Digital Innovations & Contemporary Research in SCIENCE, ENGINEERING & TECHNOLOGY
Vol. 8. No. 2, 2020

Article Citation Format
Ochei, L.C. & Ogunsanki, R. (2020): Measuring the Performance of Interoperable Service-Oriented Systems Journal of Digital Innovations & Contemp Res. In Science., Engineering & Technology. Vol. 8, No. 2. Pp 129-140

Article Progress Time Stamps
Article Type: Research Article
Manuscript Received: 11th March, 2020
Review Type: Blind Final
Acceptance: 12th June, 2020
CrossREF Member Listing :: ttps://www.crossref.org/06members/50go-live.html

Measuring the Performance of Interoperable Service-Oriented Systems

Ochei, L.C. & Ogunsanki, R.
Department of Computer Science
University of Port Harcourt
Port Harcourt, River State, Nigeria.
E-mail: laud.ochei@gmail.com; rotimi.ogunsakin@gmail.com

ABSTRACT

Businesses are increasingly facing the challenges of developing SOA-based systems that solve interoperability problems or developing systems that do not suffer from interoperability problems. The scalability of interoperable systems implies that the solution can be deployed on a large configuration. However, because an interoperable system may be built using different programming languages which may likely run on different hardware machines, a consistent system performance measurement approach is required for analyzing several performance enhancements design options. This paper presents a novel approach, client-oriented recording, for collecting and measuring the performance of the interoperable SOA-based system. The approach has been applied to a simple procurement system, and an extensive evaluation shows that the approach is effective in collecting data and measuring the performance of an interoperable SOA-based system without degrading the system's performance.

Keywords: Service-oriented systems, Interoperability, Measurement, Performance, QoS, scalable

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 options for measuring the performance of an interoperable SOA-based system
(ii) presenting an approach for measuring the performance of an interoperable SOA-based system
(iii) extensive evalution of the approach on a sample procurement ordering system

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

2. APPROACHES FOR MEASURING THE PERFORMANCE OF INTEROPERABLE SOA-BASED SYSTEMS

There are several strategies for measuring the performance of applications running on widely used programming languages like Java, python, an .NET/C#. applications. There are timin functions provided in the

2.1 Using a Timer Function

This entails adding a timer function in a business applicaton, for example, a cliwnt application tha invokes a services to palce an order) before and after the functions clllas. This timer function ca then be used to measure rhe total response time required to run an interoperability operation. Listing 1 shows an example of how to add a timer function in a Java application to measure the response time for performing a database query.

Listing 1. Java program segment to measure system performance

```
1  package test;
2  import java.io.FileInputStream;
3  import java.io.FileNotFoundException;
4  import java.io.InputStream;
5  import java.sql.Connection;
6  import java.sql.DriverManager;
7  import java.sql.PreparedStatement;
8  import java.sql.SQLException;
9  import java.text.ParseException;
10
11 public class Test {
12     public static void main(String[] args) {
13     public static void main(String[] args) throws ParseException,  SQLException,FileNotFoundException
14 {
15             long startTime = System.nanoTime();
16             // method to insert data into a database
17             methodToTime();   //Measure execution time for this method
18             long endTime = System.nanoTime();
19             //Total execution time in nano seconds
20             long durationInNano = (endTime - startTime);
21             System.out.println("The execution time for this query is : " + durationInNano);
22  }
23
```

2.2 Using a testing software package

This entails using a software testing tools to measure he end to end, total response time and performance . There are so may software testing tools such as JMeter, Empirix, Mercury LocadRunner, Selenium can be used to automate the measurement of applications. For example, JMeter, a Apache , a java-based application can be used as a load testing tool for analyzing and measuring the performance of a variety of services and applications. JMeter can be used as a unit-test tool for JDBC database connections, FTP, LDAP, web services, JMS, HTTP, generic TCP connections and OS-native processes.

2.3 Using external testing infrastructure

This entails using testing company to measure the en-end performance in a decicated netweeok. Or public internet. This would be the case when applications are hosted on the cloud. He cloud provider csn be approached to measure the performance of the application running on the r cloud platform. In orde to use this approach for measuring the performance f an interoperable interoperable SOA-based, a central repository has to be created to store the time statistics. The common eature in the above approach is that the measuremrnt is taken for individual applications. However, these approaches have limitation if used to measure the performance of interooersble SOA-bases applications because of the distributed nature of the compoents of the system. The components of the system may have been built using different programming langages, deployment platforms, and store data in didfferent repositories. A beter approach which is presented in this paper is to measure the end-to-end performance of the transactional process. Consider a scenario where a client application is used to check if an item is available before proceeding to place an order that is eventually store on a server application. In this scenario, a timer would be place before the call to check item available is made and then after the the order has been successfully saved in the repository.

3. ARCHITECTURE OF AN INTEROPERABLE SOA-BASED SYSTEM FOR MEASUREMENT

A high level model of the simulated system is captured in Figure 4.14. In our simulation, the workload is specified by the number of concurrent request in execution and not by an arrival rate. In this situation, a Closed Multiclass Queuing Network model is used.

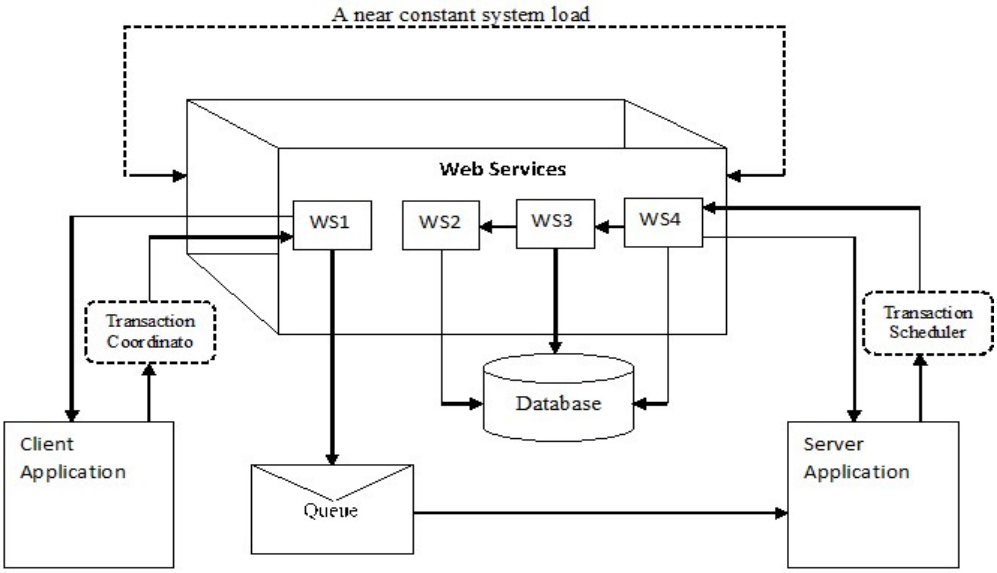

Figure 3; Architecture of the System used for measurement

4. EVALUATION

In this section we present how to measure the performance of the It is assumed that an interoperable SOA-based system has been developed before the measurement of the performance can be done. In our previous work, we presented an architecture of an interoperable SOA-based system and also a framework for design and implementation of an interoparbale 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 for our modelling the system to improve transactional support.

In this section, we present the experimental settings and procedure

4.1 Measuring the Performance of an Interoperable SOA-based System

In this section we present how to measure the performance of an interoperable SOA-based system. It is assumed that an interoperable SOA-based system has been developed before the measurement can be done. In our previous work, we presented an architecture of an interoperable SOA-based system and also a framework for design and implementation of an interoparbale 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 for our modelling the system to improve transactional support. The measurement process presented in this paper translates into three steps: specify measurement data, instrument and gather data; analyze and transform data. These steps are summaries below:

4.1.1 Specify Measurement

After observing the simulated system for a while, we have to select performance (operational) variables to be measured. A partial list of such measured quantities includes the following:

1. Length of time in the observation period
2. Number of resources in the system
3. The total busy time of a resource in the observed period
4. The total number of service request to a resource in the observation period
5. The number of request submitted to the system in the observed period
6. The total number of service completions from a resource in the observation period
7. The total number of request completed by the system in the observed period
8. Number of transactions passed/failed: This parameter simply shows the total number of transactions passed or failed.

4.1.2 Instrument and Gather Data:

After selecting the variables to measure, the system is instrumented to gather the specified measurement data. This entails setting up the simulated system in such a way that will allow us to measure and record the specified variables during the observation period. Existing approaches for obtaining information for measurements rely data collected from direct observation of the system during simulation runs with the sample program. These data will be recorded in the database and then pulled from there to do analysis and evaluation. This approach can affect the performance of the system due to overhead incurred by exchanging messages and data between the server and clients components of the system during transactions. Another approach for collecting data is to capture metrics solely based on Web server HTTP logs. The challenge is that organizations that relies solely on Weblogs will lack a clear picture of the true quality of service they offer to their clients. Also, it is very difficult to measure and report operations from IIS logs as it is usually a complicated and expensive process.

We will use a **client-oriented recording approach** to capture metrics that will be used to evaluate the performance of the system. The Client-Oriented Metric Recording approach simply entails designing a system in such a way that the client application itself will record metrics about the operations it generates. When a user initiates a set of these operations, the client can record the start time and the elapsed time before results are presented to the user. Figure 2 shows the architectural design for gathering and analyzing client metrics. Before a web service is invoked to carry any task, useful client-related metrics such as client start time, end time, transaction status, number of transactions and any other business data that will aid product support and capacity planning are recorded and then placed on a message queue for low-priority processors. This is done each time a new transaction is initiated by the user. A separate application (metrics application) retrieves, organizes and analyses these metrics details from one or more databases into useful information to allow general-purpose transaction reporting. IIS logs are no longer needed. The client-recording components and data flow are shown in red.

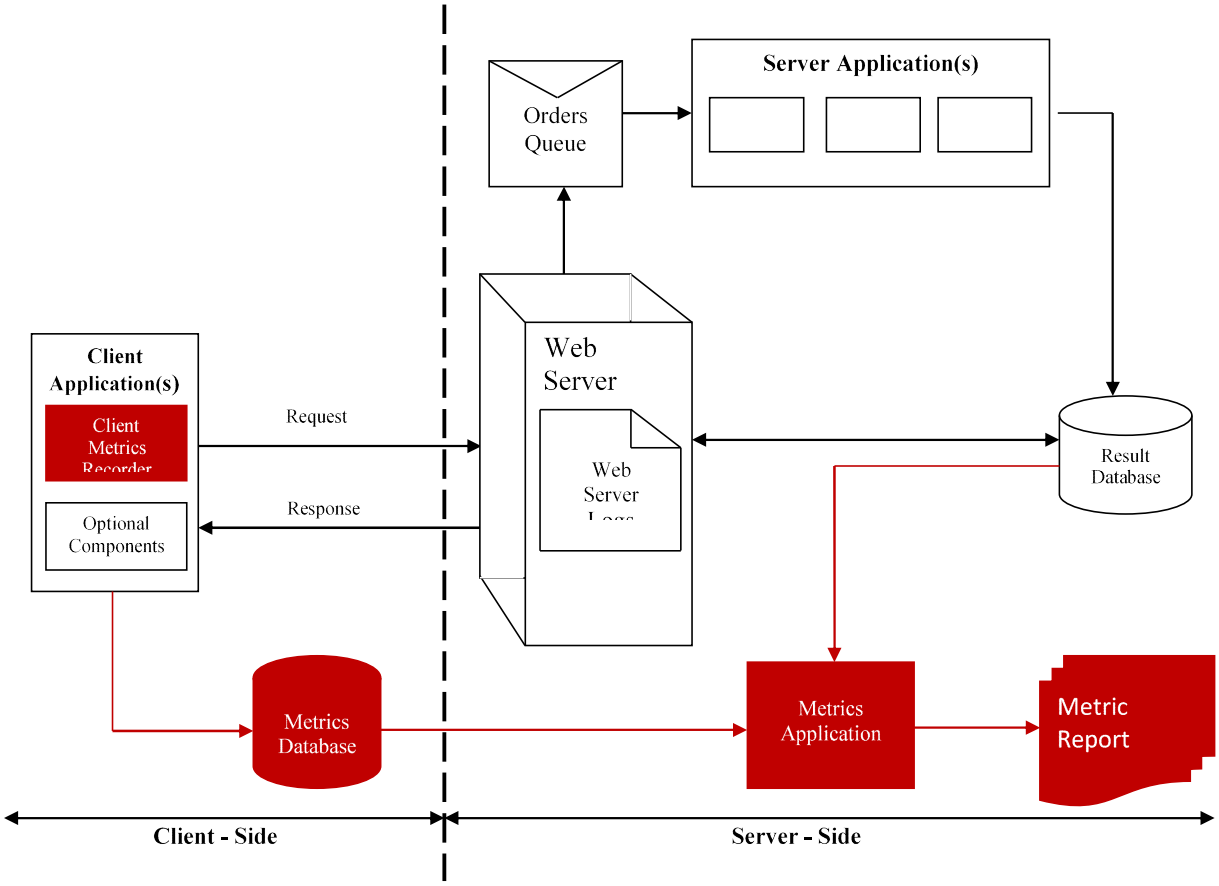

Figure 2: Web Services Transaction Architecture with Metrics (Connolly, 2004)

Most platforms (Java EE, .NET and PHP) have timing functions. We will aggregate the timing statistics from these basic timing functions for measurement. During implementation, we will add a time function in the application before and after function calls. This timer function can be used to measure the total response time required to run a transaction in a distributed environment. When measuring the performance of the framework, we will not measure individual applications (or service); rather we will measure the end-to-end application performance (such as a round trip from the .NET client (service) to the Java or PHP application server (service).

For example, in the distributed transaction scenario (i.e., the procurement system), we will customize a timer function to measure from the creation of a purchase order (from the .NET client or service) to the completion of generation of a new product request (return the processing result from the Java EE application server). This client-metrics recording approach will make it easy for us to measure the overall transaction system capacity against its key performance target: client response time.

4.1.3 Analyze and Transform Data

These raw data have to analyzed and transformed into input parameters. Typical measurement data do not coincide directly with the input parameters required for performance modelling. For modelling purposes, typical measurement data need to be analyzed and transformed to become useful (Manase, 2004). A set of derived quantities from the known measurable quantities can be obtained. A partial list includes the following:

1. 1. Response time
2. Throughput
3. Resource utilization
4. Service demand
5. Arrival rate
6. Multiprogramming level (concurrency level)
7. System Load - this parameter refers to the number of concurrent virtual users trying to access the Web Service at any particular instance in an interval of time.
8. Wait Time (Average Latency) - this parameter refers to the time it takes from when a request is sent until the first byte is received.
9. Mean of Missing Rate (MMR) - defined as the ratio of aborts over the total number of transactions was computed. That is, MMR = Na/Nt.

4.2 Obtaining Input Parameters For Modeling

The representativeness and accuracy of any modeling process depends directly on the quality of its input parameters (Manese, 2004). In this section, we will look at how to gather, analyze and transform measurement data into input parameters which will be used for modeling the quality of service for interoperable service-oriented systems. The input parameters used for the simulation experiments are – number of clients, number of transactions/request for each client, the duration (length) of delay before starting each client and the duration (length) of delay before each client makes the next transaction or request. Thereafter we will also present a sample input data used for two simulation runs, and measured and derived quantities taken from the simulation.

Table 1: Description of input parameters used for the simulation experiments.

SN	Parameters	Description	Sample data
1	S_o	Size of the maximum OrderNo to use for place an order	1000000
2	C_l	Number of clients to start	1
3	D_{clm}	The duration(length) of delay before starting each client	10
4	R_q	Number of orders(i.e., request) that each client can place	10
5	D_{req}	The duration (length) of delay before each client makes the next transaction or request.	333

We assume that **a request refers** to asynchronous web service calls while **a response refers** to a completed request received from the server.

Table 4.9. Description of measured quantities from the simulation experiments

SN	Notation	Description
1	T	Length of time in the observation period (equal to total busy time of the system)
2	K	The number of resources
3	B_i	Total busy time of the server in the observation period
4	A_i	Total number of service request(arrivals) to the server
5	A_0	Total number of request submitted to the system in the observation period
6	C_i	Total number of service completions from the server in the observation period
7	C_0	Total number of requests completed by the system in the observation period
8	N_{ord}	Total number of orders placed
9	E_{time}	Total response (elapsed)time of a request at the system
10	E_{sev}	The processing (service) time of a request at the server
11	T_{req}	Time interval all request were sent
12	T_{sev}	Time interval all request arrived at the server
13	T_{res}	Time interval all responses were received
14	T_{ord}	Time interval all orders were placed

All asynchronous requests processed by the server are captured and saved in a database before it returned to the client. After simulation, we simply count the number of requests in the database that have placed the complete number of orders that is required. The number of orders to place by each request is generated randomly before attempting to place the orders.

From the above known measured quantities, the following set of derived quantities (input parameters for analytical modelling) can be obtained.

Table 3. Description of the derived quantities

SN	Notation	Derived Quantity	Equation
1	S_i	Average service time per completion	B_i/C_i
2	U_i	Utilization of the resource(server)	B_i/T
3	X_i	Throughput (i.e., completions per unit time) of the server. This is also called the completion rate	C_i/T
4	X_0	System throughput	C_0/T
5	λ_{clt}	Invocation rate(i.e., invocations per unit time) at the client	A_0/T_{req}
6	λ_{sev}	Arrival rate (i.e., arrivals per unit time) at the server. Also called input rate or system load.	A_i/T_{sev}
7	V_i	Average number of visits(i.e., visit counts) per request to the server	C_i/C_0
8	R	Average Response time at the server	$R = \frac{\sum_{i=1}^{n} E_{time}}{A_i}$
8	R	Average Response time of the system	$R = \frac{\sum_{i=1}^{n} E_{time}}{A_0}$
9	$\overline{N}$	Concurrency level (Multiprogramming level).	$\overline{N} = \frac{\sum_{i=1}^{n} E_{time}}{T}$ $\overline{N} = \lambda_{sev} R$
10	D_i	Service Demand	$D_i = U_i/X_0$ $D_i = V_i \times S_i$

5. EVALUATION

This section presents how to evaluate the system using the measrurments obtained. the model .

5.1 Computing the Derived Quantities from the Measured Quanties

The derived quantities can be calculated from the measured quantities. The values are presented below based the analytical formulas presented in the Section 3.

Table 2. Input paramters for analytical modelling

Input paramaters	Experiment 1	Experiment 2
C_i	8	8
R_q	400	400
D_{oin}	0	0
D_{req}	400	0

Table 3. Sample measured quantities

Measured parameters	Experiment 1	Experiment 2
A_0	3200	3200
A_i	91	87
C_i	44	40
C_0	44	40
N_{ord}	283	268
E_{time}	16905	21584
B_i	1168	1550
T	1800	1800
T_{req}	180	100
T_{sev}	1165	1519
T_{res}	1109	1519
T_{ord}	1167	1549

Table 5. Sample derived quantities

Derived Quantities	Experiment 1	Experiment 2
S_i	26.52	38.75
U_i	0.649	0.861
X_i	0.024	0.022
X_0	0.024	0.022
V_i	1	1
λ_{clt}	17.78	32
λ_{sev}	0.078	0.048
R_i	185.77	248.09
R_0	5.28	6.754
$\overline{N}$	9.39	11.99
D_i	27.04	39.14

5.1.1 Computing Bounds on Performance

Service Demand of the system (D) = Average Service time = 26.52

Since utilization is 0.649, then

X = U/D ≤ 1/D for the system

X = 0.649/26.52 ≤ 1/26.52

X = 0.024 ≤ 0.037

Maximum throughput = 0.037

The upper bounds on throughput can be obtained as follows:

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

X ≤ min [0.037, N/27.04]

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

$$R \leq \max\left[N \times \max\{D_i\}, \sum_{i=0}^{K} D_i\right] \qquad \text{Equation (2)}$$

R ≥ max [N x 27.04, 27.04]

5.1.2 Information Obtained from the Simulation Model

In the first stage of the modelling process, we assumed that the system is abstract. We went on to perform operational analysis and bounding analysis. We have now gathered a reasonable amount of information to enable us to understand the system we want to model. A partial list includes the following:

1. Selecting the data to measure.
2. Configuring the system to measure and record the data.
3. Computing derived quantities from the measured data such as response time, throughput etc.
4. Carrying out operational analysis and bounding analysis on the measured data.
5. Knowing workload class and the parameters to use in specifying the workload class. The simulated system fits into an interactive workload where requests are submitted by a fixed number of clients. The workload intensity of the system is specified by the system load (and maybe also by the length of the delay before invoking each client and before each client makes a request) and not by concurrency level or arrival rate. System load is computed as the number of requests sent to the system divided by the time interval between the first request and the last request.
6. Choosing the type of Queuing network model to use. A closed multiclass queueing network model is appropriate. There could be a single or multiple queues and/or several servers.
7. Investigating the important relationship between system load and average response. It resembles an exponential curve/distribution. The relationship between the concurrency level and system load is approximately sinusoidal. In a closed model, when the number of client threads and the number of requests for each client increases in a constant rate the relationship between the concurrency level and the system load is approximately sinusoidal.

8. Identifying basic Queuing Network, Markov model, and Regression analysis as tools to model the system for improving transactional and security support
9. Investigating the limitations of using throughput metrics for analyzing the behavior and performance of the interactive workloads. Throughput metrics measures system performance for repetitive, synchronous sequences of request. The results of these benchmarks do not correlate directly with user-perceived performance. The performance of e-commerce applications depends on the speed at which the system can respond to an asynchronous stream of independent and diverse events that result from interactive user input. Response time (latency), not throughput is the key performance metric for interactive systems.

6. CONCLUSION AND FUTURE WORK

This paper presents a simulation model (that is, a sample program) 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 measure 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 prmoted 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.

REFERENCES

1. Abundo, M., Cardellini , V., Presti, F.(2020). Admission Control Policies for a Multi-class QoS-aware Service Oriented Architecture. Retrieved on November 5, 2020 from http://dl.acm.org/citation.cfm?id=2185445
2. Albahari, J.(2006): Threading in C#. O'Reilly Media, Inc. Retrieved on August 24/08/2020 fromwww.albahari.com/threading/
3. 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.
4. Cabrera L,F; Copeland, G; Feingold, M; Freund, R.W; Freund, T; Johnson, J; Sean Joyce, S; Kaler, Chris; Klein, J; Langworthy, D; Little, M; Nadalin, A; Newcomer E; Orchard, D; Ian Robinson, I; Shewchuk, J; Tony Storey, T.(2003): Web services composite application framework (ws-caf). Retrieved on October 31, 2020 from http://developers.sun.com/techtopics/webservices/wscaf.
5. Casado, R., Tuya, J., Younas, M.(2011). A framework to test advanced web service transactions. IEEE International Conference on Software Testing, Verification and Validation.
6. Chan,P., Lyu, M., Malek, M.(2006): Reliable Web Services: Methodology, Experiment and Modeling
7. 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
8. 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
9. Endo, Y., Wang, Z., Chen, J., Seltzer, M. (2020): Using Latency to Evaluate Interactive System Performance. Retrieved on 12 October, 2020 from static.usenix.org/publications/library/proceedings/osdi96/full_papers/endo/endo.ps
10. Erl, Thomas(2009). SOA Design Patterns. Pearson Education, Inc. New Jersey, USA.
11. 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.
12. Garcia-Molina, H., Salem, K.(1987). Sagas. *Proceedings of the ACM SIGMOD Conference*, SanFrancisco, CA, 1987, pp. 249-259
13. 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.
14. Kamra, A., Misra, V., Nahum, E.(2004). Controlling the performance of 3-tiered web sites; Modeling, Design, and Implementation. SIGMETRICS/ Performance '04 June 12-16, 2004, New York, NY, USA. ACM 1-58113-664-1/04/0006
15. Kounev, S and Buchmann, A.(2003): Performance Modeling And Evaluation Of Large-Scale J2EE Applications
16. Kounev, S and Buchmann, A.(2003):Improving Data Access of J2EE Applications by Exploiting Asynchronous Messaging and Caching Services
17. Kounev, S., Bender, K., Brosig, F., Huber, N.(2010): Automated Simulation-Based Capacity Planning for Enterprise Data Fabrics.
18. Kounev,S., Huber, N, Spinner, S., Brosig, F.(2006): Model-based Techniques for Performance Engineering of Business Information Systems
19. Schroeder, Bianca; Harchol-Balter, Mor; Wierman, Adam; Iyengar, Arun; and Nahum, Erich(2006): How to Determine a Good Multi-Programming Level for External Scheduling. Computer Science Department. Paper 878. Retreived on September 19 from http://repository.cmu.edu/compsci/878

20. Malrait, L, Marchnad, N., Bouchenak, S.(2009). Average Delay Guarantee in Server Systems Using Admission Control.
21. Menasce, D., Almeida V., Dowdy, L. (2004). Performance By Design: Computer CapacityPlanning by Example. Pearson Education, Inc. New Jersey, USA.
22. 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.