International Journal of Computer and Communication Technology

Volume 5 | Issue 4 Article 1

October 2014

Checkpointing Orchestrated Web Services

A. Vani Vathsala
CVR College of Engineering, JNTU, Hyderabad, India, atlurivv@yahoo.com

Hrushikesha Mohanty
Department of Computer and Information Sciences, University of Hyderabad, Hyderabad, India.,
mohanty.hcu@gmail.com

Checkpointing Orchestrated Web Services

A.Vani Vathsala
Department of Computer Science
CVR Engg College University of Hyderabad
atlurivv@yahoo.com

Hrushikesha Mohanty
Department of Computer and Information Sciences
University of Hyderabad
mohanty.hcu@gmail.com

Abstract- Web Services are built on service-oriented architecture which is based on the notion of building applications by discovering and orchestrating services available on the web. Complex business processes can be realized by discovering and orchestrating already available services on the web. In order to make these orchestrated web services resilient to faults; we proposed a simple and elegant checkpointing policy called "Call based Global Checkpointing of Orchestrated web services" which specifies that when a web service calls another web service the calling web service has to save its state. But performance of the web services implementing this policy reduces due to checkpointing overhead. In an effort to improvise this policy, we propose in this paper, a checkpointing policy which uses Predicted Execution Time and Mean Time Between Failures of the called web services to make checkpointing decisions. This policy aims at reducing the required number of Call based Checkpoints but at the same time maintains the resilience of web services to faults.

Keywords : *Checkpoints, Web Services, Mean Time Between Failures, Orchestration, Predicted Execution Time.*

I. INTRODUCTION

A service in execution may take service of another service and this may result in nested call of services. This is known as Orchestration of services. In case of such service execution pattern, if a service fails to complete (for any possible reason) then all the services dependent on the failed service are to be re-executed causing a voluminous rework.

Traditionally such a situation is handled (for avoiding re-work) by checkpointing. In our earlier work, [5], we have proposed “Call Based Checkpointing Policy" that saves status of caller services so that in case of failure of callee service, the computation at the former can be resumed at this saved point. However this method is time consuming due to the overhead of maintaining caller status at every service call.

In this work, we propose a method that does not necessitate checkpointing at every call thus reducing instances of checkpointing. The rationality on decision making is based on two factors i.e, Execution Time Prediction and Mean Time Between Failures. A caller service predicts the execution time of the callee S_1, say $PET(S_1)$. This is a possible estimate from the execution history of callee services. Let the Mean Time Between Failures of the callee be $MTBF(S_1)$. If $PET(S_1) < MTBF(S_1)$ then the caller most probably can avail the service from the callee. Hence checkpointing the caller at the service call is not required. This paper details on this concept and advocates its utility for orchestrated services in making them resilient to possible errors.

If S_1 is a composite web service, a call to S_1 might result in a nested call; whether to take a checkpoint or not while calling each of the services involved in the nested call, has to be decided. This decision at each step has to be taken without needing many computations (Execution Time predictions of the services involved in the nested call). Hence we have proposed to use the already available computations, i.e, PET of S_1, and composition operators used to compose involved web services, to take the decisions. We have proved the fact that PET of the composite service S_1 and knowledge of composition operators alone are sufficient to take these decisions.

In section II we present our analysis of work done in this area. In section III we present our basic Call based Checkpointing policy and in section IV we give a detailed description of our Execution Prediction based checkpointing policy. In subsection ‘A’ of this section we give briefly the method for calculating MTBF for web services. In subsection ‘B’ we describe the method of using Euclidean distances to predict Execution time of web services. In subsection ‘C’ we describe how to minimize the number of checkpoints to be taken. We also discuss the role of composition opera-tors and PET of a composite service in making Call based Checkpointing decisions with necessary proofs. Towards the end of this section we have demonstrated the generation of Global checkpoints of an orchestrated web service using the new Execution Prediction Based Checkpointing policy. We conclude by giving a sketch of our future work.

II. RELATED WORK

Few papers [6,7,8] have been published discussing the need and techniques for checkpointing web services. But all these works require the user to specify the exact checkpointing locations. In contrast we proposed a simple and elegant checkpointing policy[5] for orchestrated web services which specifies that whenever a web service calls another web service, the state of calling web service must be saved. But checkpointing web services at all pre specified locations (at all service calls) may lead to overzealous checkpointing that results in degradation of the performance altogether. Hence to improve the performance of composed web services with call based checkpoints, we propose Execution Prediction based

Checkpointing scheme.

The research works presented in [2,3,4] propose methods for predicting runtime of web services. These works advocate the use of predicted execution time for selection of web services to construct composite web service workflows. To the best of our knowledge there is no work which concentrates on using Predicted Execution Time and MTBF for checkpointing of web services.

Zoltan Balogh et:al presented a knowledge based approach[2] for predicting runtime of stateful web services. To predict the execution time of a web service instance, it maintains a knowledge base of possible different past cases for different combinations of input parameters. Given a web service instance, Euclidean distances are used to find out most similar past cases. The runtime for the given web service instance is predicted to be the average output value of the most similar past cases. Estimation of web services runtime is done keeping in view construction of composite web service workflows.

Zhengdong Gao uses Back Propagation Neural Networks to predict the runtime of a given web service [4]. He uses Availability, Network Bandwidth, Response Time, Reliability of the given web service as inputs to the Neural Network which produces predicted execution duration as output. The core of his work is the design and implementation of BP Neural Network which is used to predict performance of services.

In order to predict timing failures, Laranjeiro [3] proposes to use a graph based approach. He analyzes the service code and builds a graph to represent its logical structure. He then gathers time-related performance metrics during runtime. This data is used to predict if a given execution will or will not conclude in due time.

The research presented in [1] intends to provide the concept of MTTF (Mean Time To Failure) of composite web service. It describes the calculation method of MTTF of composite web services based on the workflow composition pattern. The authors use the concept of MTTF of web services to find out reliability of a given composite web service.

III.CALL BASED CHECKPOINTING POLICY

Calling a web service includes several steps and incurs considerable cost and time at run time. When the calls are nested and if there is any kind of failure the entire sequence of calls has to be re-invoked causing considerable delay in response which results in degradation of quality of the service provided. Our Call-based checkpointing policy aims at avoiding expensive re-invocations of web services and hence we propose that Checkpoints must be taken when web services interact with each other. According to our policy: "Save the state of the service requestor after sending the request. Similarly save the state of the service provider after sending the response".

A state of a web service in execution is characterized by the state of its local memory and a history of its activity. If such a state of the web service is saved on a stable storage, then the saved state is called as **a local checkpoint** for the web service. A local checkpoint that is taken most recently is called as the latest local checkpoint for the web service.

An orchestrated web service S_0, is a composition of one or more constituent web services. It has pieces of code that it executes on itself and also calls other web services based on some conditions. If the orchestrated web service is not having any active calls (time t2 in Fig 1), and is executing its own piece of code, then its latest local checkpoint gives the **latest global checkpoint** of the composed application.

When S_0, calls another web service, there may be nested chain of service calls because of which more than one web service can be active(not completed their execution) at a given point of time.(at time t1 in Fig 1 web services S_0,S_1 and S_4 are active). Hence the state of the orchestrated web service is collectively represented by the states of all active web services when a service call is in place. Thus, **Call-based Global checkpoint** for an orchestrated web service which has a service call in process, is defined as the set of the latest local checkpoints of each of the web services that are active during the call.

Motivating example

A customer requests a web service for his loan processing. This loan processing web service receives the request from the customer which consists of information like his name and requested credit amount. This loan processing web service, S_0, invokes two more web services: Loan approver web service(S_1) and accessor web service(S_2). If the amount requested is less than 10,000 S_0 calls the loan accessor web service S_2. This web service, based on some customer records, decides and reports back whether there is high risk in approving loan to the customer. If S_2 reports low risk, S_0 itself approves the loan. If the amount requested is greater than 10,000 or if S_2 reports high risk, the loan approver web service,S_1, is invoked by S_0 to enquire about the customer and report whether to approve the loan to the customer or not. If the amount requested is greater than 1,00,000 S_1 outsources the job to another web service S_3. S_1 calls another loan approver web service S_4 to take second opinion and sends back the reply to S_0. Finally loan processing web service S_0 sends back its reply to the customer. Fig 1 depicts an execution instance of this loan processing web service.■

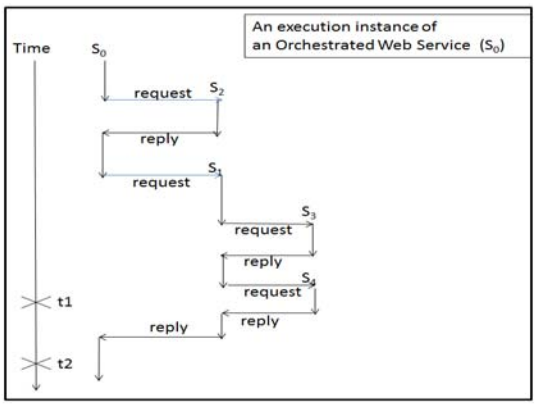

Fig 1: An Execution instance

Let C_0 represent the local checkpoints generated by S_0. Let C_1 represent local checkpoints generated by the service S_1, C_2 represent local checkpoints generated by the service S_2 and so on.S_0 might make several web service calls, while in execution. Let C_0^i represent the checkpoint generated by S_0 for i^{th} service call that it has placed. Let S_0 invoke S_1 in its i^{th} service call. Then C_1^{ij} represents the j^{th} local checkpoint taken by S_1 when it is serving i^{th} call of S_0.

To provide the service, if S_1 makes use of services provided by other web services we have three superscripts in checkpoint numbering. In general, checkpoint C_m^{ijk} indicates: S_0 invokes S_1 as part of its i^{th} service call, S_1 invokes S_m, as part of its j^{th} service call, and this is the k^{th} local checkpoint taken by web service S_m.By applying the Call-based checkpointing policy to the execution instance depicted in Fig 1, we can see that number of local checkpoints generated as part of Call based Global checkpoint is '9' as shown in Table I.

Thus we see from Table I that when the service reply is received back by S_0, the Call-based Global checkpoint reduces to the latest local checkpoint of S_0.Upon failure the application has to be rolled back to latest global checkpoint and all the messages received after the latest global checkpoint have to be replayed from the message logs. Execution of the composed application can thus continue from latest global checkpoint without re-invocation of already finished constituent web services.

IV. EXECUTION PREDICTION BASED CHECKPOINTING OF WEB SERVICES

In order to improve the performance of composed web services with call based checkpoints, we propose

TABLE 1: LOCAL CHECKPOINTS GENERATED AS PART OF CALL -BASED GLOBAL CHECKPOINTS (CBGC)

Status of execution	CBGC
invoke S_2	$\{C_0^1\}$ /* first CBGC */
End S_2	$\{C_0^1\ C_2^{11}\}$ /*End of first CBGC */
invoke S_1	$\{C_0^2\}$ /* second CBGC */
invoke S_3	$\{C_0^2, C_1^{21}\}$
End S_3	$\{C_0^2, C_1^{21}, C_3^{211}\}$
invoke S_4	$\{C_0^2, C_1^{22}\}$
End S_4	$\{C_0^2, C_1^{22}, C_4^{221}\}$
End S_1	$\{C_0^2, C_1^{23}\}$ /*End of second CBGC */
End S_0	$\{C_0^3\}$ /* third CBGC */
Total No of Local checkpoints generated = 9	

Execution Prediction based Checkpointing scheme. For each service call, this scheme decides, considering the PET and MTBF of the called web service, whether a checkpoint has to be taken on making a call to the service. Hence to implement this scheme, a caller should know the MTBF and PET of the callee.

MTBF of a service is an average measure of the time duration for which the service can run without failure. MTBF of a service has to be made public by the service itself by placing the MTBF in its WSDL(Web Services Description Language). This MTBF can then be used by the service requestors to implement the checkpointing policy. When S_0 calls another web service S_1, PET of S_1 is calculated by S_0 using Euclidean distances method as explained in subsection 'B' below.

Checkpointing Rule:

If PET (S_1) < MTBF (S_1), then S_1 will execute within its MTBF and eventually send back the reply to S_0. In such a case S_0 need not take a checkpoint while calling S_1 with anticipation that S_1 might fail.

Else if PET (S_1) ≥ MTBF (S_1) then S_1 might fail before sending a reply back to S_0 and hence S_0 must take a checkpoint before calling S_1.

A. Calculation of MTBF

Let ↘ represent the Failure rate of a web service S and let □ represent the MTBF. Then □ = 1/ ↘.

MTBF of a web service can be obtained by taking inverse of its Failure rate. Failure rate of a web service can be obtained by measuring its number of failures per unit time. (Ex: Failure rate = 5 failures in one hour. MTBF = 1/Failure rate = one hour/5 = 12 minutes).

MTBF of composite services: If a web service is a

composition of other web services, then their MTBFs will affect the MTBF of the composite service. Let θ_i represent the MTBF of a constituent web service S_i where $1 \leq i \leq n$ and n is the maximum number of web services involved in the composition.A web service may be composed of a set of web services using the three primitive operations sequence, parallel and choice as depicted in Table 2. Fig 2 shows different cases of MTBF calculation. Other composition operations can be derived from these three primitive operations.

TABLE II: MTBF CALCULATION

Composition Operation	MTBF Calculation
Sequence: $S = S_1; S_2$	$\lambda = \lambda_1 + \lambda_2$; $\theta = 1/\lambda$
Parallel: $S = S_1 \mid S_2$	$\lambda = \lambda_1 + \lambda_2$; $\theta = 1/\lambda$
Choice: $S = (S_1 + S_2)$	$\lambda = \lambda_1 * P_1 + \lambda_2 * P_2$; $\Theta = 1/\lambda$. S_1 is selected with probability P_1 and S_2 is selected with probability P_2

B. Execution Time Prediction

Execution time Prediction can be done by comparing the current execution instance with similar previous cases. Input parameter values can be compared to find out the similarity between any two execution instances. Let $I = \{i_1, i_2, \ldots i_m\}$ represent an execution instance of a web service with m input parameters. Euclidean distance can be used to find out the similarity[4]. The similarity between any two cases I_1, I_2 is computed using the following formula for finding Euclidean Distance(ED).

$$ED(I_1, I_2) = SQRT\left(\Sigma^{m}_{k=1}(i_{1k} - i_{2k})^2\right)$$

The case/instance which has the smallest Euclidean distance to the current execution instance is considered to be the most similar case.Table III depicts history of execution instances of our loan processing web service, where m=2, $I = \{i_1, i_2\} = \{LoanAmount, Risk\}$. These values are synthesized values and are based on the number of web services that will be invoked for the corresponding execution instance. Let the input parameter values for current execution instance of S_0 be Loan Amount=1,27,000 and Risk = IR.

TABLE III: EXECUTION INSTANCES OF LOAN PROCESSING EXAMPLE

Input Parameters		Parameter used for Prediction
Loan Amount	***Risk***	***Execution time***
9000	Low	2 tu
8000	High	6 tu
25,000	IR	4 tu
1,25,000	IR	7 tu
7000	Low	2 tu
IR=Irrelevant I=Invoked tu=time units		

While calculating Euclidean distances, input parameters having non-numerical values pose a problem. In such a case, map Non-numerical values to numerical values. For example, Input parameter 'Risk' in the example has fixed non-numeric values 'Irrelevant', 'Low', High'. They can be mapped to corresponding numeric values 1,2,3.Similarly input parameters that have range values, like '*LoanAmount*' in our example, where the range in which they fall is more important than the actual value, we have to map each range to a numerical value.

Calculation of Euclidean distances for this example reveals that there is one case similar to current execution instance. Take the average of execution times of similar cases to predict the execution time of the current execution instance, which is 7 time units.

C. Minimizing the number of Checkpoints

Goal of Execution prediction based Checkpointing policy is to minimize the number of local checkpoints that are generated as part of the Call-based Global checkpoint.

When a web service S_0 calls another web service S_1 and if PET (S_1) < MTBF (S_1) then it indicates that the called web service completes its execution within its MTBF. Hence the calling web service will get its reply and there is no need to take a checkpoint in the calling web service. If the called service is a composite service and results in nested calls, **decision has to be taken whether the checkpoints have be taken throughout the path of the nested call or not.**

For example, if a composite service S_1 calls two constituent services S_2 and S_3, then it must be decided whether S_1 must take checkpoints while calling S_2 and S_3. According to the policy, when S_1 calls S_2, S_1 must calculate PET (S_2) and obtain MTBF of S_2 from WSDL file of S_2. If PET(S_2)< MTBF(S_2) then there is no need to checkpoint S_1 while calling S_2. But in order to take this decision, S_1 must calculate the PET (S_2).Similar is the case of calling S_3. These calculations can be avoided if the decision can be taken by using PET (S_1) and MTBF (S_1) alone. We have that PET (S_1) < MTBF (S_1). If it can be proved that PET (S_2) < PET (S_1) and MTBF (S1)< MTBF (S2) then it can be deduced that:
PET(S2) < PET(S1) <MTBF (S1) < MTBF(S2)…(1)
Hence PET (S_2) < MTBF (S_2).Whether PET (S_2) is less than MTBF (S_2) or not can thus be found without calculating PET (S_2).

PROOF: PET of a composite service = It's Local Computation time + Time taken to place Service calls + PET of constituent services.

PET (S_1) = Local Computation time of (S_1) + Time taken to call S_2 and S_3 + PET (S_2) + PET (S_3).
Hence PET (S_1) > PET (S_2) and PET (S_1) > PET (S_3).
OR PET (S_2) < PET (S_1) and PET (S_3) < PET (S_1).
Hence first half of equation (1) is proved. We have to consider MTBF calculations to prove second half. MTBF for a composite service is calculated taking into account MTBF of its constituent services also, as shown

in Table II. Fig 2 explains MTBF calculation.

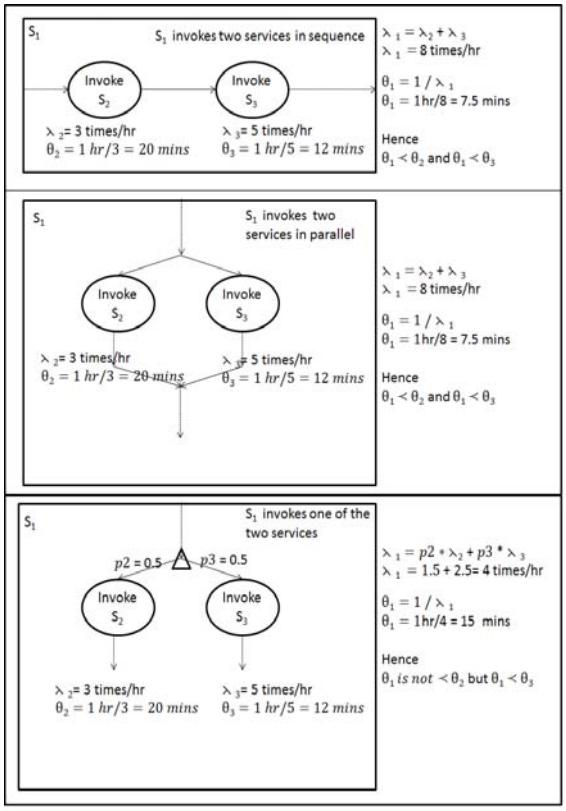

Fig 2: MTBF Calculation

Hence it cannot be generalized that θ_1 is lesser than θ_2 and θ_3. i.e, it cannot be generalized that MTBF (S_1) <MTBF (S_2) OR MTBF (S_1) < MTBF (S_3). Hence If PET (S_1) < MTBF (S_1) and if the called web service S_1 is a composite service resulting in a nested call, then equation (1) holds good if composition operation is either 'sequence' or 'parallel composition' but does not hold good in case of 'choice' operation.

■

Hence using above observation, Execution time prediction based checkpointing policy can be stated as: When a web service S_0 calls another web service S_1 then, S_0 must:

i) Obtain the MTBF (S_1) being called from its WSDL. If S_1 is a composite service, then the composite service should calculate its MTBF using the formulae briefed in previous section and make it's MTBF available in its WSDL.

ii) Use Euclidean distances to find out similar cases and take their average execution time as Predicted Execution Time, PET (S_1).

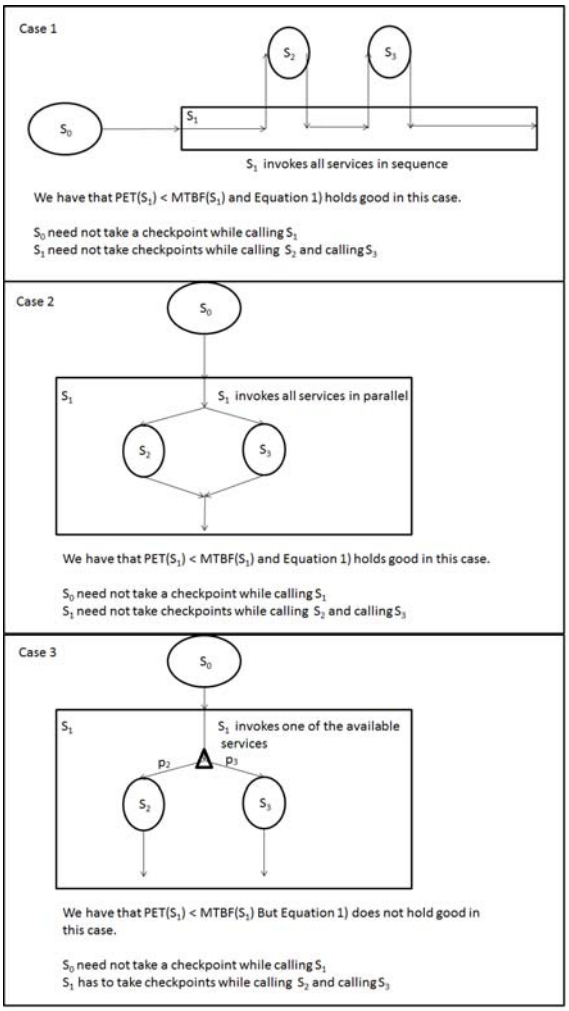

Fig 3: Execution prediction based Checkpointing Policy for basic composition operations

iii) If PET (S_1) ≥ MTBF (S_1) then S_0 has to take a checkpoint while calling S_1.
else if PET (S_1) < MTBF (S_1) then
a) If S_1 is not a composite service then S_0 need not take a checkpoint while calling S_1.
b) If S_1 is a composite service: policy is explained using the Fig 3 for basic cases of composition and using Fig 4 for combination of composition operations.

For any combination of sequence and parallel composition operations, equation (1) holds good because in any case, λ_1 is the sum of λs of constituent services. The same is explained using Fig 4. But in case of choice composition operation, equation (1) does not hold good. Hence any sub composition involving 'choice' operation leaves us with no choice other than taking checkpoints for that sub composition.

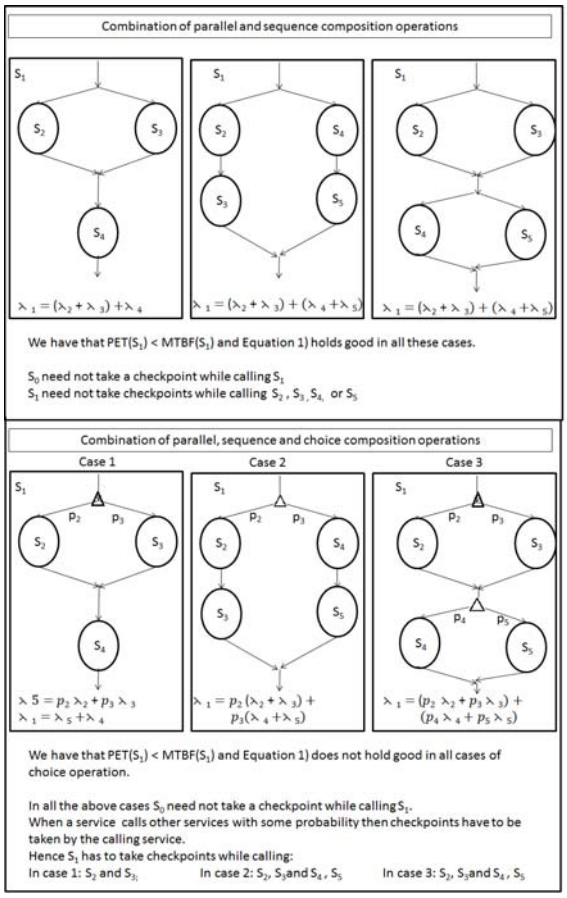

Fig 4: Execution prediction based Checkpointing Policy for combination of composition operations

But when Execution prediction based checkpointing policy is used, the number of checkpoints generated as part of Call based Global checkpoint reduces to 3 as shown in Table IV. This is because when S_0 calls S_1, S_1 calls S_3 and S_4 in sequence resulting in a nested call. Before S_0 calls S_1, S_0 calculates that PET (S_1) = 4tu and MTBF (S_1) = 5tu. Hence it is found that PET (S_1) < MTBF (S_1) and S_0 need not take a checkpoint before calling S_1.

S_1 calls S_3 and S_4 in sequence and since composition operation is 'sequence', we can see that equation (1) holds good and there is no need to calculate PET (S_3) and PET (S_4). Hence it can be deduced that PET (S_3) < MTBF (S_3) and PET (S_4) < MTBF (S_4) and S_1 need not take checkpoints while calling them. Also S_3 and S_4 need not take checkpoints after sending reply back to S_1 since their predicted execution time is less than their MTBF and they will not fail.

Thus we can see that total number of Local checkpoints generated is greatly reduced by implementing Execution time Prediction based Checkpointing.

TABLE IV: DEMONSTRATION OF EXECUTION PREDICTION BASED CHECKPOINTING

Current node visited	PET and MTBF Calculation by the caller	Action Taken
invoke S_2	PET (S_2) 2tu and MTBF (S_2) = 1tu. PET (S_2) > MTBF (S_2)	$\{C_0^1\}$ generated by S_0 /* first CBGC */
End S_2		$\{C_0^1, C_2^{11}\}$/* End of first CBGC */
invoke S_1	PET (S_1) = 4tu and MTBF (S_1) = 5tu. PET (S_1) < MTBF (S_1)	$\{C_0^2\}$ not generated by S_0
invoke S_3	PET (S_3) < MTBF (S_3) can be deduced from PET (S_1) < MTBF (S_1)	C_1^{21} not generated by S_1
End S_3	PET (S_3) < MTBF (S_3) can be deduced from PET (S_1) < MTBF (S_1)	C_3^{211} not generated by S_3
invoke S_4	PET (S_4) < MTBF (S_4) can be deduced from PET (S_1) < MTBF (S_1)	C_1^{22} not generated by S_1
End S_4	PET (S_4) < MTBF (S_4) can be deduced from PET (S_1) < MTBF (S_1)	C_4^{221} not generated by S_4
End S_1	PET (S_1) < MTBF (S_1)	$\{C_1^{23}\}$ not generated by S_1
End S_0		$\{C_0^3\}$ /* third CBGC */
Total No of Local checkpoints generated = 3		

V. CONCLUSION AND FUTURE WORK

In this paper we have proposed to use PET and MTBF of web services to decide whether checkpoints have to be taken at service calling locations. We have used Euclidean distances method to find out similar cases for the given web service execution instance and use them to estimate the execution time of the instance. If this estimated execution time of the called web service is less than its MTBF then there is no need to checkpoint its calling web service.

Recovery of applications based on checkpointing policies has been well studied in database and distributed computing fields. Due to lack of space we are not describing the implementation of recovery policy for web services. We intend to cover it in our future work.

When web services are orchestrated each service call results in creation of a new service instance and when the called service sends a reply back that service instance is destroyed. But if web services are choreographed service calls may be directed to already existing service instances. Also, when the called service sends a message to the caller, it might be in the middle

of an operation, expecting some communication from the caller. Our previous checkpointing policy does not suffice to such a scenario. Hence we propose to develop a new checkpointing policy for Choreographed services as part of our ongoing research work.

REFERENCES

[1] Tao Hu, Minyi Guo, Song Guo, Hirokazu Ozaki, Long Zheng, Kaori Ota, Mianxiong Dong.MTTF of Composite Web Services.International Symposium on Parallel and Distributed Processing with Applications.

[2] Zoltan Balogh, Emil Gatial, Michal Laclavik,Martin Maliska, and Ladislav Hluchy. Knowledge-Based Runtime Prediction of Stateful Web Services for Optimal Workflow Construction. LNCS 3911, pp. Springer-Verlag Berlin Heidelberg 2006

[3] Nuno Laranjeiro, Marco Vieira, and Henrique Madeira.Predicting Tim-ing Failures in Web Services.ISBN: 978-3-642-04204-1. Springer-Verlag Berlin, Heidelberg l'2009

[4] Zhengdong Gao, Gengfeng Wu.Combining QoS-based Service Selection with Performance Prediction. Proceedings of the 2005 IEEE International Conference on e-Business Engineering (ICEBEŠ05) 2005 IEEE

[5] A.Vani Vathsala.Global Checkpointing of Orchestrated Web Services. Submitted to RAIT 2012, ISM Dhanbad. To appear in the proceedings of RAIT.

[6] Soumaya Marzouk, Afef Jmal MaLalej, and Mohamed Jmaiel. Aspect-Oriented Checkpointing Approach of Composed Web Services. F. Daniel and F.M. Facca (Eds.): ICWE 2010 Workshops, LNCS 6385. Springer-Verlag Berlin Heidelberg 2010.

[7] Susan D. Urban, Le Gao, Rajiv Shrestha, and Andrew Courter. Achieving Recovery in Service Composition with Assurance Points and Integration ″ Rules: OTM 2010, Part I, LNCS 6426, pp. 428U437, 2010. l' Springer-Verlag Berlin Heidelberg 2010

[8] Sagnika Sen, Haluk Demirkan and Michael Goul.Towards a Verifiable Checkpointing Scheme for Agent-based Interorganizational Workflow System Docking Station Standards.

[9] Jens Happe.Predicting Mean Service Execution Times of Software Com-ponents Based on Markov Models. p 53-70, Proceedings of Lecture Notes in Computer Science 3712 Springer 2005.