IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 8, 2017, accepted September 17, 2017, date of publication September 26, 2017,
date of current version February 28, 2018.

Digital Object Identifier 10.1109/ACCESS.2017.2756867

Autonomic Web Services Enhanced by Asynchronous Checkpointing

MARIANO VARGAS-SANTIAGO[1], LUIS MORALES-ROSALES[2], SAÚL POMARES-HERNÁNDEZ[1,3,4], AND KHALIL DRIRA[5]

[1]Department of Computer Science, National Institute for Astrophysics, Optics and Electronic, Puebla 72840, Mexico
[2]Faculty of Civil Engineering, Universidad Michoacana de San Nicolás de Hidalgo, Morelia 58030, Mexico
[3]CNRS, LAAS, 54200 31031 Toulouse, France
[4]LAAS, University of Toulouse, 54200 31031 Toulouse, France
[5]LAAS, 54200 31031 Toulouse, France

Corresponding author: Luis Morales-Rosales (lamorales@conacyt.mx)

This work was supported by the National Council for Science and Technology of Mexico under Grant PDCPN2013-01-215421.
The work of M. Vargas-Santiago was supported by CONACyT through CVU under Grant 417439.

ABSTRACT The evolution of business software technologies is constant and is becoming increasingly complex which leads to a great probability of software/hardware failures. Business processes are built based on web services as they allow the creation of complex business functionalities. To attack the problem of failures presented by the use of web services, organizations are extrapolating the autonomic computing paradigm to their business processes as it enables them to detect, diagnose, and repair problems improving dependability. Sophisticated solutions that increase system dependability exist, however, those approaches have drawbacks; for example, they affect system performance, have high implementation costs, and or they may jeopardize the scalability of the system. To facilitate evolution to self-management, systems must implement the monitoring, analyzing, planning, and execution (MAPE) control loop. An open challenge for MAPE loop is to carry out in an efficient manner the diagnosis and decision-making processes, recollecting data from which the system can detect, diagnose, and repair potential problems. Also, dealt by systems dependability, specifically as fault tolerant mechanisms. One useful tool for this purpose is the communication induced checkpointing (CiC). We use CiC in attacking the dependability problem of using web services in a distributed and efficient manner. First, we present an approach for web services compositions that supports fault tolerance based on the CiC mechanism. Second, we present an algorithm aimed at web services compositions based on an autonomic computing and checkpointing mechanism. Experimental results support the feasibility of this concept proposal.

INDEX TERMS Autonomic computing, web services, autonomic systems, Internet technologies, checkpointing.

I. INTRODUCTION

Businesses are always searching for ways to improve overall competitiveness. In developing business software, new technologies help the evolution and growth of businesses. For large-scale systems, the entire system becomes less reliable and the fault rate probability augments as the number of components increase [1]. Usually under these circumstances Web Services carry out complex business processes known as compositions which frequently work with the Business Process Execution Language (BPEL) engine for orchestration [2]. These Web Services may be locally deployed or they may be used collaboratively with other Web Services and are distributed over heterogeneous environments, which this the most common situation seen in real world deployments. Distributed enterprise applications are usually built following the Service Oriented Architecture (SOA) paradigm [3]. Following the SOA architectural model for composite services, it dynamically enables the creation of distributed software intensive systems to be built from a combination of diverse independently developed services [4]. SOA may be used as the platform for distributed systems and a bridge between all these spatially separated, loosely-coupled services that can be easily discovered through a well-known interface. These interfaces are leveraged by the Web Services paradigm which follow and apply open-standards for interoperability; specifically, Web Services support service

composition and application evolution [5]. Even though Web Services are used within complex business collaborative environments, they are error prone because of unreliable Internet behavior during run-time while they are still required to function correctly and be available on demand. Failures may lead to terrible consequences such as augmenting execution time, higher costs to run applications, destroyed systems, or system breaches. As a consequence, organizations must maintain a way to make their systems or business processes as dependable as possible before they intend to automate them [6]. In anticipating these errors, organizations using core Web Services for their business processes require efficient and seamless solutions. In order to attack the problem of failures presented by Web Services, organizations are extrapolating the Autonomic Computing Paradigm into their business processes as it enables them to detect, diagnose, and repair problems and therefore improve system dependability.

IBM introduced the *Autonomic Computing* paradigm, where systems are considered to be self-manageable; these should be able to perform self-maintenance and corrective actions with minimal human intervention [7]. In order to facilitate evolution to self-management, Kephart proposed the MAPE (Monitoring, Analysis, Planning and Execution) control loop [8]. IBM introduced the notion of Autonomic Managers and Managed Elements. Autonomic Managers are responsible for interaction and communication with the outside world; this may be interpreted as human computer interaction, interactions with other elements, or as a bridge between the managed elements. The Monitoring phase focuses on recollecting data, which afterwards is used by the Analysis phase for diagnosis purposes to identify the cause of a detected symptom. The Planning phase must plan ahead, if possible, what actions to take. Finally, the Execution phase executes actions based on previous phases.

Presently, sophisticated solutions exist for dependability enhancement of Web Services. However, these approaches have drawbacks; these include: affected system performance, potentially high implementation costs [5], and/or they may jeopardize system scalability [9]. One open challenge for Web Services is increasing reliability [4]; dependability is based upon reliability, availability, security, and maintainability [10].

To facilitate evolution to self-management, systems must implement the MAPE control loop. An open challenge for the MAPE loop is to carry out in an efficient manner the diagnosis and decision-making processes, recollecting data from which the system can detect, diagnose and repair potential problems. Also, dealt by systems dependability, specifically as fault tolerant mechanisms. One useful tool for this purpose is the Communication induced Checkpointing (CiC). For Web Service composition CiC is used so that while Monitoring and Analyzing data exchanged between processes the system can save a consistent global snapshot (CGS) or rollback to a previous CGS.

In this paper, we propose the use of CiC in attacking the dependability problem of using Web Services and tackling the issues with the MAPE loop of autonomic computing in a distributed and efficient manner. CiC has proved to be a worthy method that has gained vast maturity improvements, advances especially for distributed systems by solving issues of rollback recovery, software debugging and software verification [11]. CiC focuses on building CGSs or checkpoints (one from each process) and avoiding dangerous patterns such as zigzag paths and zigzag cycles [12]. The purpose of building CGS is to checkpoint processes on nonvolatile storage to allow for system rollback or the survival of process failures. Therefore for decoupled systems, an architectural model type has been chosen; the autonomic behavior may be incorporated into preexisting non-autonomous systems, such as Web Services. Checkpointing protocols may be applied in this category. Research leads us to propose merging the following technologies.

- Autonomic Computing which provides incorporates the MAPE control loop for self-manageable systems.
- Web Services which follow open standards for interoperability.
- Checkpointing Protocols which save consistent states to which a system may rollback in the event of undesirable system function.

In this paper, we first introduce and justify the motivation for our specific area of research in previous Section. In Section II, we analyze the model for checkpointing and in Section III, we discuss related state-of-the-art works that increase Web Services dependability. Section IV analyses the environment and considerations for the proposed solution; a case study is presented as well as an algorithm. Section V illustrates the results to support our approach and shows feasibility. Finally, conclusions are drawn in Section VI where we also suggest further work research.

II. THE MATHEMATICAL MODEL

A. COMMUNICATION PATTERNS

Distributed systems have the following characteristics: there is no global notion of time, processes do not share common memory and communication is solely by message passing. In this context, distributed computation consists of a finite set of processes where $P = \{P_1, P_2, \ldots, P_n\}$, and our focus is the enhancement of Web Services. We assume that channels have unpredictable yet finite transmission delays which are both reliable and asynchronous. To have the ability of using checkpointing mechanisms, two types of events must be considered: *internal* and *external*.

Internal events are those that change the processes states; for instance a checkpoint, which is a finite set of internal events is denoted by E_i. External *send* and *delivery* events are those that affect the global state of the system. Let m be a message, $send(m)$ is the emission of m by a process $p \in P$ and $delivery(q, m)$ is the delivery event of m to participant $q \in P$ where $p \neq q$. The set of events associated to M is

$$E_m = \{send(m) : m \in M\} \cup \{delivery(p, m) : m \in M \wedge p \in P\}$$

Thus the whole set of events is

$$E = E_i \cup E_m$$

B. DEFINITIONS

1) THE HAPPENED BEFORE RELATION

This relation establishes causal precedence dependencies over a set of events. The HBR is a strict partial order (transitive, irreflexive and antisymmetric). It is defined as follows [13]:

Definition 1: The HBR, denoted by $\rightarrow$, is defined by the following three rules and it is the smallest relation on a set of events E.

1) If a and b are events of the same process, and a was originated before b then $a \rightarrow b$.
2) If a is the event *send(m)* and b is the event *delivery*(p_i, m) then $a \rightarrow b$.
3) If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$.

The partially-ordered set $\widehat{E} = (E, \rightarrow)$ models the distributed computation.

2) IMMEDIATE DEPENDENCY RELATION

The HBR, in practice, is expensive since it has to keep track of the relation between each pair of events. In order to avoid such, the Immediate Dependency Relation (IDR) identifies and attaches a minimal amount of control information per message to ensure causal ordering. The IDR is the transitive reduction of the HBR and is denoted by "$\downarrow$", defined as follows [14]:

Definition 2: Two messages a and b $\in$ E have an IDR a$\downarrow$b if the following restriction is satisfied:

$$\text{a} \downarrow \text{b} \Leftrightarrow [a \rightarrow b \wedge \forall c \in E, \neg(a \rightarrow c \rightarrow b)]$$

3) CHECKPOINT AND COMMUNICATION PATTERN CCP

Represented by its distributed computation, it consists of a set of incoming and outgoing messages and associated local checkpoints [15]

Definition 3: A communication and checkpoint pattern (CCP) is a pair $(\widehat{E}, E_i)$ where $\widehat{E}$ is a partially-ordered set modeling a distributed computation and E_i is a set of local checkpoints defined on $\widehat{E}$.

An example of a CCP is exhibited in Fig. 1; showing the *checkpoint interval* denoted I_k^x, with a sequence of events occurring at p_k between C_k^{x-1} and C_k^x $(x > 0)$

C. BUILDING CONSISTENT GLOBAL SNAPSHOTS

In distributed systems, processes take their checkpoints independently so the system can restart execution from the last saved consistent global snapshot (CGS) in case of a failure.

Definition 4: A CGS does not contain any HBR causally related checkpoints, in other words, for any pair of checkpoints A and B satisfies that:

$$\neg(A \rightarrow B) \wedge \neg(B \rightarrow A)$$

As stipulated by Netzer and Xu in [16].

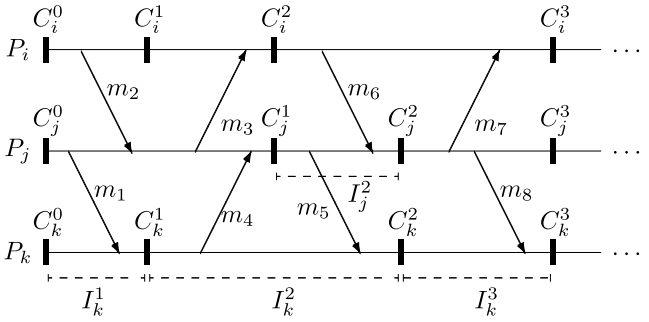

FIGURE 1. A Checkpoint and Communication Pattern (CCP) [12].

III. RELATED WORKS

Dependability has been widely researched; a whole taxonomy about it may be found in [19]. Many different aspects of the system affects its dependability, including: reliability, availability, security, maintainability and systems considered to be self-manageable [10]. To date none address the self-manageable issues for systems and there is no unified or standardized form for implementing the MAPE control loop within Web Services. Some works treat each one of the MAPE control loop phases as an individual Web Service [20], while others only tackle the self-healing feature of autonomic computing [21]. In [22], the authors attempt to address the entire MAPE control loop process by adding extra interfaces to confront functional and nonfunctional requirements of Web Services.

Marzouk *et al.*, illustrate [2], that Web Services are implemented to follow business logic; they could be deployed either locally inside an enterprise or they could cooperate in a distributed environment. Either way incorporating Web Services are crucial to the process solution and must implement fault tolerance mechanisms. The authors principal concern is ensuring self-adaptability of composite services. An approach for adapting Web Service(s) configuration based on *strong mobility* code has been provided and is achieved with generic source code transformation. It has been determined or recognized that a strong mobility methodology for Web Services BPEL processes requires periodic checkpointing. When self-adaptability is a necessity, one or several instances of the orchestration process will be migrated to the previous checkpoint and processes will be resumed starting from the interruption point; this is to say, that the system process will begin from the last captured checkpoint. Using strong mobility requires replicating services, when possible; in other cases finding similar services becomes a complicated task due to services formats. For example one service uses an integer instead of a float.

The Marzouk *et al.* [18] approach pursues a self-healing property where in case of failure the orchestration process is migrated to a different server and in case of QoS violation a subset of running instances may be migrated to a new server in order to decrease the initial host load. The authors argue that, their solution has no risk of non-deterministic

TABLE 1. Web services and checkpointing.

Parameter / Article	[2]	[6]	[17]	[18]
Goal	Ensure self-adaptivity of composite services in particular for BPEL.	Restore attacked business processes with the aid of fault-tolerance based on checkpoint and rollback.	Increase dependability of business processes.	Propose a solution for strong mobility of composed WS.
Environment	Distributed enterprises business logic.	Distributed systems (Businesses).	Distributed systems.	Web Services orchestration.
Evaluation	The travel agency (BPEL orchestration).	Recovery Time.	Recovery Time.	The travel agency (BPEL).
Used CP	?	?	?	Checkpointing policies.
Recovers	BPEL process.	Business processes.	Performance time in terms of message delivery.	Orchestration process using self-healing.

execution when recovering flow activities since a unique state is always saved for each instance. But, a recovery state is built after synchronizing all flow branches which permits saving a consistent checkpoint. Yet, in our opinion, using synchronization for constructing a consistent checkpoint makes this approach expensive because of the barrier imposed from synchronizing; hence this solution is slow and with no concurrency. For this reason, other proposed works do not use checkpointing techniques and have to restart the whole Web Service composition or orchestration when there is a fault.

Vaca and Gasca identified [6] that executing business processes are susceptible to intrusion attacks, which may cause severe faults. Fault tolerance techniques tackle such issues, decreasing risk of faults and therefore being more dependable; the aim is to achieve dependability in business processes automation. The authors claim that, to resists faults related to integrity attacks, fault tolerance techniques may be applied. Varela and Martnez proposed OPUS: Fault tolerance against integrity attacks in business processes, a framework with many capabilities developed following the Model-Driven Development (MDD) and the Model Driven Architecture (MDA). This framework has four layers: Modeling, Application, Fault Tolerance and Services. Where the Fault Tolerance layer is based on checkpointing and rollback recovery. However, the authors do not mention which checkpointing mechanism they use; often new and improved checkpointing mechanisms are proposed in state-of-the-art literature. We believe that recovery of overhead time may be reduced making use of such improved protocols.

Varela *et al.* [17] argue that companies need to communicate internally exchanging information between business logics, thus they seek to deploy a Business Process Management System (BPSM). A BPSM aids in automation of business processes, but in this context systems are error prone and may not guarantee perfect execution over time. Therefore a new paradigm Business Process Management (BPM) has arisen; it is defined as a set of concepts, methods and techniques to aid the modeling, design, administration, configuration, enactment and analysis of business processes. In implementing the business process life cycle the BPM paradigm follows diverse stages: design and analysis, configuration, enactment and diagnosis. However each stage may introduce different kinds of faults. It is indispensable for companies to gain dependability in early design stages and promote the reduction of possible faults and risks. In this work, the authors propose following traditional fault tolerant ideas such as replication and checkpointing while focusing on service-oriented business processes. Such approaches require the introduction of extra components (sensors) into the business process design and therefore extra time to check each sensor and recovery of business process service in rollback.

Table 1 summarizes related works. It exhibits the aim, the technology used and the environment under which the authors recommend solutions. Despite the differing proposals, some open questions remain such as: How to integrate Web Services in dynamic environments in an autonomic way. Are checkpoints taken for rollback recovery strategies consistent? Are the proposed solutions negatively impacting system performance?

IV. AUTONOMIC WEB SERVICES BASED ON ASYNCHRONOUS CHECKPOINTING MECHANISM

A. ARCHITECTURE

The proposed approach is suitable in distributed heterogeneous environments; it leverages the Enterprise Service Bus (ESB) infrastructure which provides and integration. Additionally, the ESB ensures interoperability and offers several features including: service discovery, intelligent routing, message processing and service orchestration assuring proper format between service providers and consumers regardless of which programming language is used [23].

Functional properties, or the functional contract, of a Web service may be exposed by the Web Service Description Language (WSDL); this dictates how the service must behave.

Whereas the non-functional properties of a Web Service are represented by the Quality of Service (QoS) parameters; this is also true for Web Service composition, which must be monitored and analyzed in order to determine whether or not the service is functioning in an adequate manner. Monitoring an individual Web Service and global Web Services compositions is challenging because of the distinctiveness presented in each case since each Web Service is unique. Performance parameters can be monitored under diverse scopes; the client may obtain various parameters including latency, throughput or error rate.

The architecture is designed to provide interoperability between diverse services and systems comprised of different technologies through standard-based adapters and interfaces that use Web Service technology (as shown in Fig. 2). Furthermore, each Web Service follows the MAPE control loop from the autonomic computing paradigm. The service layer represents the petition or the execution of a required task or service from which performance information will be extracted. In particular, we use the active component approach for monitoring and analyzing activities related to Web services performance measurements, for example we monitor non-functional requirements periodically or when processes exchange messages.

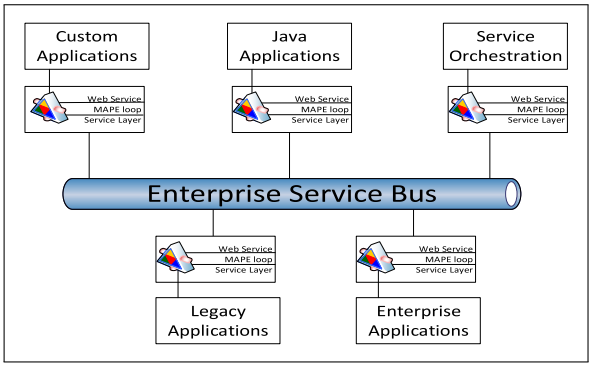

FIGURE 2. ESB with MAPE loop Architecture.

B. PERFORMANCE MEASUREMENTS

We have proposed an asynchronous checkpointing mechanism to address fault tolerance, therefore, we measure systems performances before and after implementing the CiC mechanism system architecture.

We have chosen performance measurements pertinent to network traffic to evaluate performances before and after applying CiC architecture. We measured average response time AVG_{RT} and throughput β_t , in terms of Transactions per Second (TPS), in order to measure application service and CiC performance. Also, the number of forced checkpoints where measured as performance indicator.

1) AVERAGE RESPONSE TIME (AVG_{RT})

Defined as the average time taken by a Web Service from the time the client sends a request until the time that the reply is received from the Application Server. To calculate the response time, we use two timestamps: one, when the client sends a request (t_1) and two, the time when the response is received (t_2). Then the response time is calculated as:

$$RT = t_2 - t_1 \tag{1}$$

This is done for each request/response which is in play in the system consequently obtaining AVG_{RT}.

2) THROUGHPUT (β_t)

For interactive systems, the system throughput is defined as the ratio of the total number of requests for the total time which has a correlation with the response time. We define Web Services system performance β_t as the amount of data processed by a Web Service in a given time interval.

3) NUMBER OF FORCED CHECKPOINTS

Implementing CiC brings benefits to Web services, however, there exists a trade-off for checkpoints when the system is behaving correctly and when the system suffers degradation. Plus, storage space must be considered for applications that lack resources, therefore proposed solutions must checkpoint efficiently.

For such reason we measure the number of forced checkpoints, because checkpoints must belong to a CGS, and implementing CiC we can be certain that if a checkpoint is taken it is most likely to belong to a CGS. Since, CiC follows a safe checkpointing approach by triggering certain checkpoints to avoid zigzag paths and z-cycles, such rules and safe conditions can be found in [12].

C. MAPE CYCLE

Business processes must be dependable so they are available when requested; solutions that suggest augmenting Web services dependability must also be scalable and even autonomic [22]. In this work we propose an approach which follows the autonomic computing MAPE cycle based on CiC protocols (as shown in Fig. 3).

As illustrated in Fig. 3, the Monitoring module will initiate the petition, sending a request through the Enterprise Service Bus (ESB) system, which is in charge of routing, adapting or mediating the request if necessary; this is the service layer. Then the Monitoring module computes the QoS parameters as response time and throughput. These events are then converted into XML-based messages and stored in a common knowledge base which is shared by all Web Services.

The Analyze module uses a *Diagnostic* Engine that checks the extracted information to decide if the behavior of the Web Service is normal or if it suffers any anomaly or fault. In other words, it identifies patterns in the logs by looking for specific problems that occurred [20]. If the Web service presents normal parameters then it is immediately returned from the Analysis module to the Web Service invoking a new service. However, the message is forwarded to the analysis process which checks the new aggregate values with the aim of predicting the immediate future state of the system, based

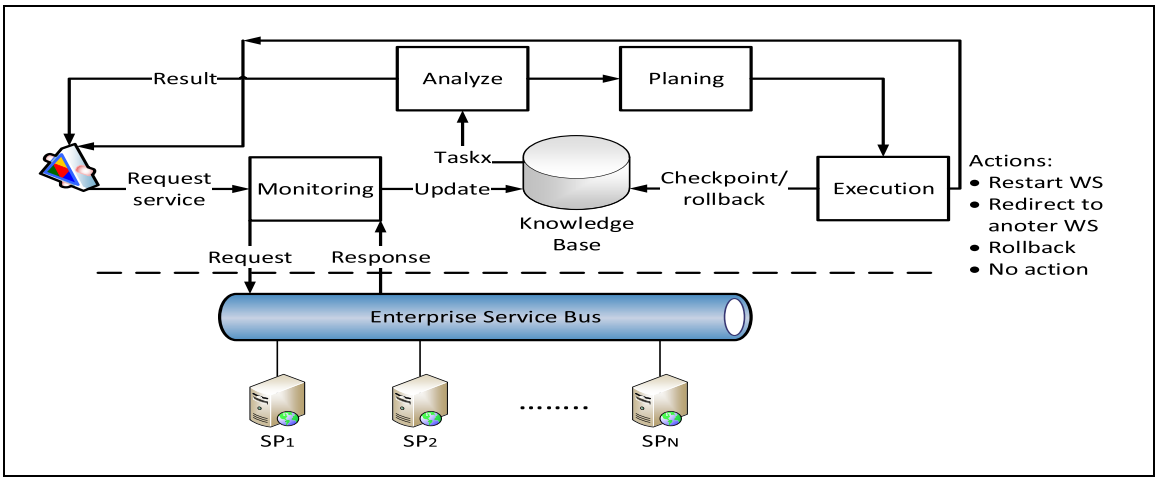

FIGURE 3. Autonomic Web services based on CiC protocols.

on Hidden Markov Model (HMM) [24], Bayesian Networks [25], [26] or any other method that supports prediction. When the QoS parameters are predicted as abnormal, an alert will be sent to the scheduler who will set up the next forced checkpoint(s). So that later the Execution module realizes, and processes them. After that if degradation occurs and the system cannot continue then it would enter the rollback recovery stage.

D. SCENARIO

Autonomic Web services based on communication induced checkpointing can be better explained through an example like the *Stock Quote* composite Web Service system. Consumers of a service solicit multiple stocks brokers to determine in which stocks they should invest; clients that pay a subscription, premium users, are able to receive real-time stock quote services. Fig. 4 depict a case where two service consumers make a petition to a stock broker or service provider, showing the need for building CGS.

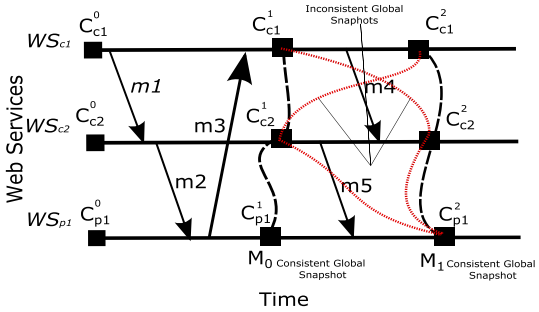

FIGURE 4. Example Scenario.

As stipulated by Netzer and Xu, when both zigzag and causal paths are present CGS cannot be identified [16]. Fig. 4 also illustrates that M_0 and M_1 build CGS while C^1_{c1}, C^2_{c2} and C^2_{p1} cannot be part of a CGS; because of messages $m4$ and $m5$; although no causal path exists between C^1_{c1} and C^2_{p1} a zigzag path has been formed by the aforementioned messages. This means that no CGS can be formed from the checkpoints involved in a zigzag path, in other words no CGS can be built that containsC^1_{c1} and C^2_{p1}.

E. ALGORITHM

We present Algorithm 1 that targets implementation of an autonomic computing paradigm and integration of a checkpointing mechanism. The aim of our algorithm is to address the MAPE loop for Web Services composition.

With Algorithm 1 we suggest initializing all variables (lines 1 to 5); line 4 builds and assigns to C the systems initial CGS. The MAPE control loop is represented by lines 6 to 31; for each Web Service, the algorithm starts the *Monitoring* step by checking Web Service policies and the process level at the time that the *CHECKPOLICIES* are called while confirming that Web Services perform based on the constraints stipulated in their policies. At any time during this computation period a new CGS may be built. *VALIDATE* is used to detect a set of symptoms returning a non-empty set when the Web Service specifications and policies are not met. When the entire composite is functioning correctly a no problem reply is returned (lines 10 and 11). Otherwise in order to have a consistent view of the system and to have a proper verification and diagnostic result, the last known CGS is retrieved from the common Knowledge Base (KB), line 13.

The vector containing the set of symptoms is *Analyzed* in order to find the best known diagnosis, retrieved from the KB; this is done for each individual Web Service within the composition. When no diagnosis is found, line 17, the symptoms are classified, line 18, and a new diagnosis is generated as well as a new plan, line 19. Contrarily, when the diagnostic is found, line 21, a *Plan* is retrieved from the KB. This is suitable for an individual Web Service, however with a composite Web Service, the system must build a global diagnostic demonstrating how the overall composite is behaving, line 26. The same case is applied to individual plans;

Algorithm 1 Autonomic Composite Web Services

1: **procedure** Initialization
2: Initially
3: $S \leftarrow \emptyset, P \leftarrow \emptyset, D \leftarrow null$
4: $C \leftarrow BUILD\_INITIAL\_CGS(WS_1, WS_2, \ldots, WS_n)$
5: **end procedure**
6: **procedure** MAPE($(WS_1, WS_2, \ldots, WS_n)$)
7: **for** $WS[i], i = 1$ to n **do**
8: $FWS[i] \leftarrow CHECKPOLICIES(m)$
9: **end for**
10: **if** $FWS[i] == \emptyset \forall FWS[i], i = 1, 2, \ldots, n$ **then**
11: **return** "No problem found for Composite WS"
12: **else**
13: GET_LAST_CGS((C))
14: **for** $\forall FWS[i], i = 1, 2, \ldots, n$ **do**
15: **if** $FWS[i] \neq \emptyset$ **then**
16: $D[] \leftarrow FIND\_BEST\_DIAGNOSTIC(FWS[i])$
17: **if** $D[] = NULL$ **then**
18: $D[] \leftarrow CLASSIFY\_SYMPTHOM(FWS[i])$
19: $P[] \leftarrow GENERATE\_PLAN(D[])$
20: **else**
21: $P[] \leftarrow FIND\_BEST\_PLAN(D[])$
22: **end if**
23: **end if**
24: **end for**
25: **end if**
26: $D_G \leftarrow BUILD\_GLOBAL\_DIAGNOSTIC(D[])$
27: $P_G \leftarrow FIND\_BEST\_GLOBAL\_PLAN(DG[]) \cap P[]$
28: **for** $action \in P_G$ **do**
29: $EXECUTE(action, P_G)$
30: **end for**
31: **end procedure**
32: **procedure** CheckPolicies(m)
33: $wsdescr \leftarrow GET\_WS\_DESCR(wsid[e])$
34: $wspolicy \leftarrow GET\_WS\_POLICY(wsdescr)$
35: $processpolicy \leftarrow GET\_PROCESS\_POLICY(pid[e])$
36: $policy \leftarrow processpolicy \cap wspolicy$
37: $S \leftarrow validate(m, policy, wsdescr)$
38: **return** (S)
39: **end procedure**

a global plan must be generated from the know plans and for the overall system, line 27. Finally, each action must be *Executed* from the overall global plan carrying out a series of actions, lines 28 to 29, such as rollback or restart a specific Web Service.

V. RESULTS AND DISCUSSION

In order to show that autonomic Web Services based on asynchronous checkpointing (specifically communication-induced checkpointing) do not have a great impact on the overall performance of the systems, we conducted several performance tests. Specifically, we measured *response time* and *transactions per second* as key performance indicators.

For testing we implemented our proposed solution using the following hardware: a workstation with a 16 GB RAM using a Windows 7 64-bit Operating System. The WSO2 Application Server was used to deploy Web Services; for performances tests, diverse concurrent Java clients were emulated in order to replicate approximate a real world setting.

A. EXPERIMENTAL RESULTS

Fig. 5 and Fig. 6 show the behavior of the system when evaluating performance. For this purpose, each scenario was executed 100 times; for 20 consumers, 2000 samples were collected and for 30, 3000 were collected in increments of 10 to reach 200 where a total of 20,000 samples was collected. Subsequently, an average was obtained based on response time and transactions per second. Response time measured the time from when the customer sent a request to the credit approval service until he received a response. Transactions per second measured how many transactions were executed over a specific period of time.

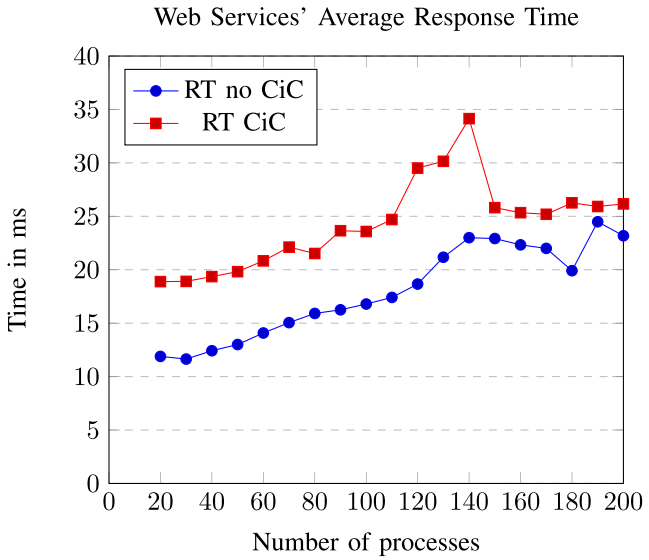

FIGURE 5. Response Time Measurement for the system implementing and without implementing CiC.

Fig. 5 indicates that although the average response time, AVG_{RT}, increased by approximately 30%, this increase was maintained for both the 20 customers requesting credit approval service and the group of 200. During the initial performance tests, we observed that with a low number of consumers, the average response time was slightly higher than when the existing solution without the CiC mechanism was applied. However when the number of clients making requests increased, i.e., more users accessing Web Service concurrently, the CiC mechanism performed better by reducing the average response time.

Experimental results for system throughput (β_t), in terms of Transactions Per Second (TPS), are shown in Fig. 6. The behavior is quite similar to the average response time exhibited in Fig. 5. In the early state of the performance

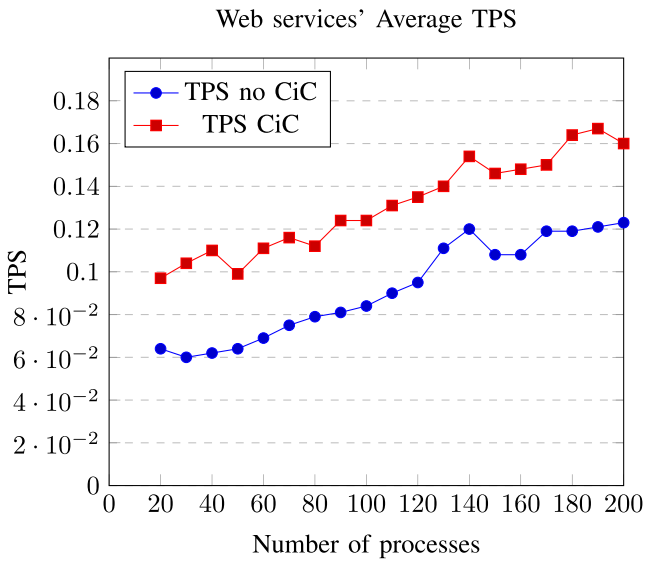

FIGURE 6. Transactions per second Measurement for the system implementing and without implementing CiC.

tests, we observed little enhancement in β in comparison to Web Services that do not implement the CiC mechanism. Nevertheless, when many clients are in play, a maximum gain is observed in throughput. Generally, boosts in all performance measurements were seen by applying our CiC approach to the existing Web Services considering a large number of concurrent users.

The aforementioned figures support an argument that our approach is scalable with low implementation cost (in terms of performance impact). The values recommended by *ITU G.1010*, which attempts to standardize the use of Web Services, are not violated for the response time. ITU stipulates the following values: preferred = $0-2$ seconds, acceptable = $2-4$ seconds and unacceptable = 4 to infinity.

Table 2 shows the incremented percentage for the response time when the CiC mechanism was implemented and when it was not executed. When 200 clients use the same applications or processes concurrently, as seen in real life scenarios, saving snapshots from the system can cause performance degradation. However, CiC adds a fault tolerance feature under which the system can restore execution from previously executed events.

Table 3 shows system performance degradation; here we compare system throughput when implementing the CiC mechanism and when not implementing it.

As evidenced from performance tests, we conclude that system performance is not affected when the mechanism of communication-induced checkpointing (CiC) is implemented; the underlying actions corresponding to the rest of the MAPE control loop should be seamlessly executed.

In Fig. 7 during the initial performance tests, we observed that with a low number of consumers, the average number of forced checkpoints was slightly higher than with a high

TABLE 2. Response time.

$\#ofProcesses$	$RTnoCiC$	$RTCiC$	$Inc\%$
20	11.89	18.89	59%
30	11.64	18.91	62%
40	12.42	19.36	56%
50	13.0	19.82	52%
60	14.08	20.83	48%
70	15.05	22.12	47%
80	15.91	21.52	35%
90	16.25	23.65	46%
100	16.79	23.58	40%
110	17.41	24.70	42%
120	18.66	29.52	58%
130	21.17	30.15	42%
140	23.00	32.14	40%
150	22.92	25.81	13%
160	22.33	25.34	13%
170	22.00	25.19	15%
180	19.91	26.26	32%
190	24.49	25.92	15%
200	23.18	26.16	12%
			38%

TABLE 3. Throughput.

$\#ofProcesses$	$TPSnoCiC$	$TPSCiC$	$Inc\%$
20	0.064	0.097	34%
30	0.060	0.104	42%
40	0.062	0.110	43%
50	0.064	0.099	35%
60	0.069	0.111	38%
70	0.075	0.116	36%
80	0.079	0.112	30%
90	0.081	0.124	35%
100	0.084	0.124	32%
110	0.090	0.131	32%
120	0.095	0.135	32%
130	0.111	0.140	30%
140	0.120	0.154	21%
150	0.108	0.146	22%
160	0.108	0.148	27%
170	0.119	0.150	28%
180	0.119	0.164	21%
190	0.121	0.167	27%
200	0.123	0.160	23%
			31%

number of consumers. However when the number of clients making requests increased, i.e., more users accessing Web Service concurrently, the CiC mechanism performed better by reducing the average number of forced checkpoints.

B. COMPARISON WITH RELATED WORK

This section presents a comparison with related works. Several of the related work only mention that checkpointing mechanisms are suitable for use in Web services [6], [17], however, do not consider service composition [5], [9]. On the other hand, other works to identify a consistent recovery point inhibit the system by introducing synchronization points [2], [18], finally, scalability is a factor that must be taken into account. According to our analysis and what some of these works report do not fulfill such properties. Summarized in Table 4. Hence, according to the results presented

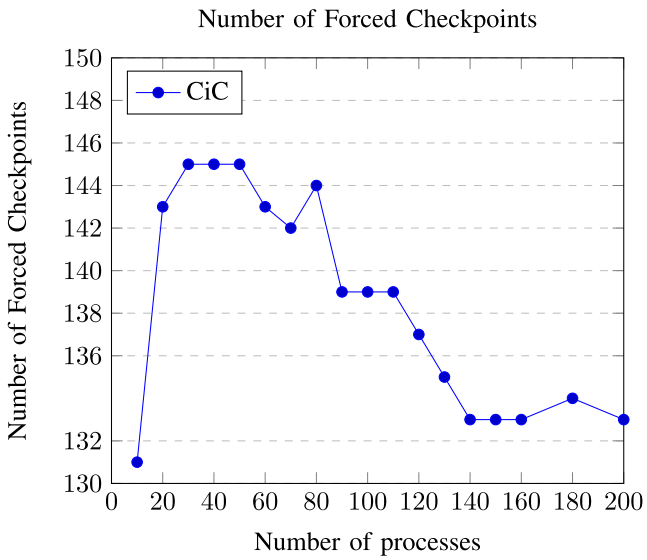

FIGURE 7. Number of forced checkpoints for 1000 messages sent.

TABLE 4. Comparison with related work.

Comparison with related Work				
Ref	WS composition	Solution	Execution	Scalable
OA	Yes	Distributed	Asynchronous	Yes
[5]	No	Centralized	Synchronous	Yes
[9]	No	Distributed	Asynchronous	No
[2], [18]	Yes	Centralized	Synchronous	Yes
[6], [17]	No	Centralized	Synchronous	Yes

in the previous section and Table 4, we can assure that our approach (OA) presents a better solution. Since it does not represent a negative impact on system performance, it is scalable, distributed, asynchronous and has low implementation cost.

VI. CONCLUSIONS AND FUTURE WORK

We have presented an algorithm aimed at improving Web services compositions based on implementing an autonomic computing and checkpointing mechanism. Additionally, we suggested an approach that implements the Monitoring Analyze Plan and Execute (MAPE) control loop within Web Services based on checkpointing protocols. Afterwards we presented a Web services composition to support fault tolerance using asynchronous communication induced checkpointing (CiC) which is domino effect free. To prove the feasibility we have presented an algorithm that leverages communication-induced checkpointing, and it is oriented for Web Services compositional interactions. Our algorithm can be applied to Web services since it supports an asynchronous communication and a non-coordinated execution.

Our approach reduces forced checkpoints by establishing certain triggering rules that we call safe checkpoint conditions. The results show that the CiC mechanism does not introduce high overhead to current Web Services compositions. We have shown how a Communication-induced Checkpointing (CiC) mechanism can improve autonomic computing.

Merging these two widely used paradigms, autonomic computing and checkpointing protocols, is a challenge that remains open. We think that an adaptive CiC based on system performance is worth consideration. The development of an Autonomic Service Bus (ASB) based on CiC protocols will be pursued. Another area of interest worth consideration is optimizing the number of checkpoints, which may be a good strategy to reduce communication overhead that is generated by communication-induced checkpointing mechanisms. The aforementioned strategy may be carried out by predicting quality of service variation, which represents how systems behave during a given period.

ACKNOWLEDGMENT

The authors would like to thank Rebekah Hosse Sullivan for taking her time and reviewing this work.

REFERENCES

[1] A. Moody, G. Bronevetsky, K. Mohror, and B. R. De Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC)*, Nov. 2010, pp. 1–11.
[2] S. Marzouk, A. J. Maalej, I. B. Rodriguez, and M. Jmaiel, "Periodic checkpointing for strong mobility of orchestrated Web services," in *Proc. World Conf. Services-I*, Jul. 2009, pp. 203–210.
[3] G. H. Alferez and V. Pelechano, "Context-aware autonomous Web services in software product lines," in *Proc. 15th Int. Softw. Product Line Conf. (SPLC)*, Aug. 2011, pp. 100–109.
[4] A. Immonen and D. Pakkala, "A survey of methods and approaches for reliable dynamic service compositions," *Service Oriented Comput. Appl.*, vol. 8, no. 2, pp. 129–158, 2014.
[5] J. Yin, H. Chen, S. Deng, Z. Wu, and C. Pu, "A dependable ESB framework for service integration," *IEEE Internet Comput.*, vol. 13, no. 2, pp. 26–34, Mar. 2009.
[6] A. J. V. Vaca and R. M. Gasca, "OPBUS: Fault tolerance against integrity attacks in business processes," in *Computational Intelligence in Security for Information Systems* (Advances in Intelligent and Soft Computing), vol. 85, L. Herrero, E. Corchado, C. Redondo, and N. Alonso, Eds. Berlin, Germany: Springer, 2010, pp. 213–222.
[7] B. Solomon, D. Ionescu, M. Litoiu, and G. Iszlai, "Autonomic computing control of composed Web services," in *Proc. ICSE Workshop Softw. Eng. Adapt. Self-Manage. Syst.*, 2010, pp. 94–103.
[8] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.
[9] I. Al-Hadid, "Airport enterprise service bus with self-healing architecture (AESB-SH)," *Int. J. Aviation Technol., Eng. Manage.*, vol. 1, no. 1, pp. 1–13, 2011.
[10] A. Dan and P. Narasimhan, "Dependable service-oriented computing," *IEEE Internet Comput.*, vol. 13, no. 2, pp. 11–15, Mar. 2009.
[11] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, "A survey of rollback-recovery protocols in message-passing systems," *ACM Comput. Surv.*, vol. 34, no. 3, pp. 375–408, 2002.
[12] A. C. Simon, S. E. P. Hernandez, and J. R. P. Cruz, "A delayed checkpoint approach for communication-induced checkpointing in autonomic computing," in *Proc. WETICE*, Jun. 2013, pp. 56–61.
[13] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978.
[14] S. P. Hernandez, J. Fanchon, and K. Drira, "The immediate dependency relation: An optimal way to ensure causal group communication," *Annu. Rev. Scalable Comput.*, vol. 6, no. 3, pp. 61–79, 2004.
[15] Y.-M. Wang and W. K. Fuchs, "Lazy checkpoint coordination for bounding rollback propagation," in *Proc. 12th Symp. Rel. Distrib. Syst.*, Oct. 1993, pp. 78–85.
[16] R. H. B. Netzer and J. Xu, "Necessary and sufficient conditions for consistent global snapshots," *IEEE Trans. Parallel Distrib. Syst.*, vol. 6, no. 2, pp. 165–169, Feb. 1995.

[17] A. J. Varela-Vaca, R. M. Gasca, N. D. Borrego, and S. P. Hidalgo, "Fault tolerance framework using model-based diagnosis: Towards dependable business processes," *Int. J. Adv. Secur.*, vol. 4, nos. 1–2, pp. 11–22, 2011.

[18] S. Marzouk, A. Maalej, and M. Jmaiel, "Aspect-oriented checkpointing approach of composed Web services," in *Current Trends in Web Engineering* (Lecture Notes in Computer Science), vol. 6385, F. Daniel and F. Facca, Eds. Berlin, Germany: Springer, 2010, pp. 301–312. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-16985-4_27

[19] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, *Human and Societal Dynamics Ebook: A Process for Developing a Common Vocabulary in the Information Security Area* (NATO Science for Peace and Security Series—E), vol. 23. New York, NY, USA: IO Press Publisher, 2007, pp. 10–51. [Online]. Available: http://ebooks.iospress.nl/volumearticle/24040

[20] S. A. Gurguis and A. Zeid, "Towards autonomic Web services: Achieving self-healing using Web services," *ACM SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–5, 2005.

[21] A. Moga, J. Soos, I. Salomie, and M. Dinsoreanu, "Adding self-healing behaviour to dynamic Web service composition," in *Proc. 5th WSEAS Int. Conf. Data Netw., Commun. Comput.*, Bucharest, Romania, 2006, pp. 206–211.

[22] W. Tian, F. Zulkernine, J. Zebedee, W. Powley, and P. Martin, "Architecture for an autonomic Web services environment," in *Proc. WSMDEIS*, 2005, pp. 32–44.

[23] D. Chappell, *Enterprise Service Bus*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2004.

[24] R. B. Halima, M. K. Guennoun, K. Drira, and M. Jmaiel, "Providing predictive self-healing for Web services: A QoS monitoring and analysis-based approach," *J. Inf. Assurance Secur.*, vol. 3, no. 3, pp. 175–184, 2008.

[25] R. Koh-Dzul, M. Vargas-Santiago, C. Diop, E. Exposito, F. Moo-Mena, and J. Gómez-Montalvo, "Improving ESB capabilities through diagnosis based on Bayesian networks and machine learning," *J. Softw.*, vol. 9, no. 8, pp. 2206–2211, 2014.

[26] R. Koh-Dzul, M. Vargas-Santiago, C. Diop, E. Exposito, and F. Moo-Mena, "A smart diagnostic model for an autonomic service bus based on a probabilistic reasoning approach," in *Proc. IEEE 10th Int. Conf. Ubiquitous Intell. Comput., 10th Int. Conf. Auto. Trusted Comput. (UIC/ATC)*, Dec. 2013, pp. 416–421.

MARIANO VARGAS-SANTIAGO received the Master in Research degree in information and telecommunications from the l'Institut National des Sciences Appliquées de Toulouse, Toulouse, France, in 2013. He is currently pursuing the Ph.D. degree in distributed systems and software engineering with the El Instituto Nacional de Astrofísica, Óptica y Electrónica, Mexico. His current research interests include the merging of two widely used paradigms: checkpointing mechanisms and autonomic computing.

LUIS MORALES-ROSALES received the Engineering degree in computer systems from the Instituto Tecnológico de Colima, Colima, Mexico, in 2001, and the Ph.D. degree in computer science from the El Instituto Nacional de Astrofísica, Óptica y Electrónica, Mexico, in 2009. His current research interests are the applications of distributed systems and intelligent computing.

SAÚL POMARES-HERNÁNDEZ received the Ph.D. degree from the Laboratory for Analysis and Architecture of Systems of Le Centre National de la Recherche Scientifique, France, in 2002. He is currently a Researcher (Professional Title) with the Computer Science Department, El Instituto Nacional de Astrofísica, Óptica y Electrónica, Mexico. Since 1998, he has been conducting research in the field of distributed systems, partial order algorithms, and multimedia synchronization.

KHALIL DRIRA received the Engineering and M.S. (DEA) degrees in computer science from the École Nationale Supérieure d'Electrotechnique, d'Electronique, d'Informatique, d'Hydraulique et des Télécommunications, Toulouse, in 1988, and the Ph.D. and HDR degrees in computer science from Université Paul Sabatier Toulouse in 1992 and 2005, respectively. Since 1992 as Charg de Rechere, he assumes a full-time research position with the Le Centre National de la Recherche Scientifique, France. His research interests include cooperative network services. His research activity addresses topics in this field focusing. He continues to be involved in national and international. He continues as a member of the program journals in the fields of software architecture as well as communicating. He has also been an Editor of a number of proceedings, books and journals, which have been published and presented, in these fields.