Time and Cost Aware Checkpointing of Choreographed Web Services

Vani Vathsala Atluri and Hrushikesha Mohanty

CVR College of Engineering and University of Hyderabad
Hyderabad, India
atlurivv@yahoo.com, mohanty.hcu@gmail.com

Abstract. Complex business processes can be realized by composing two or more web services into a composite web service. Due to the widespread reachability of Internet, more and more web services are becoming available to the consumers. Quality aware consumers look for resilience in services provisioned on Internet. This paper proposes message logging based checkpointing and recovery for web services to make them resilient to faults. It presents an algorithm that checkpoints services participating in a choreography in such a way that the execution time and cost of service constraints are always met. It identifies checkpoint locations by considering the costs involved in checkpointing, message logging and replaying for service recovery. The cost estimation is carried out using service interaction patterns and QoS values of the services involved. Performance of the proposed checkpointing strategy is corroborated with the results obtained from experiments.

Keywords: Web services, choreography, checkpointing, QoS.

1 Introduction

There are several algorithms that are proposed in literature for checkpointing distributed applications. Any checkpointing scheme has to satisfy the requirement of resilient service provisioning [13]. Checkpointing a choreographed web service needs special care so that recovery does not need a chain of restarts of component services. Hence checkpointing web services is of interest. We propose to perform checkpointing of choreographed web services at three different stages of web service development: 1. Design time 2. Deployment time 3. Run time.

A group of web services interacting with each other by means of message exchanges, to accomplish a business task are called as a choreographed web services. Choreographed web services have the information about their sequence of message based interactions, and actions to be performed by each of them, documented in a design time artefact called as **choreography** document. Using this document, we have proposed a **design time** checkpointing approach in our previous work [12] that introduces checkpoint locations in a choreography, at places where non repeatable actions are performed. In the event of transient

R. Natarajan et al. (Eds.): ICDCIT 2015, LNCS 8956, pp. 207–219, 2015.

failures(temporary failures) this checkpoint arrangement avoids chain of reinvocation of web services, specifically when a non repeatable action is executed. But, it does not handle the issue of meeting deadlines in case of transient failures. This paper addresses this issue using Quality of Service(QoS) values (like response time, reliability, cost of service etc.) and other quantities like checkpointing time, message logging time etc which can be measured at the time of **deployment**. We propose a time and cost aware checkpointing algorithm that introduces minimum number of checkpoints so that execution time and cost constraints are met even in the event of transient failures. The trade-off between number of checkpoints and recovery time is experimentally analysed. As part of our future work we intend to take up revision of checkpoint locations at **run time** using dynamically predicted QoS values and dynamic composition of web services. We have presented our approaches on response time prediction of web services in our previous works [11] [14].

This paper is organised as follows: in section 2 we discuss existing checkpointing approaches and compare them with our approach, in section 3 we give an overview of our choreography model and three stage checkpointing approach, in section 4 we brief on our proposed approach for deployment time checkpointing. Checkpointing algorithm is given in section 5 along with experimental results. Conclusion and future work are discussed in section 6.

2 Related Work

Well understood techniques[6] for checkpointing and recovery of distributed applications are not readily applicable for that of web services. This is because a recovery strategy for distributed applications requires other processes also, in addition to failed process, to rollback. Such a strategy is not suitable for composite web services since requiring chain of rollbacks of the remaining services leads to a compromise in quality of the composite service.

Fault handling strategies proposed in the field of web services are of two types [13]: Checkpointing and recovery [2], [10], [9], and Substitution [4], [3], [5]. In the proposed checkpointing and recovery strategies [2],[10], responsibility of specifying checkpointing locations in the design document is bestowed on the user. Success of such strategies lies in the knowledge and skillset of the users. In another checkpointing strategy[9], checkpointing is used to save the work of constituent services that have not failed but, the faulty constituent service has to restart from the beginning. Substitution approaches proceed by substituting the faulty web service with a functionally equivalent one. The main drawback of this approach can be seen when the invoking service itself fails. The failed instances would have to be reexecuted from the beginning. This implies recalling the invoked services again resulting in increased execution times.

QoS aware checkpointing has been proposed in various areas like embedded systems [1], and mobile computing [8]. In [1] authors propose QoS aware message logging based checkpointing of embedded and distributed systems. They formulate the problem of finding an optimal checkpoint interval, in a process, that

maximizes systems overall quality as a Mixed Integer Non-Linear Programming (MINLP) problem and provide an algorithm for finding the solution. They do not consider QoS values of other processes participating in the composition, in checkpointing decisions. In [8], authors work on mobile computing environment where in hosts going out of range transfer their checkpointed data to other hosts. It uses link reliability values to dynamically maintain superior checkpointing arrangement.

Considering QoS values of the constituent services while taking checkpointing decisions for a choreographed web service is pivotal in meeting promised deadlines. In our survey we have not come across any web service checkpointing strategy that focuses on this issue. Hence in this paper we advocate time and cost aware checkpointing strategy that makes use of QoS values of constituent services, to decide on checkpoint locations while meeting the promised deadlines.

In the next section we present concepts [12] required here for putting the proposed strategy at right perspective.

3 Checkpointing Choreographed Web Services

A choreography of web services describes in a document, called choreography document, series of interactions that are to be performed by constituent web services to accomplish common business goals. A choreography of web services is modelled as a composition of interaction patterns in our previous work. We use this model to aid in checkpointing decisions. In the following subsection we give a brief on our model, details can be obtained from [12].

3.1 Modelling Choreographed Web Services

Each participant ξ of a service choreography performs some local actions and communicates necessary information with other participants. Thus, broadly, **operations** of a service are classified into two categories: *local action* and *interaction*. An **interaction pattern**, or a **pattern** in short, is defined as a sequence of operations wherein the first operation in the sequence is an interaction. In our previous paper [12] we have proposed different types of interaction patterns and detailed on modelling service choreographies using these patterns.

To make the description in this paper self-contained, and to keep up with the space requirements, we use only one of our proposed patterns, pattern P_2 (Chose P_2 as it has maximum number of interactions) in this paper. In P_2, the initiator ξ sends an invoke message and continues to execute a sequence of operations, before getting a reply from ξ_r. Upon receiving the invoke message, ξ_r executes a sequence of operations and sends a reply message back to ξ (refer to Fig 1(a)).

In [12] we propose that service interaction patterns may be combined in different ways using composition operators to give composite patterns. We proposed four kinds of composite patterns: sequential pattern, nested pattern, iterative pattern, and concurrent pattern whose operators are ".",[ ],* and | respectively. Fig 1(f) depicts an example choreography which is modelled as a composition of

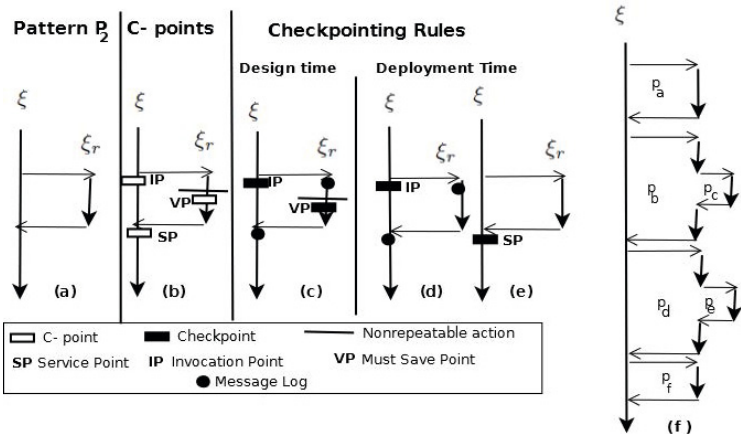

Fig. 1. Modelling service choreographies and checkpointing rules

our patterns. A composite web service is compactly represented in text using a pattern string. A **pattern String** represents a choreographed web service whose composition is expressed in terms of patterns and composition operators. The pattern string for the example choreography is $p_a.p_b[p_c].p_d[p_e].p_f$

In a step towards identifying possible checkpoint locations in a given choreography, we associate each pattern with what are called as C-points. A **C-point** is a probable checkpointing location. We define three types of C-points: service point, must save point and invocation Point. Fig 1(b) depicts C-points in pattern P_2. A *service point* is marked in the initiator of the pattern after the reply message is received. An *invocation point* is marked in the initiator of the pattern after it sends the invoke message. A *must Save point* is marked in a participant of a pattern after a nonrepeatable action, if any.

The example choreography depicted in Fig 1(f) is used to assist in illustration of concepts. In this example, web service ξ interacts with few other web services resulting in a composite web service. **We illustrate checkpointing the web service ξ in this paper, which is applicable to other constituent web services.**

To start with, we define what are called as sequential components. The part of pattern string which is delimited by "." operator is called as a **sequential component** "s" . The sequential components in our example are p_a, $p_b[p_c]$, $p_d[p_e]$, p_f which are named as s_1, s_2, s_3, s_4 respectively (Fig 2(a)).

Each sequential component is a pattern(can be either atomic or composite) and hence has C-points associated with it. Sequential components are referred to as **components** for ease of writing from here on. If the web service ξ initiates n_s number of components we have a maximum of $2n_s$ C-points to be converted into checkpoints. It may not be possible to convert all these C-points since conversion of all the C-points might result in violation of deadlines.

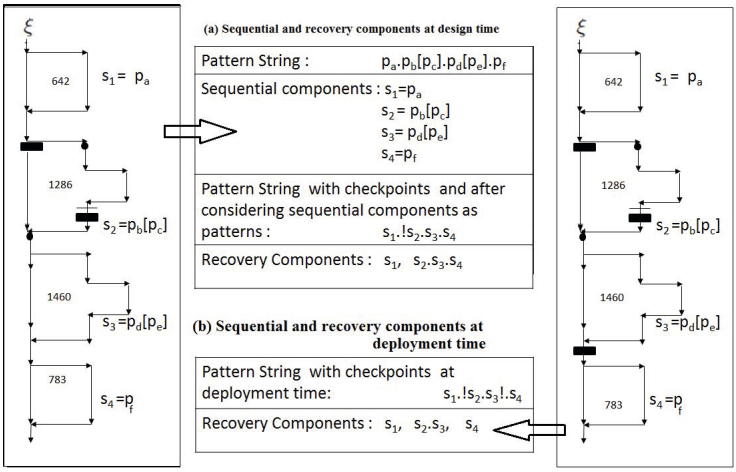

Fig. 2. A choreography and its recovery components

3.2 Recovery Components

A recovery component is defined as an execution unit that is delimited by checkpoints. A failure at anywhere in a recovery component results in rollback to the checkpoint placed at the beginning of the recovery component. In case checkpoints are inserted into a web service either at design time or later, the pattern string that reflects the choreography must reflect the checkpoint locations also. Hence we use the following notation: For a component s if its invocation point is converted to a checkpoint then symbol "!" is added to the left of it, if its service point is converted to a checkpoint then symbol "!" is added to the right of it and if both the C-Points of a component are checkpointed then the component is removed from the pattern string along with its two "!" marks. **A recovery component** is that part of a pattern string which is delimited by "!" mark.

Fig 2(a) depicts the example choreography with checkpoints inserted at design time. It's pattern string annotated with "!" marks at design time is given by $s_1.!s_2.s_3.s_4$. Thus we have initially two recovery components $s_1, s_2.s_3.s_4$ for the example choreography. Fig 2(b) depicts the choreography and its recovery components after a service point is converted into checkpoint at deployment time. By this time we have three recovery components $s_1, s_2.s_3, s_4$.

3.3 Proposed Approach on Deployment Time Checkpointing

We assume that occurrence of transient failures in a web service follow Poisson distribution with the mean failure rate given by λ. It is assumed that failures do not occur during recovery time. Each failure has to be followed by recovery of the failed service for successful completion of its execution.

Recovery of a web service has the following two overheads: **execution time overhead and cost overhead**. Execution overhead is the additional time

required during recovery that includes: i)rollback to the checkpointed state ii) replay logged messaged iii) re-execute unsaved activities. Cost overhead is the additional overhead to be paid to reinvoke a constituent web service in case its reply is not logged. Every web service has deadlines for execution time and cost which have to be met even in case of failures to provide a quality service.

Deployment time checkpointing aims at inserting minimum number of checkpoints to reduce recovery overhead in case of failures so that constraints are met. Minimum number of checkpoints ensure minimal execution time in case of failure free executions. Minimal recovery overhead requires more number of checkpoints to be inserted which results in undesirable increase in time for failure free executions (execution instances which do not fail). Hence we do not aim at minimal recovery overhead, instead we aim at minimum number of checkpoints which result in minimum overhead during failure free executions. Detailed procedure for checkpointing is presented in the next section.

4 Procedure for Deployment Time Checkpointing

Deployment time checkpointing is performed in three stages. 1) Measurement of execution time and other quantities at deployment time. 2)Collection of QoS values of constituent services. 3)Computation of recovery overhead and placement of checkpoints.

4.1 Measurement of Quantities

Initially all the participants of a choreography should insert checkpoints in their code at the locations according to design time checkpointing policy. Let ξ be the participant which is currently being checkpointed. Let T_C, T_L, T_R and T_{CR} represent checkpointing time, message logging time, message replay time and time to restore ξ to a saved state, respectively. These quantities have to be determined experimentally at deployment time. Let C_D represent the cost of service charged by ξ when service is provided with in the promised maximum execution time T_D.

4.2 Collection of QoS Values

We have identified and modeled those QoS attributes which play a crucial role in checkpointing decisions. The considered QoS attributes are response time, vulnerability and cost of service provision.

Let ξ_r be the service provider in a component s initiated by ξ. **Response time** $s.t_{rt}$ of s, is defined as the response time of ξ_r. **Vulnerability** $s.vl$ of s to failures is defined as vulnerability of ξ_r which in turn is defined using reliability of ξ_r. **Reliability** rl is defined as the success rate of the service i.e. the ratio of number of times the service is successfully delivered to total number of service invocations. $\xi_r.vl = 1 - (\xi_r.rl)$. Cost of a service is defined as the price to be paid for providing the requested service. **Cost of service** $s.ct$ of s is defined as cost of service provided by ξ_r of the component.

4.3 Computation of Recovery Overhead

Recovery overhead is measured in terms of additional execution time and cost of service to be paid, to recover ξ from failures. Recovery overhead for each of the recovery components is computed using the quantities measured at deployment time and QoS attributes defined above.

4.3.1 Execution Time Overhead

For each of the components s, initiated by the participant ξ, let $s.t_{at}$ and $s.t_{rt}$ represent average local activity time and response time of the callee. These values are determined experimentally at deployment time. For every component s, there can be only one invocation point, one service point, and one or more must save points in ξ. According to our design time checkpointing policy[12], only invocation point and must save points of a component may be converted into checkpoints, service points cannot be. Let $s.n_{mp} \geq 0$ and $s.n_{ip} \in (0, 1)$ represent number of must save points and invocation points which are converted into checkpoints at design time. Let $s.n_{sp} \in (0, 1)$ represent number of service points which are converted into checkpoints. According to the design time policy, $s.n_{sp} = 0$ indicating that zero service points are checkpointed yet.

Let $T_{pure}(s)$ represent pure execution time of a component s without including any of checkpointing and message logging times. $T_{pure}(s) = max(s.t_{at}, s.t_{rt})$. Let $T_{fixed}(s)$ represent the fixed execution time of component s after including checkpointing and message logging times for checkpoints and logs inserted at design time. Checkpointing time is given by $(s.n_{mp} + s.n_{ip}) * T_C$. These checkpoints are never revised at deployment time and run time. Hence we call it as $T_{fixed}(s)$. If an invocation point or a must save point is converted into checkpoint, the reply message received by ξ has to be logged, refer [12]. Let $s.n_{ml} \in (0, 1)$ represent number of message logs at design time for component s. $T_{fixed}(s) = T_{pure}(s) + (s.n_{mp} + s.n_{ip}) * T_C + s.n_{ml} * T_L$.

We use the notation $\alpha_0(s)$ to represent $T_{fixed}(s)$, 0 indicates that 0 C-points of s are converted into checkpoint in deployment stage. Let $T_{fixed}(\xi)$ represent fixed execution time of the participant ξ before converting any of the C-points of ξ into checkpoints in deployment stage. It is in short represented as $\alpha_0(\xi)$ where 0 indicates that 0 C-points are already converted into checkpoints in deployment stage. $T_{fixed}(\xi) = \alpha_0(\xi) = \sum_{i=1}^{n_s} \alpha_0(s_i)$ where n_s is number of components initiated by ξ.

$\alpha_k(\xi)$ represents **failure free execution time** of ξ after k C-points are converted into checkpoints and lg messages are logged in deployment stage. $\alpha_k(\xi) = \sum_{i=1}^{n_s} \alpha_0(s_i) + k * T_C + lg * T_L$.

Let $T_W(\xi)$ represent **worst case execution time** of the participant ξ which is defined as follows: $T_W(\xi) = \alpha_k(\xi) + T_{rec}(\xi)$, which includes $\alpha_k(\xi)$, and time $T_{rec}(\xi)$ to recover the web service in case of its failure. To satisfy the promised maximum execution time, $T_W(\xi)$ must be $\leq T_D$. In the next subsection we describe recovery time $T_{rec}(\xi)$ computation.

Recovery Time Computation. In a component if its invocation point is checkpointed, then the service $s.\xi_r$ is not invoked again and hence $T_{rec}(s)$ includes i) the time taken, T_{CR}, to restore the web service from its latest checkpoint, ii) message replay time T_R and iii) time to execute unsaved local activities. Maximum unsaved work in a component results when a failure occurs almost at the end of the component, i.e just before receiving reply. If s's invocation point is converted into checkpoint at design time $T_{rec}(s) = T_{CR} + T_R + s.t_{at}$, else $T_{rec}(s) = max(s.t_{at}, s.t_{rt})$. Table 2 (fifth column) gives recovery time values for the example choreography. $T_{rec}(r) = \sum_{s \in r} f(s) * T_{rec}(s)$, where $f(s) = (\lambda * T_{rec}(s))$ gives average number of failures in the component s. $T_{rec}(\xi) = \sum_{i=1}^{n} T_{rec}(r_i)$ where n is the number of recovery components of ξ.

Relative recovery time of component s is defined as: $T_{rrec}(s) = \frac{T_{rec}(s)}{T_{rec}(r)}$.

Relative recovery time of recovery component r is: $T_{rrec}(r) = \frac{T_{rec}(r)}{T_{rec}(\xi)}$.

4.3.2 Cost Overhead

Let C_{total} indicate failure free cost of service for all the components of ξ, $C_{total} = \sum_{i=1}^{n_s} s_i.ct$. Let $C_{rec}(s)$ indicates additional cost incurred in component s during recovery in case of a failure.

In a component if its invocation point is checkpointed, then the service ξ_r is not invoked again and hence there is no additional cost of invocation. Thus, $C_{rec}(s) = 0$ if $s.n_{ip} = 1$, else $C_{rec}(s) = s.ct$.
$C_{rec}(r) = \sum_{s \in r} f(s) * C_{rec}(s)$. and $C_{rec}(\xi) = \sum_{i=1}^{n} C_{rec}(r_i)$.

Relative recovery cost for s is defined as: $C_{rrec}(s) = \frac{C_{rec}(s)}{C_{rec}(r)}$.

Relative recovery cost for r is defined as: $C_{rrec}(r) = \frac{C_{rec}(r)}{C_{rec}(\xi)}$.

Let $C_W(\xi)$ indicate worst case cost of service which is defined as $C_W(\xi) = C_{total} + C_{rec}(\xi)$. Cost constraint is satisfied if $C_W(\xi) \leq C_D$.

4.3.3 Checkpointing Score

Checkpointing score cs of a component s quantifies suitability of the component for checkpointing. It is defined as the following weighted sum. Higher values of cs indicate greater need for checkpointing the component.

$$s.cs = W_1 * T_{rrec}(s) + W_2 * s.vl + W_3 * C_{rrec}(s)$$

where $\sum_{i=1}^{3} W_i = 1$ and they represent weights to be used to alter the effects of individual components on the checkpointing score. Similarly, checkpointing score cs of a recovery component r is defined as:

$$r.cs = W_1 * T_{rrec}(r) + W_2 * r.vl + W_3 * C_{rrec}(r).$$

4.3.4 Deciding on Checkpoint Locations

Recovery component r_j which has the highest checkpointing score is selected for checkpointing. From among all components of r_j, the component s_o that has got

maximum checkpointing score is selected for placing the checkpoint. Further, if local activity time of s_o is greater than or equal to s_o's response time the next checkpoint is placed at service point of s_o. Else the next checkpoint is placed at invocation point of s_o, i.e if s_o's response time is larger, the service provider of s_o is not reinvoked in the event of failure of ξ by placing a checkpoint at the invocation point (refer to Fig 1 (d,e)).

5 Time and Cost Aware Checkpointing Algorithm

We propose Time and Cost aware Checkpointing algorithm [Algorithm 1] that incrementally converts C-points into checkpoints, one at a time. It takes as input all the quantities measured and collected at deployment stage, and pattern string representing the composition with checkpoints inserted at design stage (line no 1). Algorithm starts by computing recovery overhead of all components (line nos 4,5). In each iteration it finds out recovery components and computes their execution cost and time overhead (line nos 12,14). It then computes worst case execution cost $C_W(\xi)$ and time $T_W(\xi)$ (line nos 13,16). Then it checks to see if $T_W(\xi)$ is within T_D and $C_W(\xi)$ is within C_D. If so, it terminates. Generally $T_W(\xi)$ is large and is greater than T_D. After conversion of a C-point into checkpoint, $T_W(\xi)$) decreases. Also, total checkpointing time increases. We continue converting C-points into checkpoints until time and cost constraints are met. Exact checkpointing location is decided using the deployment checkpointing strategy presented in subsection 4.3. Due to space constraints we do not present the algorithm $RecoveryComponents(ps, \xi)$ that extracts recovery components from a given pattern string.

5.1 Experimental Results

To illustrate the effectiveness of our approach, we have developed, tested and monitored the performance of sample web services using Oracle SOA suite 11g [7]. We have used weblogic server which is configured to work as SOA server. Oracle JDeveloper is used for development and deployment of web services. Oracle Enterprise Manager which is also a part of Oracle SOA suite, is used to collect performance related metrics. The weblogic server is hosted on a machine using 4GB RAM, 2.13GHz CPU and J2SDK5. Oracle 11g Database is installed on a machine having 4GB RAM and 3.00GHz CPU. All the PCs run on Windows 7 OS and are connected via high speed LAN through 100Mbps Ethernet cards.

To aid in our experimental results, we have developed the following web services: *WS1:* web service which invokes *currency converter* web service and then *calculator* web service. *WS2:* web service which invokes *currency converter* web service. *WS3:* web service which invokes *calculator* web service. *WS4:* web service takes an Indian state and returns its capital. *WS5:* web service which returns the Indian state to which a given capital city belongs. *WS6:* web service which invokes four web services in the order: WS4, WS3,WS2,WS5. For testing the efficiency of the proposed algorithm, we have used web service WS6, the

Algorithm 1. Time and Cost aware Checkpointing Algorithm

Input: Read $T_C, T_L, T_R, T_{CR}, T_D, C_D, \lambda$ and QoS values of components, pattern string ps.
$k = 0$//k indicates number of C-points checkpointed at deployment time
$lg = 0$// ml indicates number of messages logged at deployment time
for *each component* s **do**
    Compute $C_{rec}(s)$, $T_{rec}(s)$
$C_{total} = \sum_{i=1}^{n_s} s_i.ct$
$R = RecoveryComponents(ps, \xi)$
while R *is not Null* **do**
    $n = |R|$ // Number of recovery components
    for *each recovery component* $r \in R$ **do**
        Compute $C_{rec}(r)$, $T_{rec}(r)$
    $C_{rec}(\xi) = \sum_{i=1}^{n}(C_{rec}(r_i))$
    $C_W(\xi) = C_{total} + C_{rec}(\xi)$
    $T_{rec}(\xi) = \sum_{i=1}^{n}(T_{rec}(r_i))$
    $\alpha_k(\xi) = \alpha_0(\xi) + k * T_C + lg * T_L$
    $T_W(\xi) = \alpha_k(\xi) + T_{rec}(\xi)$
    if $(T_W(\xi) \leq T_D)$ *and* $(C_W(\xi) \leq C_D)$ **then**
        break. // deadlines can be met
    //else select a component from recovery component r for checkpointing
    for *each recovery component* $r \in R$ **do**
        $C_{rrec}(r) = \frac{C_{rec}(r)}{C_{rec}(\xi)}$
        $T_{rrec}(r) = \frac{T_{rec}(r)}{T_{rec}(\xi)}$.
        $r.cs = CHS(r)$ //CHS computes checkpointing score of r
    j = index of recovery component whose cs is returned by $\max_{i=1}^{n}(r_i.cs)$
    Let the recovery component r_j consist of sequential components from index l to l+h where $1 \leq h \leq n_s - l$
    for *each component* $s \in r$ **do**
        $s.cs = CHSP(s)$ //CHSP computes checkpointing score of s
    o = index of component whose cs is returned by $\max_{i=l}^{l+h}(s_i.cs)$
    if $s_o.t_{at} \geq s_o.t_{rt}$ **then**
        $s_o.n_{sp} = 1$ //Place the next checkpoint at service point of s_o
    else
        $s_o.n_{ip} = 1$ //Place the next checkpoint at invocation point of s_o
        $s_o.n_{ml} = 1$ //log the reply message received
        $lg = lg + 1$
    k=k+1
    $R = RecoveryComponents(ps, \xi)$
Print no of checkpoints inserted = k

Table 1. Execution Time Parameters for WS6

Quantity	**Value** Time in msec, Cost in units
T_C	78
T_L	24
T_R	15
T_{CR}	36
$T_{pure}, \alpha_0(\xi)$	4171, 4171+78+24=4273
T_D, C_D	6000,150
W_1, W_2, W_3, λ	0.333,0.333,0.333,.001

Table 2. Checkpointing score calculation for constituent web services

s	**WS**	$s.t_{at}$	$\alpha_0(s)$	$T_{rec}(s)$	$s.vl$	$s.ct$	$C_{rec}(s)$	cs	$f(s)$
s_1	WS4	125	642	642	0.43	25	25	(0.2533+0.43+0.555) *0.333=0.413	0.642
s_2	WS3	240	1286	$240+36+15=291$	0.56	30	0	(0.115+0.56+0)* 0.333=0.225	0.291
s_3	WS2	1024	1460	1460	0.38	25	25	(0.576+0.38+0.555)* 0.333=0.504	1.460
s_4	WS5	245	783	783	0.24	20	20	(0.308+0.24+0.444)* 0.333=0.330	0.783

Table 3. Algorithm 1 Trace for k=1

Quantity	**Value**
Q, Q_0	$\{s_1, s_2, s_3, s_4\}$
$R = \{r_1, r_2\}$	$r_1 = s_1$,$r_2 = s_2.s_3.s_4$
$C_{rec}(\xi)$	$\sum(C_{rec}(r_1), C_{rec}(r_2)) = \sum(25*0.642, 0+25*1.46+20*0.783) = 68$
$T_{rec}(\xi)$	$\sum(T_{rec}(r_1), T_{rec}(r_2)) = \sum(642*.642, 291*.291+1460*1.46+783*.783) = 3241$
$\alpha_k(\xi), T_W(\xi)$	4273 +0 = 4273, 4273+3241=7514
$C_{total}(\xi), C_W(\xi)$	100,68
r_1	$T_{rrec} = \frac{642}{3241} = 0.198, vl = 0.43, C_{rrec} = \frac{25}{68} = .5555, cs = .367$
r_2	$T_{rrec} = \frac{2534}{3241} = .8727, vl = 0.81, C_{rrec} = \frac{45}{68} = .661, cs = .780$
k, j, l, h	0, 2, 2, 2
$\max(s_2.cs, s_3.cs, s_4.cs)$	$\max(0.225, 0.504, 0.330) = 0.504$
o	3, Place checkpoint at ip of s_3 and log reply message.

choreography document of which is depicted in Fig 2(a). Numbers in the figure depict fixed execution time, $\alpha_0(s)$, of each of the components.

Table 1 depicts the values of various required quantities that are recorded and read into the algorithm. Table 2 shows checkpointing score calculation for components. Table 3 gives a trace of the algorithm for $k = 1$. According to computations component s_3 is selected for converting its invocation point into checkpoint.

Graphs in Fig 3 depict variation of important quantities computed by the algorithm with increase in k value. It can be seen that T_W and C_W cross time deadline for $k = 0$ (Fig 3(c),(d)). T_W and C_W decrease considerably after converting a C-point into checkpoint. Also, failure free execution time increases due to this checkpoint. But conversion of another C-point into checkpoint is not taken up because deadline is already met with $k = 1$ and also failure free execution time increases further with $k = 2$ (Fig 3(b)). Hence the algorithm terminates when k=1.

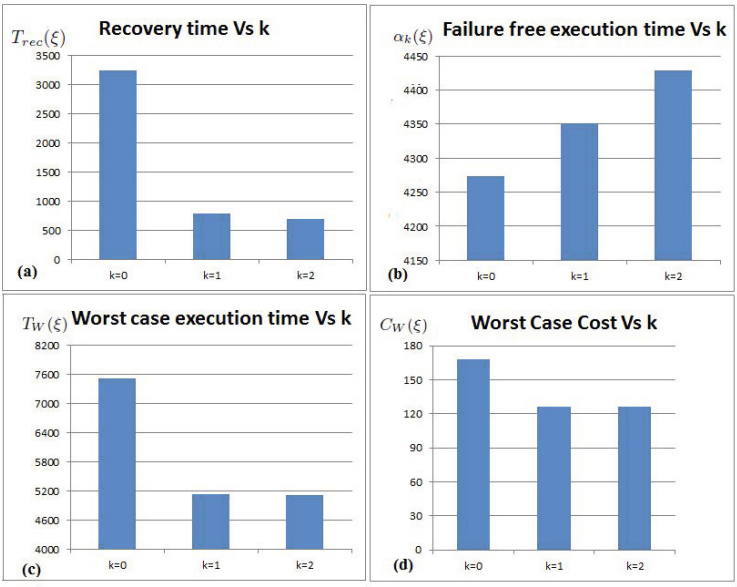

Fig. 3. Plots of important quantities in three iterations of algorithm

6 Conclusion and Future Work

In this paper we have detailed on time and cost aware checkpointing algorithm that has to be adopted at deployment stage for resilient execution of choreographed web services. We define units of execution called as recovery components and compute their checkpointing score by using execution time and other

quantities measurable at deployment time and QoS values of constituent services. It aims at introducing minimum number of checkpoints in a web service so that time and cost constraints are met during both failure free executions and failed and recovered executions. Web services operate in highly dynamic Internet where response times are bound to vary from the advertised values. We intend to take up run time revision of checkpoint locations as part of our future work. It is mainly intended to consider actual values of all response times and revise checkpoint locations accordingly.

References

1. Chen, N., Yu, Y., Ren, S.: Checkpoint interval and system's overall quality for message logging-based rollback and recovery in distributed and embedded computing. In: ICESS 2009 (2009)
2. Elnozahy, E.N(M.), Alvisi, L., Wang, Y.-M.: A survey of rollback-recovery protocols in message-passing systems. ACM Comput. Surv. (2002)
3. Ezenwoye, O., Sadjadi, S.M.: Trap/bpel: A framework for dynamic adaptation of composite services. In: Proc. WEBIST 2007 (2007)
4. Mansour, H.E., Dillon, T.: Dependability and rollback recovery for composite web services. IEEE Transactions on Services Computing (2011)
5. Liu, A., Qing, L., Huang, L., Xiao, M.: Facts: A framework for fault-tolerant composition of transactional web services. IEEE Transactions on Services Computing (2010)
6. Elnozahy, E.N.(M.), Alvisi, L., Wang, Y.-M.: A survey of rollback-recovery protocols in message-passing systems. ACM Comput. Surv. (2002)
7. Oracle White Paper, editor. Oracle SOA Suite 12c A Detailed Look. Oracle (2014)
8. Darby, P.J., Tzeng, N.-F.: Decentralized qos-aware checkpointing arrangement in mobile grid computing. IEEE Transactions on Mobile Computing (2010)
9. Rukoz, M., Cardinale, Y., Angarita, R.: Faceta*: Checkpointing for transactional composite web service execution based on petri-nets. Procedia Computer Science (2012)
10. Urban, S.D., Gao, L., Shrestha, R., Courter, A.: Achieving recovery in service composition with assurance points and integration rules. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM 2010. LNCS, vol. 6426, pp. 428–437. Springer, Heidelberg (2010)
11. Vani Vathsala, A., Mohanty, H.: Using hmm for predicting response time of web services. In: Proceedings of the CUBE International Information Technology Conference. ACM (2012)
12. Vani Vathsala, A., Mohanty, H.: Interaction patterns based checkpointing of choreographed web services. In: Proc of the 6th International Workshop on Principles of Engineering Service-Oriented and Cloud Systems. ACM (2014)
13. Vathsala, A.V., Mohanty, H.: A survey on checkpointing web services. In: Proc of the 6th International Workshop on Principles of Engineering Service-Oriented and Cloud Systems. ACM (2014)
14. Vathsala, A.V., Mohanty, H.: Web service response time prediction using hmm and bayesian network. In: Jain, L.C., Patnaik, S., Ichalkaranje, N. (eds.) Intelligent Computing, Communication and Devices. AISC, vol. 308, pp. 327–335. Springer, Heidelberg (2015)