Journal, Advances in Mathematical & Computational Sciences
Vol. 8 No. 1, March 2020
www.mathematics-computationaljournal.info

Article Citation Format
Ochei, L.C.O., Ogunsanki, R. & Wobidi, E. (2020): Modelling the Performance of Interoperable Service-Oriented Systems. Journal of Advances in Mathematical & Computational Sc. Vol. 8, No. 1. Pp 109-124

Article Progression Time Stamps

Article Type: Research Article
Manuscript Received9th Jan, 2020
Final Acceptance: 21st March, 2020

Modelling the Performance of Interoperable Service-Oriented Systems

[1]Ochei, L.C.O., [2]Ogunsanki, R. & [3]Wobidi, E.
Department of Computer Science
University of Port Harcourt
Port Harcourt, Nigeria.
E-mails: [1]laud.ochei@gmail.com; [2]rotimi.ogunsakin@gmail.com

ABSTRACT

There is an increasing number of interoperable applications and services that are composed of components that run on different platforms, programming languages, storage systems and deployment mechanisms. There are serious transactional and security limitations for interoperable systems because they involve distributed transactions that span across different application components and shared resources. The paper presents an approach for modelling an interoperable service-oriented system to provide transactional and security support to improving performance and quality of service (QOS). This approach involves writing a sample program that mimics the behaviour of a typical interoperable system, observing the system, measuring selected variables, and thereafter using operational analysis and bounding analysis to derive input parameters and more information for modelling the system. The model which is based on linear regression has been applied to a sample procurement system to predict key parameters of the system. The model is recommended for modelling an interoperable SOA-based system to improve the performance and QOS.

Keywords*:* Modelling, Performance, Interoperability, Service-oriented systems, Measurement, QoS

1. INTRODUCTION

Businesses are increasingly running distributed transactions that span across different applications, platforms (e.g., .NET, Java, PHP, etc.) and shared resources (Laudati et al, 2003). These applications are developed from services, which are reusable software components services. The traditional common e-commerce application, for example, where a purchase order must be submitted across multiple systems, is an instance of this type of application. Because they are made up of multiple web services that are implemented on different platforms, these applications are known as interoperable SOA-based systems. It is often very difficult to connect concurrent business processes running on disparate platforms into a single transaction. For example, one platform may add or update data; another platform would later access the modified or added data which can severely limit transactional capabilities across platforms (Gabhart, 2004).

This limitation becomes more acute when concurrent transactions with interleaving operations spans across different applications and resources. Most online procurement systems are composed of a set of federated services (web service components) from Java EE, .NET, PHP platforms, and legacy systems deployed in various businesses centers distributed over the internet. Hours of service interruptions often translate into millions of dollars in lost revenue.

Without a proper system management infrastructure in place, the troubleshooting process can consume days or weeks before the problem is identified and fixed, thereby degrading overall service levels. Motivated by the problems highlighted above, this paper presents an approach for measuring the end-to-end performance of an interoperable SOA-based system. The main contributions of the paper are:

(i) presenting an architecture and approach for modelling an interoperable service-oriented system
(ii) presenting a model based on linear regression for predicting the number of requests completed by the system
(iii) applying the model to predicting the number of responses sent to the client

The rest of the paper is organised as follows: Section 2 reviews related literature. Section 3 discusses the approach for modelling an interoperable SOA-based system.

2. REVIEW OF RELATED WORKS

There are several related research work that explores existing and proposed transactional model to manage web service transactions. Alrifai et al (2009) proposed an extension to the standard web service transaction framework to support concurrency control on service level. Alrifai et al (2006) proposed an extension to the WS-transaction protocol for concurrency control. This approach is complex to implement. The protocol is based on a dependency graph maintained at the server-side. It avoids direct communication between transaction coordinators which preserves security by keeping mission-critical information. This approach has cost implications because it requires two times the number of exchanges messages to reach globally correct execution.

Chen (2008) proposed a new method called two-phase locking with fairness principles (2PL-FP), which resolves the concurrent data access for both real-time and non-real time support operations. This approach still suffers from locking-based concurrency protocols- deadlock, global cycles. Choi et all (2005) proposed a protocol called web services Transaction Dependency management protocol (WTDP) that ensures consistent execution of isolation-relaxing WS-transactions. WTDP is an extension of the WS transaction specifications. There are so many web services introduced to manage the protocol. This leads cost implications to an increase in the number of exchanges message to ensure consistency.

Paul et al (2007) proposed isolation solutions relaxed web service transactions while still maintaining an acceptable level of service. The research work is relevant because it suggested what conceptually resembles an approach that our proposed model used to implement admission control. That is if the set limit of arriving requests is reached, then request are rejected and sent to a temporary queue to be submitted later. This approach has no analytical or simulated model to substantiate the proposed solution.

Shan-liang et al (2011) designed and proposed a transaction coordination model based on extending WS-BPEL. The model is only a preliminary model and to make the success rate of execution of WSC higher, the introduction of THP protocol into the model was proposed to make it more robust. The model is not validated in mathematics. A high-level model of the simulated system is captured in Figure 1. In our simulation, the workload is specified by the number of concurrent requests in execution and not by an arrival rate. In this situation, a Closed Multiclass Queuing Network model is used.

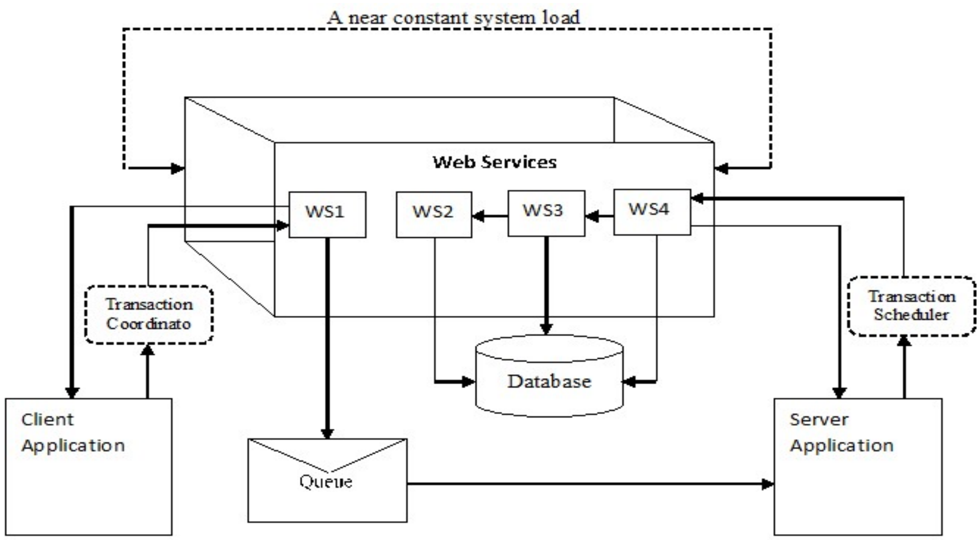

Figure 1: Architecture of the System used for measurement

3. MODELING INTEROPERABLE SERVICE-ORIENTED SYSTEMS

In this section, look at how to construct a model for the proposed system so that it will improve transactional and security support for an interoperable SOA-based system. Modelling an interoperable service-oriented system involves two phases:

i. Phase One - developing a sample system to be modeled: this phase, it is assumed that the system is abstract. This simply entails writing a simple computer program that mimics the behaviour of the procurement system that we are considering.
ii. Phase Two - deriving information about the system: this phase uses operational analysis and bounding analysis to derive more information about the system.

This paper assumes that an interoperable SOA-based system has been developed before the measurement of the performance can be done. In our previous work (Ochei et al, 2021), we presented an architecture of an interoperable SOA-based system and a framework for design and implementation of an interoperable SOA-based based on the framework. In the paper, we focus on the process of observing the system, measuring the selected variables and how to obtain input parameters that will be used to model the system to improve transactional support.

3.1 Operational Analysis

At this stage, we have measured data from the simulation runs and transformed them into input parameters, called operational variables. We will now use an approach called operational analysis to establish important relationships between these operational variables. These relationships, called operational laws, are quite general, simple and are based on readily available measurement data. In addition, we will also study the bounding behavior of the simulation model.

Based on a few simple observations of the system, we will derive more information from the system by applying these simple laws. Using this information as input to further laws and equations (such as queuing model, regression analysis, Markov models), we will gradually build up a model that represents a more complete picture of the behavior of the system. Thereafter, we will either translate the models/equations directly into algorithms (protocol) or introduce the models into certain sections of the algorithm. This algorithms and protocols when implemented will improve transactional and security support for interoperable SOA-Based systems.

Operational Laws

Operational laws are simple equations which may be used as an abstract representation or model of the average behaviour of almost any system (Hillston, 2009). The foundations of the operational laws are observable variables. These are values which we could derive from watching a system over a finite period.

Figure 2 represents a high-level model of an abstract system. We assume that the system receives requests from its environment. Each request generates a job or customer within the system. When the job has been processed the system responds to the environment with the completion of the corresponding request.

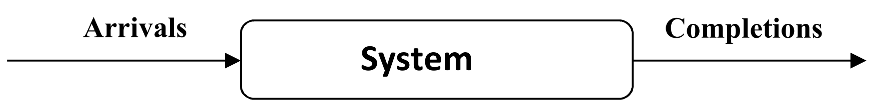

Figure 2. High level model of an abstract System.

The following is a presentation of the five operable laws that we will use in our study. The definition of the variables and notations used can be seen in section 4.10.1 and will therefore not be repeated here.

(i) **Utilization Law**:

Utilization law states that the utilization of a resource is equal to the product of the throughput of that resource and the average service requirement at that resource.

$$U_i = X_i \times S_i \quad (1)$$

If the number of completions from the resource i during the observation period is equal to the number of arrivals in that interval, then $U_i = \lambda_i \times S_i$.

(ii) **Forced Law:**

The throughput of a resource (X_i) is equal to the average number of visits (V) made by a request to that resource multiplied by the system throughput(X_0).

$$X_i = V \times X_0 \quad (2)$$

(iii) **Service Demand Law:**
Service demand, D, is defined as the total average time spent by a typical request of a given type obtaining service from a resource.

$$D = V \times S = \frac{U}{X} \quad (3)$$

(iv) **Little's Law:**
Little's law states that the average number of jobs in a system is equal to the product of the throughput of the system and the average time spent in that system by a job.

$$N = X \times R \quad (4)$$

(v) **Interactive Response Time Law:**
The law can be written as,

$$R = \frac{M}{X_0} - Z \quad (5)$$

where M is the number of clients and Z is the think time. Note that if the think time is zero, Z = 0, then the interactive response time law simply becomes Little's law.

3.2 Bounding Analysis
Bounding analysis is used to obtain upper bounds on throughput and lower bounds on response time from service demand. Bounding analysis provides information about the best possible performance that the system can have.

In general, upper bounds on throughput and lower bounds on response time that can be obtained by considering the service demands only (i.e., without solving any underlying model).

The upper asymptotic bounds on throughput are:

$$X_0 \leq \min\left[\frac{1}{\max\{D_i\}}, \frac{N}{\sum_{i=0}^{K} D_i}\right] \quad (6)$$

The lower bounds for the response time can be obtained as follows.

$$R \leq \max\left[N \times \max\{Di\},\ \sum_{i=0}^{K} D_i\right] \quad (7)$$

4. EVALUATION

In this section, we present the experimental settings, and the description of the procurement system.

4.1 Experimental settings

The procurement system was designed using a laptop with the following specification -

Hardware requirements:

All experiments have been carried out on the same computation platform, which is a Windows 10 running on a SAMSUNG Laptop with an Intel(R) CORE(TM) i7-3630QM at 2.40GHZ, with 8GB memory and 1TB swap space on the hard disk.

Software requirements:

Windows 10 operating system, SQL Server 2010 database, Microsoft Message queue and Visual Studio 2010. MS Excel was used for building the model. The programming language used was C# because C#/.NET integrates the asynchronous callback design.

4.2 Description of the Procurement System

The model transaction for the procurement system is simple: place order for products. Clients invoke one Web service to place an order asynchronously and then return a purchase number. Client response time is measured as the time difference between placing an order and when the order processing is completed.

This sample application is configured in such a way that the entire sample transaction system can run on a single machine installed with Windows 7. The following components were created to support the sample application. There are two C# projects:

1. ASP.NET web service
2. Visual studio 2010 project called TemperatureWeb that contains several ASPX pages. There are several ASPX pages:
   (i) A console application (ClientApplication) that is used as a Client application
   (ii) An ASPX page used as metrics application and yet another used as metric application.
   (iii) An ASPX page used as application sever

From implementation standpoint, a summary of all the major components of the system are:

- Three databases: procurement, products, and metrics.
- Two message queues: placeholders to hold details of the order placed by customers, and metricsdetails to hold metrics details of the transaction.
- An application server called ProcessOrder that processes the orders placed by customers.
- An application server called MetricsReport that obtains metrics details from the metricsdetails queue and logs them to the Metrics database.
- An ASP.NET Web service that defines five WebMethods: MyOrders(generates order details and places them in placeordetails queue), OrderService(generates purchase order number if product availability check is successful) , InventryService(checks if product is available, and if not it calls the supply service), SupplyService(this service generates a new product request and then serves it to the database)
- A Client application created as a console application (ClientApplication) that creates multiple threads to simulate many clients placing orders for processing. The order numbers are generated randomly.

4.2.1 Client Application- ClientApplication

A client application is used to simulate a population of clients placing orders submitting transactions of varying complexity at fixed intervals. The client application uses a number of parameters. They are given below (see Table 1):

Table 1: ClientApplication Simulation Parameters

SN	Parameters	Description
1	Scale	Size of the maximum OrderNo to use for place an order
2	Count	Number of order that each customer(client) can place
3	Gdelay	Milliseconds between problem submissions
4	Threads	Number of clients to start
5	Tdelay	Seconds between client starts

The Tdelay parameter allows a client population to be introduced slowly to the system so that the rate at which server capacity is exceeded can be determined. Each client thread creates its own instances of convert (he class that contains the web services that are invoked) and MyOrders(client recording component). Each thread submits count number of factoring problems between 0 and scale. It uses Thread.Sleep to wait for gdelay milliseconds after order processing is completed before placing another order.

The clientApplication invokes and passes ClientID to MyOrder web service method to generate and place order details on the placeorderdetails queue. The Myorders WebMethod returns the ClientID back to the client. The difference between the time the client sent the request and the time that client respond was received and measured as the Client Response Time.

4.2.2 Application Server – ProcessOrder

In this research, the application server is implemented as ASP.NET application. An application server could also be implemented as a windows service or a console application, and in the case of having a large transaction-based system there could be a collection application server. We can actually run any number of applications servers easily, together with multiple clients and Web services, on a single machine. This type of application use implement the client application may not matter as much, provided it performs the same basic operations. In our case, each ProcessOrder application reads order details from a queue, process the orders and saves the generated purchase order (PO) numbers into the procurement database. It is also possible to simulate multiple application servers by creating multiple instances of ProcessOrder application.

4.2.3 Metrics Application- The Metrics Recorder

MetricsReport is developed as an ASP.NET application. It reads mericdetails from the metricdetails queue and saves it to the Metrics database. Multiple MetricsReport can also be started. The application design is simple and the performance of the metrics infrastructure is relatively unimportant and will not degrade the system. The collection of metrics details may even be performed on distinct servers to reserve other resources for application servers.

4.3 Performing the Test

The procedure for performing the test is summarized below:

Step 1: Preparing the database

Remove any existing data in the procurement database and metric database. Make sure the product table is properly populated with sufficient data. You can manually open up the database and delete existing data. Also, inspect the product table in the product database to make sure that the quantity field contains the same value.

Step 2: Start the web service methods

Start the TestWebService so that the web service methods can be accessible. A list of all web services running on the machine including the one just started shows that web service has started.

Step 3: Starting the Client application

Start the console application named – Client application. The input parameters can be entered directly from the keyboard. Figure 3 shows the output screen of the *ClientApplication*.

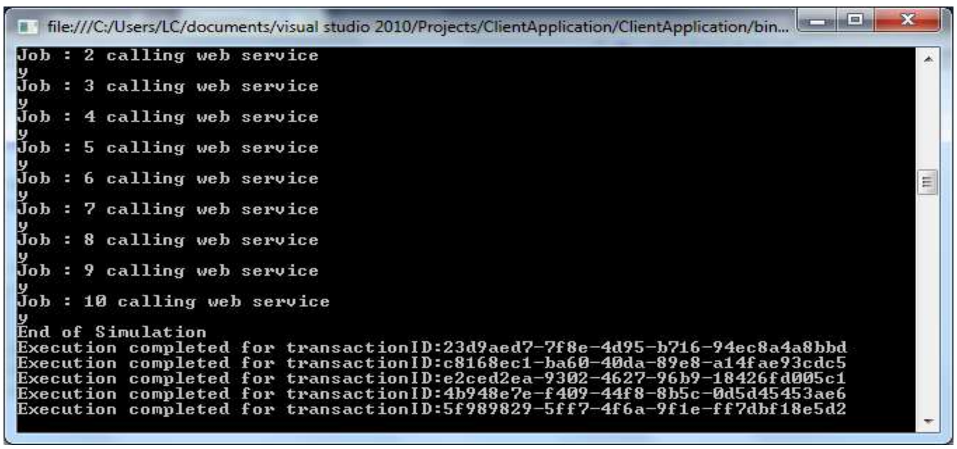

Figure 3: Client application showing processing of web service calls

The client application is used to test the capacity of the system. We will run four tests.

Test 1: A single client with a low input rate

The test is designed to examine the response time in an unloaded system, so we study the effect that the complexity of the transactions (i.e., placing several orders) has on application response time. The use of a single client prevents the accumulation of transactions in the system so that it can be used as a baseline estimate of the total server time needed to received transactions request (i.e., orders whose OrderID's are a random number between 1 and 1000000) and deliver the asynchronous response back to the client. The parameters used for this test are shown in Table 2.

Table 2. Parameters used for this test (single client with a low input rate)

Scale	1000000
Count	10
Gdelay	333
Threads	1
Tdelay	10

Journal, Advances in Mathematical & Computational Sciences
Vol. 8 No. 1, March 2020
www.mathematics-computationaljournal.info

This test creates a single client thread on the system and then places 10 orders for processing. The orders to be processed are assigned unique OrderID(or OrderNo) which are numbers generated randomly between 1 and 1000000. The client places orders and then waits for 333 milliseconds before placing the next order. The use of a single client prevents the accumulation of transactions on queues on the system so it can be used as a baseline estimate of the total server time needed to receive and process orders. Since this test creates a single client on the system, the Tdelay parameter (i.e., the interval between client starts) has no significant effect.

Test 2: Single Client with a high input rate.
This test is run to study the effect that several transactions have on the response time in an unloaded system. The parameter used for this test is summarized below:

Table 3. Parameters used for this test (single client with a high input rate)

Scale	1000000
Count	100
Gdelay	333
Threads	1
Tdelay	10

Test 3: Multiple clients with a low input rate.
This test is run to overload the system and study the effect that low input rate (a sizable number of transactions) have on the response time in an overloaded system. The parameter used for this test is summarized below:

Table 4. Parameters used for this test (single client with a high input rate)

Scale	1000000
Count	50
Gdelay	333
Threads	5
Tdelay	10

Test 4: Multiple clients with a high input rate.
This test is run to overload the system and is designed to measure the server throughput, which is the maximum number of transactions a server can complete per unit of time.

A test like this is designed to slowly ramp up the number of clients and slowly increase the transaction rate to reveal the point at which the response time degrades. The parameter used for this test is summarized on the next page:

Table 5. Parameters used for this test (single client with a high input rate)

Scale	1000000
Count	100
Gdelay	333
Threads	10
Tdelay	30

This test will create a total of 10 client threads, each behaving like the single client in the last simulation. The threads are started at 30 seconds intervals to slowly increase the load on the system so we can test the point at which response times degrade. Since multiple clients are submitting a request, a backlog will accumulate in the place order details queue when the server application is busy. This simulation may take some time (say 5 minutes) to complete depending on the configuration of the system. To be on the same level, we will observe the system for 30 minutes for all the experiments. Each client request is delayed for about a third of a second before sending another request.

Step 4: Starting the server application
1. Start the application/service that represents the Application Server. The web server must be up and running since this is a web application. Process the orders so that the orders that were queued in the ***placeorderdetails*** queue will be retrieved and processed. The basic minimum, the server report should contain the following details: whether or not the message has been received, TransactionID, CustomerNo, OrderNo, ProductNo, Quantity, Order status, Due date, and the status of the message.

Step 5: Starting the Metrics Application (metricReport)
When the client finishes, start a Metrics Application (**the application that displays metric report)** on the server and let it run until it has logged all the queued metrics reports into the Metrics database.

1. Start the metrics report application
2. The MetricReport will display metrics report after it retrieves metrics data from one of more database and stores in a central repository. A sample metrics report shows the transaction and client IDs, the transaction start time, its elapsed time in milliseconds, and the attributes the client assigned to the transaction.

5. RESULTS AND DISCUSSION

There are four scenarios that we will consider. For each scenario, we will conduct several experiments where we vary one of the independent parameters while keeping the rest constant. The scenarios are as follows:

Scenario 1: A single client with a low input rate
Scenario 2: A single client with a high input rate.
Scenario 3: Multiple clients with a low input rate.
Scenario 4: Multiple clients with a high input rate.

The input parameters for the experiments are - number of clients, the number of transactions/requests for each client, the duration of delay before starting each client, and the duration of delay before each client makes the next transaction or request.
The following assumptions have been made in the experiment:

i. In the graph that we will present later, each data point is the average of seven simulations runs, where the length of time in each run is equivalent to the total busy time of the system (that is, the time interval between the first client request and the last client response).
ii. The length of time in the observation period is 30 minutes (or 1800 seconds). In some cases, it may be very close or almost equal to the total busy time, which will make the system utilization to be 1(100%).
iii. The metrics – response time, throughput, concurrency level is measured only for the successful request (that is request whose responses have been received at the client).

iv. We assume that each request/transaction is executed asynchronously.
v. (V) The tests described in this research were conducted in a controlled environment; so the numbers presented here may not match the results that you get when you run the tests in your environment.
vi. The system simulated is a multi-tier application, closed model, used to model QoS. The different web services are also modeled as resources (or service centers) in the system. Multiple resources are present in the system.
vii. Since the simulation period is fixed (30 minutes) and so arrivals and completions may be lost before receiving server and after receiving service.
viii. The total number of service completions from the resource is equal to requests completed by the system. That is, $C_i = C_i$
ix. The simulation takes place in three phases as follows:

Phase 1 – client application calls a web service to place orders in a queue.

Phase 2 – server application retrieves orders from the queue and calls web services to processes it. the result is stored in queues and/or databases.

Phase 3 – metric application retrieves the results and analyses it.

5.1 Analyzing Experimental Results: Multiple Clients with Low Input Rate
The experimental result presented in this section conform to the scenario where there is multiple clients threads with a high input rate. Apart from this, this experimental result also conforms to the first phase of the simulation process where the client application invokes a web service.

This web service invocation is simply a request to place orders for a product. These requests might be successful or not successful depending on the multiprogramming level (concurrency) and on the service demand of the resource (in this case, the web service places the order). Successful request is kept in the queue for server application to process later. In this section we are going to present the measured quantities and derived quantities.

Table 6. Measured quantities and derived quantities.

Exp	Input Parameters				Measured Quantity					
	C_l	R_q	D_{cln}	D_{req}	A_0	A_i	C_0	E_{time}	N_{ord}	T_{sev}
1	1	50	5	50	50	50	21	5826	156	571
2	2	100	10	100	200	77	30	8632	251	969
3	3	150	15	150	450	79	32	15157	254	994
4	4	200	20	200	800	78	31	11960	251	1301
5	5	250	25	250	1250	85	38	26289	274	1421
6	6	300	30	300	1800	73	26	10663	230	1151
7	7	350	35	350	2450	98	51	23949	304	1109
8	8	400	40	400	3200	76	29	8846	246	1111
9	9	450	45	450	4050	79	32	18223	241	1286
10	10	500	50	500	5000	76	29	13712	239	1336

The above data was collected based on a randomly set number of orders that can be placed by each request at the server. Our analysis shows that one request cannot process more than four orders. Therefore, those requests that contain more than 4 orders are not completed and therefore are not sent back to the client application. This accounts for the reduction in the number of request responses that are received by the client. The derived quantities are presented in the table below.

The table below shows all the operational variables that have been computed. The computation was done automatically by exporting the data recorded in MS Access into MS Excel 2007. We can carry out operational analysis (analyze the relationship between operational laws and the operational variables) and bounding analysis (bounds on throughput and response time) on the computed operational variables.

Table 7. Computed operational variables

Exp	S_i	U_i	X_i	R_0	$\overline{N}$	D_i	Invocation Rate
1	27.2	0.32	0.02	116.52	3.24	16	0.09
2	32.3	0.54	0.02	112.11	4.8	27	0.21
3	31.07	0.56	0.02	191.87	8.43	28	0.46
4	41.97	0.73	0.02	153.34	6.65	36.5	0.62
5	37.4	0.79	0.03	309.29	14.61	26.34	0.88
6	44.27	0.64	0.02	146.07	5.93	32	1.57
7	21.75	0.62	0.03	244.38	13.31	20.67	2.21
8	38.32	0.62	0.02	116.4	4.92	31	2.89
9	40.19	0.72	0.02	230.68	10.13	36	3.15
10	46.07	0.75	0.02	180.43	7.62	37.5	3.75

Further analysis can be done in MS Excel by plotting graphs and charts to study the relationships that exist between the operations variables.

5.2 Performing Bounding Analysis

In our simulation, we assume that there is one resource (i.e., server) and so we compute only one value for the service demand. Although we can calculate service demand in different ways, we rely on the principle that the service demand is the total average service time of a request at a resource. In our simulation, this is simply equal to the mean service of the resource.

Service Demand of the system (D) = 27.2

X = 1/D for the system
X = 1/27.2
X ≤ 0.037

Maximum throughput = 0.037

The upper bounds on throughput can be obtained using equation (4.6) and (4.7) as follows:

$$X_0 \leq \min\left[\frac{1}{\max\{D_i\}}, \frac{N}{\sum_{i=0}^{K} D_i}\right]$$

X ≤ min [0.037, N/27.2]

The lower bounds for the response time can be obtained as follows

$R \leq \max\left[N \times \max\{Di\}, \sum_{i=0}^{K} D_i\right]$
R ≥ max [N x 27.2, 27.2]

5.3 A Model for Predicting the Number of Requests Completed By The System
In this section, we build a model for predicting the number of responses that are sent to the client based on the number of requests that the server (i.e., resource) receives. This is shown in the graph below. To do this we plot a graph as shown below. The data points obtained in the simulation experiment are shown in the graph by the smooth line and marker. A trend line is added to these points using MS Excel 2007 (i.e., by right-clicking on the dashed line and selecting Add Trend Line). A polynomial trend line is selected because visual inspection indicates a polynomial relationship between the number of requests that arrives at the server and the number of responses received at the client. The linear regression performed generates the following relationship:

$Y = 0.014X^2\ 1.45X + 58.09$

The R^2 value (coefficient of determination provided by MS Excel 2007) obtained is 0.992. This value is very close to 1, which means that the regression line adequately models the observed data.

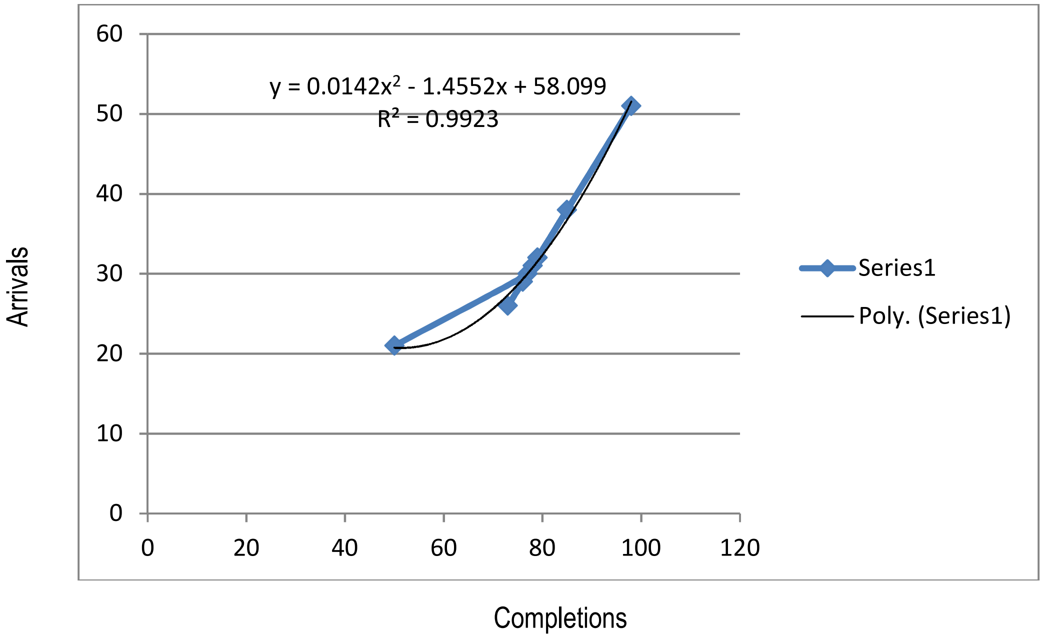

Figure 3: Regression line for predicting the number of requests completed by the system

5.4 Applying the Model to Predict the number of responses sent to the client

Now that we have a model, we can use it to predict the number of responses that are sent to the client based on the number of requests that the server (i.e., resource) receives. Assuming that the server receives 40 requests (that web services), then we can predict the number of responses that are sent to the client as follows:

That is if, X = 40 requests, then Y = 1600 (0.014) - 1.455 (40) + 58.09

Y = 22.4 – 58.2 + 58.09 = 22.29 responses (sent to the client).

Again, if X = 67 requests, then Y = 4489 (0.014) - 1.455(67) + 58.09

Y = 62.846 – 97.485 + 58.09 = 23.451 responses (sent to the client).

6. CONCLUSION AND FUTURE WORK

This paper presents a simulation model to obtain more information about a typical service-oriented system. We have designed and implemented a service-oriented system. The sample system used in the study is a simple purchase-order system (a subsystem of a procurement system) composed of components (i.e., web services) built-in .NET platform. The measurement process involved in gathering data from service-oriented systems has also been presented. Specifically, we explained how to specify the measurement data, instrument the system and gather the specified variables, and analyze and transform the measured data into input parameters required for modelling.

In future, we plan to integrate a multitenancy component into the architecture presented in this paper, and then do a comparative analysis with existing multitenancy architectures to evaluate how multitenancy components affect transactional and security support for interoperable SOA-based applications. This will be especially useful in cloud environments where resources sharing is promoted, while at the same time ensuring that there is isolation between two or more components of the system or one or more tenants accessing the system (Ochei et al, 2019).

REFERENCES

1. Alrifai M., Dolog P., Balke W., Nejdi W. (2009): Distributed Management of Concurrent Web ServiceTransactions. IEEE Transactions on Services Computing, vol. 2, no. 4, pp. 289-302, October-December, 2009.
2. Shan-liang, P., Ya-Li, L., & Wen-juan, L. (2011). A framework for ensuring consistency of Web Services Transactions based on WS-BPEL. International Journal of Modern Education and Computer Science, 3(4), 47.
3. Chen, H.(2008).Transaction Management Issues in Web Service-Oriented Electronic Commerce Systems: Performance Evaluation. Published by SAGE. Simulation. Retrieved on August 29,2020 from http://sim.sagepub.com/cgi/content/abstract//84/6/263
4. Choi, S; Kim, H, Jang H; Kim, J; Su Myeon Kim Su M; Junehwa Song, J; Yoon-Joon Lee(2008): A framework for ensuring consistency of Web Services Transactions. Information and Software Technology 50 (2008) 684–696. Available online at www.sciencedirect.com
5. Gabhart K. (2004): Java/.NET Interoperability via Shared Databases and Enterprise Messaging. Retrieved on January 31, 2011 from http://www.devx.com/interop/Article/19952/0/page/2.
6. Garcia-Molina, H., Salem, K.(1987). Sagas. *Proceedings of the ACM SIGMOD Conference*, San Francisco, CA, 1987, pp. 249-259
7. K. Haller, H. Schuldt, and C. Türker. Decentralized coordination of transactional processes in peer to peer environments. ACM Press, in Proc. of the 14th ACM Intl. Conference on Information and Knowledge Management (CIKM 2005), pages 36--43, Bremen, Germany, Nov. 2005.
8. Kounev, S and Buchmann, A.(2003):Improving Data Access of J2EE Applications by Exploiting Asynchronous Messaging and Caching Services
9. Kounev,S., Huber, N, Spinner, S., Brosig, F.(2006): Model-based Techniques for Performance Engineering of Business Information Systems
10. Ochei, L.C., Ogunsakin, R., Wobidi, Echebiri (2020): Architectural Framework for Improving QoS of Service for Interoperable Service-Oriented systems. Computing, Information Systems, Development Informatics & Allied Research Journal. Vol 11 No 2, Pp 125-144. Available online at www.isteams.net.cisdijournal

11. Menasce, D., Almeida V., Dowdy, L. (2004). Performance By Design: Computer CapacityPlanning by Example. Pearson Education, Inc. New Jersey, USA.
12. Gabhart K. (2004): Java/.NET Interoperability via Shared Databases and Enterprise Messaging. Retrieved on January 31, 2011 from http://www.devx.com/interop/Article/19952/0/page/2.
13. Laudati P.; Loeffler W: David Aiken, Arkitec, Keith Organ, Arkitec, Anthony Steven, Mike Preradovic, Wayne Citrin, Peter Clift,(2003): Application Interoperability: Microsoft .NET and J2EE. Microsoft Corporation.
14. Ochei, L. C., Petrovski, A., & Bass, J. M. (2019). Optimal deployment of components of cloud-hosted application for guaranteeing multitenancy isolation. *Journal of Cloud Computing*, *8*(1), 1-38.
15. Hillston, J.(2010). Performance Modeling. Lecture notes on performance Modeling. School of Informatics, The University of Edinburgh, Scotland, UK. Retrieved on July 31, 2012 from http://www.inf.ed.ac.uk/teaching/courses/pm/