International Journal on Future Revolution in Computer Science & Communication Engineering
Volume: 3 Issue: 12
ISSN: 2454-4248
383 – 387

Dynamic Checkpointing of Composite Web Services

Vani Vathsala A
Dept of CSE, CVR College of Engineering, Hyderabad, India
e-mail: atlurivv@yahoo.com
Tel.: +919866586106

Abstract Web services provide services to their consumers in accordance with terms and conditions laid down in a document called as Service Level Agreement (SLA). Web services have to abide by these terms and conditions failing which, SLA faults result. Fault handling of web services is a key mechanism using which SLA faults can be avoided. We propose fault handling of choreographed web services using **checkpointing and recovery.** We propose checkpointing in three stages: design, deployment and **dynamic** checkpointing. We have presented first two stages of checkpointing in our earlier publications. In this paper we discuss the need for dynamic checkpointing and, various factors to be considered while revising checkpoint locations dynamically. We also propose a framework for implementing dynamic checkpointing.

Keywords- *web service, web service, checkpointing, Service level Agreement*

*****

I. INTRODUCTION

A web service is a piece of software that provides a service and is accessible over Internet. If a web service provides a service without invoking another web service, then it is called as an atomic web service. Otherwise it is called as a composite web service. Participating services (or participants) of a composite service exchange messages according to the sequence specified in a document, called as choreography document, and the composite service is called as choreographed service.

A business application to be developed as a composite web service is engineered in the following three stages: Design, development & deployment, and execution.

Design: List of roles of various participating services, actions to be performed by them, Sequence of interactions (message exchanges) among the participating services and, details of data items exchanged are specified in a document called as Choreography document.

Development and Deployment: When a web service is installed on its web server and is ready to accept requests, the web service is said to be deployed. Web services advertise their Quality of Service (QoS) attributes like response time, cost of service etc at the time of deployment. These advertised QoS values aid service consumers in selection of suitable services.

Execution: When a service consumer invokes a composite web service, all the constituent services are executed in the sequence given in the design document.

Web services need to be equipped with a fault handling mechanism to provide reliable services. Checkpointing is a time tested technique that has been used in several areas like databases, distributed computing etc for handling faults [7]. Checkpointing is a proactive technique which prescribes to save the state of an application so as to enable its recovery in case of any failure of the application at a later time. A failed application rolls back to a previously checkpointed state and continues its execution from there on.

We have not come across any work on policy-based checkpointing of composite web services to ensure efficient handling of faults and subsequently avoid SLA faults. Any web service checkpointing scheme has to consider the following important characteristics of web services [9]:

1. Composite nature of web services and non repeatability of actions. 2. Compliance to SLA. 3. Dynamic selection of constituent web services. 4. Dynamic nature of the Internet and web server environments. Considering the above characteristics of web services, we propose the following three stage checkpointing strategy, refer to Figure 1:

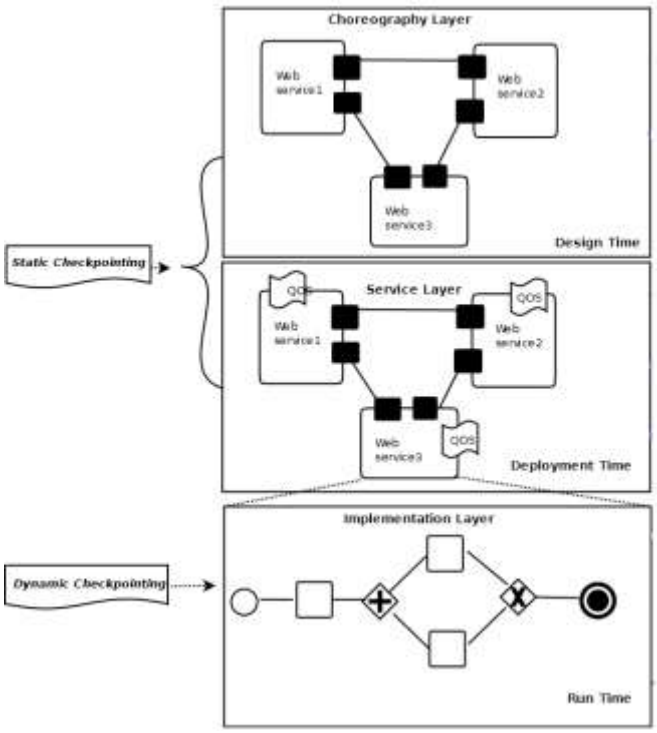

Figure 1. Three Stage Checkpointing of a Composite Web Service

 ISSN: 2454-4248
383 – 387

In the first stage, called **design stage[1]**, we propose checkpointing locations using choreography document. In this stage checkpoints are placed in such a way that web services performing non repeatable actions are not invoked again in the event of failure of the invoking web service.

In second stage of checkpointing done at deployment time called as **time and cost aware checkpointing [2],** we propose checkpointing locations in a composite web service such that time and cost deadlines as specified in SLA are met even in case of transient failures and subsequent recovery.

Web services advertise their Quality of Service attribute values in order to aid consumers in selecting web services that suit their requirements. We use these advertised values in deciding checkpoint locations. These checkpoints and checkpoints generated at design time are inserted into web services in their code before they are deployed on their servers.

For certain web services, actual QoS values tend to deviate from advertised QoS values (particularly response time values). For certain other composite web services, some of the constituent web services are selected dynamically. In such cases, there is a need to revise checkpointing locations at run time for efficient execution of web services. Hence, in this paper we propose **dynamic checkpointing** which revises checkpointing locations in a composite web service at run time. It uses predicted response time values to decide on checkpoint locations.

This paper is organized as follows: Section II presents a study on contemporary work in the field of dynamic checkpointing and urges the need for checkpointing web services at run time. Section III elucidates the need for dynamic checkpointing, and section IV explains the propounded run time checkpointing strategy. Section V presents the framework designed to perform dynamic checkpointing, section VI presents conclusion and future work.

II. RELATED WORK

In this section we present a survey on dynamic checkpointing strategies proposed in web services and related areas.

Authors of [3] propose an adaptive algorithm named MeanFailureCP+ that deals with checkpointing of grid applications with execution times that are unknown apriori. The algorithm modifies its parameters, based on dynamically collected feedback on its performance. MeanFailureCP+ monitors dynamically the number of jobs processed during a monitoring interval of predefined length and based on this feedback modifies subsequent job length estimates in such a way that the checkpointing overhead is minimized without significantly penalizing the system fault-tolerance. The approach allows for periodic modification of checkpointing intervals at run-time, when additional information becomes available.
This algorithm increases checkpointing interval by a value given by the end user when remaining execution time is lesser than average failure interval. Else, if job failure interval is lesser than remaining execution time, it reduces checkpoint interval by a value given by end user. In modified version of the algorithm it does not ask the user to give exact job length value, but estimates it from current number of requests. If number of jobs in last interval >= number of jobs in current interval, it reduces the job length by 0.1%, otherwise increases it by 0.1%.

In [4], authors propose an adaptive task checkpointing based job scheduling scheme for grid environments; Whenever a grid resource broker has tasks to schedule on grid resources, it makes use of the fault index (No of jobs not successfully completed gives fault index). They propose to maintain and update the fault index of all available resources of the grid. The fault index of the grid resource will suggest its vulnerability to faults (i.e., higher the fault index, higher is the failure rate). The Fault Tolerant Schedule Manager (FTScheduleManager) maintains fault index history. A centralized checkpoint manager CPManager maintains information of partially executed tasks by the grid resources.
It maintains details of last successful checkpoints taken by jobs. Grid resource broker allocates jobs to resources based on fault index.

H.E.Mansour and T.Dillon [5] propose a service oriented reliability model that dynamically calculates the reliability of a composite web service and places checkpoints in the composite web service using expected recovery time.

Work proposed in [6] uses predicted server load to adjust checkpoint frequency for high throughput data services. The authors present programming and runtime support called SLACH for building multi-threaded high-throughput persistent services. In order to keep in-memory objects persistent, SIACH employs application-assisted logging and checkpointing for log-based recovery while maximizing throughput and concurrency. SIACH adaptively adjusts checkpointing frequency based on log growth and throughput demand to balance between runtime overhead and recovery speed.

We propose a holistic approach to decide on checkpoint locations which considers 1) dynamic QoS values of all the invoked web services 2)failure rate variations of the web service to be checkpointed and 3) provision of dynamic composition of the web service to be checkpointed.

III. NEED FOR DYNAMIC CHECKPOINTING

Revision of checkpoint locations is required for those web services for which the following conditions hold good: 1. Actual response time varies largely from the advertised response time. 2. Dynamic composition of constituent web services of Ɛ at run time is facilitated. 3. Failure rate of Ɛ deviates significantly from the projected failure rate at deployment.

In the following subsections we detail upon these three scenarios.

A. Actual Vs advertised response time values

Deployment checkpointing proposed in [2], makes use of response time of the invoked services in deciding checkpointing locations for a composite web service Ɛ. The considered response times are either advertised by providers or measured by the service requesters/ consumers. In any case, response time value considered at deployment time represents an average of a set of values collected under varied conditions. However, at run time, response times vary from the considered average values. At times, this difference may be considerable due to variations in the underlying network traffic and computing environment.

In case there is a significant difference between the considered average response time values and the actual values at run time of a constituent web service of Ɛ, revising checkpoint locations in Ɛ would improve its performance. There are two important scenarios to be considered here:

1. Actual Response time of a web service is significantly greater than its advertised average response time. This scenario demands addition of checkpoint locations to satisfy constraints on execution time even when an instance of Ɛ fails and recovers.
2. Actual Response time of a web service is significantly lesser than its advertised average response time. In this scenario, removal of some checkpoint locations results in improved execution time of failure free instances of Ɛ.

Hence to improve performance of a checkpointed composite web service we propose to revise checkpointing locations at run time. In the following subsection we state another reason which urges for dynamic revision of checkpoint locations.

B. Dynamic composition

Certain composite web services allow for dynamic selection of some of their constituent web services. Dynamic composition can be done 1) either from a list SD of statically discovered (at deployment time) web services or 2) from a dynamically discovered and selected web services. In both the cases, it's not known at deployment time which web service would be selected at run time. Hence in first case, we have proposed to use the worst case advertised QoS values (maximum value in case of response time and cost, and minimum value in case of reliability) of the web services which are in the static list SD. In case of discovery of a service dynamically, we would not even have a list of shortlisted web services at deployment time and hence we propose to use maximum permissible QoS values.

In both the cases, there would be large differences in QoS values considered at deployment and those considered at run time. Hence checkpoint locations have to be revised dynamically to allow for better performance of the composite web service Ɛ.

C. Change in failure rate

After deployment, there may be changes in the failure rate of the composite web service Ɛ. If difference between the new failure rate and the failure rate projected at deployment is significant, there would be considerable differences in recovery overhead of each of the recovery components. Hence to tune the performance of Ɛ accordingly, checkpoint locations have to be revised.

We advocate revision of checkpoint locations dynamically, due to the above specified reasons. In the following section we describe the proposed strategy for revision of checkpoint locations

IV. RUN TIME CHECKPOINTING STRATEGY

In this section, we present our proposed Run time checkpointing Strategy. In our work proposed in [2], we introduced deployment time checkpoint locations in a composite web service Ɛ by making use of **advertised** QoS values of Ɛ and also those of web services invoked by Ɛ.

Ideally, **actual** response times of web services should be used for dynamic checkpointing. But, actual response time of a web service can be obtained only after its invocation. Hence predicted response time which would be almost equivalent to actual value is used. Response times of web services differ from time to time; at peek business hours response time may increase due to heavy traffic, leading to slow message flow.

We have proposed a traffic aware response time prediction strategy in [8] that considers equi-length time intervals (of length **t** time units). Each interval characterizes a network condition. Here, for discussion, we have considered 10mins time interval. In practice, the interval can be decided considering a day long traffic patterns. If **t**=*10mins*, then each hour would be divided into 6 intervals making up to a total of 144 time intervals per day. At the beginning of each time interval, response time of each of the web services invoked by Ɛ is predicted.

Since predicted response time values remain same in an interval, run time checkpointing algorithm is to be run at the beginning of every time interval to adjust checkpoints in $\mathcal{E}$. If the web service Ɛ is dynamically composed from a statically discovered set **SD** of services, then we predict response time of each of the web services in the set **SD**. If service provider $Ɛ_r$ in a component *s* of Ɛ is dynamically selected from a static list $\mathbf{SD_s}$ where $\mathbf{SD_s}$ is in **SD**, then instead of using maximum value of the **advertised** response times of the probable web services $\mathbf{SD_s}$ we propose to use maximum value of the **predicted** response times for computation of recovery time overhead for the component *s*. Other QoS values (Reliability and Cost of service) of $Ɛ_r$ are same as those taken at deployment.

If Ɛ is dynamically composed from a set of services discovered at run time, there would be no list of probable web services at our disposal and hence we cannot predict any response time values. Hence, in this case, we propose to use maximum permissible values initially at the beginning of each time interval. When actual execution of Ɛ takes place and actual response time of a dynamically selected constituent service of Ɛ becomes available, we propose to adjust checkpoint locations in the remaining components to be executed. This strategy is not to be included in run time checkpointing algorithm which is run at the beginning of each time interval,

but is instrumented into the code of web service Ɛ. Current failure rate of Ɛ is also considered for revision of checkpoints to decide whether revision of checkpoints is to be carried out or not.

V. FRAMEWORK FOR REVISION OF CHECKPOINTS

In this section, we describe the proposed framework required for introducing checkpoints at run time in a composite web service Ɛ.

Three main components of this framework are: Prediction Middleware (PM), Failure rate Measurement Module (FMM) and Dynamic Checkpointing Module (DCM). Figure 2 depicts the framework required for revision of checkpoints at runtime.

Prediction Middleware (PM) predicts response time values of web services invoked by Ɛ. At the beginning of each time interval *T*, PM would predict response time values for each of the web services possibly invoked by Ɛ, according to the given choreography. PM mainly consists of two modules: Black Box Module and White Box Module.

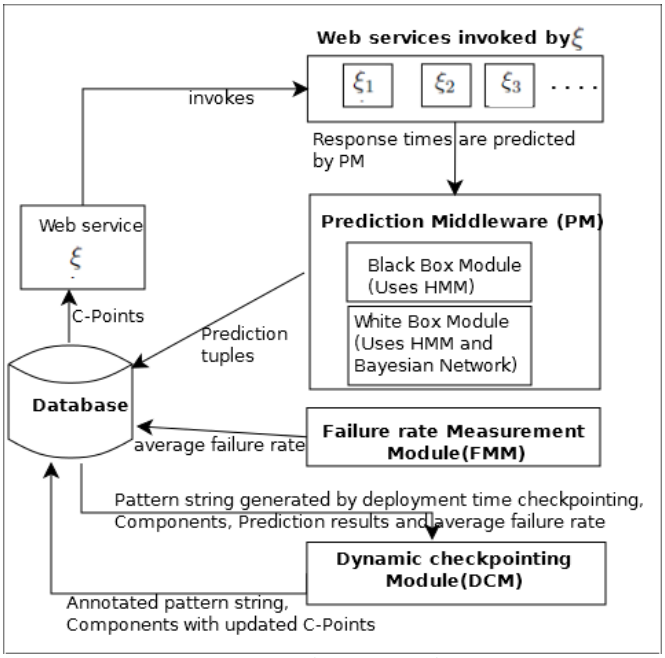

Figure 2: Framework for Revision of Checkpoints

If a web service $Ɛ_r$, invoked by Ɛ does not reveal its implementation details (internal structure of the web service, web server environment details like average waiting time in a given time interval), then PM uses black box approach to predict response time of $Ɛ_r$. If a web service $Ɛ_r$, invoked by Ɛ does reveal its implementation details, then PM uses white box approach to predict response time of $Ɛ_r$. The prediction results are made available in the form of tuples called as prediction tuples and are stored in the database at Ɛ. These predicted values are used for checkpointing decisions made during the time interval **t**.

Each prediction tuple ***pt*** is represented as **($t, Ɛ_r, ta, t_{prt}$)**
where
t is the time interval.
$Ɛ_r$ is the invoked web service.
ta is the type of approach used. **ta**=1 for black box approach and **ta**=2 for white box approach.
t_{prt} is the predicted response time in msecs.

If white box approach is used for predicting response time of a web service $Ɛ_r$ for each time interval **t,** as many predictions would be done as there are equivalence classes for input vector of $Ɛ_r$ (All input vector values that result in same execution path are grouped under one equivalence class. Each equivalence class results in different response time.) But maximum predicted response time value among them for **t** would be used to represent the predicted response time of Ɛr for the time interval **t.**

The latest expected failure rate λ_t of Ɛ would also be made available by a module called as **Failure Management Module (FMM)** at the beginning of each time interval **t.** We do not discuss here, about the approaches to be used for computing λ_t , any of the classic approaches may be used to determine λ_t .

Dynamic Checkpointing Module (DCM) takes up the task of revising checkpoint locations at run time by invoking Run time checkpointing algorithm. DCM invokes the algorithm at the beginning of every time interval **t**, after prediction tuples are made available by PM. DCM finally updates the components with modified C-points (Checkpoints), and stores them in the database. When an instance of Ɛ starts executing, it saves its state at all places where C-points are set.

VI. CONCLUSION AND FUTURE SCOPE

In this paper we have argued upon the need to checkpoint a composite web service Ɛ dynamically, the reasons being: difference in actual and advertised response times of web services invoked by Ɛ, failure rate variation of Ɛ at run time and provision of dynamic composition of Ɛ.

We have also presented a framework that mainly consists of three modules: prediction module, dynamic checkpointing module and failure rate management module.
As part of future work, we propose to develop the run time checkpointing algorithm that revises checkpoints in a composite web service at run time.

REFERENCES

[1] Vani Vathsala A, HrushikeshaMohanty, "*Interaction patterns based checkpointing of choreographed web services".* In the Proceedings of the 6th International Workshop on Principles of Engineering Service Oriented and Cloud Systems (PESOS 2014),Hyderabad, India, pp 28–37,**2014.**

[2] Vani Vathsala A, HrushikeshaMohanty, *"Time and cost aware checkpointing of choreographed web services",* In the Proceedings of the 11th International Conference on Distributed Computing and Information Technology (ICDCIT 2015), India, pp 207–219,**2015.**

[3] Chtepen M, Dhoedt B, De Turck F, Demeester P, Claeys F, Adaptive checkpointing in dynamic grids for uncertain job durations. In the Proceedings of the 31st International

 ISSN: 2454-4248
383 – 387

Conference on Information Technology Interfaces, (ICITI) pp 585–590, **2009.**

[4] Nazir, Qureshi, and Manuel. Nazir B, Qureshi K, Manuel P, " *Adaptive checkpointing strategy to tolerate faults in economy based grid*". The Journal of Supercomputing, Vol.50, Issue 1, pp 1–18, **2009**

[5] HE Mansour, TDillon, "*Dependability and rollback recovery for composite web services*". IEEE Transactions on Services Computing Vol 4, Issue 4,pp 328–339,**2011.**

[6] Zhou J, Zhang C, Tang H, Wu J, Yang T, "*Programming support and adaptive checkpointing for high-throughput data services with log-based recovery"*, In the Proceedings of IEEE/IFIP International Conference on Dependable Systems and Networks pp 91–100, 2010.

[7] Chandy KM, Lamport L, "*Distributed snapshots: determining global states of distributed systems*". ACM Transactions on Computer Systems Vol 3, Issue 1, pp 63–75, 1985.

[8] Vani Vathsala and Hrushikesha Mohanty, "*Using hmm for predicting response time of web services*", In the Proceedings of the CUBE International Conference, Pune, India, pp 520–525, 2012.

[9] Vani Vathsala A, HrushikeshaMohanty, "*A Survey on checkpointing web services".* In the Proceedings of the 6th International Workshop on Principles of Engineering Service Oriented and Cloud Systems (PESOS 2014),Hyderabad, India, pp 11–17,**2014.**