Autonomic Web Services Based on Asynchronous Checkpointing

Mariano Vargas-Santiago, Luis Morales-Rosales, Saúl Eduardo Pomares Hernández, Khalil Drira

Autonomic Web Services Based on Asynchronous Checkpointing

Mariano Vargas-Santiago[1], Luis Morales-Rosales[2], Saul Pomares-Hernandez [243] and Khalil Drira [34]

[1]*Department of Computer Science, National Institute for Astrophysics, Optics and Electronic (INAOE) , Tonantzlintla, Puebla, Mexico*
[2]*Faculty of Civil Engineering, Universidad Michoacana de San Nicolas de Hidalgo, Morelia, Michoacan, Mexico*
[3]*CNRS, LAAS, 7 avenue du Colonel Roche, Toulouse, France*
[4]*Univ de Toulouse, LAAS, Toulouse, France*
Email: mariano.v.santiago@inaoep.mx, lamorales@conacyt.mx, spomares@inaoep.mx, khalil@laas.fr

The evolution of business software technologies is constant and becoming more and more complex which leads to a greater probability of software/hardware failures. In particular, business processes are based on Web services, as they allow the creation of complex business functionalities. To attack the problem of failures presented by Web services organizations are extrapolating the autonomic computing paradigm to their business processes as it enables them to detect, diagnose, and repair problems and therefore improve their dependability. Nowadays there exist sophisticated solutions that increase systems dependability. However, such approaches have some drawbacks, for example, they affect the systems performance, they have a high implementation costs, and/or they can jeopardize the scalability of the system. To evolve systems to self-management, systems must implement the Monitoring, Analyzing, Planing and Execution (MAPE) control loop. In this paper we propose to tackle the MAPE loop of autonomic computing in a distributed and efficient manner by using communication-induced checkpointing (CiC), attacking the dependability problem for Web services. Our contribution is twofold. First, we present an approach for Web services compositions that gives support to fault tolerance based on CiC mechanism. Second, we present an algorithm aimed at Web services compositions based on autonomic computing and checkpointing mechanism. Experimental results show the feasibility of the former.

Keywords: Autonomic computing; Web Services; Autonomic Systems; Internet Technologies; Checkpointing

1. INTRODUCTION

Businesses are always searching for ways to improve their overall competitiveness. In business software, technologies help the evolution and growth of their businesses. For of large-scale systems, the entire system becomes less reliable and the fault rate probability augments as the number of component counts increment [1]. Usually under these circumstances Web services carry out complex business processes, known as compositions; frequently working with the Business Process Execution Language (BPEL) engine for orchestration [2]. These Web services can be locally deployed or they can collaborate with other Web services, over distributed heterogeneous environments, most common situation seen in real world deployments. Distributed enterprise applications are usually built following the Service-Oriented Architecture (SOA) paradigm in general [3]. Following the SOA architectural model for composite services it enables, even dynamically, the creation of distributed software intensive systems built from a combination of diverse independently developed services [4]. SOA can be seen 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; in specific Web services support service composition and application evolution [5]. Nonetheless, even though Web services are used

within complex business collaborative environments they are error prone because of the unreliable Internet behavior during runtime and yet are required to function correctly and be available on demand. Failures may lead to terrible consequences for instance augmenting the execution time, higher costs of the running applications, live lost, systems destroyed, and system breaches. As consequence organizations must obtain a way of making their systems (in many cases business processes) as dependable as possible before they intent to automate them [6]. To anticipate these errors organizations which have as core Web services for their business processes, require efficient and seamless solutions. For such purpose to attack the problem of failures presented by Web services organizations are extrapolating the autonomic computing paradigm to their business processes as it enables them to detect, diagnose, and repair problems, and therefore improve their dependability.

IBM introduced the *Autonomic Computing* paradigm, where systems are considered to be self-manageable these should be able to give themselves maintenance and corrective actions with minimal human intervention [7]; for this purpose of evolving to self-management, Kephart proposed the MAPE (Monitoring, Analysis, Planning and Execution) control loop [8]. In achieving such IBM introduces the notion of autonomic managers and managed elements. Autonomic managers are in charge for the interaction and communication with the outside world, in other words interpreted as human computer interaction and interactions with other elements and also, as a bridge with the managed elements. The *Monitoring* phase is in charge of 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 regarding previous phases.

Nowadays there exists sophisticated solutions for Web services dependability enhancement. However, such approaches have some drawbacks for example they affect the systems performance, they have a high implementation costs [5], and/or they can jeopardize the scalability of the system [9]. One open challenge for Web services is to increment their reliability [4] a particular feature of dependability which considers many more aspects like: reliability, availability, security, maintainability among others [10].

In this paper we propose to tackle the MAPE loop of autonomic computing in a distributed and efficient manner by using communication-induced checkpointing (CiC), attacking the dependability problem for Web services. CiC has proved being worthy and has gain a great level of maturity specially for distributed systems, solving issues like rollback recovery, software debugging and software verification among others [11]. In general, CiC aims at building *consistent global snapshots* (CGS) or simply checkpoints (one from each process) avoiding dangerous patterns: like zigzag paths and zigzag cycles [12] as stipulated by the authors. The purpose of building CGS is to checkpoint process on nonvolatile storage to rollback or survive process failures. As consequence, for decoupled systems, an architectural model type is chosen; the autonomic behavior can be incorporated into preexisting non-autonomous systems, such as Web services. In this category is that the checkpointing protocols can be applied. The research is aiming to merge the following technologies:

- Autonomic Computing which provides the MAPE loop for self-manageable systems.
- Web Services which follow open standards for interoperability.
- Checkpointing Protocols which save consistent states to which the system can rollback to in case of undesired functioning of the system.

We first introduce and give the motivation to be studied, and then in Section 2 we analyze the model for checkpointing. In Section 3 we discuss the related work that increase Web services' dependability. Section 4 analyses the environment and considerations around our proposed solution, a case study is presented as well as an algorithm. Finally Section 5 gives the results to support our approach, showing its feasibility. Conclusions are drawn in Section 6 where we also suggest further work.

2. THE MATHEMATICAL MODEL

2.1. Communication Patterns

Distributed systems have the following characteristics, there is no global notion of time, processes do not share common memory and communicate solely by message passing. In this context distributed computation consists of a finite set of processes $P = \{P_1, P_2, \ldots, P_n\}$, in our particular case Web services. We assume that channels have an unpredictable yet finite transmission delay and that these are 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 state, for instance a checkpoint, a finite set of internal events are denoted by E_i. External events are those that affect the global state of the system, for instance *send* and *delivery* events. 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

$$E = E_i \cup E_m$$

2.2. Definitions

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 2.1. *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.

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 the minimal amount of control information per message to ensure causal ordering. The IDR is the transitive reduction of the *HBR*, and it is denoted by "$\downarrow$", defined as follows [14]:

DEFINITION 2.2. *Two messages a and $b \in E$ have an IDR $a\downarrow b$ if the following restriction is satisfied:*

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

Checkpoint and communication pattern CCP. It is represented by its distributed computation consists on a set of incoming and outgoing messages and associated local checkpoints [15]

DEFINITION 2.3. *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 sequence of events occurring at p_k between C_k^{x-1} and C_k^x $(x > 0)$

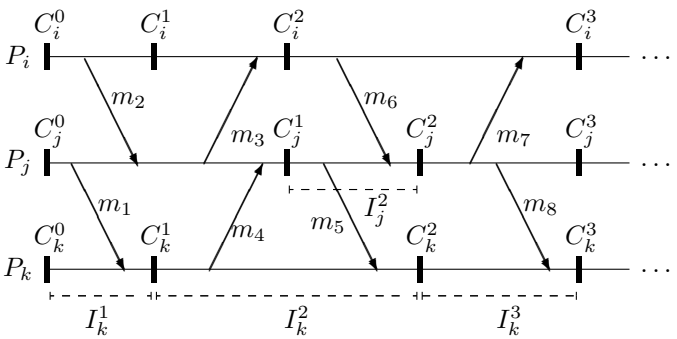

FIGURE 1. A Checkpoint and Communication Pattern (CCP) [12].

2.3. Building Consistent Global Snapshots

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

DEFINITION 2.4. *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].

3. RELATED WORKS

Dependability has been a wide area of research, a whole taxonomy about it can be found in [19]. Many different aspects of the system affects its dependability: reliability, availability, security, maintainability and consider systems to be self-manageable [10]. Yet, neither today to address the self-manageable issues for systems there is no unified or standardized form; nor for implementing the MAPE loop within Web services. Some work treat each one of the MAPE loop phases as an individual Web service [20]; others only tackle the self-healing feature of autonomic computing [21]. In [22] the authors suggest to address the entire MAPE loop by adding extra interfaces to confront functional and non-functional requirements of Web services.

Marzouk et al., illustrate in [2], that Web services are implemented to follow a business logic, they could be deployed either locally inside an enterprise or they could cooperate in a distributed environment, either way Web services are crucial and must implement fault tolerance mechanisms. The authors principal concern is about ensuring self-adaptivity of composite services. They provide an approach for adapting Web service composition based on *strong mobility* code; achieved with generic source code transformation. The authors identified that the strong mobility approach for Web services-BPEL processes requires periodic checkpointing. When self-adaptivity is a necessity one or several instances of the orchestration process will be migrated to the previous checkpoint and resumed starting from the interruption point; beginning from their last captured checkpoint. However using strong mobility requires replicating services, if possible, but in other cases finding similar services becomes a complicated task. This due to services format, for instance, one service uses an integer instead of a float.

Marzouk et al. discuss, in [18], that their approach pursues the self-healing property whereas 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 execution when recovering flow activities

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 (Businesss).	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 delivery of messages.	Orchestration process using self-healing.

since they always save a unique state for each instance. But, a recovery state is built after synchronizing all flow branches which permit saving a consistent checkpoint. Yet, to our believe 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. It is because, other proposed works do not use checkpointing techniques and have to start the whole Web service composition or orchestration.

Varela et al. identified, in [6], that while executing business processes they are susceptible to intrusion attacks, which can be the cause of severe faults. Fault tolerance techniques tackle such issues, decreasing risk of faults and therefore being more dependable, with the aim of achieving dependability before business processes automation. The authors claim that, to resists faults related to integrity attacks fault tolerance techniques can be applied. Varela and Martínez proposed OPUS, 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 mentioned which checkpointing mechanism they use; often, new and improved checkpointing mechanisms are proposed in the literature, we believe that the recovery overhead time can be reduced making use of such improved protocols.

In [17], Varela et al. argue that companies need to intercommunicate exchanging information between business logics, thus deciding to deploy what is called Business Process Management System (BPSM). BPSM aids to automate business processes, but in this context systems are error prone and can not guarantee a perfect execution over time. Therefore a new paradigm called Business Process Management (BPM) arises, defined as a set of concepts, methods and techniques to aid the modeling, design, administration, configuration, enactment and analysis of business processes. For the business processes life cycle the BPM paradigm follows diverse stages: design and analysis, configuration, enactment and diagnosis however each stage may introduce different fault kinds. For companies a mean to gain dependability in early design stages, is indispensable; promoting the reduction of possible faults and risks. In this work, the authors propose to follow traditional or classic fault tolerant ideas such as replication, checkpointing among other, focusing on the service-oriented business processes context. However, such approach requires the introduction of extra components (sensors) into the business process design, extra time to check each sensor and recovery of business process service in rollback.

Table 1 summarizes the related works. It exhibits the aim, the technology used and the environment under which the authors propose their solutions. Despite of what the different proposals address with their solutions, some questions remain open such as: How to integrate Web services in dynamic environments in an

autonomic way? Are the checkpoints taken for rollback recovery strategies consistent? Are solutions impacting negatively systems' performance?

4. AUTONOMIC WEB SERVICES BASED ON ASYNCHRONOUS CHECKPOINTING MECHANISM

4.1. Architecture

The proposed approach is suitable in distributed heterogeneous environments; it leverages the Enterprise Service Bus (ESB) infrastructure which provides an integration backbone for systems integration. Additionally the ESB ensures interoperability and offers several features such as: service discovery, intelligent routing, message processing and service orchestration assuring proper format between service providers and consumers, no matter which programming language they are written on [23].
In general the *functional properties or functional contract* of a Web service are exposed by the Web Service Description Language (WSDL), in other words how the service must behave. Whereas the *non-functional properties* of a Web service are represented by the Quality of Service (QoS) parameters, also for Web service composition, which must be monitored and analyzed in order to conclude if the service is behaving in an adequate form or not. Monitoring an individual Web service, and global Web services compositions is challenging because of the particularity presented in each case; since each Web service is unique. Performance parameters can be monitored under diverse scopes, for instance by the client side obtaining parameters like latency, throughput and error rate to name a few.
The architecture is designed to provide interoperability between diverse services and systems having different technologies through standards-based adapters and interfaces that use Web services technology (as shown in Fig. 2). Furthermore, each Web service follows the MAPE loop from the autonomic computing paradigm. The service layer represents the invocation or the execution of a required task or service from which performance information will be extracted.

4.2. Performance measurements

We proposed an asynchronous checkpointing mechanism to support fault tolerance, therefore, we measure systems' performance before and after implementing the CiC mechanism system architecture.

We have chosen the performance measurements pertinent to network traffic to evaluate the performances before and after applying CiC architecture. We measured the average response time α_t and the throughput β_t, in terms of Transactions Per Second (TPS), for measuring the application services and CiC performance.

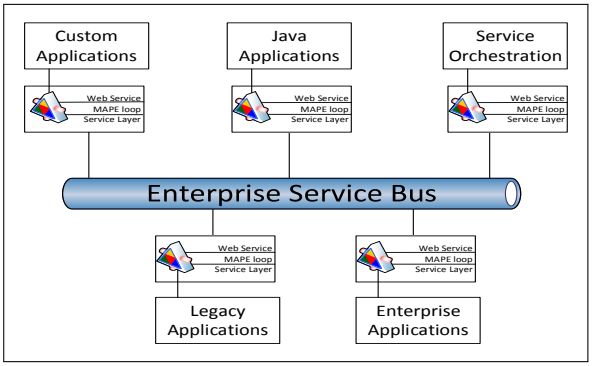

FIGURE 2. ESB with MAPE loop Architecture.

4.2.1. Average response time (AVG_{RT})
It is defined as the average time taken by Web service from the time of sending request by a client till the time of receiving the reply from the Application Server. To calculate the response time we use two timestamps, when the client sends a request (t_1) and 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 in the system in play. Latter we average all exchanged messages in the system, as consequence obtaining AVG_{RT}.

4.2.2. Throughput (β_t)
For interactive systems, the system's throughput is defined as the ratio of total number of request to the total time, which has a correlation with response time. We define the Web services systems performance β_t as the amount of data processes by a Web service in a given time interval.

4.3. 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 system; which is in charge of routing, adapting or mediating the request if necessary, in other words the service layer is called. Then the *Monitoring* module computes the QoS parameters as are response time and the throughput. These events shall be converted into XML-based messages and stored in a common knowledge base, shared by all Web services. The *Analyze* module uses a *Diagnosis Engine* that checks the extracted information to decide if the behavior of the Web service is normal or if it suffers any

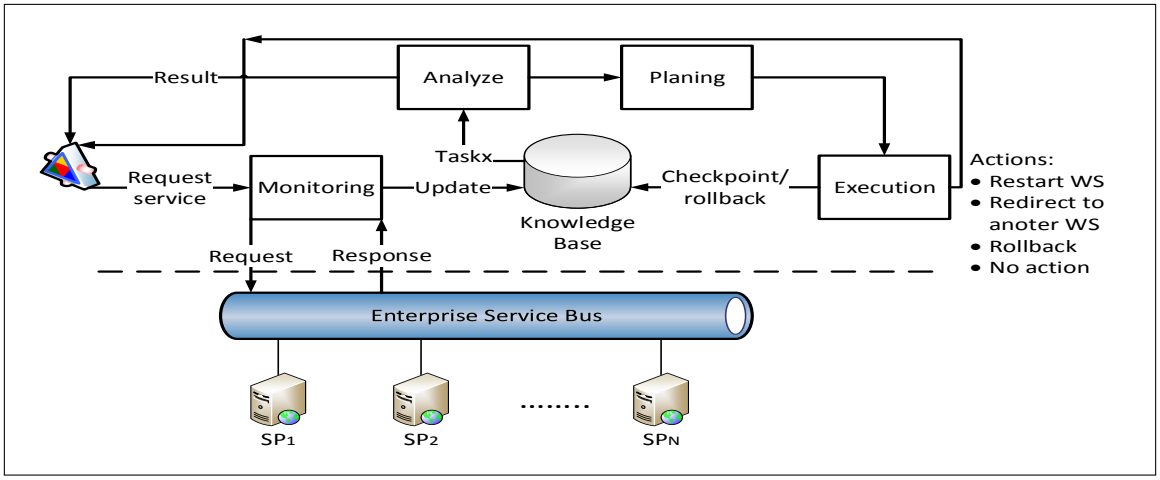

FIGURE 3. Autonomic Web services based on CiC protocols.

anomaly or fault. In other words, identifies patterns in the logs looking for specific problems that occurred [20]. If the Web service presents normal parameters then it is immediately returned from *Analysis* module to the Web service invoking a service. However, the message is forward to the analysis process that checks the new aggregate values with the aim of predicting the immediate future state of the system, based 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 schedule the next forced checkpoint(s). So that later the *Execution* module realizes them. After that if the degradation happens and the system can not continue then we would enter the rollback recovery stage.

4.4. Scenario

The autonomic Web services based on communication-induced checkpointing can be better explained through an example like the *Stock Quote* composite Web service system. Where many consumers of a service invoke multiple stocks brokers to find out on which to invest, for clients that pay a subscription, premium users, they are able to receive in real-time the stock quote service. The most simple case is shown in Fig. 4 where two service consumers make a petition to a stock broker or service provider, however it shows the need for building consistent global snapshots (CGS). As stipulated by Netzer and Xu in presence of zigzag paths and causal paths cannot constitute a CGS [16]. The Fig. depicts 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 $m5$ for instance, although no causal path exists between C^1_{c1} and C^2_{p1} a zigzag path does 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 contains C^1_{c1} and C^2_{p1}.

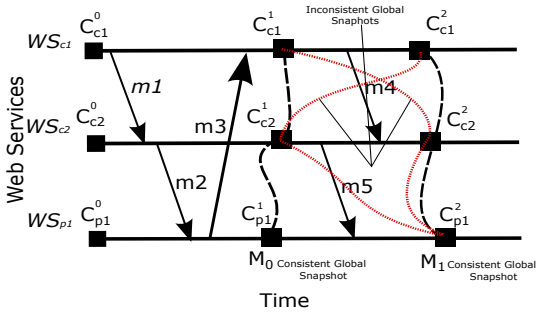

FIGURE 4. Example Scenario.

4.5. Algorithm

We present an Algorithm 1 that aims at implementing the autonomic computing paradigm and integrating checkpointing mechanism. The aim of our algorithm is to tackle Web services composition.

Algorithm 1 suggests to initialize all variables (lines 1 to 5), for instance line 4 builds and assigns to C the systems initial consistent global snapshot (CGS). The MAPE loop is represented by lines 6 to 31; for each web service the algorithm starts the *Monitoring* by checking the web service policies and process level at the time of calling the *CHECKPOLICIES*. Checking that Web service satisfy the constraints stipulated in their policies. During any time of this computation period a new CGS can be build. *VALIDATE* is used to detect a set of symptoms returning a non-empty set when the Web service specifications and policies are not met. For the best case, when the entire composite is functioning correctly a no problem 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 the last known CGS is retrieved from the common *Knowledge Base* (KB), line 13.

Then the vector containing the set of symptoms is *Analyzed* to find the best known diagnosis, retrieved from the KB, this is done for each individual Web

Algorithm 1 Autonomic Composite Web Services

```
1: procedure INITIALIZATION
2: Initially
3:     S ← ∅, P ← ∅, D ← null
4:     C ← BUILD_INITIAL_CGS(WS_1, WS_2, ..., WS_n)
5: end procedure
6: procedure MAPE((WS_1, WS_2, ..., WS_n))
7:     for WS[i], i = 1 to n do
8:         FWS[i] ← CHECKPOLICIES(m)
9:     end for
10:    if FWS[i] == ∅ ∀ FWS[i], i = 1, 2, ..., n then
11:        return "No problem found for Composite WS"
12:    else
13:        GET_LAST_CGS((C))
14:        for ∀FWS[i], i = 1, 2, ..., n do
15:            if FWS[i] ≠ ∅ then
16:                D[] ← FIND_BEST_DIAGNOSTIC(FWS[i])
17:                if D[] = NULL then
18:                    D[] ← CLASSIFY_SIMPTOMS(FWS[i])
19:                    P[] ← GENERATE_PLAN(D[])
20:                else
21:                    P[] ← FIND_BEST_PLAN(D[])
22:                end if
23:            end if
24:        end for
25:    end if
26:    D_G ← BUILD_GLOBAL_DIAGNOSTIC(D[])
27:    P_G ← FIND_BEST_GLOBAL_PLAN(DG[]) ∩ P[]
28:    for action ∈ P_G do
29:        EXECUTE(action, P_G)
30:    end for
31: end procedure
32: procedure CHECKPOLICIES(m)
33:    wsdescr ← GET_WS_DESCR(wsid[e])
34:    wspolicy ← GET_WS_POLICY(wsdescr)
35:    processpolicy ← GET_PROCESS_POLICY(pid[e])
36:    policy ← processpolicy ∩ wspolicy
37:    S ← validate(m, policy, wsdescr)
38:    return (S)
39: end procedure
```

service of 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 individual Web service, however concerning a composite Web service the system must build a global diagnostic of how the overall composite is behaving, line 26. Same case is applied to individual plans a global plan must be generated from the known 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, like rollback, restart a specific Web service etc.

5. RESULTS AND DISCUSSION

In order to show autonomic Web services based on asynchronous checkpointing (specifically communication-induced checkpointing) does not have a great impact on the overall performance of the systems, we performed several performance tests. Specifically, we measured response time and transactions per second as a key performance indicators. For testing we implement our proposed solution within the following hardware: on a workstation with 16 GB RAM with Windows 7 64-bit as operating system. The WSO2 Application Server was used to deploy to Web services, and for performance tests diverse concurrent Java clients were emulated, in order to have an approximate real world deployment.

5.1. Experimental results

Fig. 5 and Fig. 6 show the behavior of the system when evaluating its performance. For this purpose several iterations of the scenarios were performed (in particular each scenario was executed 100 times). That is, for 20 consumers, 2,000 samples were collected, for 30, 3,000 were collected in increments of 10 to reach 200 where 20,000 samples were collected. Subsequently, their average was obtained for the response time and for the transactions per second. The response time measures the time from when the customer sends a request to the credit approval service until he receives his response. Transactions per second measure how many transactions are executed over a certain period of time.

From Fig. 5 in a quantitative way, it can be observed that although the average response time AVG_{RT} is increased, approximately 30%, this increase is maintained for both 20 customers requesting the credit approval service and 200 clients. We observed during the initial emulation with low number of consumers, the average response time is slightly higher than the existing solution without the CiC mechanism. However, with increasing clients' requests, more user using Web services concurrently, our CiC mechanism performs better with reduced average response time. Interactive environments present diverse challenges, one of them has to due with their resource usage, yet our approach remains constant even when the number of consumers increases.

The emulation results for the systems' throughput (β_t), in terms of Transactions Per Second (TPS), are shown in Fig. 6. Its behavior is quite similar to the one exhibit by the average response time. In the early stage of the emulation we observe not much 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, the enhancements in all the performance measurements were observed by applying our approach on the existing Web services considering many concurrent consumers.

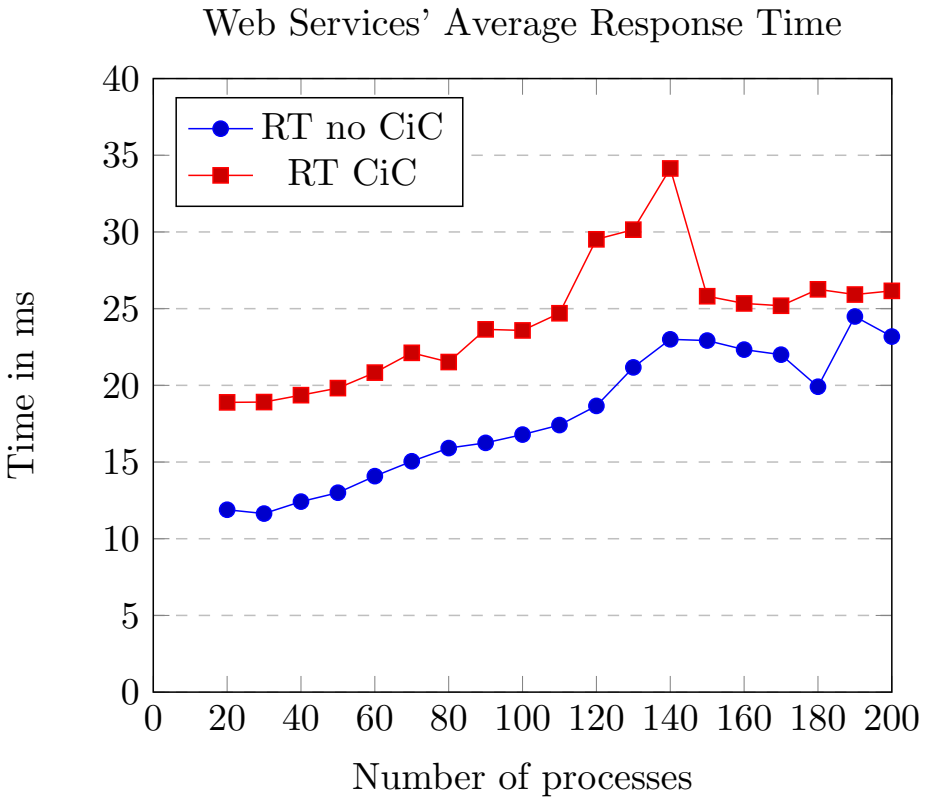

FIGURE 5. Response Time Measurement for the system implementing and without implementing CiC.

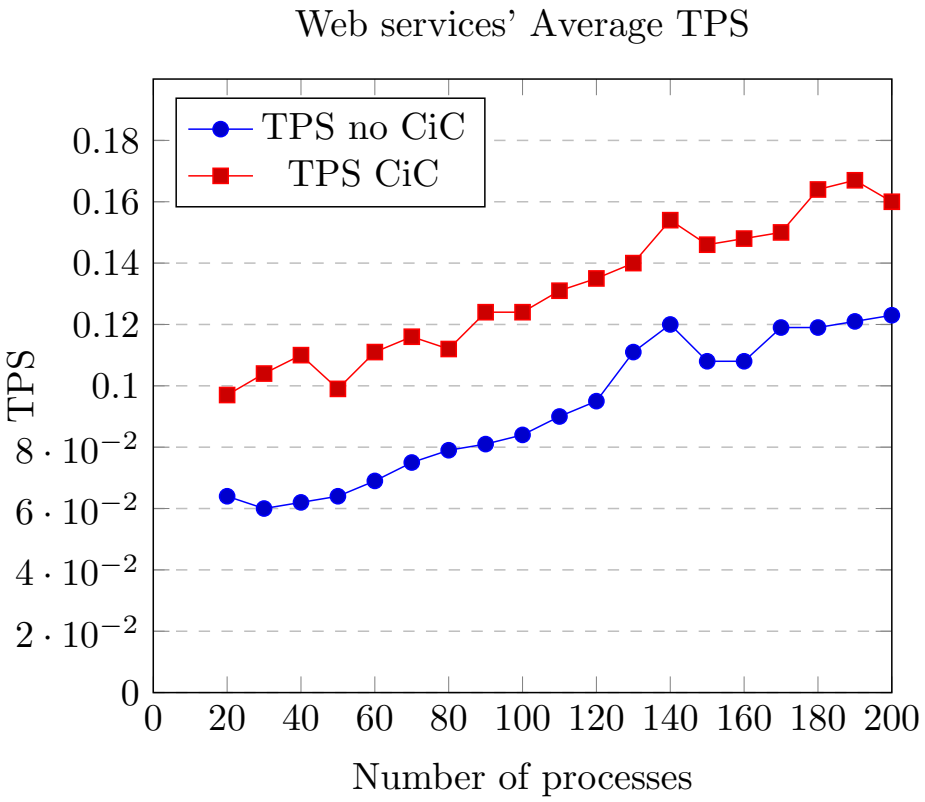

FIGURE 6. Transactions per second Measurement for the system implementing and without implementing CiC.

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%

The aforementioned figures give guide to argue that our approach is scalable and with low implementation cost (in terms of performance impact). Since 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 while adopting or not the CiC mechanism. When 200 clients use concurrently the same scenario, case most seen in real life, saving snapshots from the system incurs some performance degradation. However, adding a fault tolerance feature under which the system can restore its execution from previous executed events.

Table 3 shows the systems' performance degradation, here we compare system throughput when implementing the CiC mechanism and when not implementing it. Therefore, given evidence from the performance tests we can 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 loop should be seamlessly executed.

6. CONCLUSIONS AND FUTURE WORK

First, we present an algorithm aimed at Web services compositions based on autonomic computing and checkpointing mechanism. Second, we suggested an approach that implements the *Monitoring Analyze Plan and Execute (MAPE)* loop within Web services based on checkpointing protocols. Afterwards we presented a Web services compositions to support fault tolerance using an asynchronous communication-induced checkpointing (CiC) which is domino effect free. To prove the feasibility we have presented an algorithm that leverages the communication-induced checkpointing, and it is oriented for Web services compositions 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.

Merging these two widely used paradigms, autonomic computing and checkpointing protocols, is a challenge that remains open. However, we showed in this paper how a CiC mechanism can help upon autonomic computing . Besides, we consider that an adaptive CiC based on the systems performance will be considered. Therefore, the implementation developing of an Autonomic Service Bus (ASB) based on CiC protocols will be taken into account.

Another area of interest concerns optimizing the number of checkpoints, a good strategy since it reduces a large amount of communication overhead that is generated by the communication-induced checkpointing mechanisms. The aforementioned strategy can be carried out by predicting quality of service variation, which represents how the systems behave during a certain period.

ACKNOWLEDGMENT

This work was supported by the National Council for Science and Technology of Mexico (CONACYT)

through the project ID PDCPN2013-01-215421.

REFERENCES

[1] Moody, A., Bronevetsky, G., Mohror, K., and Supinski, B. D. (2010) Design, modeling, and evaluation of a scalable multi-level checkpointing system. *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*, pp. 1–11. IEEE.

[2] Marzouk, S., Maalej, A., Bouassida, I., and Jmaiel, M. (2009) Periodic checkpointing for strong mobility of orchestrated web services. *Services-I, 2009 World Conference on*, pp. 203–210. IEEE.

[3] Alferez, G. H. and Pelechano, V. (2011) Context-aware autonomous web services in software product lines. *Software Product Line Conference (SPLC), 2011 15th International*, pp. 100–109. Ieee.

[4] Immonen, A. and Pakkala, D. (2014) A survey of methods and approaches for reliable dynamic service compositions. *Service Oriented Computing and Applications*, **8**, 129–158.

[5] Yin, J., Chen, H., Deng, S., Wu, Z., and Pu, C. (2009) A dependable esb framework for service integration. *Internet Computing, IEEE*, **13**, 26–34.

[6] Vaca, A. and Gasca, R. (2010) Opbus: Fault tolerance against integrity attacks in business processes. In Herrero, l., Corchado, E., Redondo, C., and Alonso, n. (eds.), *Computational Intelligence in Security for Information Systems 2010*, Advances in Intelligent and Soft Computing, **85**, pp. 213–222. Springer Berlin Heidelberg.

[7] Solomon, B., Ionescu, D., Litoiu, M., and Iszlai, G. (2010) Autonomic computing control of composed web services. *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pp. 94–103. ACM.

[8] Kephart, J. and Chess, D. (2003) The vision of autonomic computing. *Computer*, **36**, 41–50.

[9] Al-Hadid, I. (2011) Airport enterprise service bus with self-healing architecture (aesb-sh). *International Journal of Aviation Technology, Engineering and Management (IJATEM)*, **1**, 1–13.

[10] Dan, A. and Narasimhan, P. (2009) Dependable service-oriented computing. *Internet Computing, IEEE*, **13**, 11–15.

[11] Elnozahy, E., Alvisi, L., Wang, Y.-M., and Johnson, D. (2002) A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys (CSUR)*, **34**, 375–408.

[12] Simon, A., Hernandez, S. P., and Cruz, J. P. (2013) A delayed checkpoint approach for communication-induced checkpointing in autonomic computing. *WETICE*, pp. 56–61. IEEE.

[13] Lamport, L. (1978) Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, **21**, 558–565.

[14] Hernandez, S. P., Fanchon, J., and Drira, K. (2004) The immediate dependency relation: an optimal way to ensure causal group communication. *Annual Review of Scalable Computing*, **3**, 61–79.

[15] Wang, Y.-M. and Fuchs, W. K. (1993) Lazy checkpoint coordination for bounding rollback propagation. *Reliable Distributed Systems, 1993. Proceedings., 12th Symposium on*, pp. 78–85. IEEE.

[16] Netzer, R. and Xu, J. (1995) Necessary and sufficient conditions for consistent global snapshots. *IEEE Transactions on Parallel and distributed Systems*, **6**, 165–169.

[17] Vaca, A., Gasca, R., Borrego, N., and Hidalgo, S. P. (2011) Fault tolerance framework using model-based diagnosis: towards dependable business processes. *International Journal on Advances in Security*, **4**, 11–22.

[18] Marzouk, S., Maalej, A., and Jmaiel, M. (2010) Aspect-oriented checkpointing approach of composed web services. In Daniel, F. and Facca, F. (eds.), *Current Trends in Web Engineering*, Lecture Notes in Computer Science, **6385**, pp. 301–312. Springer Berlin Heidelberg.

[19] Avizienis, A., Laprie, J., Randell, B., and Landwehr, C. (2007) Basic concepts and taxonomy of dependable and secure computing. *Nato security through science series e human and societal dynamics*, **23**, 10.

[20] Gurguis, S. and Zeid, A. (2005) Towards autonomic web services: Achieving self-healing using web services. *ACM SIGSOFT Software Engineering Notes*, **30**, 1–5.

[21] Moga, A., Soos, J., Salomie, I., and Dinsoreanu, M. (2006) Adding self-healing behaviour to dynamic web service composition. *Proceedings of the 5th WSEAS International Conference on Data Networks, Communication and Computers, Bucharest, Romania*, pp. 206–211.

[22] Tian, W., Zulkernine, F., Zebedee, J., Powley, W., and Martin, P. (2005) Architecture for an autonomic web services environment. *WSMDEIS*, pp. 32–44. Citeseer.

[23] Chappell, D. (2004) *Enterprise service bus*. O'Reilly Media, Inc.

[24] Halima, R. B., Guennoun, M. K., Drira, K., and Jmaiel, M. (2008) Providing predictive self-healing for web services: a qos monitoring and analysis-based approach. *Journal of Information Assurance and Security*, **3**, 175–184.

[25] Koh-Dzul, R., Vargas-Santiago, M., Diop, C., Exposito, E., Moo-Mena, F., and Gómez-Montalvo, J. (2014) Improving esb capabilities through diagnosis based on bayesian networks and machine learning. *Journal of Software*, **9**.

[26] Koh-Dzul, R., Vargas-Santiago, M., Diop, C., Exposito, E., and Moo-Mena, F. (2013) A smart diagnostic model for an autonomic service bus based on a probabilistic reasoning approach. *Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC)*, pp. 416–421. IEEE.