Comparing Performance of Web Service Interaction Styles: SOAP vs. REST

Pavan Kumar Potti
pavan.aryan@gmail.com

Sanjay Ahuja
sahuja@unf.edu

Karthikeyan Umapathy
k.umapathy@unf.edu

Zornitza Prodanoff
zprodano@unf.edu

University of North Florida
1 UNF Drive, Jacksonville, FL 32224, USA

Abstract

This paper presents a comparative performance evaluation of two Web service implementations: one is based on SOAP and the other on Representational State Transfer (REST). Simple Object Access Protocol (SOAP) and REST-based development approaches handle service interactions quite differently. SOAP is a standardized framework for constructing and processing messages independent of the technological capabilities of the receiver and can work on top of a variety of application layer protocols such as RPC, HTTP, or SMTP, whereas, REST is a set of principles for designing Web applications (HTTP as the underlying protocol). We built SOAP and REST-based Web services that perform Create, Read, Update, and Delete (CRUD) operations on a database and retrieve local files. We utilized response time and throughput metrics to compare the performance of these Web services. We found that, on average, REST has better performance compared to SOAP, though not all results were statistically conclusive. As an ancillary outcome, we found that developing Web services using SOAP was easier, due to considerable tool support. However, developing Web services using REST was time consuming and difficult due to the necessity of in-depth knowledge of HTTP and rudimentary tool support.

Keywords: Web Service, SOAP, REST, Interaction Style, RESTful, Performance

1. INTRODUCTION

In this paper, we investigate two Web service interaction paradigms: SOAP and Representational State Transfer (REST), in order to assess effectiveness of their data transfer capabilities. These varied approaches to develop Web service solutions have attracted a lot of debate both in academic and practitioner communities. Choosing service interaction style is a major architectural decision for designers and developers, as it influences the underlying requirements for implementing Web service solutions (Pautasso, Zimmermann, & Leymann,

2008). While the major software infrastructure providers such as Microsoft and IBM provide tool support for developing SOAP-based Web services, there have been an increasing number of advocates for the *RESTful* approach in the development of Web service solutions, where REST is used in conjunction with Universal Resource Identifier (URI) and HTTP (Note that SOAP has its own application layer protocol provisions and does not necessarily operate on top of HTTP, hence the label "SOAP-based Web services").

The SOAP-based Web service standard stack includes various standards such as WSDL, WS-BPEL, WS-Choreography, WS-Transaction, WS-Security, WS-Addressing, and many more developed by standardization organizations such as W3C and OASIS. Thus, SOAP-based Web service development essentially involves understanding relevant standards specification and using the right set of toolkits to develop solutions. This constantly increasing parade of standards and associated technologies often creates challenges for developers in terms of conceptual understanding and navigation of the standards space.

On the other hand, the RESTful approach espouses that Web service solutions can be developed by simply representing and exposing system's resources, and by transferring data over HTTP. A service is considered as a resource that can be identified and located by a URI and different operations can be performed on the resources using HTTP methods. In contrast, SOAP-based development focuses on exchange of communication and actions that occur between services. Thus, SOAP suggests a communication-oriented model (Umapathy & Purao, 2007) whereas REST suggests a resource-oriented model for designing interactions among Web services. See appendix A for overview of SOAP and REST, and a review of related work.

Although SOAP and REST are both present ways for building Web services, they differ in the manner data are processed and services offered. SOAP is XML-based message exchanging protocol for distributed computing, whereas REST is a design principle for Web-based applications that closely adheres to client-server architecture and advocates using bare minimum HTTP methods. Therefore, comparing these two technologies is not a trivial task. We developed a SOAP-based Web service and a RESTful Web service. Both services perform series of data exchange operations on a database server. In this article, we compare SOAP and REST interaction styles based on data transfer performance of two alternative Web service implementations using metrics such as response time and throughput. This article provides a neutral assessment of the performance and services offered to developers and architects by SOAP and REST methodologies for developing Web services.

The remainder of the paper proceeds as follows: first we provide a discussion of experimental methods and set up. Next, we discuss results of the experiments and statistical analysis followed by a discussion of results implications.

2. EXPERIMENTAL SETUP AND METHOD

Web services are most commonly used for exposing functionalities and data to other applications. Thus, service interactions comprise of information flows into and out of a service. Complex service interactions would involve an exchange of dynamic content generated by retrieving and updating data from databases. Frequent service calls, therefore, can increase service processing time and create throughput bottlenecks. Perceived performance issues with service interactions would be response times and throughputs (Cherkasova, Fu, Tang, & Vahdat, 2003). We compare SOAP and REST using response time and throughput as performance metrics.

In order to facilitate comparison of performance of SOAP and REST interaction styles, we developed two Web services that perform–*create*, *read*, *update*, and *delete* operations on a database. However, one service, called CustomerInfoSOAP, uses SOAP technology, whereas another service, called CustomerInfoREST, takes advantage of REST principles. From here on, CustomerInfo service will be used to refer to both SOAP- and REST-based services. We created a 'Customer' table in the database (Oracle 10g) containing following attributes: First name, Last name, SSN, CustomerID (primary key), Salary, Email, Active status, Mobile, City, and Country.

A Customer program was developed (using Java) to manage the above specified customer details. This Customer program was used as the basis to develop various functionalities offered by the CustomerInfo service. The CustomerInfo

service offers five functionalities: getCustomer (obtains a particular customer record from the database based on given CustomerID), addCustomer (creates a new customer record using given customer information), updateCustomer (updates an existing customer record using given information), deleteCustomer (deletes an existing customer record based on given CustomerID), and getTheFile (retrieves and returns a specified file stored locally in the server). The first four functions are used for measuring response time and the fifth function is used for measuring throughput.

Client applications were developed (using Java) to invoke and interface with the appropriate service functionalities offered by CustomerInfo service. The interaction between the client application and CustomerInfo service was used as the basis for comparing performance of SOAP and REST. The client application interacted with the CustomerInfo service using both wired and wireless connections. For this experiment, two wired client machines and two wireless client machines were used. Wired clients were connected to the service via a 10/100 Full duplex Ethernet modem. Wireless clients were connected to the service through an access point using 802.11g protocol. The server that hosts service, database server, and all clients were located in the same room. Appendix B provides hardware and software configurations and Figure C1 in appendix C depicts the experimentation setup.

Measuring Response Time

The methodology used to measure response time for each service function was same. The general scenario for measuring service response times involves a client application invoking an appropriate functionality provided by the CustomerInfo service along with relevant data at an instance of time A (measured in milliseconds). The service receives the request, processes it, connects to the database, performs the requested operation on the database, and sends an appropriate response to the client. The client receives the response completely at some time B (measured in milliseconds). The response time was measured as the difference between times A and B.

The concept of multithreading was used to simulate multiple clients accessing the service at a given time. Each service request sent from a client was a thread, and each thread had a different identification number (threadID). Each thread was initialized sequentially. To make sure each thread performs operations on the intended customer records, rather than all threads focusing on the same customer, each thread requested specific operation on the customer ID based on the thread ID.

In general, the number of service requests per client depends on the application that produces them. Since we are not modeling specific application requests, to more realistically mimic client behavior, for this experiment, we randomly allocated the number of service requests per client, namely: 1, 2, 3, 4, 10, 12, 13, 14, 15, 16, 18, 19, and 20. That sequence was produced by an on-line random number generator evaluated by (Kenny, 2005). The first experiment was conducted with one service request for each client, thus, a total of two service requests. Following that, four service requests were sent with the next experiment, i.e., two threads (service requests) on each client. Subsequent runs were conducted with 6 service requests (3 threads on each client), 8 service requests (4 threads on each client), 20 service requests (10 threads on each client), so on up to 40 service requests (20 threads on each client). Table C1 in appendix C provides a list of experiment runs along with the number of requests made by each client.

The response time was calculated for each thread separately. For every run, the arithmetic mean of the thread response times was measured and considered as the response time for that run. For example, for a run of 10 service requests, the response time for each thread was measured and the arithmetic mean of 10 response times was calculated and recorded as the response time for 10 service requests. Figure C2 in appendix C provides the skeleton of the code for measuring response time.

As per HTTP specification, GET, PUT, and DELETE methods have idempotence property (HTTP, 1999). The idempotent methods produce same results, whether it is executed once or multiple times (Wikipedia-Idempotence, 2011). GET method is idempotent safe, as it is typically used for retrieving a resource without resulting in any side-effects (HTTP, 1999). In the context of HTTP PUT method, modifying a resource state (for e.g., updating a customer name from "Smith" to "Jones") is considered idempotent; because the final resource state will be same no matter how many times the operation is performed. Similar, argument can be made for

the HTTP DELETE method. Thus, according to HTTP specification, multiple HTTP PUT and DELETE requests are not allowed. This has implications for CustomerInfoREST service as updateCustomer and deleteCustomer functionalities rely on HTTP PUT and DELETE methods correspondingly. We were able to make multiple service requests using multithreading for getCustomer and addCustomer functionalities for both services, however, were not able to make multiple requests for updateCustomer and deleteCustomer functionalities for CustomerInfoREST service.

Using HTTP PUT and DELETE methods as a part of a sequential request, however, is considered to be non-idempotent (HTTP, 1999). We created a client application that invokes addCustomer, getCustomer, updateCustomer, and deleteCustomer service functions in a sequential order. The client application first invoked addCustomer service function with relevant data, upon receiving a response, invoked getCustomer function, upon receiving a response, invoked updateCustomer function, upon receiving a response, finally invoked deleteCustomer function. Similar to other service functions, response times for composition of all four functions were measured for multiple service requests using multithreading. Figure C3 in appendix C provides the skeleton of the code for measuring response time for composition of all four service functions. Therefore, we use response time measures for getCustomer, addCustomer, and composition of all four functionalities as the basis to compare performance of the service.

Measuring Throughput

Throughput is, typically, defined as the data processed per second. We measure throughput as the number of application bytes per second and the number of clients per second. Throughput was measured using getTheFile function, which retrieves a specified file. Ten different image files ranging in size from 76 KiloBytes (KB) to 5 MegaBytes (MB) were used for measuring throughput. Files of type .png were stored in the local directory of the server in which CustomerInfo services were hosted. The client invokes getTheFile function and sends a service request for a file with the filename. The service processes the request, retrieves the file from the local drive, and sends the file to the client. System time stamp was recorded by using the getTime() Java method before invoking getTheFile and another timestamp of the system clock was recorded again after receiving the requested file. The difference between the two times was considered as the response time. Throughput in KB per second was calculated by dividing the file size in KB by the response time in seconds. Throughput in clients per second was calculated by dividing the number of clients by the response time in seconds. Figure C4 in appendix C provides the skeleton of the code used for measuring throughput.

Since there are multiple requests to the same files (including both accesses to the database table and image files), the effects of caching needed to be considered. We anticipated that physical memory caching of retrieved from the server's hard drive files (including database Table files) would have the largest effect on response times. Most of the accesses to cached files were following the exact same pattern for both the REST-based and SOAP-based implementations. The only difference in caching for the two implementations resulted from the fact that REST based experiments were conducted first, that is, the very first accesses to each file took longer to retrieve by the REST implementation. That is, we are taking a pessimistic approach in estimating response times, where our reported REST based implementation response times, would have been even faster if files were located in physical memory before the experiments were conducted. We must also note that we did not implement Web caching, as our Web server was attached to the same LAN as the clients for all experiments.

3. RESULTS

Response times for multiple service requests were gathered for getCustomer, addCustomer, and all four functions. Throughputs were gathered for multiple service requests accessing image files using getTheFile function.

Response times for getCustomer function

The getCustomer function enables a client to request for information about a customer by providing customer ID. The service gets the specified customer details from the database and sends the response to the client. Multithreading was used to depict multiple clients requesting the service at the same time. Figure C5 in appendix C depicts SOAP vs. REST

comparative chart of the response time in milliseconds against the number of simultaneous service requests for wired clients. Figure C6 in appendix C depicts the same for service requests from wireless clients. From the graphs, it can be observed that for the getCustomer function, REST had better response times than SOAP as the number of simultaneous requests increased. Graphs also indicate that response times for wireless clients were better than for wired clients. Links were underutilized for this experiment (i.e. are not bottleneck and carry little amount of traffic as compared to the next experiment). These results are not surprising since difference in network speeds between Fast Ethernet and 802.11g had little effect on performance. The effect of link speed differences is overshadowed by the effects of the performance capabilities client computers. This happened because the wireless clients were running on computers that had a newer configuration (dual and virtual cores) compared to those where the wired clients executed. From figure C6, it can be noted that SOAP was competitive until the number of simultaneous service requests gets greater than 30.

Response times for addCustomer function

The addCustomer function enables a client to add new customer data to the database. The service adds the new customer details to the database and sends a response to the client to inform successful completion of the process. Figure C7 depicts SOAP vs. REST comparative chart for wired clients and figure C8 depicts for wireless clients. From the graphs, it can be observed that for the addCustomer function, REST had better response times than SOAP as the number of simultaneous requests increased. Similar to getCustomer function, SOAP had better response times for wireless clients than for wired clients.

Response times for all four functions

The all four functions involved invocation of addCustomer, getCustomer, updateCustomer, and deleteCustomer functions in a sequential order. Thus, the client makes service requests in a sequential order, and service fulfills each request and sends a response after completion of the request. Response time was calculated for completion of all four functions. Similar to other functions, multithreading was used to make multiple simultaneous service requests. Figure C9 and C10 depicts SOAP vs. REST comparative chart for response times against the number of multiple service requests for wired clients and wireless clients, correspondingly. Similar to other getCustomer and addCustomer results, REST had better response times than SOAP as the number of simultaneous requests increased. Among the four functions, the getCustomer function constituted the majority of the response time, affecting the overall functionality and response time, accounting for the major performance difference. Similar to getCustomer and addCustomer functions, SOAP had better response times for wireless clients than for wired clients.

Response times for getTheFile function

The getTheFile function enables a client to retrieve a file stored locally in the server that hosts the CustomerInfo service. When the client makes the request for the file, the service retrieves and responds with the requested the file. Figure C11 depicts SOAP vs. REST comparative chart for response times against the file sizes in KB for wired clients and figure C12 depicts the same for wireless clients. From the graphs, it can be observed that response times for REST were comparatively better than SOAP response times, which is in keeping with the general trend observed with response times with other functions discussed before. However, as a departure to observed trend, wireless clients had higher response times than wired clients. Previously discussed functions requested text data, whereas the getTheFile requested image files (*.png). Thus, as the payload size of service response increases wireless clients may incur higher response times than that of wired clients for same service requests.

Throughput in KiloBytes per Second

Throughput can be defined as the average rate of successful data transmission over a channel. Throughput for CustomerInfo service was measured using the getTheFile function by requesting image files of sizes ranging from 76 KiloBytes to 5083 KiloBytes. Throughput for each file was calculated using the following formula: Throughput (bytes per second) = file size/response time in seconds. Figure C13 and C14 depicts SOAP vs. REST comparative chart for throughputs in KB per second against to varying file sizes for wired and wireless clients,

respectively. From below two figures, it can be observed that as the file size increased, REST has a higher throughput than SOAP. Figures also indicate that throughput for service requests for wired clients were higher than from wireless clients as it is expected because of link speed differences.

Throughput in Clients per Second

The following formula was used for calculating throughput expressed as clients per second: Throughput (clients per second) equals number of clients/response time in seconds. Similar to throughput in KB per second, getTheFile function was used for measuring throughput in clients per second and multiple service requests of files of varying sizes was made using multithreading. Figure C15 depicts SOAP vs. REST comparative chart for throughputs in clients per second against to varying file sizes for wired and wireless clients and figure C16 depicts the same for wireless clients. It can be observed that as the number of simultaneous service requests increased, REST has a higher throughput than SOAP. Similar to throughput in KB per second, throughput in clients per second was higher for wired clients than wireless clients.

Statistical Analysis

In order to assess whether there is a statistically significant difference between SOAP and REST in terms of their population means, for each conducted experiment we performed an independent samples t-test using SPSS software (SPSS, 2011). Due to space limitation, we have provided details of the statistical analysis, descriptive statistics, t-test results, effect size analysis and power size analysis in appendix E (see appendix D for tables).

Independent samples t-tests were conducted for each response time and throughput experiment groups. At the 5% level of significance, only addCustomer response time experiment with wired clients, and throughput in KB per second experiment with wireless clients were significantly different (i.e., $p < 0.05$). Difference between REST and SOAP groups for other experiments were not statistically significant at the 5% level of significance. The comparison of means reveals that REST had a lower response times and higher throughput than SOAP for all experimental groups.

Results of experiments with an insignificant difference, smaller than typical effect size and low power (less than 0.5) should be considered as inconclusive (Onwuegbuzie & Leech, 2004). Therefore, an insignificant difference cannot be interpreted as there is no statistical difference between REST and SOAP. Rather it indicates changes in experimental design and conditions may be necessary to reach conclusive results. Thus, regarding experiments with insignificant findings, future experiments should consider using either sample size larger than 13 per group or different experimental set up to observe a significant difference between REST and SOAP groups. We conclude that service developed using RESTful interaction style performed better than service developed using SOAP interaction style for addCustomer function with wired clients and throughput in KB per second with wireless clients.

4. DISCUSSION

The results of the experiments indicate that REST has better response times and throughput than SOAP. However, the difference between REST and SOAP were statistically and practically significant only for addCustomer (wired) and throughtput in KB per second (wireless) experiments. Response times can be affected by server processing capabilities and network bandwidth (Cherkasova, et al., 2003). Throughput can be affected by a number of parameters, including network capability, transmission channel, network congestion (number of shared applications), distance between computers, payload size, and processing technique to handle a payload (Choudhury & Gibson, 2006; Zhu, Davis, Chan, & Perreau, 2011). In our experimental setup, both services were tested using the same set of payload sizes, client and server machine configurations, number of clients, and number of service requests. All clients and server machines were located in the same room. Both services used HTTP as the underlying protocol for exchanging messages and files. To ensure that services have same processing capabilities, both SOAP-based and REST-based services were hosted in the same server machine. Services were tested in varied network environments (wired and wireless). Thus, experiment set up ensured that only difference is the technique used by services to process and respond to messages, i.e., service interaction techniques.

Regarding response time experiments, for all wired clients, REST consistently had lower response times than SOAP. In regard to wireless clients, SOAP and REST were competitive, however, on average REST had a better response time. Newer network configuration of the wireless clients helped SOAP to be competitive in comparison to REST. Apart from network bandwidth, response time would be affected by the processing time at the service-side. One of the main difference between REST and SOAP style interactions is that, for SOAP messages, the actual payload is included inside the envelope element, whereas, for REST entire message is the payload. Thus, SOAP service would have to perform additional processing to extract the payload information. Similarly, when sending a response message, SOAP service would have to perform additional processing to construct a SOAP formatted message. The SOAP client machines also would have to perform additional processing to create and to read the message. This additional processing time incurred towards retrieving information from the message and embedding response into the message, may explain higher response times for SOAP service.

Similar to response time experiments, REST on average performed better than SOAP for throughput experiments for both wired and wireless clients. Throughput experiments were conducted using image files of various sizes. There are considerable differences among REST and SOAP on handling messages with image files. REST considers the image file as a resource and includes the URL of the resource in the response message. Client machines can access the resource via URL and download the file. SOAP standard has an attachment feature that allows transmission of attachments along with a SOAP message. The SOAP attachment feature allows creation of a compound message structure consisting of a primary SOAP envelope part and secondary parts for including attachments (SOAP-Attachment, 2004). A compound structured SOAP message must contain one and only one primary part and zero or more secondary parts. Thus, every SOAP message with an attachment would contain a primary part regardless of whether an XML encoded message is included along with the attachment. Therefore, in comparison to REST service, SOAP service would have to perform additional processing to encode the image file as an attachment into a SOAP message. The SOAP client machines would have to perform additional processing to decode the message and access the image file. This additional processing time for encoding/decoding attachments from a message along with larger payload size due to compound structure could possibly explain lower throughput for SOAP service.

The sophistication of SOAP standard is contributing towards higher processing time and larger payload, which subsequently affects response time and throughput of the service. The simplicity and light-weight approach can be attributed to RESTful services better performance than SOAP-based services. However, there is another important side to this comparison, which should not be ignored before selecting a particular interaction style for designing services. As SOAP is a well-accepted industry standard, there are numerous specialized tool support provided by software vendors. Software vendors provide "out of the box" products to allow anyone with basic understanding of web services to develop SOAP-based services. These tools help developers with developing services easier and faster, thereby increasing productivity. Tool support available for RESTful approach is rudimentary and not as matured as the SOAP-based approach. Developers need to have basic understanding of HTTP, REST principles, and web services to develop RESTful service. The lack of tool support means developers would have to spend a considerable amount of time towards developing RESTful services, thereby reducing productivity.

Taking results of the experiments and practical implications into consideration, we provide recommendations for selecting REST and SOAP based interaction styles. The RESTful approach would be appropriate when the bandwidth needs to be limited as it does not utilize any headers along with the payload. The RESTful approach espouses stateless service by maintaining resource state information at server-side and application state information at the client-side. The RESTful approach would be recommended when the service needs to be stateless, i.e., each service interaction is independent of other interactions. The RESTful approach would be a good choice when service needs to be developed with minimal vendor-based products, whereas, developing SOAP-based service without relying on tool support would be very difficult, due to complex associations between standards.

SOAP-based interaction style would be appropriate when the service must address complex non-functional and QoS requirements, including security, reliability, and routing. There are many standards developed on top of SOAP to support those requirements. As RESTful approach supports only service interaction (Issarny et al., 2011), thus, developers would have to hard code these requirements into their applications (Tyagi, 2006). SOAP-based services would be recommended when existing services needs to be aggregated into a composed service. Standards such as WS-BPEL (WS-BPEL, 2007) allow developers to specify a sequence of service invocations and exchange of input and output data between services. SOAP-based services are a good choice when the service needs to maintain contextual information and conversation state with partnering services. These requirements are supported by standards such as coordination (WS-Coordination, 2009) and choreography (WS-CDL, 2005).

One of the limitations of this study is that only two out of twelve experimental groups were revealed to be statistically different and rest of the groups were inconclusive. This limitation can be attributed to smaller sample size that is affecting effect size and statistical power. Another limitation related to inconclusive result and experiment design is the focus on CRUD operations. Payload (customer data and image files) used for CRUD operations may not have been sufficient to create substantial differences between REST and SOAP interaction styles. CRUD scenarios used in this study did not necessitate usage of other additional standards. Usage of additional standards can create a considerable difference in payload size between SOAP and REST interaction styles. Database and CRUD scenarios are conceptually closer to RESTful as it considers these operations as resources and exposes them as a service. SOAP-based approach could have a conceptual advantage over enterprise application integration scenarios which involves complex business transactions, maintaining conversation states, and conducting secured and reliable message exchanges. Thus, as a part of future work, we intend to compare REST and SOAP interaction styles in both CRUD and enterprise application integration scenarios using a larger sample size.

5. REFERENCES

Alonso, G., Casati, F., Kuno, H., & Machiraju, V. (2004). *Web Services: Concepts, Architectures and Applications*. Berlin: Springer-Verlag.

Cherkasova, L., Fu, Y., Tang, W., & Vahdat, A. (2003). Measuring and characterizing end-to-end Internet service performance. *ACM Transactions on Internet Technology (TOIT), 3*(4), 347-391. doi: 10.1145/945846.945849.

Choudhury, S., & Gibson, J. D. (2006, 7-10 May 2006). *Payload Length and Rate Adaptation for Throughput Optimization in Wireless LANs.* Paper presented at the Vehicular Technology Conference (VTC), Melbourne, Australia.

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences* (2nd. ed.). Hillsdale, NJ: Lawrence Erlbaum Associates.

Faul, F., Erdfelder, E., Lang, A.-G., & Buchner, A. (2007). G*Power 3: A flexible statistical power analysis program for the social, behavioral, and biomedical sciences *Behavior Research Methods, 39*(2), 175-191.

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* Doctor of Philosophy, University of California, Irvine. Retrieved from http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

HTTP. (1999). Hypertext Transfer Protocol -- HTTP/1.1 Retrieved June 3, 2011, from http://www.w3.org/Protocols/rfc2616/rfc2616.html

Issarny, V., Georgantas, N., Hachem, S., Zarras, A., Vassiliadis, P., Autili, M., . . . Hamida, A. B. (2011). Service-oriented middleware for the Future Internet: state of the art and research directions. *Journal of Internet Services and Applications, 2*(1), 23-45. doi: 10.1007/s13174-011-0021-3

Kenny, C. (2005, April 2005). Random Number Generators: An Evaluation and Comparison of Random.org and Some Commonly Used Generators Retrieved September 4, 2011,

from http://www.random.org/analysis/Analysis2005.pdf

Mulligan, G., & Gracanin, D. (2009, 13-16 Dec. 2009). *A comparison of SOAP and REST implementations of a service based interaction independence middleware framework.* Paper presented at the Winter Simulation Conference (WSC), Austin, TX, USA.

Onwuegbuzie, A. J., & Leech, N. L. (2004). Post Hoc Power: A Concept Whose Time Has Come. *Understanding Statistics, 3*(4), 201-230. doi: 10.1207/s15328031us0304_1

Pautasso, C., Zimmermann, O., & Leymann, F. (2008). *Restful web services vs. "big"' web services: making the right architectural decision.* Paper presented at the International conference on World Wide Web, Beijing, China.

SOAP-Attachment. (2004, 8 June 2004). SOAP 1.2 Attachment Feature Retrieved July 2, 2011, from http://www.w3.org/TR/soap12-af/

SOAP-Primer. (2007). SOAP Version 1.2 Part 0: Primer Retrieved February 2, 2011, from http://www.w3.org/TR/soap12-part0/

SPSS. (2011). IBM SPSS Statistics. Armonk, NY, USA: IBM. Retrieved from http://www-01.ibm.com/software/analytics/spss/products/statistics/

Tyagi, S. (2006). RESTful Web Services Retrieved May 28, 2011, from http://www.oracle.com/technetwork/articles/javase/index-137171.html

Umapathy, K., & Purao, S. (2007). A Theoretical Investigation of the Emerging Standards for Web Services. *Information Systems Frontiers, 9*(1), 119-134.

Wikipedia-Idempotence. (2011, 30 May 2011). Idempotence Retrieved June 3, 2011, from http://en.wikipedia.org/wiki/Idempotence

WS-BPEL. (2007, May 9). Web Services Business Process Execution Language (WS-BPEL) Retrieved June 10, 2008, from http://www.oasis-open.org/committees/download.php/23964/wsbpel-v2.0-primer.htm

WS-CDL. (2005, 9 November). Web Services Choreography Description Language (WS-CDL) Candidate Recommendation. Retrieved April 30, 2008, from http://www.w3.org/TR/ws-cdl-10/

WS-Coordination. (2009, 2 February 2009). Web Services Coordination (WS-Coordination) Version 1.2 Retrieved July 5, 2011, from http://docs.oasis-open.org/ws-tx/wscoor/2006/06

Zhu, G., Davis, L. M., Chan, T., & Perreau, S. (2011, Jan. 31 2011-Feb. 3 2011). *Trade-offs in energy consumption and throughput for a simple two-relay network.* Paper presented at the Communications Theory Workshop (AusCTW), 2011 Australian, Melbourne, Australian.

zur Muehlen, M., Nickerson, J. V., & Swenson, K. D. (2005). Developing web services choreography standards - the case of REST vs. SOAP. *Decision Support Systems, 40*(1), 9-29.

APPENDIX A. BACKGROUND

Overview of SOAP

SOAP is a communication protocol for exchanging messages among distributed applications regardless of their implementation specific semantics and programming platform. SOAP specifies XML-based framework to construct messages that can be transmitted over a variety of transportation protocols such as HTTP and FTP (SOAP-Primer, 2007). A SOAP message must have an envelope as its root element (SOAP-Primer, 2007). An envelope element can contain two sub-elements: header and body. The body is a mandatory element used for encoding information being conveyed. The information can be encoded either using document-style or RPC-style (Alonso, Casati, Kuno, & Machiraju, 2004). The header is an optional element used for providing contextual information related to processing the message. Thus, the body element is used for specifying actual payload and header element is used for specifying the value-added services such as security and transactional context (Alonso, et al., 2004). For more information about SOAP, refer to (SOAP-Primer, 2007).

Overview of REST

REST is an architecture style for designing and developing Web-based applications. The concept and architectural principles of REST were outlined by Roy Fielding in his Ph.D. dissertation. As per REST design principles, Web-based applications are built on top of stateless client-server architecture, where in, services offered by the server are considered as resources that can be identified by their URL (Tyagi, 2006). For example, if a client requests access to a resource (ex.: a Web page) using a URL, the server transmits the resource to the client along with links for accessing other relevant resources. If the client navigates to one of those links, then a transfer from one state to another has occurred, thus, the name REpresentational State Transfer (REST). Following the above argument, Web services can be considered as resources. Web service clients can access these resources through particular representations (URLs), and transfer data and other application content that describe the action to be performed on the resource (Tyagi, 2006). Web services developed following REST principles are called *RESTful services*. For more information about REST, refer to (Fielding, 2000).

Related Work

The debate between SOAP-based Web service development and taking a RESTful approach in the development of Web service solutions has been extensively argued among the practitioner community. In the academic community, recently few studies have focused on this important design choice. Pautasso et. al (Pautasso, et al., 2008) provide a conceptual comparison of SOAP- vs. RESTful web services based on technical differences. Their analysis indicates that in comparison to REST, SOAP-based development involves fewer design decisions but there are many alternatives to consider for each decision due to standardization and tool support availability. They also suggest that choosing a RESTful approach would eliminate series of decisions and alternatives to consider for supporting advanced functionality such as choreography, and QoS. However, providing such functionality support using REST would incur significant technical risk and development effort. Zur Muehlen et. al. (zur Muehlen, Nickerson, & Swenson, 2005) provide a comparison of SOAP and REST from the context of cross-organizational workflows and conclude that both provide different but technically valid ways to solve the problem. Mulligan and Gracanin (Mulligan & Gracanin, 2009) compared SOAP and REST-based implementations to support interactions between a middleware application and its peripheral devices. Their test results indicate that REST implementations are more efficient in terms of network bandwidth utilization and latency. However, their investigation was based on the context of supporting a specific middleware application; thus, their findings cannot be generalized for the context of designing and developing Web service solutions.

While there has been some discussions comparing SOAP- and REST-based approaches for supporting interactions among Web services, there is a lack of empirical studies that compare these technologies based on performance metrics. We intend to address this gap, in this paper.

APPENDIX B. HARDWARE AND SOFTWARE CONFIGURATIONS

Following is the hardware and software configurations used for both service and client applications:

- Service configurations
  - Hardware configurations of the hosting server
    - Processor: Intel ® Pentium ® 4 CPU 3.00 GHZ 2.99 GHz
    - RAM: 0.99 GB
    - Operating System: Microsoft XP Professional Version 2002 Service Pack 3
    - Hard Drive: Maxtor 6Y080M0
    - Network Adapter: Broadcom NetXtreme 57xx Gigabit Controller
  - Software configurations
    - Integrated Development Environment (IDE) used for developing services: Netbeans version 6.7
    - Application server used for hosting services: GlassFish 3 Prelude
    - Programming platform used for developing services: Java, Java Development Toolkit (JDK) 1.6
    - Application programming interface (API) for SOAP: Java API For XML-based Web Services (JAX-WS) 2.0
    - Application programming interface (API) for REST: Java API for XML RESTful Services (JAX-RS) 1.1
    - Database: Oracle 10g
- Client configurations
  - Hardware configurations of wired clients
    - Same as the hardware configurations for the hosting server
  - Hardware configurations of the first wireless client
    - Processor: Intel ® Dual Core CPU T2050 @ 1.73 GHz
    - RAM: 1.99 GB
    - Operating System: Microsoft XP Professional Version 2002 Service Pack 3
    - Hard Drive: Hitachi HTS541060G9SA00
    - Network Adapter: Dell Wireless 1390 mini-card
  - Hardware configurations of the second wireless client
    - Processor: Intel ® Core ™ i5 CPU M430 @ 2.27 GHZ 2.27 GHz
    - RAM: 4.00 GB
    - Operating System: Microsoft XP Professional Version 2002 Service Pack 3
    - Hard Drive: ST9320325AS
    - Network Adapter: Atheros AR5B93 Wireless Network Adapter
  - Software configurations
    - IDE used for developing client application: Netbeans version 6.7
    - Programming platform used for developing client application: Java, JDK 1.6

APPENDIX C. FIGURES

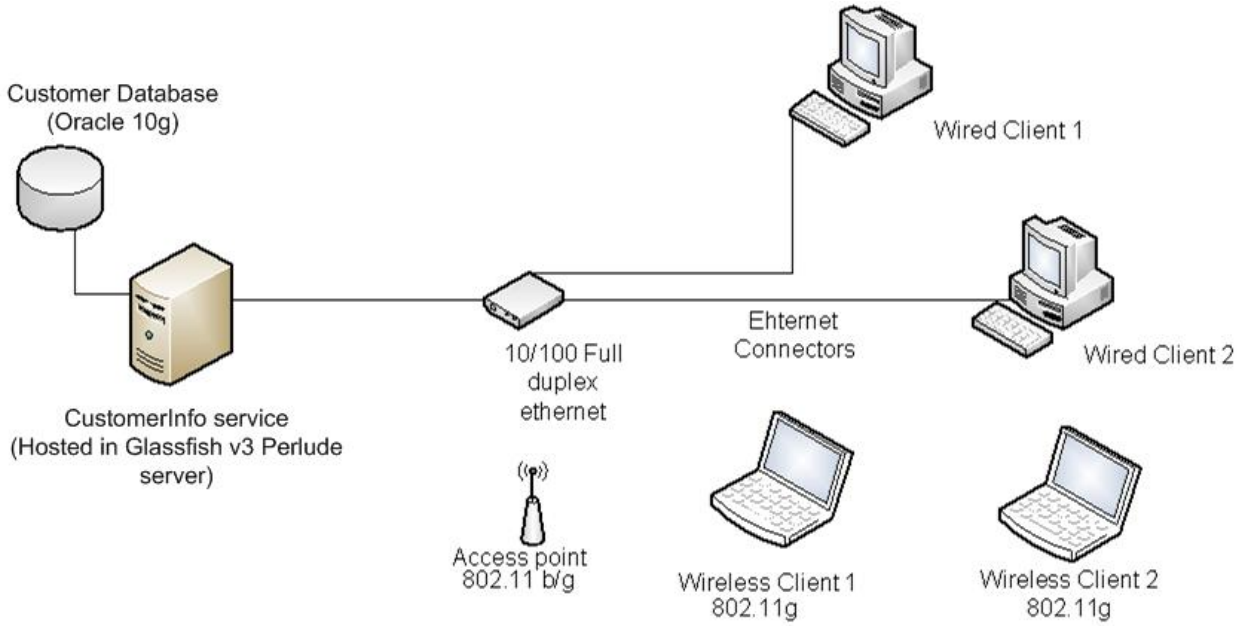

Figure C1. Experimental setup for running CustomerInfo service and client applications

```
Client Implementation Thread class
{
        Thread run method
        public void run ( )
        {
                someMethod (with correlation ID);
        }
        someMethod ( )
        {
                System time in milliseconds A of a particular thread X;
                Code for Operation;
                System time in milliseconds B of a particular thread X;
        }
}
```

Figure C2. Skeleton code for measuring response time

```
Client Implementation Thread class
{
        Thread run method
        Public void run ( )
        {
                publishMethod (with correlation ID);
                retrieveMethod (with correlation ID);
                modifyMethod (with correlation ID);
                deleteMethod (with correlation ID);
        }
        publishMethod ( )
        {
                System time in milliseconds A, of a particular thread X; (Timer started for the thread based on
                correlation ID)
                Code for operation with customer ID;
```

```
        }
        retrieveMethod ( )
        {
                Code for operation with customer ID;
        }
        modifyMethod ( )
        {
                Code for operation with customer ID;
        }
        deleteMethod ( )
        {
                Code for operation with customer ID;
                System time in milliseconds A, of a particular thread X; (Timer end for the thread based on
                correlation ID)
        }
}
```

Figure C3. Skeleton code for measuring response time for composition of all four service functions

```
Client class
{
        System time in milliseconds;-
        Code for requesting and getting the file;
        System time in milliseconds;
}
```

Figure C4. Skeleton code for measuring throughput

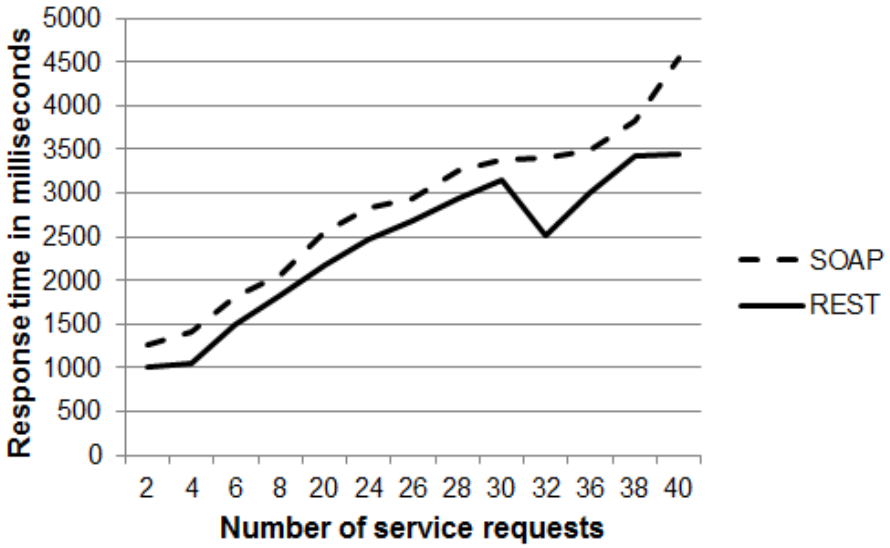

Figure C5. Response times for getCustomer function service requests from wired clients

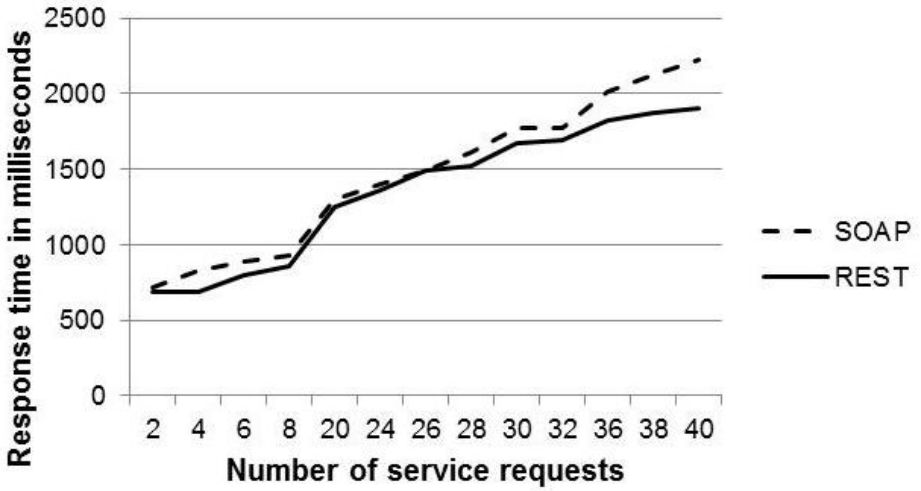

Figure C6. Response times for getCustomer function service requests from wireless clients

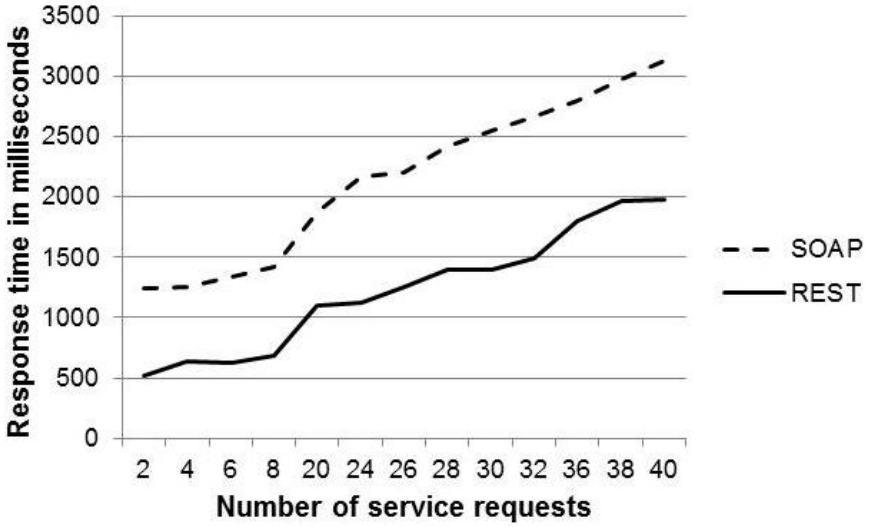

Figure C7. Response times for addCustomer function service requests from wired clients

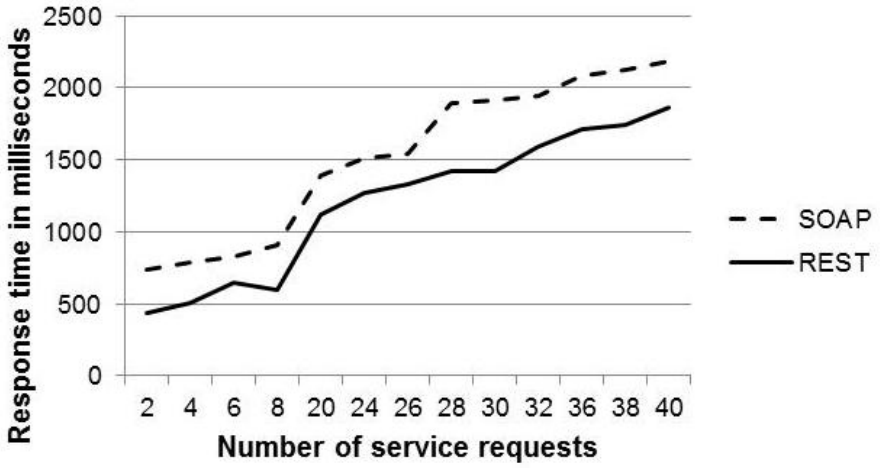

Figure C8. Response times for addCustomer function service requests from wireless clients

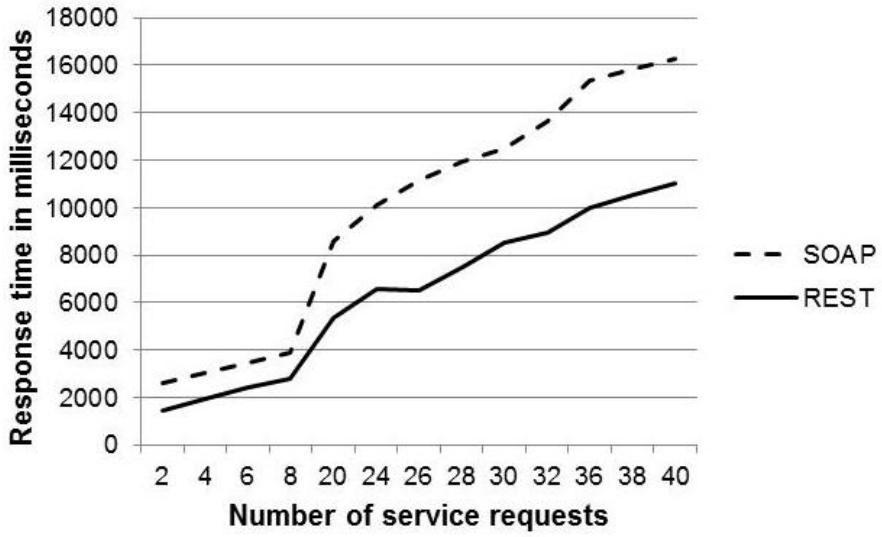

Figure C9. Response times for all for four functions service requests from wired clients

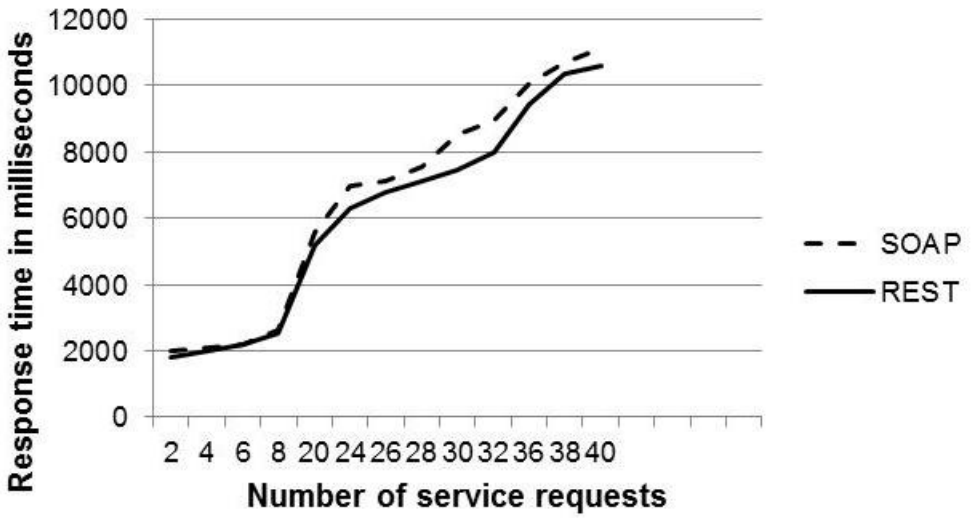

Figure C10. Response times for all four functions service requests from wireless clients

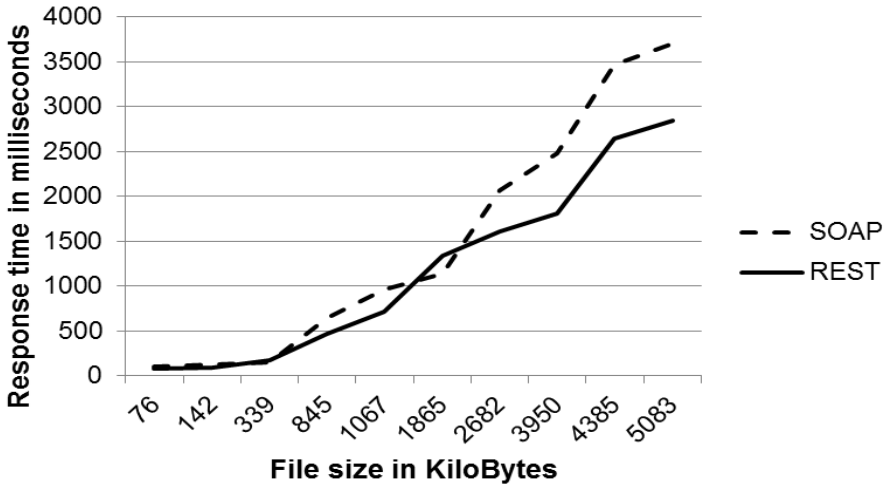

Figure C11. Response times for getTheFile function service requests from wired clients

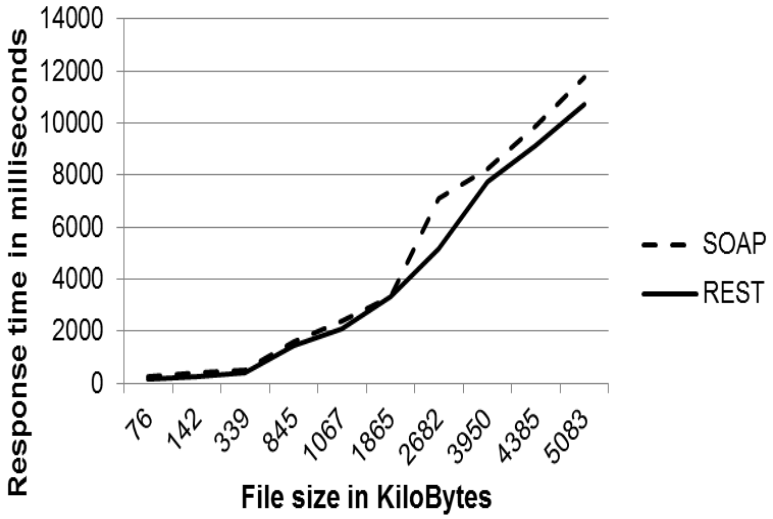

Figure C12. Response times for getTheFile function service requests from wireless clients

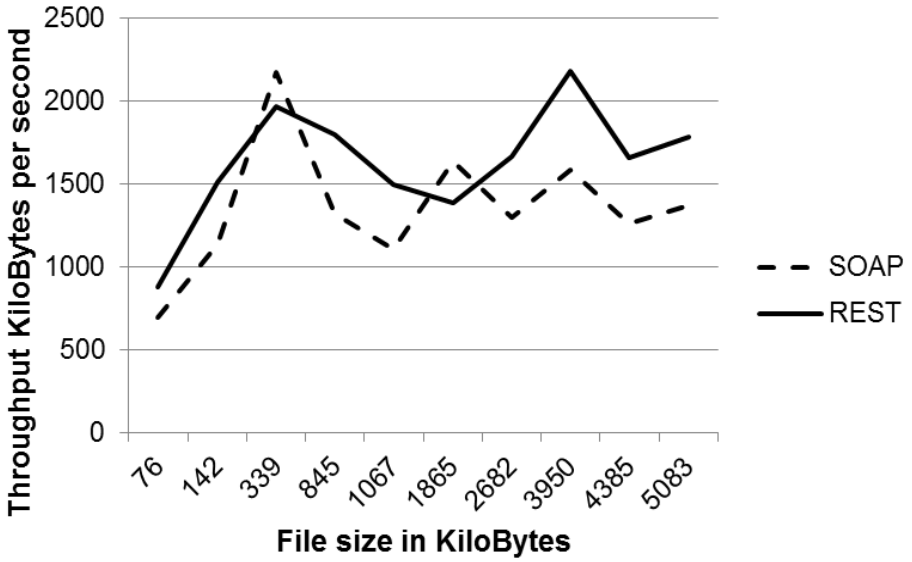

Figure C13. Throughput in KB per second for service requests from wired clients

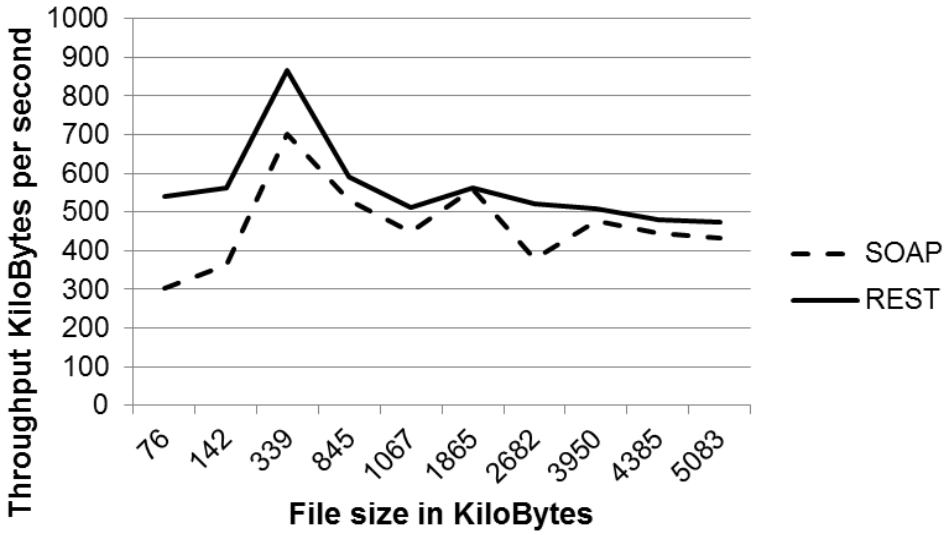

Figure C14. Throughput in KB per second for service requests from wireless clients

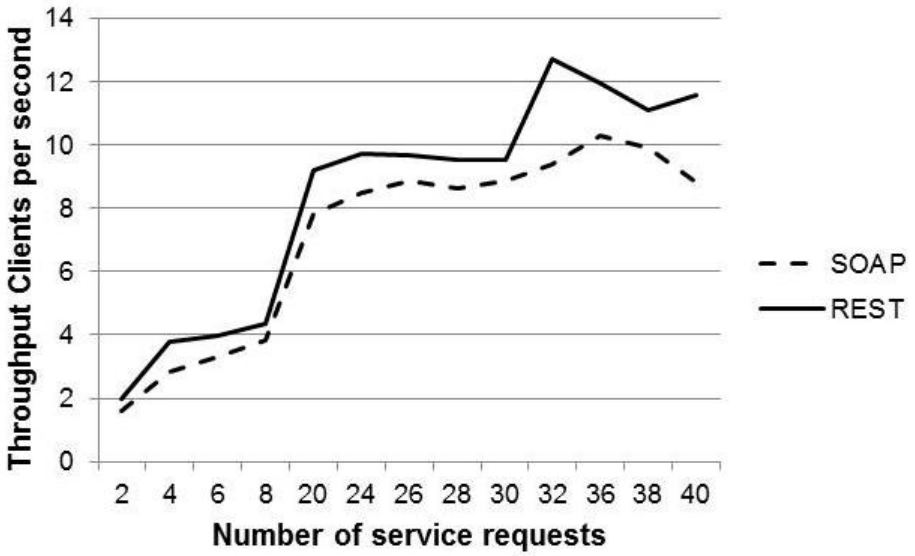

Figure C15. Throughput in clients per second for service requests from wired clients

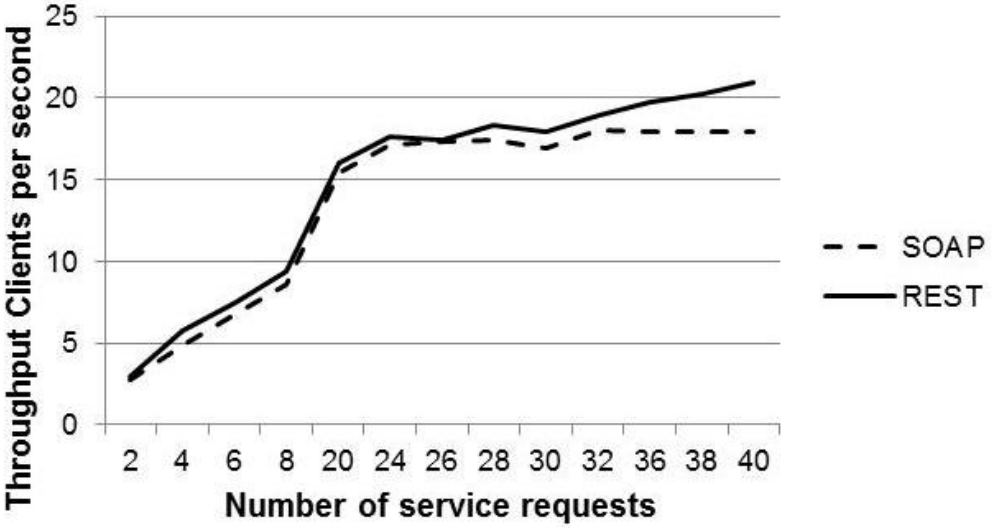

Figure C16. Throughput in clients per second for service requests from wireless clients

APPENDIX D. TABLES

Table D1. Experimental runs and data collection for response time

Experimental Run	Number of Service Requests		
	Client1	Client2	Total
1	1	1	2
2	2	2	4
3	3	3	6
4	4	4	8
5	10	10	20
6	12	12	24
7	13	13	26
8	14	14	28
9	15	15	30
10	16	16	32
11	18	18	36
12	19	19	38
13	20	20	40

Table D2. Descriptive statistics for response times and throughput in clients per second experimental groups

Experimental Groups		N	Mean	Std. Error Mean	Std. Deviation	Skewness		Kurtosis	
						Statistic	Std. Error	Statistic	Std. Error
Response time – getCustomer function	Wired clients	26	2616.815	179.810	916.855	-0.139	0.456	-0.575	0.887
	Wireless clients	26	1412.249	93.831	478.446	-0.158	0.456	-1.203	0.887
Response time – addCustomer function	Wired clients	26	1691.391	147.687	753.059	0.284	0.456	-0.834	0.887
	Wireless clients	26	1365.740	105.959	540.289	-0.238	0.456	-1.170	0.887
Response time – All four functions	Wired clients	26	8149.871	899.897	4588.591	0.156	0.456	-1.092	0.887
	Wireless clients	26	6356.798	631.045	3217.713	-0.184	0.456	-1.364	0.887
Throughput in clients per second	Wired clients	26	7.758	0.650	3.313	-0.593	0.456	-0.974	0.887

	Wireless clients	26	14.296	1.149	5.857	-0.921	0.456	-0.760	0.887

Table D3. Descriptive statistics for response times for getTheFile and throughput in KB per second experimental groups

Experimental Groups		N	Mean	Std. Error Mean	Std. Deviation	Skewness		Kurtosis	
						Statistic	Std. Error	Statistic	Std. Error
LOG Response time – getTheFile function	Wired clients	20	2.856	0.129	0.576	-0.468	0.512	-1.349	0.992
	Wireless clients	20	3.317	0.142	0.637	-0.487	0.512	-1.210	0.992
LOG Throughput in KB per second	Wired clients	20	3.160	0.028	0.123	-0.881	0.512	1.114	0.992
	Wireless clients	20	2.700	0.022	0.098	0.144	0.512	1.501	0.992

Table D4. REST and SOAP group descriptive statistics

Experimental Groups			N	Mean	Std. Error Mean	Std. Deviation
Response time – getCustomer function	Wired clients	REST	13	2403.775	231.907	836.152
		SOAP	13	2829.856	270.869	976.633
	Wireless clients	REST	13	1356.847	126.946	457.712
		SOAP	13	1467.651	141.624	510.634
Response time – addCustomer function	Wired clients	REST	13	1227.748	141.531	510.299
		SOAP	13	2155.034	187.143	674.754
	Wireless clients	REST	13	1204.787	139.028	501.274
		SOAP	13	1526.693	152.099	548.402
Response time – All four functions	Wired clients	REST	13	6430.974	938.153	3382.558
		SOAP	13	9868.767	1414.801	5101.136
	Wireless clients	REST	13	6136.475	878.411	3167.156
		SOAP	13	6577.121	937.841	3381.435
LOG Response time – getTheFile function	Wired clients	REST	10	2.814	0.185	0.584
		SOAP	10	2.899	0.189	0.597
	Wireless clients	REST	10	3.273	0.216	0.682
		SOAP	10	3.361	0.197	0.622

LOG Throughput in KB per second	Wired clients	REST	10	3.203	0.034	0.107
		SOAP	10	3.117	0.041	0.129
	Wireless clients	REST	10	2.744	0.024	0.075
		SOAP	10	2.656	0.033	0.103
Throughput in clients per second	Wired clients	REST	13	8.389	0.992	3.577
		SOAP	13	7.127	0.841	3.033
	Wireless clients	REST	13	14.822	1.704	6.144
		SOAP	13	13.770	1.596	5.755

Table D5. Independent samples t-tests results

Experimental Groups: REST vs. SOAP			**Levene's Test for Equality of Variances**		**t**	**df**	**Sig. (2-tailed)**
			F	Sig.			
Response times – getCustomer function	Wired clients	Equal variances assumed	0.245	p>0.05	-1.195	24.000	p>0.05
		Equal variances not assumed			-1.195	23.444	p>0.05
	Wireless clients	Equal variances assumed	0.151	p>0.05	-0.583	24.000	p>0.05
		Equal variances not assumed			-0.583	23.718	p>0.05
Response times – addCustomer function	Wired clients	Equal variances assumed	1.529	p>0.05	-3.952	24.000	**p<0.01**
		Equal variances not assumed			-3.952	22.343	p<0.05
	Wireless clients	Equal variances assumed	0.178	p>0.05	-1.562	24.000	p>0.05
		Equal variances not assumed			-1.562	23.809	p>0.05
Response times – All four functions	Wired clients	Equal variances assumed	3.172	p>0.05	-2.025	24.000	p<0.05
		Equal variances not assumed			-2.025	20.843	p<0.05
	Wireless clients	Equal variances assumed	0.113	p>0.05	-0.343	24.000	p>0.05
		Equal variances not assumed			-0.343	23.898	p>0.05
LOG Response times – getTheFile function	Wired clients	Equal variances assumed	0.001	p>0.05	-0.324	18.000	p>0.05
		Equal variances not assumed			-0.324	17.991	p>0.05
	Wireless clients	Equal variances assumed	0.097	p>0.05	-0.301	18.000	p>0.05
		Equal variances not assumed			-0.301	17.851	p>0.05
LOG Throughput in KB per second	Wired clients	Equal variances assumed	0.094	p>0.05	1.613	18.000	p>0.05
		Equal variances not assumed			1.613	17.428	p>0.05
	Wireless clients	Equal variances assumed	1.029	p>0.05	2.184	18.000	**p<0.05**
		Equal variances not assumed			2.184	16.449	p<0.05
Throughput in clients per second	Wired clients	Equal variances assumed	0.407	p>0.05	0.970	24.000	p>0.05
		Equal variances not assumed			0.970	23.374	p>0.05
	Wireless	Equal variances assumed	0.051	p>0.05	0.451	24.000	p>0.05

	clients	Equal variances not assumed			0.451	23.898	p>0.05

Table D6. Comparison of means test results

Experimental Groups		Mean Difference	Std. Error Difference	95% Confidence Interval of the difference		Pooled Std. Deviation	Effect Size (d)	Post Hoc Power
				Lower	Upper			
Response time – getCustomer function	Wired clients	-426.082	356.582	-1162.031	309.868	909.110	0.469	0.209
	Wireless clients	-110.804	190.192	-503.341	281.732	484.896	0.229	0.087
Response time – addCustomer function	Wired clients	-927.286	234.635	-1411.549	-443.022	598.205	1.550	0.966
	Wireless clients	-321.907	206.066	-747.206	103.393	525.367	0.613	0.323
Response time – All four functions	Wired clients	-3437.792	1697.584	-6941.434	65.849	4328.007	0.794	0.494
	Wireless clients	-440.646	1284.972	-3092.697	2211.406	3276.048	0.135	0.063
LOG Response time – getTheFile function	Wired clients	-0.086	0.264	-0.640	0.469	0.591	0.144	0.061
	Wireless clients	-0.088	0.292	-0.701	0.525	0.653	0.135	0.059
LOG Throughput in KB per second	Wired clients	0.086	0.053	-0.026	0.197	0.119	0.726	0.336
	Wireless clients	0.088	0.040	0.003	0.172	0.090	0.977	0.543
Throughput in clients per second	Wired clients	1.262	1.301	-1.423	3.946	3.316	0.381	0.154
	Wireless clients	1.052	2.335	-3.767	5.871	5.953	0.177	0.072

APPENDIX E. STATISTICAL ANALYSIS

The assumptions of population independence and Gaussian populations were tested. As REST and SOAP implementations were never executed together, data for each sample was gathered independently, moreover, REST and SOAP would never be implemented together (populations are also independent). Thus, the assumption of independence was not violated. The assumption of normal distribution was tested using observation of normal probability plots, histograms with normal curve, and the combination of skewness and kurtosis coefficients. Data gathered for response time (getCustomer, addCustomer, and all four functions) experiments and throughput in clients per second experiment indicate that dataset follow nearly normal distributions. However, response time for getTheFile function and throughput in KB per second experiments indicated that dataset did not follow a normal distribution. Data gathered for these experiments was transformed using LOG transformation function available within SPSS. Investigation of LOG transformed dataset revealed to follow a normal distribution. Descriptive statistics for the untransformed experimental groups are provided in the table D2 and for the transformed experimental groups are provided in the table D3.

Independent samples t-tests were conducted for each response time and throughput experiment groups. Table D4 provides REST and SOAP group statistics. Table D5 shows the results of the independent samples t-tests. Levene's test for assumption that the variances of the two groups are equal indicates that assumption is not violated (i.e., $p > 0.05$) for all experimental groups. Therefore, the equal variances assumed t-test statistics was used for analysis. From table D5, it can be observed, at the 5% level of significance, only addCustomer response time experiment with wired clients, and throughput in KB per second experiment with wireless clients were significantly different (i.e., $p < 0.05$). Difference between REST and SOAP groups for other experiments were not statistically significant at the 5% level of significance. The comparison of means (see table D6) reveals that REST had a lower response times and higher throughput than SOAP for all experimental groups.

Following Cohen's guidelines (Cohen, 1988), effect size (Cohen's d) was calculated to determine the magnitude of difference between REST and SOAP groups. Effect size was calculated by dividing the mean differences for REST and SOAP groups by the pooled standard deviation. The pooled standard deviation is calculated as the square root of the average of the squared standard deviations of REST and SOAP groups. Table D6 shows the effect size measure for all experimental groups. A larger than typical effect size ($d > 0.8$) was detected for response times for addCustomer function (wired) and throughput in KB per second (wireless) experiments. A typical effect size ($0.5 < d < 0.8$) was detected for response time-addCustomer (wireless), response time-all four functions (wired), and throughput in KB per second (wired) experiments. A smaller than typical effect size ($d < 0.5$) was detected for the rest of the experimental groups. Thus, the mean difference between REST and SOAP for response times for addCustomer function (wired) and throughput in KB per second (wireless) experiments are of both statistical and practical significance.

Post hoc statistical power analysis was performed using G*Power software (Faul, Erdfelder, Lang, & Buchner, 2007) to determine the likelihood of finding statistical difference between REST and SOAP for the given sample size and observed effect size. Typically, post hoc power between 0.5 and 0.8 are considered as adequate power and greater than 0.8 as high power (Onwuegbuzie & Leech, 2004). High power was observed for response time-addCustomer function (wired) experiment, thus, there is a high probability of observing similar findings in future experiments with a similar structure, effect size, and standard deviation at the 5% level of significance. Adequate power was observed for throughput in KB per second (wireless) experiment, thus, there is a moderate chance of observing similar findings in future experiments with a similar structure, effect size, and standard deviation at the 5% level of significance.