.conf - Delegate User Info

While the Delegate is registered with RC, certain information needs to be submitted. *userinfo.conf* stores all this information. This information is used only while registration.

```
Registered = YES
Name = Shashikant
Email = smkore@iitk.ac.in
Region = India
City = Kanpur
Link = Cable
OS = Linux
Priv = Ddc
```

Registered = YES is the flag to indicate status of registration process. *YES* indicates the registration has been done. Otherwise it will be *NO*. The fields **Name = Shashikant**, **City = Kanpur**, **Email = smkore@iitk.ac.in** are self-explanatory. **Region = India** is the region in which Delegate is hosted. Currently, we have defined the following regions- *Africa, Asia, Asia Pacific, Europe, India, Latin America, Middle East, North America.* The granularity is varying - India and Asia are together. In case, the city can fit into multiple regions, it is advised to choose the one with finer granularity. For example, *Kanpur* comes in *India*, which in turn comes in *Asia*. In this case, the region is chosen to be *India*.

The connectivity to Internet is specified with **Link = Cable**. Currently, the type of link can be one of the following - *Cable Modem, Dial Up 33kbps, Dial Up 56kbps, DSL, ISDN, T1, T2 and Wireless.*

The concept of Delegate rights is explained in Section 2.3.3. The privilege of the Delegate is given in the line **Priv = Ddc** - *Ddc* indicated Dedicated, *Usr* indicates User and *Adm* indicates Administrator privilege. **OS = Linux** indicates the operating system used by the Delegate.

F.3 proxy.conf - Proxy configuration

The Delegate can operate behind HTTP proxy. All the proxy related information is in *proxy.conf*. The content of the file looks like this.

```
UseProxy =  YES
ProxyURL =  myproxy.iitk.ac.in
ProxyPort =  7890
ProxyUser =  sachin
ProxyPass =  tendulkar
NoProxy =  www.cse.iitk.ac.in
```

UseProxy = YES is the flag to indicate use of proxy. If it is *NO*, the remaining parameters are unrelated. **ProxyURL = myproxy.iitk.ac.in** gives the address of the proxy and **ProxyPort = 7890** gives the port number of proxy server. ProxyUser = sachin and ProxyPass = tendulkar indicate proxy user name and password respectively, giving the proxy authorization. In case, some site has to be accessed without proxy it is specified with **NoProxy = www.cse.iitk.ac.in**. Why will the Delegate need it? Consider a Delegate running in one of the PCs of CSE network which come under the domain of *cse.iitk.ac.in* and it uses proxy for accessing Internet. Now, if the RC is located in the same domain i.e. *cse.iitk.ac.in*, then the RC must be accessed without proxy. Hence, the arrangement.

Bibliography

[1] ActiveWatch from Mercury Interactive *http://www-svca.mercuryinteractive.com/products/activewatch/*

[2] Apache Ant : A Java-based build tool *http://ant.apache.org/*

[3] Berners-Lee, T., Fielding, R. and H. Frystyk *RFC 1945: Hypertext Transfer Protocol – HTTP/1.0* MIT/LCS, UC Irvine, MIT/LCS, May 1996

[4] Berners-Lee, T., Masinter, L. and McCahill, M. *RFC 1738: Uniform Resource Locators (URL)* CERN, Xerox PARC, University of Minnesota, December 1994

[5] Compact Index of HTML Tags *http://www.willcam.com/cmat/html/crossref.html*

[6] eValid Monitoring Services *http://www.soft.com/eValid/Services/Monitoring/*

[7] FarSight from Empirix *http://www.empirix.com/*

[8] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and Stewart, L. *RFC 2617: HTTP Authentication: Basic and Digest Access Authentication* Northwestern University, Verisign, Inc., AbiSource, Inc., Agranat Systems, Inc., Microsoft Corporation, Netscape Communications Corporation, Open Market, Inc., June 1999

[9] Fu, Y., Cherkasova, L., Tang, W. and Vahdat A. *EtE: Passive End-to-End Internet Service Performance Monitoring* General Track: 2002 USENIX Annual Technical Conference, Monterey.

[10] GD Graphics Library *http://www.boutell.com/gd/*

[11] Gerrard, P. *Risk-Based E-Business Testing - Part 2 : Test Techniques and Tools* *http://www.evolutif.co.uk/articles/EBTestingPart2.pdf*

[12] Gomez *http://www.gomez.com/*

[13] Hower, R. *Web Site Test Tools and Site Management Tools* *http://www.softwareqatest.com/qatweb1.html*

[14] Java Service Wrapper *http://wrapper.sourceforge.net/*

[15] JavaCC HTML Parser *http://www.quiotix.com/downloads/html-parser/*

[16] Jalote, P. *An Integrated Approach to Software Engineering.* Narosa Publishing House, 1991.

[17] Jalote, P. *Fault Tolerance in Distributed Systems.* PTR Prentice Hall, 1994.

[18] JpGraph - OO Graph Library for PHP *http://www.aditus.nu/jpgraph/index.php*

[19] MySQL : Open Source Database *http://www.mysql.com/*

[20] peerReview from Porivo *http://www.porivo.com/*

[21] PHP : General purpose scripting language *http://www.php.net/*

[22] Rajamony, R. and Elnozahy, M. *Measuring Client-Perceived Response Times on the WWW* 3rd USENIX Symposium on Internet Technologies and Systems (USITS), March 2001, San Fracisco.

[23] Web Site Perspective from Keynote Systems *http://www.keynote.com/*