PROFILING AND TESTING PROCEDURES FOR A NET-CENTRIC DATA PROVIDER

Derik Pack

SPAWAR System Center Charleston, North Charleston, SC
derik.pack@navy.mil

ABSTRACT

A key focus for the success of Net-Centric operations is the testing procedures for web services and the environments where those web services exist. Quite often the ability of a given service to reach a specific performance goal is dependent on many factors found in the operating system itself, the language used to implement the service, the service's code quality, and related applications servers and services. A failing in the design of many test procedures is to capture one particular measure of performance while failing to quantify the many variables that affect that measure of performance. This often leads to lost development cycles trying to achieve a small performance increase in one part of the system while overlooking several other easily modifiable system components that could increase performance far more significantly. This paper presents testing procedures and examples from the development of the Net-Centric Diplomacy (NCD) initiative of Horizontal Fusion. The examples will primarily focus on the web services created by the initiative and the backend environment interactions that take place. Through this description, the reader will realize the interrelated nature of many different types of testing procedures and the necessity of good test design in order to find the most efficient means to address a given goal.

1. INTRODUCTION

With the advent of web services, the paradigm on the web is shifting from a server-to-client model to a model where web based components are combined to build distributed applications. For the purpose of this work, a web service is defined as any service that is accessible through the use of standard web protocols like Extensible Markup Language (XML) and Simple Object Access Protocol (SOAP0. This also implies the use of facilitating specifications like Web Services Descriptive Language (WSDL) and Universal Description, Discovery, and Integration (UDDI) in order to specify the interface to the service [1, 3]. The maturation of these standards will allow businesses and governments to design applications that achieve far more than a platform independent interface to a given data set. These web services will be able to register with, naturally discover, and use other web services that can deliver information or a function that would benefit the originating organization of the service. The resulting composable applications would allow for a true service-oriented architecture (SOA) where defined business processes and policies could be executed by a set of loosely coupled services built on top of available software infrastructure [2]. Such a paradigm shift in web design would have vast implications. Effective use of services could result in a lower cost of development, higher component reuse, process streamlining, and smooth integration paths [4].

In order for web services to reach this point, several impediments need to be overcome. Collectively, these issues can be thought of as areas of future work for a distributed component based application. The issues are broken into two groups: standards barriers and technical

Approved for public release 30 September 05

barriers. The standards barriers include non-maturity of standards and semantic issues. This area covers the misunderstanding of standards, and policy and interoperability issues that are taking place in the adoption of web services. The technical barriers to adoption of web services include security, performance, quality of service and reliability, and transaction support [4, 5]. Proposing a solution to all these barriers to adoption is well outside the scope of this paper. The purpose of this work is limited to the discussion of performance and in some instances quality of service and reliability of web services. The scope is limited to these areas because they are heavily affected when trying to surmount other barriers to adoption. They should, in many cases, be considered the most important design goals for a usable net-centric system. Unfortunately, few realize the complexity that must be taken into account when attempting to quantitatively measure the performance and reliability when dealing with web services. The basic performance measures and procedures need to be studied and defined for a basic system in order to facilitate a more complex distributed environment.

The rest of the paper will highlight the NCD initiative as an example of a net-centric data provider based upon web services. The choice of performance measures and procedures that were used to test this initiative will be explained. Section 2 will give a short example of the NCD web services and backend. Section 3 will define the testing measures and procedures used. Section 4 will give some example results from NCD and Section 5 will give conclusions and future areas of work.

2. NET-CENTRIC DIPLOMACY

Net-Centric Diplomacy (NCD) is the Department of State initiative in the Horizontal Fusion Portfolio. NCD provides Department of State cable and biographic reports to Horizontal Fusion's Federated Search. NCD implemented the Intelligent Federated Index Search (IFIS) WSDL and other Horizontal Fusion specifications to create a search web service that can be accessed by the Federated Search client. The specifications detail security, dynamic discovery, messaging, and authentication of services within the Horizontal Fusion Collateral Space. A full list of these specifications can be found in the Horizontal Fusion Developer Reference and Guidance [6, 7, 8]. A full description of the entire NCD implementation is beyond the scope of this paper, but a summary is provided (Figure 1). Figure 1 shows requests coming to NCD from Federated Search. These requests are received by the Net-Centric Diplomacy Search Web Service (NCDSWS). NCDSWS is the piece of the architecture that implements the Horizontal Fusion specifications. It validates the digital signing of SOAP messages it receives, checks the security information, and determines if the query is valid. If the request passes all these tests, it is passed to the Post Data Retrieval Web Service (PDRWS) which translates the requests to SQL and accesses the database to retrieve the information. The database returns the results to PDRWS which sends them back to NCDSWS to return to Federated Search.

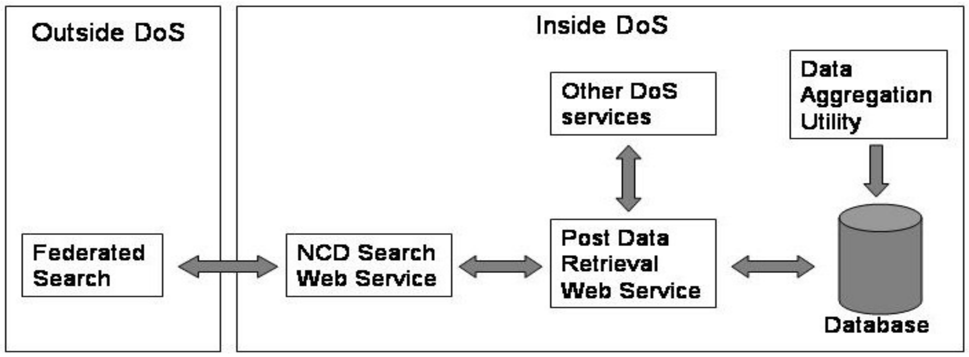

Figure 1. NCD Architecture

The prime advantage of this layered architecture is the benefit to Department of State's other web services. Since they are on the same trusted network as the PDRWS, they can directly access it without going through the security checking that is mandated by the Horizontal Fusion specification.

The architecture is implemented using Apache Axis' SOAP engine, JAVA 1.4.2 SDK, Apache Tomcat, and MS SQL Database 2000.

3. PERFORMANCE MEASURES AND PROCEDURES

One of the many goals in testing NCD is to quantify the boundaries of performance for the services that exist. In reviewing standard testing procedures for web applications, performance testing often focused on stressing the user interface. When dealing with web services, this standard for performance testing will no longer hold. Although performance tools exist that directly stress web services, two secondary considerations exist that must be considered. The first of these considerations is all the other services and application servers that a service calls in order to fulfill its function. These services and application servers affect the overall performance of any web service that calls them. In many instances the organization creating a service will not have direct control of its dependencies. Downtime on the part of a service's dependency will also cause downtime in that service. The second consideration is external specifications for a service. Essentially, the business processes that define the use of a service as an application reside outside the service. A WSDL defines the interface to a service, but the valid use of an implementation of that interface is not specified. These external specifications can have an effect on the performance of a service that cannot easily be seen using non-customizable testing tools. A prime example of external specifications is a web service that implements a query syntax. The query syntax may allow for highly recursive but semantically meaningless queries that would decrement the performance of the service if multiple client applications sent them. This issue is as much an initial design issue as a testing issue. With the composable nature of services, one must be wary of making one's service dependent on other services that may have such problems. In order to overcome these problems, NCD's testing procedures are based upon understanding and maximizing the performance through the use of characterization testing and profiling of a service's many dependencies along with testing the web service directly.

The procedure for testing performance during development is two-stage. The first stage is to define metrics that directly measure some element of a web service's performance. The second-stage is to create tests that measure individual system components to determine the best methods to increase overall system performance through the defined metrics. The following sections will be constrained to the metrics and tests used in the development stages of the project. As the project has progressed into an operational phase, different metrics and tests must be used in order to maintain the highest uptime available. This led to a set of diagnostic tools for the operational environment. These tools serve as a dashboard to monitor the internal and external services and servers that NCDSWS relies upon that are maintained by other initiatives or organizations. The results from these tools are used to replicate problems that occur in the operations environment in the development environment where the following tests and procedures are used.

3.1 First Stage Testing

Design of the first stage tests started with researching the differences between the error states of many web applications and web services. Web servers tend to reach their break point when so oversaturated with requests that they can no longer service them. This can cause the server itself to go down or simply report the unavailability for a large majority of its requests. The deserialization of SOAP requests is far more processor intensive; and as a result, the number of requests that will cause a web service to fail is far lower than for a web server. To compound the problem, web service errors do not always map to Hypertext Transfer Protocol (HTTP) service codes, and the application environment and programming language can cause unforeseen behavior depending upon their configuration. After considering these factors, the following metrics were defined for the first stage testing:

- *Round Trip Time (RTT)*: The time required for a request to be sent from a client, processed by the server and returned
- *Error*: Incorrect results or error messages received from the web service
- *Connections per Second (CPS)*: The number of connections that are being sent to the web application each second

RTT was used as a metric because it gave the most accurate simulation of the time the client would spend waiting for results. Error can be attributed to many different sources including incorrect functionality of the web service or web server and database failure. For the purpose of our testing, error was specified as anything that was not a correctly returned result. Measuring error consisted of logging to determine the most likely cause of error and capturing the percentage of errors for a set number of connection and query attempts. CPS is used because it gives a quantitative measure of a given amount of load. It was also believed that this metric could be used to find the optimal operational conditions for the server.

After finding these metrics, a survey was conducted among several different stress testing utilities to determine which ones had the best abilities to capture all this information. In the end, NCD opted to develop its own test harness (NCD LoadTest Utility) in order to better catch and analyze incorrect results and to initiate self-developed test cases where CPS could be explicitly set and controlled. Effective testing using the test harness requires a server or servers hosting the web services and a separate equivalent server running the test harness which collects data from queries it sends. During testing, processor use due to other applications is limited on the testing server to ensure results remain objective.

The test harness provided the following types of tests: continuous tests, ramped tests, burst tests, and adaptive tests. Continuous tests allowed the user to set the CPS and the time of the test. The test would then run at the defined connection rate until finished. Ramped tests allow the user to set a start and end CPS and a number of steps to take between the start and endpoint along with a time to stay at each step. The test increments its rate as it progresses until it reaches the maximum rate. Burst tests are a one time burst of a set number of connections a second. These tests are used to find average RTT for burst traffic the server could theoretically receive. Adaptive tests allow the user to specify a start point and search for the steady state CPS that the server can maintain. All these tests report back the RTT, CPS, and error.

3.2 Second-Stage Testing

Second stage testing consists of testing the code, application servers, runtime environment, and operating system to determine what modifications to these components can increase overall system performance that is measured in the first-stage tests. Several examples of testing in each of these areas will be provided.

Code testing is the most obvious method of improving performance. This is most often done with unit (regression) testing and profiling. Profiling will be focused on here because of its usefulness in conjunction with some of the first stage tests. Profiling tools give a developer insight into the amount of time spent in each method during code execution, Central Processing Unit (CPU) usage, number of objects created, and memory allocation. Profiling is especially useful for finding unused sections of code and discovering memory leaks. On occasion it may be necessary to start a web service inside a profiler while applying a load in order to identify a very slow memory leak.

When dealing with an application server, testing is not really required as long as the limitations and best settings of the application server are known. An example would be an Apache Tomcat server that provides the web container for the web services. In order to provide faster servicing of requests, Domain Name System (DNS) lookup was disabled in the server's configuration file.

Depending on the programming language, testing the runtime environment will not be necessary. For the case of NCD, it was important to examine the runtime environment because of the use of Java. The performance of the Java Runtime Environment (JRE) was affected not only by its configuration settings, but also the hardware the Java Virtual Machine (JVM) was running on. After configuring the runtime environment to use the server JVM, garbage collection monitoring was employed. This test allowed the developer to determine the throughput drop due to garbage collection and helped to select the best garbage collection algorithm to use for the given system hardware.

Tests taking place in the operating system are typically used to monitor memory and CPU usage. These tests are especially useful when using the test harness to test for several days continuously. They can correlate any unusual results that take place while sending results.

3.3 Procedures

Testing procedures for NCD were initiated with first stage testing. Cycles of burst, continuous, and ramped testing were conducted until failure levels were reached. These levels were based on whether RTT and error exceeded certain thresholds. The initial thresholds for error were either complete unresponsiveness of the server or a percent error greater than 15%. The initial failure threshold for RTT was an average RTT for a test greater than 90 seconds. Each cycle of testing would be repeated on the same server instance. After the repeat of a test, if the results from the later test were worse than the initial test, second stage testing would be used to determine if there was a memory leak, application error, or configuration problem.

After finding and correcting a problem, a few cycles of first stage testing would be repeated. If the results remained consistent for these cycles, then testing was limited to continuous testing with increasing time limits. Sustained testing over several hours helped to pinpoint problems in memory management and repetitive connections to backend data sources. If no irregularities were found in RTT and error rate after several hours, then the tester proceeded to adaptive testing.

The adaptive test was given a range for the highest RTT and error that is reasonable for the service to reach. These ranges were considered the highest values possible for the system that would still allow it to be effectively used by a user. The test then attempted to find the highest CPS where those values existed. If the CPS generated RTT and error lower than this range, the CPS increased. If CPS generated RTT and error higher than this range, the CPS decreased. This testing usually ran with CPU monitoring enabled, and lasted for at least forty-eight hours. The results for this test were used to generate a histogram to determine the optimum CPS for system.

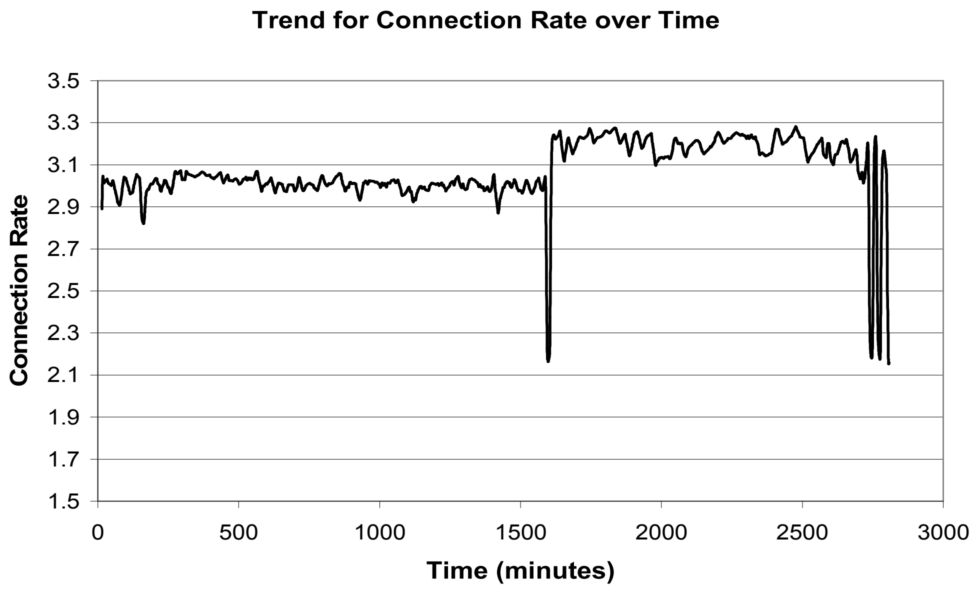

Figure 2. Connection Rate Fluctuation

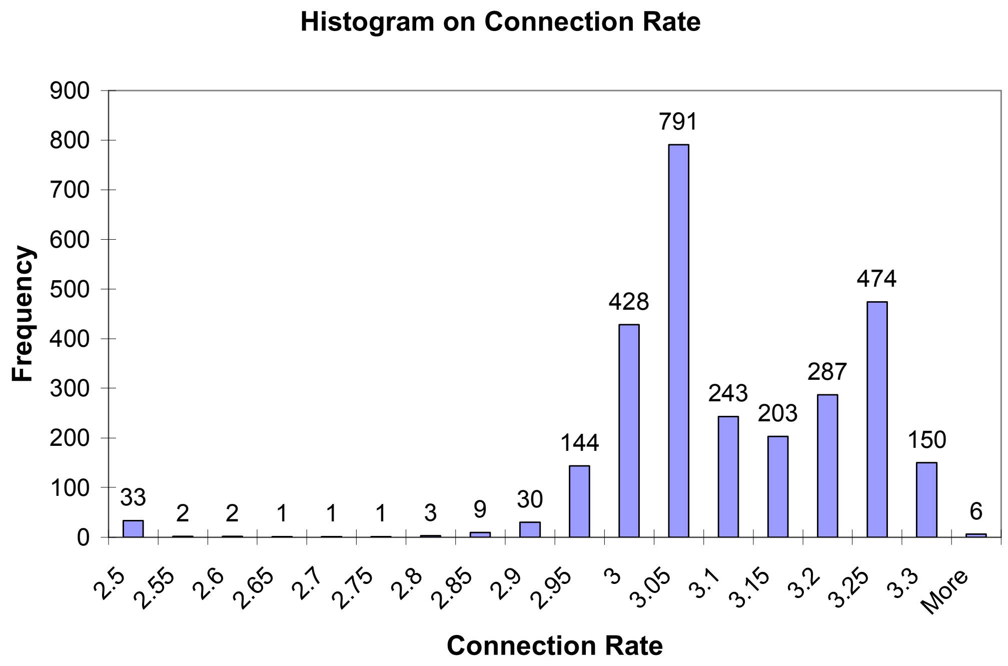

Figure 3. Connection Rate Histogram

4. TESTING EXAMPLE

A testing example is given to illustrate the usefulness of effective testing tools and plans. In the example, testing has proceeded to the point where adaptive testing is taking place. The connection rate in the test will increase or decrease to achieve a RTT between 3.5 and 4.5 seconds and an error rate that is less than 0.05% for a given one minute sample of queries. An error was defined as any query that did not return a result or returned an incorrect result. The maximum test time is set at 48 hours. The connection rate over time is shown in Figure 2. A histogram of CPS is shown in Figure 3. Although the histogram yielded a relatively high concentration between 2.95 and 3.3 connections a second, Figure 2 shows downward rate spike at around 26 and 48 hours. Although these spikes accounted for less than 0.34% of operating time, secondary testing was used to find possible causes. The accompanying second stage testing, including garbage collection and CPU monitoring, did not reveal an underlying factor that caused this fluctuation. This fluctuation was logged for further review and monitoring.

Future plans include attempting to replicate the results in another development environment and designing operational testing to monitor for such aberrations. Looking at the rest of the results from the adaptive test showed that the mean CPS was 3.06 with a 99% confidence value of 0.01.

5. CONCLUSIONS

The greatest conclusion that can be realized from the testing procedures is that even though exhaustive testing is not possible, testing is still iterative and time intensive. Various levels and types of tests had to be repeated in order to characterize the architecture's performance and to find implementation errors and flaws. A major benefit of development testing was the realization of the bottlenecks within system components. This knowledge was vital to the development of the operational system monitoring tools. With these tools, the ability to diagnose failure within a loosely coupled web services architecture was facilitated.

NCD will also continue its ongoing activities in developing its test harness. This tool has helped in testing functionality and measuring performance of various web services. The ability to test functionality was extremely important since anyone can generate client classes from the accompanying web service's WSDL. This means that clients can submit requests that are syntactically correct but semantically meaningless. The ability to test for such problems added robustness to the initiative's web services.

The last area of continued research and development for NCD will be in developing test cases that better characterize the operational environment. Differences between the testing and production environment like database size and server configuration can cause characterization curves to be incorrect. By closely modeling the end environment, these problems will be minimized.

6. REFERENCES

[1] K. Gottschalk, S. Graham, H. Kreger, J. Snell, "Introduction to Web services architecture", *IBM Systems Journal*, pp. 170-177, November 2, 2002.

[2] M. S. Pallos, "Service-Oriented Architecture: A Primer", *eAI Journal*, pp. 32-35, December 2001.

[3] Glossery of XML Key Management Requirements. http://www.w3.org/2003/glossary/subglossary/xkms2-req/

[4] M. Chen, A. Chen, B. Shao, "The implications and impacts of web services to electronic commerce research and practices", Journal of Electronic Commerce Research, VOL. 4, NO. 4, 2003

[5] P. Vita, "Challenges in the Adoption and Diffusion of Web Services in Financial Institutions", Working Paper CISL# 2004-07, 2004, http://web.mit.edu/smadnick/www/wp/2004-07.pdf

[6] Horizontal Fusion Developer's Reference, http://horizontalfusion.dtic.mil/docs/specs/20041118_Final_Developers_Ref.pdf

[7] Horizontal Fusion Developer's Guidance, http://horizontalfusion.dtic.mil/docs/specs/20041118_Final_Developers_Guide.pdf

[8] Horizontal Fusion Standards and Specifications, http://horizontalfusion.dtic.mil/docs/specs/20041112_HF_Standards.pdf