International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 12, No.1 (Oct-2022)

http://dx.doi.org/10.12785/ijcds/120199

Assessing the Performability of a Fault Tolerant Architecture for Web Services Solution using Software Fault Injection

Festus O. Oliha[1]

[1]*Department of Computer Science, University of Benin, Nigeria*

Received 22 Jan. 2022, Revised 15 Jul. 2022, Accepted 23 Jul. 2022, Published 31 Oct. 2022

Abstract: The presence of fault(s) in any web services-based system threatens its performability and dependability at large. To ensure dependability in the presence of faults or failure, service-oriented systems are implemented on scalable fault-tolerant architectures. However, the performability of these service systems under the influence of fault is vital for assessment to determine the behavioural impact on service provision and delivery. In this study, the performability of a fault-tolerant architecture implemented on software agent's capabilities has been assessed via a compile-time fault injection technique under a replica-fault load in terms of service time efficiency, response stability, and computational effort. The assessment and findings empirically established that the impact of software fault injection is no degradation to the performability of the architecture solution. That is, the architecture performability under a fault load is significantly time-efficient in service delivery with guaranteed response stability at a reduced computational overhead (high throughput) compared to without fault injection. The recognition of the evaluation, results, analysis, and emphasis will serve as a veritable tool in increasing the opportunity of building web services solutions on a fault-tolerant architecture with appreciable performability for service-oriented communities.

Keywords: : Fault Tolerance, Performance, Fault Injection, Fault-tolerant Architecture, Software Agents, Web Services, Performability

1. Introduction

The cutting-edge standards in building quality and dependable service-oriented systems are web services, fault tolerance, performance, reliability, and scalability [1], [2], [3], [4]. Service-oriented systems (service systems) are highly dependent on web services with fault-tolerant capabilities to ensure robustness in both business logic and persistent service provisions. K. Farj et al. [5] conveyed that "one of the obstacles of the adoption of the Web service paradigm in such composed systems is the problem of assessing their overall service quality, since services are inherently distributed, heterogeneous, and are often invoked with little understanding of their reliability and performance". That is, web services are usually distributed over the Internet without the guarantee that all parts of the service are highly reliable, thus, SOAP-based solutions are particularly associated with tolerance and performance overheads in the presence of faults [6].

Fault tolerance and performance are critical attributes for service solutions, but ascertaining their performability under a fault load is an assessment relevant to this work. Performability is an attribute of software quality and a product of fault tolerance and performance. Performance identifies as a key attribute of dependability in terms of speed, throughput, and regularity of responses characterized by event arrival [1]. Bala and Chana [7] underlined the necessity of fault tolerance in Service-Oriented Computing (SOC) and a plethora of researchers have published their interests in ensuring the dependability of services offered across the internet and cloud at large [5], [6], [8]. In a topical perspective regarding the advancement of SOC with the cloud, [9] highlighted that "there has been a paradigm shift from trying to avoid failures at all costs to embracing faults as opportunities for making the system more resilient. The amount of effort put into fault tolerance and resilience of service-based applications is often determined by the trade-offs between development effort, costs for redundancy, availability, and consistency". Regardless, the performability of service systems is threatened by faults, and their occurrence results in erroneous system states that may cause the failure of some components or the entirety of the system. This critically drags the notion that fault tolerance must be considered, implemented, and assessed to determine the impact of faults on the system's tolerance mechanism and its behavior under fault loads [10], [11]. One way of assessing this impact and determining the resilience of service systems against fault threats is via the simulation of software fault injection.

Fault injection plays an important role and has been singled as a feasible solution and veritable tool to expose a system's dependability in terms of performability (fault tolerance and performance) – that is, software fault injection

E-mail address: oliha_festus@uniben.edu

techniques have been largely used as means for evaluating the dependability of systems in presence of certain types of faults [3], [4], [12], [13].

Fault-tolerant approaches enable the handling of faults but, "being able to identify the magnitude of fault tolerance in a system would be a useful analysis tool for its performance" [14], while increasing efforts in determining and studying how faults could impact the behavior of fault-tolerant service systems under a certain fault type. This conception, as a result, motivated the researcher to examine the performability of the web service-based solution implemented on a fault-tolerant architecture under a fault condition. Further objectives were to: implement a fault-tolerant architecture adopting the software agent's capability for coordination and adaptability; then assess the architecture solution via fault injection to analyze its impact on the solution's performability. The rest of the paper is organized as follows: Section 2 gives background information on the underlying technologies, tools, and techniques for building a fault-tolerant web services solution with related literary works; Section 3 covers the proposed architecture, its descriptions, and workability: Section 4 entails the experiments, evaluation results, and analysis; it concluded with section five.

2. Background Information

Service-Oriented Architecture (SOA) in similar customs is described by the Open Group as "an architectural style that supports service orientation for a community of service providers and consumers for mutual value" [15]. It is a technological paradigm for development and delivery of software functions as services callable by itself or other services. Explicating this description leaves SOA to "an evolution of distributed computing based on the request/response design paradigm for synchronous and asynchronous applications" [16]. It summarizes a paradigm advancing distributed applications with service-oriented technological changes. SOA was designed with the use of Web Services as one of the core standards broadly used in business settings due to their significant capabilities in the integration and interoperability of business solutions. SOA is realized or implemented via web services technologies to create and deploy constructive blocks of service functions which are readily available on the internet and accessible via standard sets of protocols [17]. Thus, service on its own is described as a unit of work done by a service provider to achieve the desired result for a service consumer.

Studies show that web services (or a collection of them) are a realization of SOA [15], [16], [18], denoted as software entities that are specified by universal resource identifiers, and their general interfaces are defined and interpreted using XML, in which other systems, services, or entities are allowed to communicate with by a resolute behavior and following its service descriptions and definitions [17]. Quoting [19], "web service model's main roles are as service providers, service consumers and service registry with core artifacts being services and service descriptions based on a find-bind-use approach". In web service-based systems, service solution is characteristically dynamic so that services are discovered, selected, and composed, possibly at runtime. However, web services based on SOAP solutions are associated with tolerance and performance issues in the presence of faults [5], [6], [8].

Similar studies exposed that "web services provide a better solution to solve the problems of platform dependency and incompatibility issues across various technologies but, the technology itself is also lagging reliability and performance" [10], [18], [20]. This reiterates the necessity of a fault-tolerant attribute in service systems to ensure the delivery of available and reliable services over a period of time. Fault tolerance (FT) is a critical feature and has remained a key issue to service systems dependent on web services. FT is one way to ensure service availability for consumption and this is achievable via different techniques exposed in terms of redundancy – an approach that encompasses replication as a strategy in achieving FT either as active or passive replication; diversity with N- Version Programming – building replicas with different design diversity by different vendors [11], [21], [22], [23], [24], [25]. The degradation associated with service-based systems arises with the performance tradeoffs in tolerating fault(s) or the impact of its component failure – yielding undesirable performance in service systems. The impact of switching or recovering from failed replicas is consequential on response time which adversely affects the performability of the system.

A. Fault Injection: Tools and Techniques

The uninterrupted nature and availability of service systems make them more susceptible to faults that intend to drive the system in erratic behavior and thereby, threaten the survivability of the system's entirety. Faults are unavoidable defects that alter the goal of any services system. Faults could be classified as hardware/physical (permanent, transient, or intermittent), or software [26]. A fault-tolerant service system aims to perform suitably, its desired functionality irrespective of fault occurrence to ensure efficient service provision, availability, and delivery. To assess FT of service solutions, fault injection plays a vital role and several classes of fault injection have been identified in the literature [27]":

- Hardware fault injection, where the actual hardware system is affected by external physical sources;
- Simulation-based fault injection, where the target system and the faults are modeled and simulated with a fault simulator;
- Emulation-based fault injection, where the target system is emulated (usually with FPGAs) and faults are injected in the emulator".

A plethora of software fault injection (SFI) techniques and tools (Jaca, RIFLE, Xception, LFI, LIFTING, etc.) exists in aiding the introduction of faults into a system solution to evaluate the performance of the system while assessing the fault-tolerant mechanism under a simulated fault. A detailed survey of simulation-based tools for fault injection and its tolerance techniques was summarized in [25], [27], noting fault injection as a veritable tool that permits the evaluation of systems dependability. Fault injection in a software-based solution is usually time-based – either at compile-time or runtime. Compile-time encompasses code mutation or insertion at a specific location of the solution's source codes. Fault injection experiments the system's tolerance capability with emphasis to expose its behavior and possible weakness or other salient faults. The goal, however, is to assess the impact of faults on the system behavior towards improving performability and dependability.

B. *Performability in Service Systems*

Dependability in service systems particularly dependent on web services subsumes service attributes which collectively enables the system to avoid service delivery failure that can frequently and severely result in the system than acceptable. These attributes are (but are not limited to) availability, reliability, performance, and maintainability which are also subsumed as software quality attributes [27]. Reliability and availability connote fault tolerance but combined with performance connote the performability of the system. Performability in service systems is adjudged relatively over an interval of time rather than an instant of time: where the solution performs continuously in a foreseeable behavior without interruption under a simulated or real condition.

Thus, the research scope is aligned to performability attributes of FT and performance for building dependable systems. FT, connoting the means (among others: fault prevention, fault forecasting, and fault removal) towards ensuring and increasing the dependability of service systems against undesired threats – faults, error, and failure. Performance is the timing attributes involved in service delivery and a means measured via throughput and response time with regularity in responses over a period of time with tools like JMeter, SoapUI, and Storm [8], [18], [28], [29]. Emulated faults are usually injected to assess the system's performability with these metrics.

To establish the performability of service systems dependent on web services under a fault load, the researcher proposed software agents as a feasible solution towards ensuring dependability in terms of FT and performance. Software agents have been underlined in literature as the fundamental for developing fault-tolerant-based software solutions [30], [31].

C. *Software Agents and Fault Tolerance*

Software agents have predominantly gained adoption and popularity over the years in distributed systems implementation and details of its approaches were documented in correlated literature [30], [32]. Software agents are entities with distributed and partitioned characteristics capable of performing specific tasks autonomously [33]. The distributed entity is a property common to SOA and software agents in building distributed systems. One of the descriptions of SOA regarding other related computing paradigms such as Agent-Oriented Computing (AOC) was highlighted in [14] as "an architectural style whose goal is to achieve loose coupling among interacting software agents", noting how important their roles are on behalf of their owners as providers and consumers. A collection of two or more denotes multi-agents and key attributes such as autonomy, reactivity and pro-activeness were noted in related studies [7], [10], [24], [34].

Explicating software agents with relevance to FT in service systems based on web service solutions, a communication challenge burdens the amalgamation of SOA and AOC paradigms but, a solution addressing this has over the decade witnessed realization with the sociability attribute and even supported that FT is improved by replicating agent services in the related field of research [35], [36], [37]. However, [30] asserted that "fault tolerance is fundamental for the development of agent-based applications", and it is noteworthy that, this study emphasizes software agents as the top consideration – given their coordination and adaptability capability to changing requirements and environments towards addressing the trending issue of performability associated with fault-tolerant service systems.

D. *Related Works*

In [38], it was warned that fault-tolerant systems particularly with a replication approach for message-exchanging are considered with performability issues and proposed a pipeline approach to mitigate the performance overhead but, how the system behaved was with little transparency under faulty scenarios.

A fault injection toolkit was developed in [5] to test the dependability of service systems in terms of FT and performance without modification to the system being tested. The impact of fault tolerance protocol deployed at a service client was studied via the toolkit and packets were tempered and monitored given their focus on fault injection on the network layer and not the application layer.

Also, [9] exposed that most SFI models are not suitable as they do not cater to the dynamic aspect of software failure. Thus, a structural approach with fault injection-driven development was suggested for applying SFI.

The work of [24] emphasized that "it is important to eliminate failures or minimize the impact of failure" and proposed an algorithm to identify an optimal fault-tolerant candidate for critical configurations of the software system. However, their FT approach was based on replication by exploiting application behavioral characteristics – performance. Their work employed configurations using suitable FT candidates to improve the reliability of fault tolerance of a software system. They employed fault injection in their assessment for reliability.

In another related effort, [11] vented on fault injection as a tool for performance testing of web services in a composite manner. Their architecture captured "coverage criteria to guide fault injection testing of performance-related issues in composite web services by generating fault injection configurations that follow the defined test criteria for systematic fault injection and effective evaluation" of performance in terms of response time but the impact on its FT was not highlighted.

Summarily, it is construed from the review of related literary works that:

1) fault injection plays a vital role towards assessing the behaviour of fault tolerant systems under a fault load
2) the performability of fault tolerant architecture under fault injection is worthy of assessment to determine the system's survivability and dependability.

Consequently, the study proposes a research approach that adopts software agents to amalgamate coordination and management of logical time replica activities during execution.

3. Methodology - The Proposed Approach

This research aligns itself with an approach that systematically describes a service system from an architectural context through designs, implementation, deployment, experiment, and assessment.

A. The Proposed Architecture

Reiterating SOA by its defined functionality, Figure 1 depicts an architecture proposed with capabilities of a fault-tolerant system with acceptable performance under a fault load.

The architecture proposed is an integrated framework amalgamating replication, diversity, and N-Version techniques to ensure efficiency in service availability and delivery. It subjects logical redundant services to the coordination of Multi-Agent Services (MAS) depicting core components of the architecture and also uniquely active for service replica management and fault handling. The architecture components are highlighted as follows:

1) **Client Interface:** for service requests and responses
2) **Service Integration Gateway:** Web Service Integration Gateway (WSIG) component provides communication handshakes between web services and agent services. It receives service requests as SOAP messages, converts them into Agent Communication Languages (ACL) messages for agent services and activities, and then return ACL messages as SOAP responses to the response handler for client consumption.
3) **Agent Management Services (AMS)** is the environment for managing all software agents designated for the system's task – a multi-agent system. This environment is provided via the Java Agent DEvelopment (JADE) platform hosting agent services in which agents can live and execute their designated services. JADE is the most suitable platform for developing MAS [37], [39]. AMS is the major strength and heart of the architecture highly responsible for MAS including creation, registration, behavior, communication, deletion, and agency in accordance to the Foundations for Intelligent Physical Agents (FIPA) specification. It is also saddle with coordination of all logical activities involving replica group creation, replica addition, and replica process management for every replica solution present. In this way, a faulty replica or faults are detected and handled via a client-transparent method. This coordination mechanism is unique to the proposed architecture for managing fault tolerance and response stability in service delivery.
4) **Fault Manager:** this component is burdened with the role of stressing the solution with faults and communicating faulty logs with the MAS coordinator for appropriate actions. Notably, the MAS Component coordinates the services of several software agents in managing faults and replica services. Thus, the fault manager is bound under the services of MAS to manage the FT scheme/mechanism towards testing (injecting faults) the dependability of the service solutions within its confinements.
5) **Replica Solution:** is an N-Versioned scheme for building vendor-transparent replica service solutions by different service vendors or providers. In the proposed architecture, each replica solution stands as a stack of at least four N-Version sets of replica groups with n-1 of them active and running concurrently for each group and a passive standby providing the computational services with logical activities managed by MAS.
6) **Replication Solution Selection Scheme:** is a crucial component of the architecture responsible for solution response selection from n number of Replica Solution (RS) services. As observed in the literature, performance is degraded as a result of latency – delayed response, possibly due to switching from faulty replicas during fault occurrence. Owing to this, AMS employed a selection mechanism to manage replica solution results from n replica sets.
7) **Message Bus:** communicates error message logs with notification from RS services to the fault manager.

B. Architectural Deployment Specification

The technical specifications for knitting several components of the proposed architecture are captured in Figure 2. That is, the specification and logic designs required for implementation and their interfaces with emphasis on their

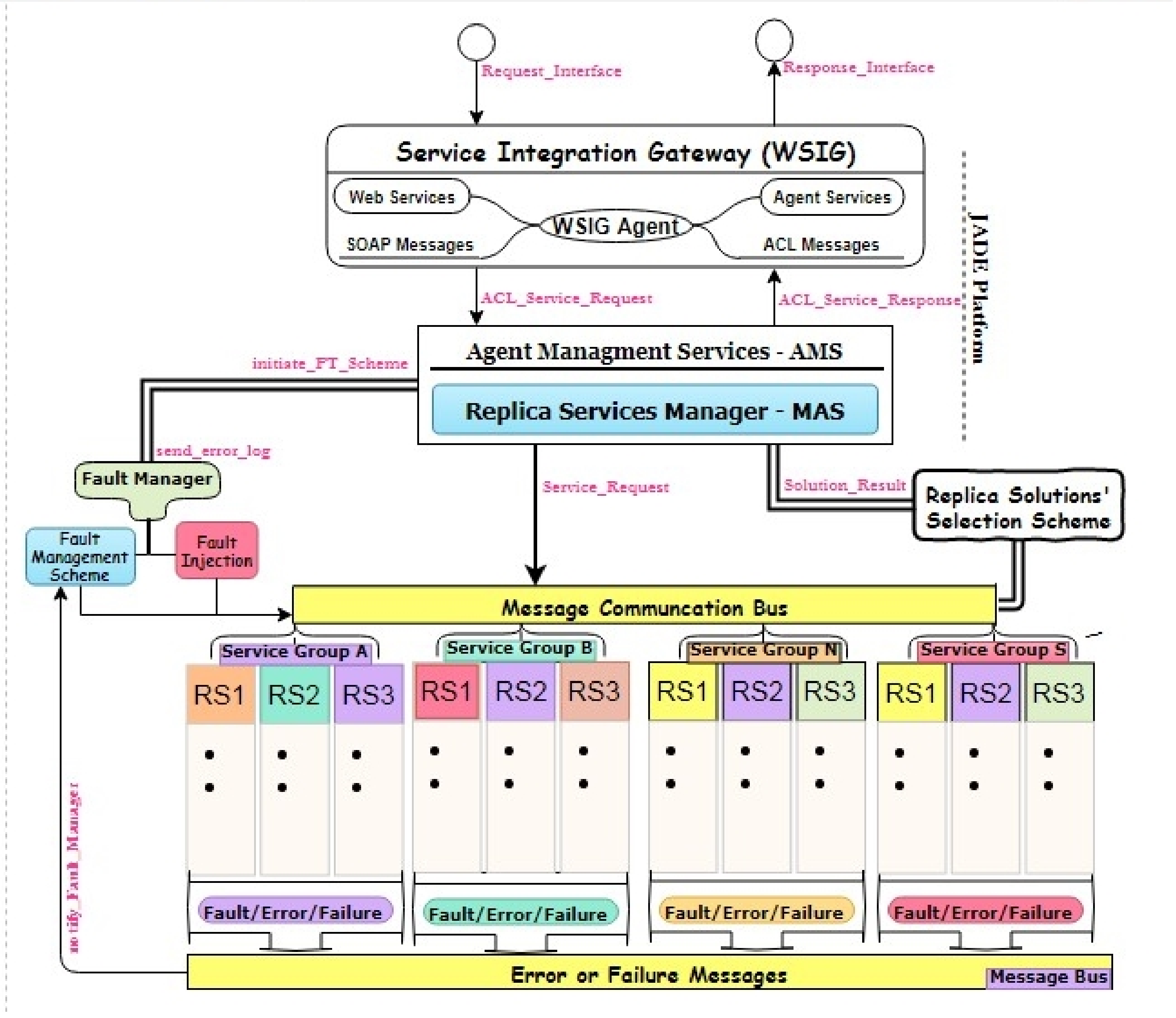


Figure 1. The Proposed Architecture for Building Fault-Tolerant Service Solution.

bindings and communications from a deployable client space and backend services

A simple test study was emphasized in [40] stressing the need to ensure that the heart of the architecture is heavily-weighted (such that the client service is a simple solution service for request and response) to be able to assess the dependability of the service provider in terms of availability, response time stability, and the unit of the request processed per second – throughput. Figure 3 depicts the replica management by MAS with a selection scheme for quick response.

Owing to this, the functionality of the proposed architecture was modeled after a lightweight test case – a simple Grade Point Average (GPA) calculator, which agrees with the work of [40], [41]; shifting the workload from the client end to the architecture solution to over-engage it with large-scale simultaneous requests such that latency is increased on service responses to expose the tolerance mechanism under a fault load.

C. Fault-Tolerant Mechanism and Injection

The fault-tolerant approach aligned in this study encompasses a replication technique with diversity based on N-Versioned programming. Replication is a major fault-tolerant approach towards ensuring service availability by building replicas. With this FT mechanism, replica solutions were built with diversity employing the N-Version technique of at least four N-Versioned sets of replica groups with n-

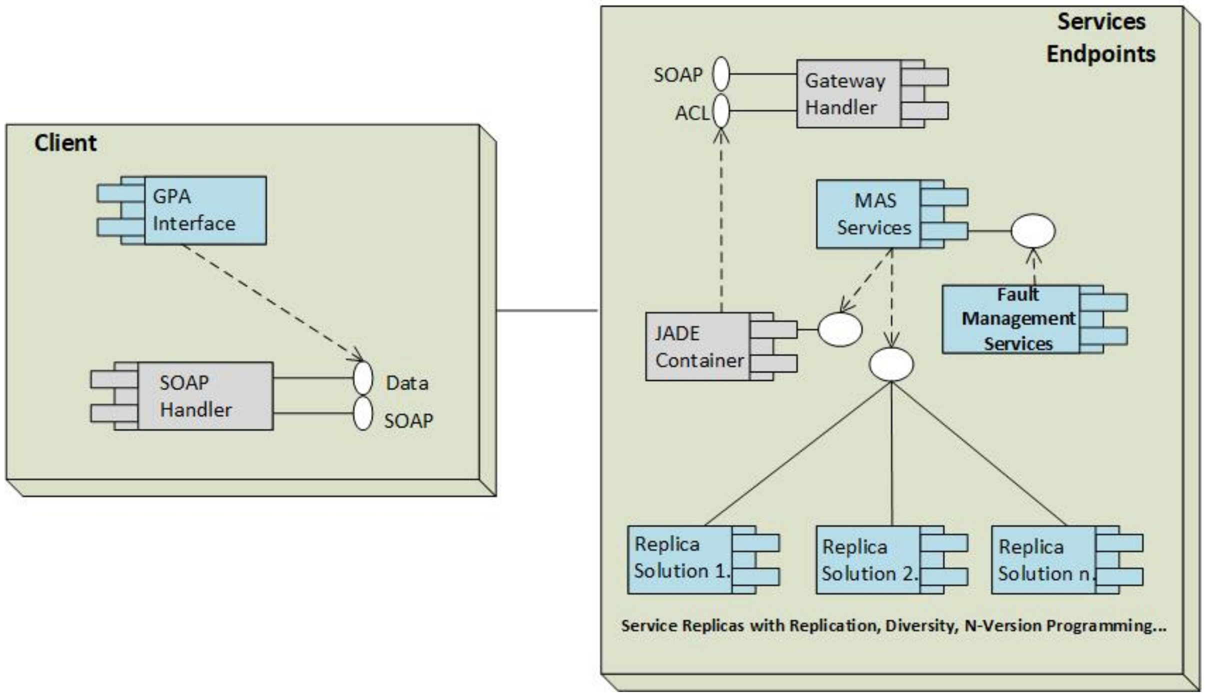


Figure 2. Deployment Specification for Implementing the Proposed Architecture Solution.

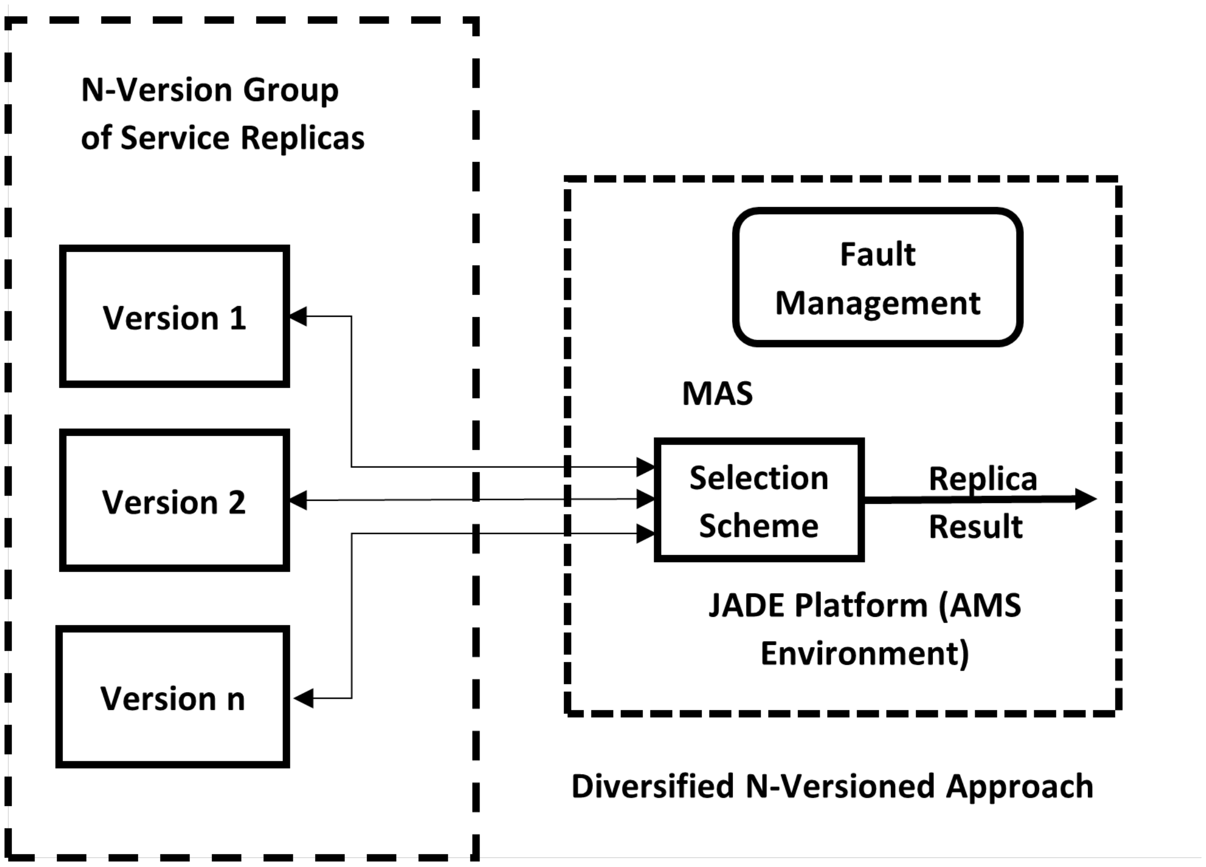

N-Version Group of Service Replicas
Version 1
Version 2
Version n
Fault Management
MAS
Selection Scheme
Replica Result
JADE Platform (AMS Environment)
Diversified N-Versioned Approach

Figure 3. Replica Solution Management

1 of them active and running concurrently for each group and a passive standby as captured in Figure 1. To tolerate fault within each group, service replicas were anchored with a design and code diversity to handle service defects and fault propagation associated with the replication approach [42]. That is, the propagation of faults to other replicas are avoided due to diversity in service vendors, and should there be a faulty or crashed service replica solution, others will actively continue a transparent service provision due to agent coordination mechanisms also providing fault management services to affected replica(s). This FT mechanism being managed by MAS ensures that services are readily available irrespective of faults or their occurrence to enhance service delivery efficiencies in the proposed fault-tolerant architecture solution.

Explicating the notion that, fault-tolerant mechanisms may cause service delivery inefficiencies in service systems, this work aligns itself with multi-agent services in implementing logical activities associated with services replicas when faults occur and replica switching and selection is done without performance overhead in service delivery as shown in Figure 2. The efficiency is in the coordination capability of MAS to ensure performance while avoiding delays in switching from a faulty replica to another among the active or passive replicas. This does not only ensure FT but guarantees the reliability of the results and availability of services. However, a concerning note on this service delivery efficiency is how accurate and consistent services remain in the presence of faults, and fault injection is one way to ascertain this notion.

Fault injection is apposite in assessing the reliability of service-based solutions via any SFI scheme. Given the fact that the performability of service systems is threatened by faults, their occurrence can lead to erroneous system states that may cause the failure of some components or their entirety. Thus, the SFI technique – static code-based fault model (code mutation/insertion) was simulated in injecting replica crash faults at compile time and identifying replica service solutions with the erroneous state at runtime [43], [44], [45], [46], [47], [48]. Employing this technique, Figure 4, and Figure 5 captured the steps required for fault injection and notification of a replica crash type.

The pseudocode follows a structural construct with a Boolean controlled mechanism on lines 12 and 13. If the Boolean value is set (true), then the fault injection construct is activated on the selected sets of replica service groups or processes. If the crash service occurs for the targeted replica solutions, then the AMS is informed with a construct of error messages as an ACL message notification via the fault manager module for potential actions.

D. Configuration and Deployment

The JADE platform was configured as the agency for housing multi-agent services and necessary files were extracted into appropriate directories to set up parameters, system environment, and user variables before subjecting

Pseudocode 1a: Pseudocode for Fault Injection via Code Modification/Insertion

```
Output: Error/Failure Message
Inputs: Array of replica Services
Initialization:    import ArrayList properties;
           <ReplicaResultProcess> replicaResult = new ArrayList();
           <Boolean> inject_fault_group = new ArrayList<>();
           <Thread> replicasThread = new ArrayList<>();
Step1: Get Repplica Size - n
1.      private void faultManager(ActionSender something, …,
          ACLMessage msg) {
2.        getReplica(n);
3.        if (replicaSize < n);
4.          addReplica(0);
5.          …;
6.          addReplica(n-1);

Step2: Add Process Threads Groups and Fault Module
7.          faultThread F0 = createGroup(0, …, msg); //active
          replicas
8.          …;
9.          faultThread Fn = createGroup(n-1, …, msg); //passive
          replica
10.       replicasThread.add(F0);
11.       replicasThread.add(Fn);
             …;
12.       injectfault_group.add(true);
13.       injectfault_group.add(false);
             …;
14.       F0.start(); //start replica group threads
15.             Fn.start();
             }
```

Figure 4. Pseudocode 1a: Fault Injection via Code Mutation

Pseudocode 1b Pseudocode for Fault Injection via Code Modification/Insertion

```
Step3: get Error message
16.       private void getErrorMessage(){
17.         try{
18.             for(faultThread t: replicasThread)
19.                 t.join();
20.             int i = 0; //active replica groups and processes
21.             for (ReplicasResultProcess re : replicasResult){
22.                 Boolean[] pr = re.getInterrupted();
23.             int j = 0;      // for passive replica groups
24.             for (Boolean pr1 : pr) {
25.                 if(pr1){   //if process failed
26.                     if(I == (replicasThread.size()-1))  //passive failure response
27.                             logger.log(Level.INFO, "Passive Process Output
                             (Process{0}) :
                          Failed", new Object[]{j});
28.                     else
29.                     logger.log(Level.INFO, "Group - {0} -> Process Output
                          (Process{1}) : Failed", new Object[]{i, j});
30.             }
31.             j++;
32.            }
33.             i++;
34.         }
35.       }catch(Exception ex){
36.        logger.log(Level.SEVERE, "Observation Check "+ex.getMessage(), ex);
37.     }
38.   }
Step4:  Send Notification
39.      private void sendNotification(Action actExpr, …, ACLMessage request,
            int Performative, Object result){
40.         ACLMessage notification = prepareNotification(actExpr, request,
            performative, result);
41.         send(notification);
42.     }
```

Figure 5. Pseudocode 1b: Fault Response and Notification

the fault-tolerant web services solution to testing. The diagram in Figure 6 depicts a successful configuration and the technical gluing of several software components to enable service provision and consumption of the proposed solution. Web services were deployed on the Universal Description, Discovery, and Integration (UDDI) servers as agent services and vice versa with multi-agent capabilities.

4. Experimental Study and Results

Using the compile SFI, a replica crash fault was injected into the service solution and subjected to a performance test to ascertain its impact. The experiment was a simulation of a performance evaluation using Apache JMeter with performance indicators – response time, throughput, response stability (standard response deviation). Two versions of the solution were configured at 25,000 per unit time at a large scale from random ten services request channels for testing – that is, without a fault load and with fault injection. The solution's behavioral data from the performance evaluation were captured as depicted in Figure 7, and Figure 8.

Figure 7 and Figure 8 are the evaluation summary reports from the experiments of the architecture solution without fault injection (Figure 7) and with fault injection (Figure 8). In both figures, a total of 25,000 sample service requests were distributed randomly through ten different service request channels. The reports captured the average, minimum, and maximum response times for each request channel with an overall of 4072, 28, and 10885 for the architecture solution without a fault load and 1173, 27, and 2565 for the architecture solution with fault injection. Other evaluation attributes captured were error%, throughput, request size per second, and standard deviation which represents the regularity or stability of service responses to requests. It tells the rate of variation from the expected service responses to requests. For Figure 7, it shows that an average of 5.35 KB/sec request size was processed at a ramp-up time of 2.4/sec for throughput with a standard deviation of 655.06 at an error rate of 0.00%. Also, Figure 8 shows that an average of 18.6 KB/sec request size was processed at a ramp-up time of 8.5/sec for throughput with a standard deviation of 165.80 at an error rate of 0.00%. The error rate indicates a 100% error-free evaluation, meaning no arbitrary or incoherent (byzantine) responses to service requests for the entire sample requests of 25,000. To appreciate the results, the maximum response time (which is the worst case), standard deviation (response stability or regularity), and throughput (computational effort) were considered for graphical representations and further analysis in line with standard practices for performance evaluation.

From Figure 7 and Figure 8, graphs were plotted for both versions of the architecture's solution as shown in Figure 9 and Figure 10: where SWOF(x) represents the architecture's Solution without Fault Injection and SWFI(y) represents the Solution with Fault Injection. The experiment results for performance attributes of response time and rate of the regularity of responses – response stability were captured in Figure 9 and Figure 10.

Although throughputs for both scenarios were fascinatedly observed from the experiment results to be an average of 2.4sec (SWOF) and 8.5sec (SWFI). To buttress the impact of fault injection on the experiment, the response time and response stability were selected in consonant with performance evaluation practices in literary work [19].

However, it was evident in Figure 9 and Figure 10 that the performability of the SWFI is better appreciated because responses are more stable with better response time when the fault was injected into the architecture's solution – an indication of reliability in service response to a service request in the presence of faults, or faulty/failed replica(s). Thus, performance was capped within an acceptable rate of regularity in service delivery of fault-tolerant service systems but in statistical terms, the rate in regularity difference with response time for SWFI solution over SWOF is empirically grey. Thus, statistical interpretation was paramount.

A. Result Analysis

Consider x_i and y_i to be sample variables, such that x and y are the solution versions without and with fault injection, and i denotes the assessment attributes for performance – response time, response stability, and throughput. The goal is to examine the architecture solution without fault injection (x) against the solution with fault injection (y), and determine whether or not injecting fault into the architecture solution will:

1) ***for i = 1***; reduce the response time efficiency of the solution
2) ***for i = 2***; worsen the response stability of the solution, and
3) ***for i = 3***; increase computational overhead – throughput of the solution

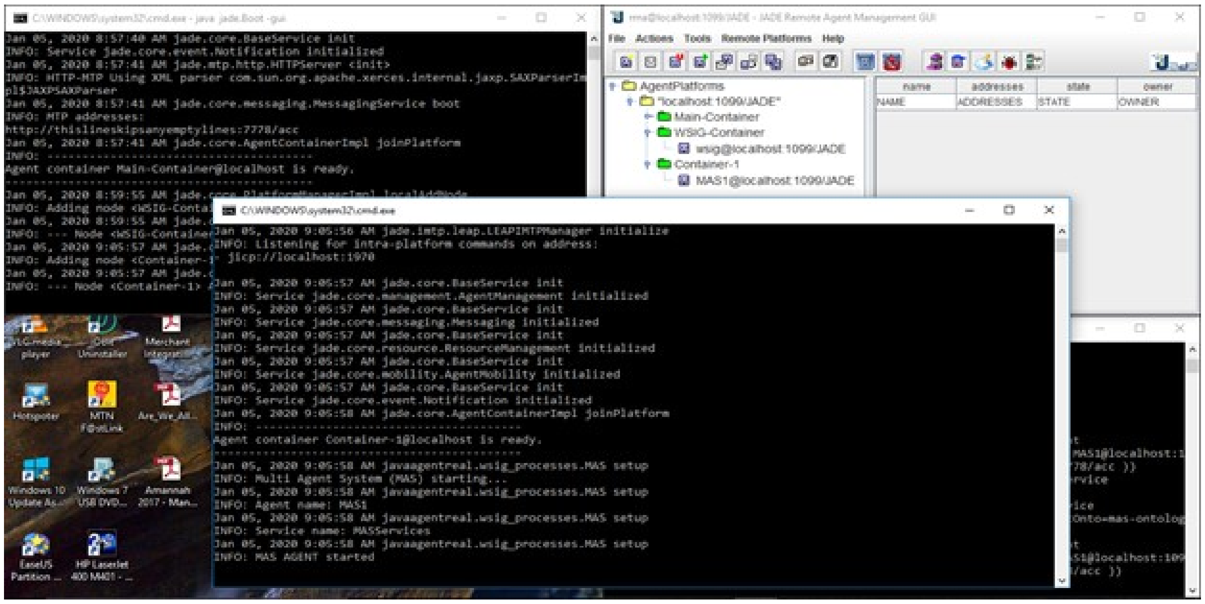
Thus, for each of the performance instances, the following hypotheses are tested:

instance 1: for i = 1, the following hypothesis holds:

1) ***H_0:*** $\mu_x = \mu_y$ indicating no significant impact of fault injection on response time efficiency for SWF and SWFI.
2) ***H_1:*** $\mu_x > \mu_y$ indicates that the impact of fault injection significantly increases time efficiency in SWFI over SWOF.
3) ***H_2:*** $\mu_x < \mu_y$ indicates that the impact of fault injection significantly reduces time efficiency in SWFI over SWOF.

instance 2: for i = 2, the following hypothesis holds:

1) ***H_0:*** $\mu_x = \mu_y$ indicating no significant impact of fault injection on response stability for SWOF and SWFI.
2) ***H_1:*** $\mu_x > \mu_y$ indicates that the impact of fault injection significantly guarantees response stability

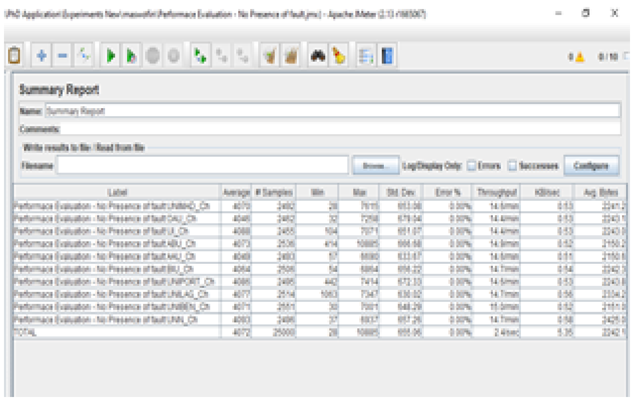

C:\WINDOWS\system32\cmd.exe - java jade.Boot -gui
rma@localhost:1099/JADE - JADE Remote Agent Management GUI
File Actions Tools Remote Platforms Help
AgentPlatforms
"localhost:1099/JADE"
Main-Container
WSIG-Container
wsig@localhost:1099/JADE
Container-1
MAS1@localhost:1099/JADE
name addresses state owner
NAME ADDRESSES STATE OWNER
C:\WINDOWS\system32\cmd.exe

```
Jan 05, 2020 8:57:40 AM jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
Jan 05, 2020 8:57:41 AM jade.mtp.http.HTTPServer <init>
INFO: HTTP-MTP Using XML parser com.sun.org.apache.xerces.internal.jaxp.SAXParserIm
pl$JAXPSAXParser
Jan 05, 2020 8:57:41 AM jade.core.messaging.MessagingService boot
INFO: MTP addresses:
http://thislineskipsanyemptylines:7778/acc
Jan 05, 2020 8:57:41 AM jade.core.AgentContainerImpl joinPlatform
INFO: --------------------------------------
Agent container Main-Container@localhost is ready.
--------------------------------------------
Jan 05, 2020 8:59:55 AM jade.core.PlatformManagerImpl localAddNode
INFO: Adding node <WSIG-Conta
Jan 05, 2020 8:59:55 AM jade.
INFO: --- Node <WSIG-Containe
Jan 05, 2020 9:05:57 AM jade.
INFO: Adding node <Container-1
Jan 05, 2020 9:05:57 AM jade.
INFO: --- Node <Container-1>
```

```
Jan 05, 2020 9:05:56 AM jade.imtp.leap.LEAPIMTPManager initialize
INFO: Listening for intra-platform commands on address:
- jicp://localhost:1970

Jan 05, 2020 9:05:57 AM jade.core.BaseService init
INFO: Service jade.core.management.AgentManagement initialized
Jan 05, 2020 9:05:57 AM jade.core.BaseService init
INFO: Service jade.core.messaging.Messaging initialized
Jan 05, 2020 9:05:57 AM jade.core.BaseService init
INFO: Service jade.core.resource.ResourceManagement initialized
Jan 05, 2020 9:05:57 AM jade.core.BaseService init
INFO: Service jade.core.mobility.AgentMobility initialized
Jan 05, 2020 9:05:57 AM jade.core.BaseService init
INFO: Service jade.core.event.Notification initialized
Jan 05, 2020 9:05:58 AM jade.core.AgentContainerImpl joinPlatform
INFO: --------------------------------------
Agent container Container-1@localhost is ready.
--------------------------------------------
Jan 05, 2020 9:05:58 AM javaagentreal.wsig_processes.MAS setup
INFO: Multi Agent System (MAS) starting...
Jan 05, 2020 9:05:58 AM javaagentreal.wsig_processes.MAS setup
INFO: Agent name: MAS1
Jan 05, 2020 9:05:58 AM javaagentreal.wsig_processes.MAS setup
INFO: Service name: MASServices
Jan 05, 2020 9:05:58 AM javaagentreal.wsig_processes.MAS setup
INFO: MAS AGENT started
```

Figure 6. Successful Configuration and Deployment of Solution Services.



Name: Summary Report
Comments:
Write results to file / Read from file
Filename

Label	Average	# Samples	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
Performace Evaluation - No Presence of fault UNIMAID_Ch	4070	2482	28	7615	653.08	0.00%	14.6/min	0.53	2241.2
Performace Evaluation - No Presence of fault OAU_Ch	4046	2482	32	7258	679.04	0.00%	14.4/min	0.53	2243.5
Performace Evaluation - No Presence of fault UI_Ch	4088	2455	104	7071	651.07	0.00%	14.4/min	0.53	2240.0
Performace Evaluation - No Presence of fault ABU_Ch	4073	2536	414	10885	666.68	0.00%	14.9/min	0.52	2150.2
Performace Evaluation - No Presence of fault AAU_Ch	4049	2493	57	6680	633.67	0.00%	14.6/min	0.51	2150.6
Performace Evaluation - No Presence of fault BU_Ch	4064	2506	54	6864	656.22	0.00%	14.7/min	0.54	2242.7
Performace Evaluation - No Presence of fault UNIPORT_Ch	4086	2495	442	7414	672.33	0.00%	14.5/min	0.53	2243.8
Performace Evaluation - No Presence of fault UNILAG_Ch	4077	2514	1063	7347	630.02	0.00%	14.7/min	0.56	2334.2
Performace Evaluation - No Presence of fault UNIBEN_Ch	4071	2551	30	7001	648.29	0.00%	15.0/min	0.52	2111.0
Performace Evaluation - No Presence of fault UNN_Ch	4093	2486	37	6937	657.26	0.00%	14.7/min	0.58	2425.0
TOTAL	4072	25000	28	10885	655.06	0.00%	2.4/sec	5.35	2242.1

Figure 7. Evaluation Summary Report without a Fault Load.

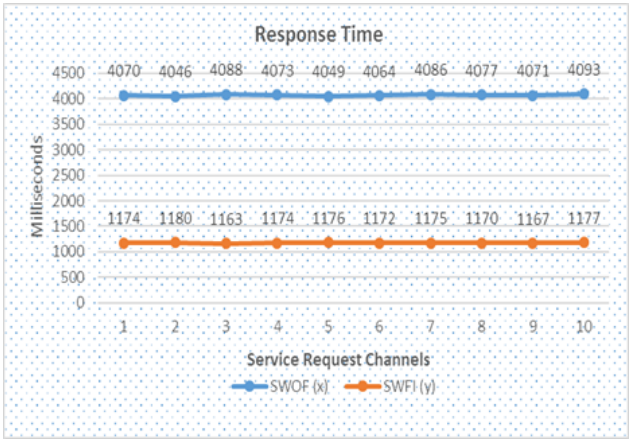
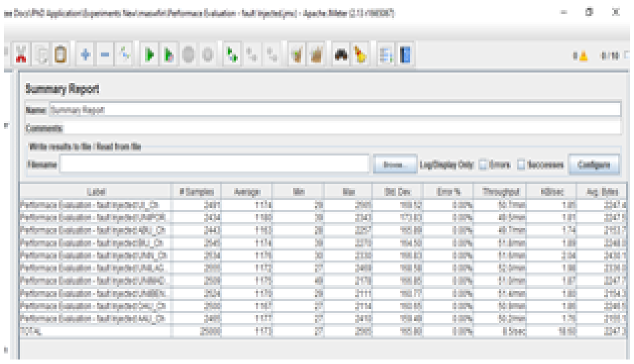


Figure 9. SWOF vs SWFI Line Graph for Response Time.



Name: Summary Report
Comments:
Write results to file / Read from file
Filename

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
Performace Evaluation - fault injected UI_Ch	2491	1174	29	2565	169.52	0.00%	50.7/min	1.85	2247.4
Performace Evaluation - fault injected UNIPOR	2434	1180	39	2343	173.83	0.00%	49.5/min	1.81	2247.5
Performace Evaluation - fault injected ABU_Ch	2443	1163	28	2257	165.89	0.00%	49.7/min	1.74	2153.7
Performace Evaluation - fault injected BU_Ch	2545	1174	39	2270	164.50	0.00%	51.8/min	1.89	2248.0
Performace Evaluation - fault injected UNN_Ch	2534	1176	30	2330	166.83	0.00%	51.6/min	2.04	2430.1
Performace Evaluation - fault injected UNILAG	2555	1172	27	2469	168.58	0.00%	52.0/min	1.98	2336.0
Performace Evaluation - fault injected UNIMAID	2509	1175	48	2178	166.85	0.00%	51.0/min	1.87	2247.7
Performace Evaluation - fault injected UNIBEN	2524	1170	29	2111	160.77	0.00%	51.4/min	1.80	2154.3
Performace Evaluation - fault injected OAU_Ch	2500	1167	27	2114	160.65	0.00%	50.9/min	1.86	2248.5
Performace Evaluation - fault injected AAU_Ch	2465	1177	27	2410	159.49	0.00%	50.2/min	1.76	2155.1
TOTAL	25000	1173	27	2565	165.80	0.00%	8.5/sec	18.50	2247.3

Figure 8. Evaluation Summary Report with Fault Injection.

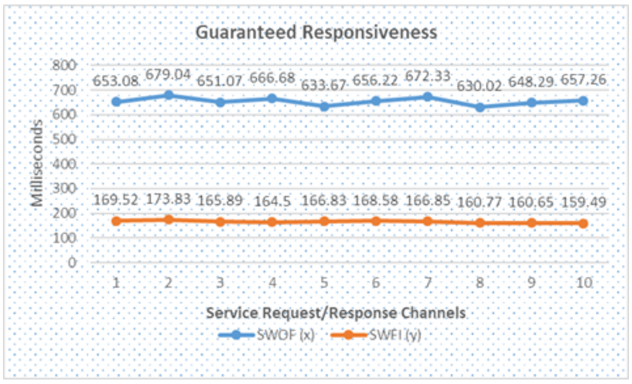


Figure 10. SWOF vs SWFI Line Graph for Guaranteed Responsiveness.

in SWFI over SWOF.

3) ***H*$_2$*:*** $\mu_x < \mu_y$ indicates that the impact of fault injection significantly degrades response stability in SWFI over SWOF.

instance 3: for i = 3, the following hypothesis holds:

1) ***H*$_0$*:*** $\mu_x = \mu_y$ indicating no significant impact of fault injection on computational overhead for SWFI over SWOF.
2) ***H*$_1$*:*** $\mu_x > \mu_y$ indicates that the impact of fault injection significantly reduces computational overhead in SWFI over SWOF.
3) ***H*$_2$*:*** $\mu_x < \mu_y$ indicates that the impact of fault injection significantly increases the computational overhead in SWFI over SWOF.

Two (response time and response stability) of the three categories of hypotheses are to be tested for each performance attribute (instances). Thus, let x represent the normal solution SWOF and y represent the affected solution SWFI. To validate the hypothesis, the t distribution test was adopted for analysis because of the sample size (n = 10). Thus, x and y are normally distributed with means μ_x and μ_y and with the same variance $\sigma_x = \sigma_y$ and a random sample of sizes n_x and n_y. Thus, the sample means and variance are denoted by $\bar{x}$ and $\bar{y}$, S_x^2 and S_y^2, then the difference between two means for both versions with the same samples on the stated hypothesis is given by [49]:

$$t = \frac{(\bar{x}-\bar{y})-(\mu_x-\mu_y)}{\sqrt{(n_x s_x^2 + n_y s_y^2)}} \sqrt{\frac{n_x n_y (n_x+n_y-2)}{n_x+n_y}} \tag{1}$$

Where $n_x + n_y - 2$ represents the degrees of freedom. Since the sample sizes are equal i.e. $n_x = n_y = 10$; therefore, Equation 1 can be summarized to:

$$t = \frac{(\bar{x}-\bar{y})-(\mu_x-\mu_y)}{\sqrt{(n_x s_x^2 + n_y s_y^2)}} \sqrt{90} \tag{2}$$

Assuming that $\sigma x = \sigma y$, then, Equation 2 becomes Equation 3 respectively:

$$t = \frac{(\bar{x}-\bar{y})}{\sqrt{(n_x s_x^2 + n_y s_y^2)}} \sqrt{90} \tag{3}$$

But, the sub-component is calculated in Equation 4

$$n_x s_x^2 = \sum_{i=1}^{n} (x-\bar{x})^2 \tag{4}$$

Also, Equation 5 becomes:

$$n_y s_y^2 = \sum_{i=1}^{n} (y-\bar{y})^2 \tag{5}$$

TABLE I. ns^2 Computation for Response Time

Size	SWOF (x)	$x-\bar{x}^2$	SWFI (y)	$y-\bar{y}^2$
1	4070	2.89	1174	1.44
2	4046	660.49	1180	51.84
3	4088	265.69	1163	96.04
4	4073	1.69	1174	1.44
5	4049	515.29	1176	10.24
6	4064	59.29	1172	0.64
7	4086	204.49	1175	4.84
8	4077	28.09	1170	7.84
9	4071	0.49	1167	33.64
10	4093	453.69	1177	17.64
	$\bar{x}$= 4071.50	$n_x s_x^2$ = 2192.10	$\bar{y}$= 1172.80	$n_y s_y^2$ = 225.60

TABLE II. ns^2 Computation for Response Stability

Size	SWOF (x)	$x-\bar{x}^2$	SWFI (y)	$y-\bar{y}^2$
1	653.08	2.84	169.52	14.66
2	679.04	589.23	173.83	66.24
3	651.07	13.66	165.89	0.04
4	666.68	141.94	164.50	1.42
5	633.67	445.04	166.83	1.30
6	656.22	2.11	168.58	8.35
7	672.33	308.49	166.85	1.34
8	630.02	612.36	160.77	24.22
9	648.29	41.94	160.65	25.41
10	657.26	6.22	159.49	38.45
	$\bar{x}$= 654.77	$n_x s_x^2$ = 2163.85	$\bar{y}$= 165.69	$n_y s_y^2$ = 181.43

It is vital to calculate the value of t and observe if it exceeds its critical value of 2.228 for the sample size of 10 at 0.025 from the students t table, and also determine the confidence limits for $\mu_x - \mu_y$. First, the subcomponents were calculated. Table I and Table II show the computation of ns^2 for both solution samples with performance attributes – response time and response stability. The computed t values are given in Table III with subcomponent calculation details in Table IV.

The computed t values from Table III for both versions of the built architecture solution in terms of response stability and response time are 95.81 and 559.31, and the critical value of t from the student's t table for a one-tailed sample size of 10 at 0.025 (95% confidence level) is 2.228. Thus, interpreting the performance attributes instance validates the following hypothesis:

instance 1: for i = 1, the calculated value of t is 559.31, which is greater than 2.228. Therefore, $H_1 : \mu_x > \mu_y$ is valid and accepted; indicating that the architecture performability under a fault load (SWFI) is with a significant increase in time efficiency over SWOF.

instance 2: for i = 2, the calculated value of t is 95.81, which is greater than 2.228. Therefore, $H_1 : \mu_x > \mu_y$ is valid

TABLE III. Computed t Value

Variables	Time Efficiency	Response Stability
$(\bar{x}-\bar{y})$	2898.90	489.08
$n_x s_x^2 + n_y s_y^2$	2417.70	2345.28
$\sqrt{n_x s_x^2 + n_y s_y^2}$	49.17	48.43
$\frac{(\bar{x}-\bar{y})}{\sqrt{(n_x s_x^2+n_y s_y^2)}}$	58.96	10.10
t	559.31	95.81

TABLE IV. Computed Confidence Limit

Variables	Time Efficiency	Response Stability
$2.228 * (\sqrt{n_x s_x^2 + n_y s_y^2})/(\sqrt{90})$	11.55	11.37
α	2887.35	477.70
β	2910.44	500.45
$\%Performance = (\alpha \sqrt{x}) * 100$	70.91	72.96

and accepted; indicating that the architecture performability under a fault load (SWFI) significantly guarantees response stability over SWOF.

instance 3: for i = 3, the observed throughputs from the resulting experiment were 2.4 (SWOF) and 8.5 (SWFI), which are both greater than 2.228. Therefore, $H_1 : \mu_x < \mu_y$ is valid and accepted; indicating that the architecture performability under a fault load (SWFI) is with reduced computational overhead in SWFI over SWOF.

This significant impact on the performance instances is caused by the injection of fault into the sample solution and it's important to calculate the significant percentage impact where the 95% confidence limit for $\mu_x - \mu_y$ is denoted by:

$$|t| < 2.228 \quad (6)$$

substituting Equation 2 in Equation 6 reduces Equation 6 to:

$$\alpha < \mu_x - \mu_y < \beta \quad (7)$$

where α and β are the lower and upper limits respectively and given by:

$$\alpha = (\bar{x}-\bar{y}) - 2.228(\sqrt{n_x s_x^2 + n_y s_y^2})/\sqrt{90} \quad (8)$$

$$\beta = (\bar{x}-\bar{y}) + 2.228(\sqrt{n_x s_x^2 + n_y s_y^2})/\sqrt{90} \quad (9)$$

The impact of fault injection or the presence of fault shows from Table IV, that the architecture solution can only ascertain α unit of significance ($\alpha/\beta\%$) for response time and response stability. Thus:

1) the performability of the architecture solution under a fault load (SWFI) is time-efficient with service response delivery by about 70.91% over SWOF with a confidence limit of 95%.
2) the performability of the architecture solution under a fault load (SWFI) is guaranteed with response stability by about 72.96% over SWOF with a confidence limit of 95%.
3) the observed performability of the architecture solution under a fault load (SWFI) is with a significant reduction in computational overheads by about 77.98% over SWOF with a confidence limit of 95%.

The statistical results are in harmony with the graphical results and this further validates the statistical instrument asserting that the the performability of the implemented fault-tolerant architecture under a fault load is significantly worthy of dependability in the presence of faults or failure of some service replicas.

B. Discussions

Analyzing the impacts of faults in fault-tolerant software systems has received extensive reviews for some time now. Unfortunately, how they impact the system opens research for evaluation of the overall system's performability under a given fault load. Studies on FT in service systems reveal two approaches such as replication and diversity with N-Versioned programming and most often include empirical evaluation of actual fault rates. In concordance, the proposed methodology implemented a fault-tolerant service solution on software agent technologies with more concentration on evaluating the performability of the architecture solution under a given fault – replica crash, to ensure service availability and improved performance.

The proposed architecture is limited to FT of replica crash fault load at the application layer. It does not provide support for multiple fault injection techniques. Thus, it can only accommodate fault injection at compile time for the targeted system and hence, a fault outside the application layer may result in different performability outcomes. However, several studies quantified FT under faults for applications spanning large configurations and considered response time and availability of services as their performance metrics [5], [9], [48]. With a focus on the application layer, this study used the compile-time technique for fault injection and checked the performability with key attributes of the observed throughput, response time, and response stability on two versions of the proposed architecture solution – with and without fault injection.

The experimental study and results were graphically presented for both versions in Figure 9 and Figure 10 denoting the performability in the SWOF and SWFI versions in regards to response time and stability in responsiveness. It was graphically greyish to empirically contrast the performability with the impact of fault injection on SWFI over SWOF for a period of time. As emphasized in [50], a statistical interpretation was paramount to ascertain the empirical and significant difference. With service quality attributes of throughput, the response time (time efficiency), and response stability (guaranteed responsiveness) [51], [52], [53], [54], [55], the experimental study and results were statistically analyzed with indications that the architecture solution demonstrates robustness in performability. Thus, findings established that:

1) the performability of the architecture solution under a fault load (SWFI) is time-efficient with service response delivery by about 70.91% over SWOF with a confidence limit of 95% at a high throughput – reduced computational effort.
2) the performability of the architecture solution under a fault load (SWFI) is guaranteed with response stability by about 72.96% over SWOF with a confidence limit of 95%.

The research findings assert, therefore, that the architecture solution is significantly time-efficient in the presence of a fault and guarantees the regularity of service responses by about 70.91% and 72.96% with a confidence level of 95% – undoubtedly as a result of the fault-tolerant mechanisms implemented via software agent services. This implies that the performability of the architecture solution (solution's behavior) without and under a fault load unveiled a matching uniformity with appreciable stability or regularity in responses and this aligns with related efforts in [5], [10], [11], [24], [25], but is contrary to the performability issues observed in other efforts [21], [22], [51], [52].

The findings further established that the performability of web service solutions was improved averagely by about 71.9% regardless of the impact of fault injection. The experimental study and findings thus, contribute to the pool of knowledge steered towards the field of service-oriented communities for the advancement and deplorability of dependable service systems.

5. Conclusion

In this study, the influence of a replica-fault load on the performability of a fault-tolerant architecture implemented on software agent's technologies for web services solutions has been simulated and assessed via a compile-time software fault injection technique. The study's assessment and findings established that the architecture (and its solution) is significantly worthy of performability in the presence of fault while emphasizing the adoption of software agents as a feasible solution for building dependable service systems on fault-tolerant architectures. The simulation may not exactly mirror the real-world environment but, the resultant conclusion is a significant advancement in performability assessment with veritable analysis for building fault-tolerant service systems with good performance for service providers and consumers.

Further study is required to determine the performability of the architecture solution under the influence of a fault load outside the application layer for service-based systems.

Acknowledgment

No grant was received for this research. No conflict of interest.

References

[1] E. Morris, W. Anderson, S. Bala, D. Carney, J. Morley, P. Place, and S. Simanta, "Testing in Service-oriented Environments," Technology and System Solutions (RTSS) Program, Software Engineering Institute, Technical Report, Feb. 2010.

[2] N. Laranjeiro, M. Vieira, and H. Madeira, "A Robustness Testing Approach for SOAP Web Services," *Journal of Internet Services and Applications*, vol. 3, no. 2, pp. 215–232, 2012.

[3] D. Grela, K. Sapiecha, and J. Strug, "A Fault Injection-based Approach to Assessment of Quality of Test Sets for BPEL Processes," in *Proceedings of the International Conference on Evaluation of Novel Approaches to Software Engineering*, Angers, France, Jul. 2013, pp. 81–93.

[4] C. Pham, L. Wang, B. C. Tak, S. Baset, C. Tang, Z. Kalbarczyk, and R. K. Iyer, "Failure Diagnosis for Distributed Systems using Targeted Fault Injection," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 2, pp. 503–517, 2017.

[5] K. Farj, Y. Chen, and N. A. Speirs, "A Fault Injection Method for Testing Dependable Web Service Systems," in *IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing 2012*, ser. ISORC 2012. Shenzhen, China: IEEE, Apr. 2012, pp. 47–55.

[6] A. C. C. Machado and C. A. G. Ferraz, "Guidelines for Performance Evaluation of Web Services," in *WebMedia '05: Proceedings of the 11th Brazilian Symposium on Multimedia and the web*, ser. WebMedia '05 2005. Minas Gerais, Brazil: ACM, Dec. 2005, pp. 1–10.

[7] A. Bala and I. Chana, "Fault Tolerance - Challenges, Techniques and Implementation in cloud Computing," *International Journal of Computer Science*, vol. 9, no. 1, pp. 288–293, 2012.

[8] H. Hamad, M. Saad, and R. Abed, "Performance Evaluation of RESTful Web Services for Mobile Devices," *International Arab Journal of e-Technology*, vol. 1, no. 3, pp. 72–78, 2010.

[9] L. Feinbube, L. Pirl, and A. Polze, "Software Fault Injection: A Practical Perspective," in *Dependability Engineering*, ser. IntechOpen Book Series. IntechOpen, 2018, pp. 47–60. [Online]. Available: https://doi.org/10.5772/INTECHOPEN.70427

[10] A. K. Pandey, A. Kumar, and S. Shukla, "A Novel Framework for Reliable and Fault-tolerant Web Services," *International Journal of Recent Technology and Engineering*, vol. 7, pp. 67–73, 2019.

[11] J. Qian, H. Wu, H. Chen, C. Li, and W. Li, "Fault Injection for Performance Testing of Composite Web Services," *International Journal of Performability Engineering*, vol. 14, no. 6, pp. 1314–1323, 2018.

[12] M. C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault Injection Techniques and tools," *IEEE Computer*, vol. 30, pp. 75–82, 1997.

[13] F. Bessayah, A. Cavalli, W. Maja, E. Martins, and A. W. Valenti, "A Fault Injection Tool for Testing Web Services Composition," in *Proceedings of the 5th International Academic and Industrial Conference on Testing – Practice and Research Techniques*, ser. LNCS 2010. Windsor, UK: Springer, Sep. 2010, pp. 137–146.

[14] B. Kannan and L. E. Parker, "Fault-tolerance Based Metrics for Evaluating System Performance in Multi-Robot Teams," in *Proceedings of Performance Metrics for Intelligent Systems Workshop*, Aug. 2006, pp. 1–8.

[15] J. Erickson and K. Siau, "Web Services, Service-Oriented Computing, and Service-Oriented Architecture: Separating Hype from Reality," *Journal of Database Management*, vol. 19, no. 3, pp. 42–54, 2008.

[16] S. Arya and M. Yadav, "Service–oriented Architecture: Concepts and Challenging Issues," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4, no. 7, pp. 1219–1224, 2014.

[17] M. Khanbabaei and M. Asadi, "Principles of Service-Oriented Architecture and Web Services Application in order to Implement Service-Oriented Architecture in Software Engineering," *Australian Journal of Basic and Applied Sciences*, vol. 5, no. 11, pp. 2046–2051, 2011.

[18] S. Gupta and P. Bhanodia, "A Flexible and Dynamic Failure Recovery Mechanism for Composite Web Services using Subset Replacement," *International Journal of Science and Research*, vol. 3, no. 12, pp. 1886–1890, 2014.

[19] A. S. Mustafa1 and Y. S. Kumaraswamy, "Performance Evaluation of Web-Services Classification," *Indian Journal of Science and Technology*, vol. 7, no. 10, p. 1674–1681, 2014.

[20] S. Mumbaikar and S. Padiya, "Web Services Based on soap and rest Principles ," *International Journal of Scientific and Research Publications*, vol. 3, no. 5, pp. 1–4, 2013.

[21] N. Laranjeiro and M. Viera, "Deploying Fault-Tolerant Web Services Compositions," *International journal of computer systems science and engineering*, pp. 1–4, 2008.

[22] A. Carzaniga, A. Gorla, and M. Pezze, "Handling Software Faults with Redundancy," in *Architecting Dependable Systems VI*, ser. LNCS 5835. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 148––171. [Online]. Available: https://doi.org/10.5772/INTECHOPEN.70427

[23] S. M. Hosseini and M. G. Arani, "Fault-tolerance Techniques in Cloud Storage: A Survey," *International Journal of Database Theory and Application*, vol. 8, no. 4, pp. 183–190, 2015.

[24] P. Kumari and P. Kaur, "A Survey of Fault Tolerance in Cloud Computing," *Journal of King Saud University – Computer and Information Sciences*, vol. 33, no. 10, pp. 1159–1176, 2018.

[25] M. R. Chinnaiah and N. Niranjan, "Fault Tolerant Software Systems using Software Configurations for Cloud computing," *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 7, no. 3, pp. 1–18, 2013.

[26] D. Khurana and A. P. Shukla, "A Survey on Fault Injection and Debugging," *International Journal of Innovations in Engineering and Technology*, vol. 7, no. 3, p. 283–288, 2016.

[27] M. Kooli and G. D. Natale, "A Survey on Simulation-Based Fault Injection Tools for Complex systems," in *2014 9th IEEE International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*. Santorini, Greece: IEEE, May 2014, pp. 1–6.

[28] P. M. Broadwell, "Response Time as a Performability Metric for Online Services," Ph.D. dissertation, The University of California, California, USA, 2004.

[29] S. Hussain, Z. Wang, I. K. Toure, and A. Diop, "Web Service Testing Tools: A Comparative Study," *International Journal of Computer Science*, vol. 10, no. 1-3, pp. 641–647, 2013.

[30] G. K. Saha, "Transient Fault Tolerance in Mobile Agent Based Computing," *INFOCOMP Journal of Computer Science*, vol. 4, pp. 1–11, 2005.

[31] G. K. Saha, "Software Based Fault Tolerant Computing using Redundancy," *International Journal of the Computer, the Internet and Management*, vol. 17, no. 3, pp. 41–46, 2009.

[32] A. B. Alvi, M. A. Hashmi, Z. H. Chuban, M. Atif, and I. Ahmed, "Adaptive Byzantine Fault Tolerance Support for Agent Oriented Systems: The BDARX," *International Journal of Advanced and Applied Sciences*, vol. 6, no. 2, pp. 57–64, 2009.

[33] M. Calisti, F. Dignum, R. Kowalczyk, F. Leymann, and R. Unland, "Service-oriented Architecture and Multi-Agent Systems Technology," in *Dagstuhl Seminar Proceedings 10021*. Schloss Dagstuhl, Germany: Leibniz-Zentrum für Informatik, Jan. 2010, pp. 1–16.

[34] K. Potiron, A. E. F. Seghrouchni, and P. Taillibert, *From Fault Classification to Fault Tolerance for Multi-Agent Systems*. Springer, London: Springer, Jun. 2013.

[35] F. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*, ser. Wiley Series in Agent Technology. West Sussex, England: John Wiley and Sons, Inc., Aug. 2007.

[36] A. O. Erlank and C. P. Bridges, "A Hybrid Real-Time Agent Platform for Fault-Tolerant, Embedded Applications," *Auton Agent Multi-Agent Syst*, vol. 32, p. 252–274, 2018.

[37] S. Dahling, L. Razik, and A. Monti, "Enabling Scalable and Fault Tolerant Multi Agent Systems by Utilizing Cloud Native Comput-

ing," *Autonomous Agents and Multi-Agent Systems*, vol. 35, no. 10, pp. 1–27, 2021.

[38] A. S. S. Guna, "Performability Issues of Fault Tolerance Solutions for Message-Passing Systems: the Case of RADIC," Ph.D. dissertation, Universitat Autonoma de Barcelon, California, USA, 2009.

[39] P. Leitao, S. Karnouskos, L. Ribeiro, J. Lee, T. Strasser, and A. W. Colombo, "Smart Agents in Industrial Cyber-Physical systems," in *Proceedings of the IEEE*. IEEE, May 2016, pp. 1086–1101.

[40] M. O. Shafiq, y. Ding, and D. Fensel, "Bridging Multi-Agent Systems and Web Services: Towards Interoperability between Software Agents and Semantic Web Services," in *In Proceedings of the 10th IEEE International Conference on Enterprise Distributed Object Computing (EDOC)*. Hong Kong, China: IEEE Computer, Oct. 2006, p. 85–96.

[41] F. O. Oliha, "A Fault Tolerant Architecture for Web Services Solutions," Ph.D. dissertation, University of Benin, Benin City, Nigeria, 2018.

[42] S. Aghaei, M. R. Khayyambashi, and M. A. Nematbakhsh, "A Fault-Tolerant Architecture for Web Services," in *2011 International Conference on Innovations in Information Technology*. Abu Dhabi, United Arab Emirates: IEEE, Apr. 2011, p. 53–56.

[43] N. Looker, M. Munro, and J. Xu, "Testing Web Services," in *Proceedings of the 16th IFIP International Conference on Testing of Communicating Systems*. Oxford: IEEE, Apr. 2004, pp. 1–5.

[44] H. Zaide, R. Ayoubi, and R. Velazco, "A Survey on Fault Injection Techniques," *The International Arab Journal of Information Technology*, vol. 1, no. 2, p. 171–186, 2004.

[45] M. Hossain, "Web Service-Based Software Implemented Fault Injection," *Information Technology Journal*, vol. 5, no. 1, p. 138–143, 2006.

[46] R. Ramakrishnan, J. Anbarasi, and V. Kavitha, "Soapui and SOAP Sonar Testing Tool using Vulnerability Detection of Web Service," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 2, no. 11, p. 6995–7002, 2014.

[47] K. Umadevi and S. B. Rajakumari, "A Review on Software Fault Injection Methods and tools," *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 3, no. 3, p. 1582–1587, 2015.

[48] K. Nagaraja, X. Li, R. Bianchini, R. P. Martin, and T. D. Nguyen, "Using Fault Injection and Modeling to Evaluate the Performability of Cluster-Based Services," in *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, ser. USITS 03. Seattle, WA, USA: USENIX Association, Mar. 2003, pp. 1–14.

[49] P. G. Hoel, *Introduction to Mathematical Statistics*, ser. 5th Edition. USA: John Wiley and Sons, Inc., Jun. 1996.

[50] F. Oliha, "Guaranteeing Performance in a Fault Tolerant Architecture Solution using Software Agent's Coordination," *Journal of Information and Communication Technology*, vol. 21, no. 4, p. 595–625, 2022.

[51] D. Zuquim, G. Garcia, and M. B. F. D. Toledo, "An Architecture for Fault-Tolerant and Service-Based Business Processes," in *IEEE 11th International Conference on Computational Science and Engineering*, ser. Brazilian workshop on Business Process Management. Gramado, Brazil: IEEE, May 2007, pp. 1–13.

[52] A. Sari and M. Akkaya, "Fault Tolerance Mechanisms in Distributed Systems," *International Journal of Communications, Network and System Sciences*, vol. 8, p. 471–482, 2015.

[53] M. I. Ladan, "Web Services Metrics: A Survey and A Classification," in *International Conference on Network and Electronics Engineering IPCSIT, Vol. 11*. Singapore: IACSIT Press, Jan. 2011, pp. 93–98.

[54] A. Bora and T. Bezboruah, "A Comparative Investigation on Implementation of RESTful Versus SOAP-Based Web Services," *International Journal of Database Theory and Application*, vol. 8, no. 3, pp. 297–312, 2015.

[55] M. Kumar, "Various Factors Affecting Performances of Web Services," *International Journal of Sensor and Its Applications for Control Systems*, vol. 3, no. 2, p. 11–20, 2015.

Festus O. Oliha is an academician with the Departement of Computer Science, University of Benin, Nigeria. He is into Software Engineering and Quality Assurance for Service-Oriented Systems. His research interest centres around performability computing for software systems and data solutions. He is a member of ACM and IEEE. ORCID address at https://orcid.org/0000-0001-5772-6919.