An Implementation of Replication Oriented Architecture (ROA) for Web Service Scalability

*Obilikwu, P. O. and Terwase, V. S.
Department of Mathematics and Computer Science, Benue State University, Makurdi, Nigeria
*Corresponding author: poblikwu@gmail.com

doi: https//doi.org/10.46912/napas.176

Abstract
Web services provide application to application integration across different platforms. However, the consumption of web services generates request traffic that must be attended to by an instance of the web server without fail. To guarantee dependability of the web service, the instances of the web service are replicated as a way of scaling the web service. The Replication Oriented Architecture (ROA) has been designed and implemented using the Java Enterprise application development platform and interesting results have been obtained. Improvements in the PHP scripting language have made it a popular programming language for web and enterprise application development. In this paper, an implementation of the ROA architecture using PHP is done. The implementation is simulated on the Apache Jmeter and results compared to the results obtained in the Java implementation. The results show that both application development platforms achieve web service scalability as a quality of service (QOS) expected of a web service. In specific terms, 50.9% at 95.0% confidence level improvement in response time was achieved when PHP is used which compares favorably with 22.5% improvement at 95.0% confidence level achieved on the Java platform.

Keywords: Web service, scalability, replication, availability, cluster, quality of service, Java, PHP

Introduction

The configuration of a web service requires a server instance of the web service. The carrying capacity of an instance, also referred to as a virtual server, depends on its configuration which includes memory, weights, administrator listener port, http listener port among other properties. Based on the configuration, the web server instance is given a weight which defines the number of requests it can handle among other things. Creating multiple server instances is a way of scaling the web service because a single server instance cannot scale to accommodate very large number of client connections (Braveti. Gilmore. Guidi, Tribastone, 2008).

An instance may fail and may need to be repaired automatically or otherwise. While awaiting repairs, the request will have to be transferred to another server instance. This ensures the availability of the web service at any given time. The need for several instances also arises when the number of requests outweighs the weight of the virtual server or the instance configuration is not the same. Creating several instances of web services defines its scalability.

There are empirical evidences that for the current generation of web server applications, multiprocessor platforms do not provide the needed scalability to handle large traffic volumes. A scalable web server architecture is therefore key to enabling web services to handle the ever increasing traffic loads (Marian, Birman and Rennese, 2006). The scalability of web services can be achieved by designing high quality architectures and replication is one such architecture. Replication is one of the most widely researched concepts in distributed computing and its initial use was in data replication. Data replication has been used to improve data access performances in distributed systems and it has been proved as capable of increasing data availability while reducing user waiting time, enhancing fault tolerance and ultimately improves scalability (Gopinath and Sherly, 2018).

A web service is a software component which is seen as a service it offers (ThiBen and Brambring, 2018). Web service is a self-contained component, which is published, located and invoked over the web. To achieve the goal of interaction across different platforms and programming languages, web service architecture defines standards for service definition (Web Service Description Language) and service interaction (Simple Object Access Protocol). (ThiBen and Brambring, 2018). The components of a web (Dustdar and Schreiner, 2005) service performs a discrete set of related functions. (Digvijaysinh, 2017) and there are no single web service standard.(Tilkov, 2005). Rather, the associated protocols aim to provide a means of describing data and behavior in a manner machine can process (Taylor and Harrison, 2009). When it is necessary to combine the functionality of several web service we speak of a composition service (Alonso et al, 2004). Composite services are recursively defined as aggregation of elementary and composite service (Dustdar and Schreiner. 2005).

Registry Infrastructure

With the number of services growing in today's computing environment, the need for meaningful cataloguing is a must. More so, as the number of consumers and services grow, remembering which service provides what functionality and the location of the service endpoint in order to send messages becomes very important. In the case of a small-scale SOA model, dependability upon high degree of human knowledge and interaction may be required but for a growing or large-scale SOA model, there is the need for a more effective and sustainable approach. In order to employ the full potentials of web service, the web service paradigm must be supported by an appropriate service publication and discovery infrastructure (Pilioura, Kapos and Tsalgatidou, 2004). At present, the most prevalent standard for WS publication and discovery is the Universal Description Discovery and Integration (UDDI) specification.

The UDDI information structure has four levels: The top level is the business entity level that provides the general data about the company, such as its address, a short description, contact information and other general identifiers. Associated with each business entity is a list of business services, including the description of each service and the categories of the service, for instance purchasing, shipping etc Also, within a business service, one or more binding templates provides more technical information about the web service.

Benefits of Web Services Scalabilty

Relying on a single instance of a web service is risky. It may become unavailable due to

failure or even overloading. To reduce this inherent risk, many instances should be created thereby making the web service scalable and available at all times. The importance of scalability is even more obvious given the fact that web services are inherently poorly scalable (Birman, 2005a, Birman, 2006, Cignek *et al*, 2006). Scalability talks about the ability of web services to be massively deployed on a large scale on different platforms and still maintain availability (Ekoubase and Onibere 2011).

Scalability assumes that instances of web services can be created. A web service is thus composed of several instances. In the event that an instance is "deceased", the dependability quality property of web services requires that the load on the "deceased" instance is transferred to another instance. Scalability therefore comes handy in balancing load assuming the load exceeds the weight of an instance. An instance of a web service among other configuration properties has a weight. The weight of an instance determines the resources it needs to be instantiated. There are several approaches for the composition of web services. One prominent example is the Business Process Execution Language (BPEL) for web services (OASIS, 2007).

The preceding discussion laid a foundation for this study in terms of a background study. The rest of this paper is organized as follows: Section 2 reviews related work necessary to better understand the technical aspects of this work, with an emphasis on the replication technique as being central to achieving web service scalability. Section 3 describes the details of the procedures and tools used to show that the PHP software development environment can be used to build scalable web service applications. In Section 4, experiments using both java and PHP development environments will be performed using the procedures and the APACHE Jmeter as a tools. The results of the experiments will be discussed with the aim of comparing the results obtained in both application development environments Section 5 concludes the paper and outlines current and future lines of research.

Review of Related Literature

Replications, clustering and parallel computing (Loukopolous, Lampsas and Ahmad 2005) (Loukopolous, Lampsas and Ahmad 2005) are well known as solutions to problems of database scalability. They have been used to enhance the performance and availability of databases. Related to the discussion on how replication, clustering and parallel computing has been used to enhance the performance and availability of databases are Database Management Systems (DBMS) (Perez, Garcia-Carballeira, Carretero, Calderon and Fernandez (2010); Mobile Systems (Tu, Li, Xiao, Yen. and Bastani (2006) and Large-scale systems and data grid systems (Ranganathan and Foster, 2001; Chervenak, Deelman, Foster, Guy, Hoschek, Iamnitchi, Kesselman, Kunst, Ripeanu, Schwartzkopf, B, Stockinger. and Tierney (2002).

All of the solutions related to achieving might have been first used to enhance the performance and availability of databases but they have found its application in other areas especially web service scalability. All the solutions rely on the ability to create multiple copies of data or service and to have multiple computers (that may or may not be at the same location) working together to provide the user with access to data or service. In this way, the solutions are closely interconnected and sometimes referred to interchangeably. However, there are subtle differences that are worth reviewing.

Database Replication Techniques

Replication has its origin in database theory and distributed database systems (Goel, Sunshant, Buyya & Rajkumar, 2006; Ghemawat et al. (2003); Birman, 2004, 2005a). Data replication can be categorized into two namely static replication and dynamic replication (Mokadem, and Hameurlain 2015). In a distributed environment, data replication can take place in a distributed storage this includes: (1) Distributed DBMS (2) Peer-to-Peer Systems (3) Data Grids (4) World Wide Web

Replicating database defines a database where multiple copies of some data items are stored at multiple sites or nodes (Goel, Sunshant, Buyya and Rajkumar, 2006). Poor scalability can result into poor system performance, hence the need to evolve better replicating strategies to improve performances especially in distributed systems. Data replication is a major technique used in distributed system to meet the challenges of high availability and improved data access performance. Data replication increases data availability, reduces user waiting time, increases

fault tolerance and improves scalability. Static data replication strategies follow a deterministic approach were the number of replicas to be created and the node to place the replica is well defined and pre-determined. i.e when and where to create replicas are determined before commencing the execution of the application. In static data replication is done randomly on randomly chosen nodes for a fixed number of times. Some examples of static replication approaches in a cloud includes: (a) Google File System (b) Hadoop Distributed File System (c) Amazon Dynamo.

Ghemawat *et al.* (2003) designed Google File system (GFS) used for scalable distributed file systems in data intensive applications. This file system support reliable, efficient access to large set data using big cluster of cheap hardware. The GFS implements a static distributed data algorithm for Google cloud. In GFS, the replicas in the multiple chunk servers are dynamically maintained. The limitation of this approach is that a fixed replica number is used for all files which may not approach replication properly.

Hadoop Distributed File System (Bui, Shujaat, Eui-Nam, and Sungyoung 2016), is a storage component developed by Apache Hadoop. This follows a static distributed replication policy to provide availability and reliability of data. The number of replica size for each file size is configured at the time of file creation. The placement of replicas is done in such a way that two replicas are stored in two separate nodes in the same local rack and one in a separate remote rack. The hadoop replication strategy improves data reliability, availability and network bandwidth utilization. The drawback of this approach is that access behaviour is not taken into consideration for replicating data. Dynamic replication policy is considered in such scenario where the replicas for each data is decided based on access popularity of data (Wei et al. (2010); Abad et al. (2011).

Clustering Architectures

Scaling web services means that as many instances of a web service as required are created to make the system fault tolerant. With redundant instances, it is easy to fail over from a "deceased" instance to another live instance. By default, the instances of a web service runs on one or more computers called clusters. Parallel computing is a particular example of cluster computing where multiple computers work together to provide some function or function whereby multiple computers each perform some sub-function. In the context of databases, a common example of parallel computing is a parallel database. Here multiple computers each process a subset of a query based on a subset of the data that they have access to. Sharded databases, or most NoSQL databases (MongoDB, Cassandra for example) are examples of this kind of system.

There are many kinds of clustering with implementations in different tiers. In every tier, it may be named differently meaning the same thing like virtualization, partitioning, mirroring. Some of the clustering tiers include:

(1) *AS Clustering:* Application servers (and HTTP servers) support clustering with varying capabilities. Some of them can make session state replication across AS instances. Some of them have load-balancers to distribute coming requests. Some of them can have transparent fail-over feature etc.

(2) *Hardware Clustering:* RAID is such technology used for both performance and reliable disks.

(3) *OS (Server) Virtualization:* We know many OS-level virtualization programs that can run many logical servers within same physical server at the same time.

(4) *DB Clustering:* Many DBMS supports clustered database instances via different topologies. We can add JDBC-level cluster libraries to this category that simulate clustered database feature.

Replication Oriented Architecture (ROA)

Replication is a mechanism whereby data or a web service is made available in more than one piece. In the simplest case of replication, there is a master and a slave. This master and slave arrangement could be in one computer or multiple computers. Where replication is implemented in more than one computer, the master copy could be in one computer and the slave in another. The computers may be in the same location or they may be located in geographically disperse area and connected via a communication network. The master and slave are first synchronized and after that, any change to the master is replicated to the slave. The changes may be replicated synchronously or asynchronously. Databases provided replication natively (semi-synchronous replication in MySQL) or one could use additional software (for example, Galera or Tungsten). Disk

mirroring in either hardware or software may also be considered as forms of replication.

In situations where replication is implemented in more than one computer, the computers are said to be a cluster. Clustering is therefore a generic term used to describe a class of techniques where many computers work collectively, and perform some function or functions. For example, if data is replicated between two locations then it is possible for a database to access each data set and answer queries submitted to it. In such a system, a copy of data can be in one location and accessed by one database instance and data in a either the same or a different location accessed by a second database instance. Then these two database instances would be considered to be a database cluster. In this example, each database instance is able to completely answer queries against the data. Oracle Real Application Clusters (RAC) is an example of a system of this kind.

Drawing from the experiences garnered from replicating databases, several specifications have been developed for the replication of web services. In the work of Farouk, O., Badawy, O., Youssef, M. [34] they proposed an architecture that integrate replication and clustering to provide reliability, availability and scalability of web service. In their approach, they posit that replication and clustering are needed to achieve availability and scalability. In their architecture, they proposed an N-tier architecture where components were divided into Four: Clients, Interface servers, Application servers, and Database Servers. They deployed replication in the interface servers and both clustering and replication for the application servers. However, this architecture with its manifold benefits is complex to implement and interface issues may present a challenge.

(Liu *et al*, 2004) presented in his research a frame work to publish up-to-date QOS information for web services in which the success depends on the mechanism of the feedback from the users about the quality of service they consume.

Jaeger, Goldman and Muhl (2004) proposed a mechanism which could be more efficient by using an aggregation scheme for QOS aspects. The scheme and approach as proposed by the authors has a challenge in that, service for composition are chosen sometime before execution, the QOS parameters changes during service execution, the QOS demands of a user may be violated even if no issues are found during service selection time. Replication is seen by this author as a possible solution to deal with dynamic QOS on performance, high availability and fault tolerance. Several copies of service are used instead of running single copies. There are a lot of other replication architectures or strategies example includes: The Gossip Architecture or Quorum Consensus, the Double Quorum architecture, and Cassandra. However, these strategies are only worth mentioning but are not considered.

According to ThiBen and Brambring (2018), originally these replication strategies were designed for data/database but soon there was a transfer to object replication. However, there exist approaches to implement these concepts into service replication. Ye and Shen (2005), discussed the implementation of reliable web service by using active replication. In his architectural approach, proxies for separating a user from the web service. In this approach, the proxy accepts all request from the user and is responsible for ensuring consistency in the execution of the several replicas. A user can only view a single proxy on which it sends a request to, but in the background, the proxy sends data or request across all other proxies or recipients simultaneously (multicasting), each recipient or proxy hiding one of the web services of the group. This approach focuses on reliability of web services and only active replication is implemented. However, when trying to look at several QOS aspects, this approach is too inflexible. In a similar case, (Chan *et al*, 2007) approach is focused only on reliability, other QOS is not dealt with. Quality of Service (QOS) is a broad concept that can involve a number of context-dependent non-functional properties such as privacy, reputation and usability (Liu, Ngu and Zeng, 2004). More so, in the work of (Salas, Perez-Sorrosal, Patino and Peris, 2006) web service replication as an approach was carried out with a goal of providing highly available web service in a wide area network. Again, this approach made use of active replication to achieve high availability and introduced a multicast mechanism to communicate between replicas.

In (Ekoubase and Onibere, 2011), the proposed architecture for web service scalability is server side and it has been noted from the web service solution test presented that the scalability

of web service is significantly better when built on ROA. This view is also supported by (Thiben and Brambring, 2018).

In the work, it was shown that ROA improves the web scalability by 31.7% with 90% of confidence. The web service architecture as proposed by Ekuobase and Onibere (2011) was however implemented on the Java Enterprise Application platform to neglect of other competing enterprise application development platforms.

Php Web and Enterprise Application Development Platform

PHP is currently one of the most popular languages used in open source community and in industry to build large web-focused applications and webservices (Dudhe and Sherekar, 2014). PHP has evolved over the years from a scripting language to an Object Oriented Programming (OOP) Language thereby providing the web development community with all the powerful benefits of OOP. With the evolution of PHP from the very first version to the current stable version 8.0, several aspect of the language has evolved: the use of libraries, removal of some functions, stability of user interfaces (Kyriakakis and Chatzigeorgiou. 2014).

PHP has gained maturity over the years by the number of growing open source community and the number third party libraries and APIs used. In terms of speed, PHP there has been tremendous improvement starting from versions 5.6 to the stable 8.0. In terms of frameworks, PHP boast a lot of open web frameworks such as symphony, cakePHP, Zend, Laravel and others thereby adding speed to web development which eases the development of SOAP and RESTful Webservices.

Methodology

The purpose of scalability testing is to check whether our system scales appropriately to the changing load. It is expected that a larger number of incoming requests should cause proportional increase in response time. The proposed architecture will be built to reflect this property.

Proposed Web Service Architecture

We chose to design a simple fictional web application called Students information system that pushes data to a backend that exposes its functionality as a web service based on our proposed web service replication architecture to test for scalability. The Use-Case and corresponding class diagram of the application are depicted in Figures 1 and 2.

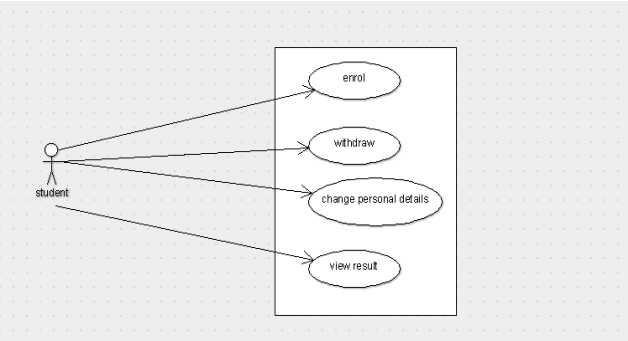

Figure 1: Use-case Diagram: Student Information Management System.

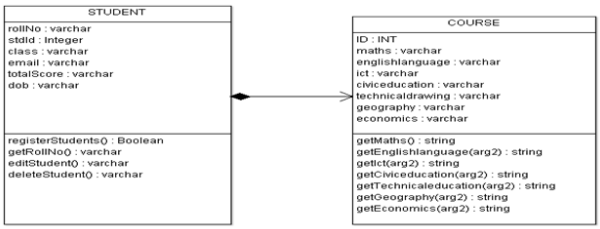

Figure 2: Class Diagram: Student Information Management System.

In this implementation, a three-tier components architecture is proposed. The components are web component, application component and the database or backend component. The architecture is depicted in Figure 3.

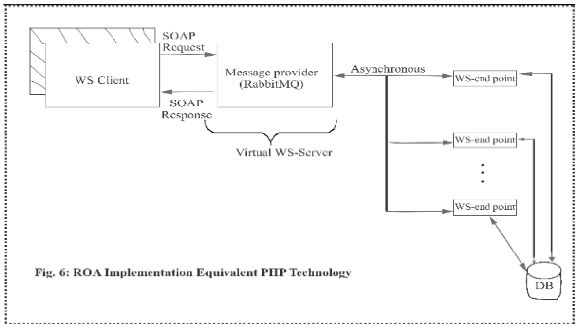

Fig. 6: ROA Implementation Equivalent PHP Technology

A cluster with a single node in the application server is used with two instances created. Each instance with its configuration is deployed as a multi-tier application. The goal of this solution is to find metrics and check if the solution scales appropriately in response to increasing load based on the architectural design.

The message brokers system is used due to the following advantages:

(1) To process background jobs. When application needs to process a lot of data. E.g an email notification from an online e-commerce system.
(2) To process message later when application is to.
(3) Scaling.

Hardware Tools: the tools used here were HP Pavilion Notebook with the following configurations: IntelCore (TM)i5-2410M,CPU@2.30GHZ Dual Core, 6.0GB installed memory and 600GB of Hard disk was used for developing the prototype.

Software Tools: We chose to discuss our software tools under operating system, Integrated Development Environment (IDE), Development language. We chose to use XAMPP server not because it is silver bullet but because of familiarity and its good features with respect to deploying large web applications solutions. Windows 10 operating system was used not because of lack of other operating systems, but just for familiarity.

Rabbitmq: is a message oriented middleware tool that allows to communicate and exchange data by sending and receiving messages. It uses the AMQP (Advanced Message Queuing Protocol).

Docker: is a container management software with images, volumes and container. An image is a blueprint (structure with instructions) for building a container. Images are made up of layers.

Backend Database: by this we mean the relational database of choice of our application. There were several DBMS but in a small to medium application, the choice of MySQL database was deemed adequate for our solution.

Language and IDE: the language of implementing our application was Java programming and the Integrated Development Environment (IDE) of choice was Visual Studio. This tool was chosen not because it was better than other development environments like PHPStorm or Netbeans IDE but because of its familiarity and seamless compatibility with PHP and Mysql.

Simulation Tools: Apache JMeter 5.0 was used as a simulation tool. This is because of its extensive features and very vast array of listeners and comprehensive GUI. The researcher is aware of other simulation tools like SOAPUI, SOAPPro, MatLab etc

Implementation

In order to test ROA architecture as designed, Rabbitmq was installed with docker and our application deployed. Our Application was a fictitious Web Service application with a Mysql backend and able to display request of its content in JSON for other application to consume. The message queing middleware (RabbitMQ) is akin to Java Message Service in java. The request and response is handled by the RabbitMQ. A file

called Dockerfile was created in order to provide docker configuration for our PHP web service application and its backend. The Dockerfile is a file that is used in building an image this file consist of various configurations for the rabbitmq that serves as a blueprint for a Docker image. The application was then simulated using Apache JMeter and the response time of the application was determined. We kept the number of request (sample size) similar with the Java ROA solution.

Apache Jmeter is simulation software that is designed to test and measure the performance and functional behaviour of client applications. It is one of the most popular and widely used open source, freely distributed testing application. JMeter was developed by Stephano Mazzochi of the Apache Software Foundation. It was primarily designed to test the performance of Apache JServ which was later substituted with the Apache Tomcat Project (Emily, 2008). Ever since its first release, JMeter has since developed and evolved to load-test FTP servers, database servers, java servlets and objects. JMeter is written in java and is highly extensible through a provided Application Programming Interface (API). JMeter works by simulating at client side of a client/server application. JMeter has been widely accepted as one of the best performance or load testing simulation tools for web applications and various companies have adopted Apache JMeter as a performance testing tool (Emily, 2008). Some of the companies includes: SharpMind of Germany for functional and regression testing, AOL for load testing of websites, ALALOOP of France has used JMeter since 2008 for performance testing of many web applications.

Results and Discussion

The results obtained from the two experiments performed on Apache Jmeter are depicted in tables 1 and 2. Table 1 shows the test results the ROA architecture built using PHP while Table 2 shows the results of the architecture using Java. Both experiments used the same dataset and ramp up period.

Table: 1: Scalability Table for our application solution build on ROA JAVA solution

No. of runs	No. of Sample size(Virtual Users)	Ramp-up Period (in seconds)	Throughput in Minutes	Per request Throughput in Minutes	Per request Throughput in (ms) (T_{ij})	Mid-Response of sample size(ms)	Computational Strength-per unit request in $(ms)^{-2}$	Performance Degradation$(ms)^{-2}$
1	1	1	234.375	234.375	0.0651	256	0.000254313	
2	5	2	182.704	36.5408	0.05075	37	0.001371652	
3	10	4	163.488	16.3488	0.04541	41	0.001107642	0.00026401
4	50	24	127.081	2.54162	0.0353	42	0.000840483	0.000267159
5	100	50	120.817	1.20817	0.03356	42	0.000799054	4.1429E-05
6	150	80	113.182	0.75454666	0.03144	40	0.000785986	0.000013068
7	200	95	126.828	0.63414	0.03523	40	0.00088075	0.000094764
8	250	120	125.433	0.501732	0.03484	39	0.000893397	0.000012647
9	300	148	121.933	0.40644333	0.03387	39	0.000868469	0.000024928
10	350	170	123.786	0.35367428	0.03439	41	0.000838659	0.00002981
11	400	200	120.26	0.30065	0.03341	39	0.000856553	1.7894E-05
12	450	230	117.489	0.26108666	0.03264	42	0.000777044	7.9509E-05
13	500	248	121.165	0.24233	0.03366	44	0.000764931	1.2113E-05
14	550	270	122.409	0.22256181	0.034	39	0.000871859	0.000106928
15	600	298	120.985	0.20164166	0.03361	44	0.000763794	0.000108065
16	650	320	97.638	0.1502123	0.02712	44	0.000616402	0.000147392
17	700	345	121.876	0.17410857	0.03385	39	0.000868063	0.000251661
18	800	390	123.196	0.153995	0.03422	36	0.000950586	0.000082523
19	900	400	135.027	0.15003	0.03751	41	0.000914817	3.5769E-05
20	1000	430	139.636	0.139636	0.03879	40	0.000969694	5.4877E-05
								π=9.13637E-05

Table 2: Scalability Table for our application solution build on PHP ROA Equivalent.

No. of runs	No. of Sample size(Virtual Users)	Ramp-up Period (in seconds)	Throughput in Minutes	Per request Throughput in Minutes	Per request Throughput in (ms) (T_{ij})	Mid-Response of sample size(ms) (R_{ij})	Computational Strength-per unit request in $(ms)^{-2}$	Perfomance Degradation$(ms)^{-2}$
1	1	1	652.174	652.174	0.181159444	15	0.012077296	
2	5	2	5769.231	1153.8462	0.320512833	44	0.007284383	
3	10	4	1073.345	107.3345	0.029815139	470	6.34365E-05	0.007220946
4	50	24	5988.024	119.76048	0.0332668	217	0.000153303	-8.98668E-05

Table 2: Cont.

5	100	50	121.065	1.21065	0.000336292	21	1.60139E-05	0.000137289
6	150	80	113.065	0.753766667	0.00020938	19	1.102E-05	4.99391E-06
7	200	95	126.834	0.63417	0.000176158	19	9.27149E-06	1.74849E-06
8	250	120	125.41	0.50164	0.000139344	16	8.70903E-06	5.62463E-07
9	300	148	121.957	0.406523333	0.000112923	16	7.0577E-06	1.65133E-06
10	350	170	123.829	0.353797143	9.8277E-05	15	6.5518E-06	5.05898E-07
11	400	200	120.243	0.3006075	8.35021E-05	15	5.56681E-06	9.84993E-07
12	450	230	117.586	0.261302222	7.2584E-05	14	5.18457E-06	3.82238E-07
13	500	248	121.163	0.242326	6.73128E-05	14	4.80806E-06	3.76512E-07
14	550	270	122.354	0.222461818	6.17949E-05	14	4.41392E-06	3.94131E-07
15	600	298	120.959	0.201598333	5.59995E-05	14	3.99997E-06	4.13958E-07
16	650	320	122.002	0.187695385	5.21376E-05	14	3.72411E-06	2.75852E-07
17	700	345	121	0.172857143	4.80159E-05	14	3.42971E-06	2.9441E-07
18	800	390	123.189	0.15398625	4.2774E-05	13	3.2903E-06	1.39401E-07
19	900	400	110.29	0.122544444	3.40401E-05	13	2.61847E-06	6.71833E-07
20	1000	430	139.612	0.139612	3.87811E-05	13	2.98316E-06	-3.64691E-07
								π=0.000404522

The results obtained are as shown in tables 1 and 2 were subjected to calculation using MS-Excel application. It is also important to note that the ramp-up period was chosen at random but it was chosen in a way to mimic real life load on the application solution.

We apply the mathematical model for estimating the computational strength of an application with increasing request as

$$\left|S_{i,j} - S_{i,j}\right| < \varepsilon \qquad \ldots\ldots\ldots\ldots\ldots\ldots \quad (1)$$

(Ekuobase, and Onibere, 2013).

Where ε represents the user degradation tolerance. The computational strength S_{ij} for an application i, for j request per unit time is given by:

$$S_{i,j} = C * \left(\frac{T_{i.j}}{R_{i,j}}\right) \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2)$$

(Ekuobase, and Onibere, 2013).

Where C is a constant denoting server and hardware strengths, $\mathbf{T_{ij}}$=application throughput per unit request, $\mathbf{T_{ij}}$ is converted to per milliseconds (ms), $\mathbf{R_{ij}}$ =application mid response time set of request or sample in milliseconds (ms), i represents the application, where i=0(1) , 0 for the conventional approach solution and 1 for the ROA. j=number of samples.

C=1, because we are comparing the computational strength with each other under the same hardware and software.

Let the samples, X and Y be the performance degradation at the Java ROA approach and our PHP ROA approach and their means μ_x and μ_y. We seek whether or not our solution built on PHP ROA approach will improve scalability.

We proposed hypotheses: H_1: $\mu_x<\mu_y$ (java ROA approach is not significantly scalable to ROA built on PHP). H_2: $\mu_x>\mu_y$ We make certain assumptions:

(1) That our sample size is normally distributed.
(2) Our sample size is small. In our case 20 (i.e. n=20).
(3) Our data points (sample size) are the same.
(4) We have the same variance.

$$T - value = \sqrt{\frac{\left|\mu_x - \mu_y\right|}{\frac{Sx^2}{n_1} + \frac{Sy^2}{n_2}}} \qquad \ldots\ldots\ldots\ldots\ldots \quad (3)$$

Where n1and n2 are the number of samples, S_x and S_y are the standard deviation and S_x^2 , S_y^2 is the variance between the two samples.

From Table 1 and 2, μ_x= 0.0000913637, μ_y=0.0004045, $n_1=n_2=20$, S_x^2= 0.000000007612, S_y^2=0.000002896

Substituting these values in equation (3),

$$\text{T-value} = \frac{0.000313136}{0.000381026} = 0.821.$$

Using the Null Hypothesis, H_0 (There are no significant difference between the samples), using the 0.05 (α level) probability,=> 95% times the null hypothesis will be rejected and only 5% the null hypothesis will not be rejected.

The degree of freedom (df)=$(n_1-1)+(n_2-1)$ (for an independent test). =38.

The critical value from the two tailed T-Test=2.021.

Also, using descriptive statistics in our Microsoft Excel to calculate the confidence level for the mid response time, we calculate the confidence interval for Table1. The Application solution built on ROA Java solution) result as 22.5% improvement with 95.0 confidence level. Furthermore, we also calculated the confidence interval for Table 2. The Application solution built on ROA PHP solution) 50.9% improvement at 95.0 confidence level.

We conclude that since our T value is lower than the critical value, we don't reject the null hypotheses. It therefore means there is nothing statistically significantly different between the samples from both architectures.

Conclusion

Based on our analysis, we generally conclude that there are no differences in the scalability of web services built with ROA using Java and its equivalent in PHP. However, it was observed from literature reviewed that web services developers are now tilting towards the use of PHP for developing web services due to its popularity and available of several frameworks.

This paper did not delve deeply into some of the areas that are considered novel in the general area of web services but this work is a foundation based on which such areas will be studied in our future research. Such areas include the following among others:

(1) The area of building web services as a microservice.
(2) Docker/ kubernetes (Container Orchestration) for large container management of web services deployed in containers.
(3) Web Services security. Web services being distributed across different platforms on the internet faces the challenge of security.

References

Abad, Cristina L., Yi Lu, and Roy H. Campbell. (2011). "DARE: Adaptive data replication for efficient cluster scheduling. ", in cluster computing (CLUSTER), 2011 IEEE international Conference on, IEEE 159-168.

Birman, K. (2005): Can Web Services Scale up? IEEE Computers. 38(10):107-110.

Birman, K., Cignek, R. (2006): The Untrustworthy Web Services Revolution. IEEE Computer.39(2):98-100.

Braveti, M. Gilmore, S. Guidi, C. Tribastone, M. (2008). Replicating webservices for scalability. Trustworthy global Computing. ISBN: 978-3-540-78662-7. Volume 4912.

Bui, Dinh-Mao, Shujaat Hussain, Eui-Nam Huh, and Sungyoung Lee. (2016): Adaptive replication management in HDFS based on supervised learning." IEEE Transactions on Knowledge and Data Engineering 28(6): 1369-1382.

Chan, P.P.W. (2007). Reliable Web Services: Methodology, Experiment and Modeling. Proceedings of the IEEE International Conference on Web Services (ICWS 2007), Salt Lake City, USA.

Chande, Suresh (2003). Semantic Wev & Web service software technology Laboratory Nokia Research Centre.

Chande, Suresh (2003): Semantic Web and Web Service Software Technology Laboratory Nokia Research Centre.

Chervenak, A., Deelman, E., Foster, I., Guy, L., Hoschek, W, Ianmnitchi, A., Kesselman, C., Kunst, P., Ripeanu, M., Schwatzkopf, B., Stockinger, K., and Tierney, B. (2002): 'Giggle: a framework for constructing scalable replica location service', supercomputing.

Digvijaysinh, R. (2017). "REGISTRY FOR RESTful WEB SERVICE: RESTRegistry." International Journal of Research-Granthaalayah, 5(7), 128-135. https://doi.org/10.5281/zenodoo.835523.

Dudhe, A, & Sherekar, S. (2014). Performance Analysis of SOAP and RESTful Mobile Webservices in Cloud Environment. International Journal of Computer Applications, 975,8887.

Duster, S., Schreiner, W. (2005)." A Survey on Web Services Composition", International Journal of Web and Grid Services, Vol. 1, No. 1, pp.1-30.

Ekuobase, G. and Onibere, E. (2011). Architecture for Scalable Web Services Solution. Canadian Journal of Pure and Applied Sciences. 5(1):1449-1453.

Ekuobase, G. and Onibere, E. (2013). Scalability of Web Services Solution built on ROA. Canadian Journal of Pure and Applied Sciences. vol. 7, No. 1 pp.2251-2270.

Ghemawat, Sanjay, Howard Gobioff and Shun-Tak Leung. (2003): "The Google File System. 19th ACM Symposium on operating system principles 37(5): 29-43

Halili, E. H. (2008). Apache JMeter: A Practical Beginner's Guide to Automated Testing and Performance for Websites. Birmingham, U. K. Pack Pub.

Http://www.omg.org/technology/documents/spec_Catalog.htm.

Jaeger, M.C., Goldman, R. G., Muhl, G. (2004): QOS Aggregation for Web Service Composition using Workflow Patterns. Proceedings of the 8th IEEE International Enterprise Distributed Object Computing Conference (EDOC'04), Monterey, USA.

Liu, Y., Ngu, A., Zeng, L. (2004): QOS Computation and Policing in Dynamic Web Service Selection. Proceedings of the 13th International World Wide Web Conference (WWW 2004), New York City, USA.

Marian, T., Birman, K., van Rennese, R. (2006): A Scalable Service Architecture. 25th IEEE Symposium on Reliable Distributed Systems(SRDS'2006)0-7695-2677-2/06.

Mokadem, R., and Hameurlain, A. (2015).'Data Replication strategies with performance objective in data grid systems: A Survey', International Journal of Grid and Utility Computing, vol.6, No.1 pp. 30-46.

OASIS, (2007). Web Service Business Process Executor Language Version 2.0 OASIS Standard. http:'//docs.oasis-open.org/wsbpel/2.0/OS/wsbpet V 2.0-OS. pdf.

P. Kyriakakis and A. Chatzigeorgiou, "Maintenance Patterns of Large-scale PHP Web Applications," IEEE International Conference on Software Maintenance and Evolution, PP. 381-390, 2014.

Perez, J. M., Garcia-Carballeira, F., Carretero, J., Calderon, A. and Fernandez, J. (2010):"Branch Replication Scheme: A new model for data replication in Large Scale Data Grids", Future Generation Computer Systems, Vol. 26. no.1, pp.12-20.

Piloura, T. Kapos, G. D. & Tsalgatidou, A. (2004). PYRAMID-S: A Scalable Infrastructure for Semantic Web Service Publication and Discovery. 14th International Workshop Research Issues on Data Engineering: Web Services for e-Commerce and e-Government Applications. Proceedings. doi:10.1109/ride.2004.1281698. source: IEEE Xplore.

Ranganathan, K. and Hameurlain, A. (2001). "Identifying dynamic replication strategies for a high performance data grid', International Workshop on Grid Computing.

Salas, J., Perez-Sorrosal, F. Patino, M.M., Peris, J.R. (2006). WS-Replication: A Framework for Highly Available Web Services. Proceedings of the 15th International World Wide Web Conference (WWW 2006), Edinburgh, Scotland.

ThiBen, D. Brambring, T. (2018): Improving Quality of Web Services By using Replication. IADIS International Journal on WWW/Internet. VOL. 7. No. 1. pp. 26-43.

Tilkov, S. (2005): Web Services Overview, GISOA. stefan.tilkov@innoq.com, innoq Deutschland GmbIT, http://www.innoq.com

Tu, M., Li, P. Xiao, L., Yen, I. L. and Bestani, F.B. (2006). '" Replica Placement Algorithm for Mobile Transaction Systems", IEEE Transactions on Knowledge and Data Engineering, Vol. 18, No. 7, pp. 954-970.

Van Steen, M., & Piere, G. (2010): 'Replicating for performance case studies'.

Web Services: A Survey of Web Services. citeseerx.1st.psu.edu/viewdoc/download? DOI=10.1.1.97

Wei, Qingsong, Bharadwaj Veeravalli, Bozhao Gong, Lingfang zeng, and Dan Feng. (2010) "CDRM: A cost effective dynamic replication management scheme for cloud storage cluster", in cluster Computing (CLUSTER), 2010 IEEE International Conference on: 188-196.

White, T. (2012). Hadoop: The definitive guide. "O' Reilly Media, Inc".

Ye, X. and Shen, Y. (2005). A Middleware for Replicated Web Services. Proceedings of the IEEE International Conference on Web Services (ICWS'05), Orlando, USA.