

Economic Strategies for Composing Complex Web Services

D. Alvarez¹, R. Torres², S. Cruz¹, and J. Mendoza¹

¹ Department of Information Technology, Pontifical Catholic University of Chile, Santiago, Chile

² Department of Computer Science, University of Concepción, Concepción, Chile

ABSTRACT

Web services are gaining importance and are often used as a standard approach for integrating diverse miscellaneous services often distributed over the network. Hence, it necessitates attaining high levels of reliability and availability without being affected by any service, network or infrastructure failures. To accomplish this, we propose an efficient mechanism which will narrow the plausibility of fault detection by dynamic web service selection among similar services. The Quality-of-service (QoS) can have an important effect on web service composition service reliability. In this paper, we perceive the rationale for the failure of the web services and resolve it by considering the QoS attributes like response time, execution time, etc. The proposed modeling approach indicates a possible reduction in the failure probability as well as a significant improvement in the execution duration of the complex web services composition. This has been proved through the results achieved with an experimental setup.

Keywords: Failure Probability, Quality-of-service (QoS), Web services, Response time, Execution duration.

I. INTRODUCTION

Web services composition are used to manage business processes. It is dynamic in nature. Complex web service composition can be framed by identifying and integrating diverse web services distributed over the network. Web Services is a service offered to facilitate communication between the Service provider and Service consumer over the web. Web services can be defined and published by the service consumer and it has been discovered and invoked for completing the task by the service consumer. Two or More web services combined to complete a task is called composite web services or web services composition. Reliability achievement is a big issue in service oriented computing; it is not like traditional standalone machine.

Web services are spread across the web which is remotely developed and hosted making it difficult to locate. It may also happen that the service becomes unavailable without prior notice. Also its performance can dynamically change according to various reasons like change in workload of servers, unstable network connectivity or communication links etc.

Faults may occur in service oriented systems due to various reasons. There can be fault in its framework like power fluctuations or any flaws made during the designing and implantation phase such as any software bug. The defect can also be introduced manually by the administrators. There can also be an issue that the service requested is not available or there is a very high response time.

In this paper, we present a methodology which will detect the fault in the web service composition and then optimize the results obtained by minimizing the faults. For example, if a client requests for a set of services from the server, for instance, a user wants to buy a book online. He will make a request and try to contact one of the various servers available (like amazon.com, ebay.com, flipkart.com). There can be a failure in the web service invocation, for instance, the server may not be available at that moment due to various reasons like maintenance or server is down or the server might be taking a lot of time to reply to the request or the particular requested web service is not offered by the server. Hence in such cases to minimize this tendency of failure we can redirect the request to another server.

Service availability indicates whether a service is available or not ie. whether a response to the service invocation has been successfully received. The Response Time is the duration from the time the request has been sent till the time the response has been received.

The remnant part of the paper is constructed as follows: Section 2 provides a synopsis of the works related to this field. Section 3 introduces the proposed model. The composition approach for fault detection, optimization, and reduction of failure probability is discussed in the proposed approach. Finally the conclusion and future direction are discussed.

II. RELATED WORKS

In this section, first will rundown and go through some of the related works and investigations on QoS management and fault tolerance mechanism prior to examining the affiliation amidst our research work and standardization efforts for the web services composition technique.

There are some previous researches on reliability and fault tolerance and the evaluation of the existing web service composition. Researchers [1] scrutinize the predicament of employing the optimal and best strategy for fault tolerance to build and establish a reliable service oriented system by mapping the user requirements and prerequisites. It also examines the fault tolerance policy selection for the business processes which are semantically homologous.

Various research experiments and investigations have been carried out on the subject of QoS aims selection of web services and composition. A QoS-aware middleware platform [4] proposed and it reports the issues of web services discovery from the repository for composition to maximizing the satisfaction of users by considering the QoS properties.

QoS management has been extensively studied and analyzed in the horizon of web service composition system [7], [5], [6]. The primary concern of this research is on the subsequent matters: QoS specification to acknowledge characterization of functional performance and QoS attributes. [1], [2], [3], [4] discusses the QoS properties providing the non-functional characteristics for web service selection when more than one service is present with similar and comparable properties.

In [3] various QoS issues on web services (Availability, Response time, Throughput, Security properties like authentication, confidentiality, data integrity and non-repudiation) are discussed from the point of view of the service provider and service user. It states that the service users and providers need to be able to engage in QoS negotiation. The providers should monitor the load they receive from the users and should check whether they meet the service level agreement (SLA). The user must also check the quality-of-service they obtain.

In [8] the fault tolerant infrastructure for web services is studied which can be used for interoperability among applications and for redistributing on the web with high reliability and availability demands. It must provide a crystal-clear fault tolerance for the clients. The principle objective of this is to assure transparency in fault tolerance for end users utilizing the active replication technique.

In contemporary, there has been a growing interest in processes which can be implemented by invoking multiple web services. The client can cite their preferences and constraints and based on this the selection of services are dynamic in nature for service composition and enforced by discovering the suitable services available at the runtime [2] [4]. Negotiation techniques have also been used to attain and come up with a practical and reasonable solution.

A hybrid model is proposed [9] to enhance the performance of complex composite web services. With the help of this the rate of failure can be computed and also the reliability can be determined. However this does not work towards achieving the minimal error.

The works mentioned above do not aim towards minimizing the probability of failure in web services which has been taken into consideration in this paper.

III. PROPOSED MODEL

In this section of paper, we begin by first describing the architecture and its various entities. The different QoS attributes for the web services are discussed followed by the required algorithms.

1. Architecture

The service provider is accountable for the providing the web services to the end users. The service consumer is the client who is the end user or web application making the request for the web services and utilizing these services created by the service provider. All the communication taking place will be tabulated in the service register which will be later used for analyzing purpose.

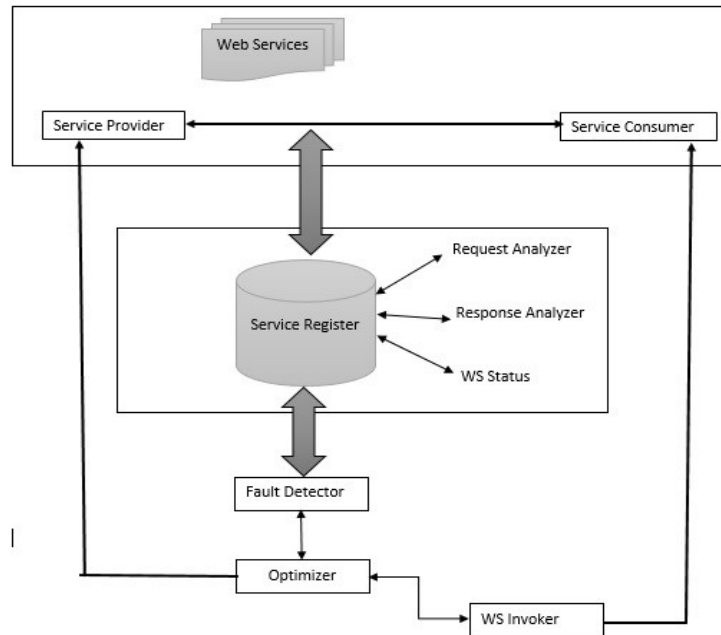


Figure 1 Architecture Diagram

The request analyzer will be calculating the amount of time taken to transfer the request from the consumer to the provider and to process this request. The response analyzer will be calculating the time required to send the requested web service from service provider to the service consumer. The WS Status will mention the status of the service provider concerning the web service request.

The request will be sent to all the service providers one at a time, and this data will be stored in the service register which will later forward its data to the fault detector. It will be detecting the fault in the web service invocation by various service providers which will be used to calculate the failure probability. This result will then be examined by the optimizer which will try to minimize this failure probability previously obtained by the fault detector. This can be done by redirecting the request to another nearby web service provider which will provide the similar homogeneous web services.

The WS Invoker is used for invoking the web service from the web service provider as instructed by the optimizer and will then send this service to the web service consumer.

2. Quality Criteria for Web Services

In the present scenario, there are multiple candidates available for web services with similar functionalities. In such cases, QoS attributes impart some non-functional features for dynamic web service discovery. Based on the previous research methods and experimental results [2], [3], [4] we exemplify the following QoS attributes for the web services:

1. **Service Availability.** The service availability determines the possibility of the end user to access the web service. The value of service availability is calculated as the total duration for which service is available during the last t seconds.
2. **Service Execution Duration.** The service execution duration is computed as the expected duration in seconds required by the service consumer to send the request and receive its results.
3. **Service Response time.** The Response time measures the duration receiving the response from the server.
4. **Service Failure probability.** Failure occurs if either at the server side the request was not successfully executed or when the end user does not receive its corresponding response. Hence the service failure probability is computed as the ratio of the number of services which were not successfully executed to the total number of web services.

3. Algorithm

Algorithm 1 Sending request to the service provider.

Input: SP: service provider.

WS: web service.

Output: request sent.

```
1 int SPNumber = |SP|;
2 int WSNumber = |WS|;
3 int ser=0;
4 for (j=0; j<SPNumber; j++) do
5 ser++;
6 for (i=0; i<WSNumber; i++) do
7 sendurl (ser, SPNumber[j], WSNumber[i]);
8 end for
9 end for
```

In Algorithm 1, the web service consumer is sending the request for the web service to the web service provider one at a time. The ser is the service index for the service provider which is being requested by the service consumer.

Algorithm 2. Finding Response Time

Input: SP: service provider.

WS: web service.

Output: RT: response time

```
1 Open Connection
2 int code: getResponseCode;
3 string response: getResponseMessage;
4 long StartTime: Current time of system
5 long Response Time: Current time – Start Time;
6 long x1: ResponseTime%1000;
7 long x2: ResponseTime /1000;
8 return ResponseTime;
```

In Algorithm 2, the Consumer bind with Provider to invoke the functionalities of web services. The response i.e. Response Code, Response Message from the network is obtained. The response time is calculated as the duration of the connection being established till it sends the corresponding requested web service back to the service consumer.

Algorithm 3. Fault Detection

Input: RT: response time

Output: F: failure

```
1 int lim: Time limit of RT
2 if (RT>lim) then
3 code=408;
4 response= Timeout;
5 flag= true;
6 else if (code=404) then
7 flag= true;
8 end
9 if (flag==true) then F=true;
10 else F=false;
11 return F;
```

In Algorithm 3, the failure can occur if the response time exceeds time limit or if the web service is not available. The 408 is the request timeout error. It is an http status code which implies that the request the service consumer sent to the service provider took longer than the web service provider was prepared to wait.

The 404 is the Not Found Error Http status code. This error is shown when the requested web service by the service consumer is not available at the service provider.

Algorithm 4. Minimizing failure probability**Input:** SP: service provider.

WS: web service.

Output: request sent.

```

1  int SPNumber = |SP|;
2  int WSNumber = |WS|;
3  for (i=0; i<WSNumber; i++) do
4    for (j=0; j<SPNumber; j++) do
5      curService=WSi;
6      curServer=SPj;
7      Send request of curService to curServer.
8      if(code==200) then
9        break;
10     end
11  end

```

In Algorithm 4, the request for the web service made by the service consumer is sent to the service provider. If due to any reason like the service was unavailable on that service provider or its response time is exceeding the time limit etc. another request will be made to send the same service but this time to the next service provider. This will ensure that the availability of web services increases and hence there will be significant reduction in failure probability.

IV. EXPERIMENTAL SETUP

The experimental setup has various computers or nodes acting as the server. The client will send a request for a web service. This request will be henceforth forwarded to the servers nearby one at a time.

The setup is first laid as show in Figure 2. There are two servers and three clients. Each client here is sending a request for a web service to the server which is the service provider and in reply it is receiving a response either success or failure from the web server. Here it can be noticed that there are many service invocation requests resulting in failure.

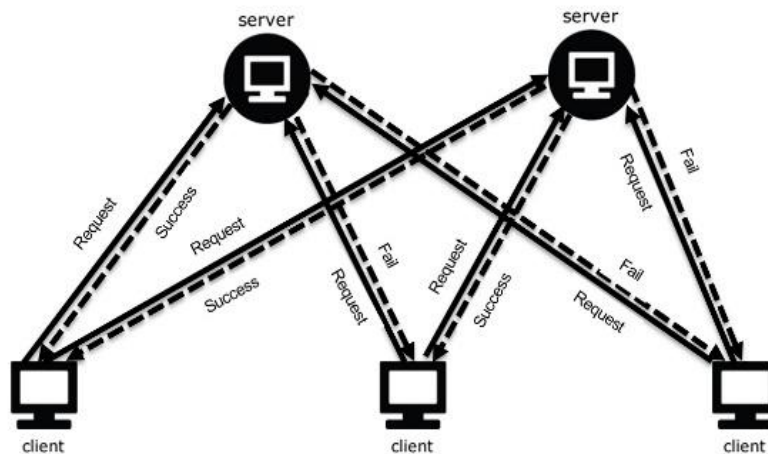


Figure 2. Client server Setup

There can be various causes of failure now based on the Quality-of-Service (QoS) attributes as discussed in section 3 of this paper.

1. When the web service is unavailable on the server

In this scenario, it may happen that the particular web service requested by the service consumer is not available with the service provider. This can be due to various reasons like the server is down, network connectivity issues, server maintenance is taking place or it may be permanently shut down.

Service availability is the possibility that a service invocation will be executed successfully with a response. The value of service availability can be determined from the former data of service invocations as the ratio of the number of successful execution of web service against the total number of request made. Hence this unavailability of the requested web service on the service provider will eventually lead to an error message received by service consumer hence acting as the failure.

2. When the web service has exceeded its response time

Here the web service provider is available unlike in the first scenario but is taking time way too long to respond to the request made by the service consumer than it was supposed to. Hence due to this increase in response time, it will increase the execution duration of the web composition and the service consumer will have to wait for long which is unfavorable.

Response time is the total amount of time it takes to respond to a request for web service. It is the duration from the point of time the request has been sent till the time its response has been received.

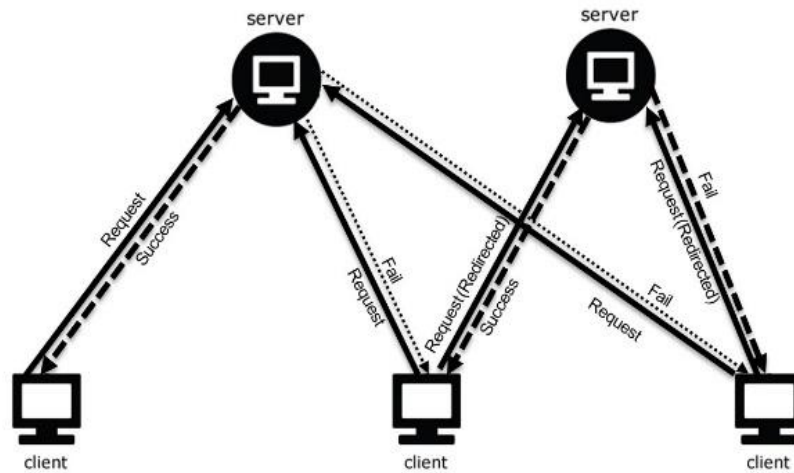


Figure 3. Improved Client server Setup

Hence in such cases to reduce the risk of failure the client's request for that particular web service will be forwarded to other nearby servers providing similar web service as per request. This can be seen in Figure 3 which is an improved version of the Figure 2. As it can be seen in the previous figure the second client is first receiving a failure in response from first server and then receiving a success from the second server. So instead of doing so much we can directly redirect the service request to the third server which will eventually result in success, hence reducing response time as well.

This will thus lead to the successful invocation of web services and is efficient to implement for the execution of web services. Hence it increases the success probability for invocation of web services by reducing the failure probability.

V. RESULT AND DISCUSSION

Our proposed model and algorithm have been tested on a wide range of the web service request which was randomly generated. Experiments have been carried out to examine and compare the solutions which were obtained after applying the proposed methodology.

The following table given below states some of the response status codes based on the request made by the service consumer to the service provider where 200 represents the successful execution of web service, and others represent a failure.

Table 1 Response status codes and message

Response Code	Response Message
200	OK
400	Bad Request
404	Not Found
408	Request Timeout

Table 2 (a) & (b), presents the experimental results for only one request made for five web services with two service providers available. It returns the response time along with the status for each request indicating the outcome as success or failure. The web service index identifies the web service, and the server hit name identifies the service provider to which the request for the web service has been sent.

Table 2 Experimental results of web service for one request
(a) Before implementing failure reduction

WEB SERVICE INDEX	STATUS	RESPONSE TIME	OUTCOME	SERVER HIT NAME
WS1	408	2.179sec	FAIL	Ser1
WS2	200	0.454sec	SUCCESS	Ser1
WS3	404	1.199sec	FAIL	Ser1
WS4	403	1.157sec	FAIL	Ser1
WS5	200	0.604sec	SUCCESS	Ser1

(b) After implementing failure reduction

WEB SERVICE INDEX	STATUS	RESPONSE TIME	OUTCOME	SERVER HIT NAME
WS1	408	1.658sec	FAIL	Ser2
WS2	200	0.454sec	SUCCESS	Ser1
WS3	200	0.568sec	SUCCESS	Ser2
WS4	403	0.166sec	FAIL	Ser2
WS5	200	0.889sec	SUCCESS	Ser1

Table 2 (a), shows the experimental result before implementation of fault reduction. Each service here is sent to a single service provider whereas, in Table 2 (b), the web services have been redirected to another service provider on the occurrence of a failure.

For instance, it can be observed that in Table 2 (b), the services which had previously failed as shown in Table 2 (a) have now been redirected to server 2. It can also be noted that the number of web services successfully executing has increased.

Further, the experiment was extended to a different number of requests and then their aggregated values for response time was used to calculate mean response time. The number of failure and success outcome for all requests is being noted, and hence the failure probability for each web service is being calculated.

Table 3 Experimental results of web services for 100 request
(a) Before implementing failure reduction

WEB SERVICE INDEX	MEAN RESPONSE TIME	NUMBER OF FAIL OUTCOMES	NUMBER OF SUCCESS OUTCOMES	FAILURE PROBABILITY
WS1	1.456sec	42	58	0.42
WS2	0.961sec	24	76	0.24
WS3	0.872sec	21	79	0.21
WS4	2.256sec	55	45	0.55
WS5	0.567sec	19	81	0.19

(b) After implementing failure reduction

WEB SERVICE INDEX	MEAN RESPONSE TIME	NUMBER OF FAIL OUTCOMES	NUMBER OF SUCCESS OUTCOMES	FAILURE PROBABILITY
WS1	1.032sec	26	74	0.26
WS2	0.211sec	18	2	0.18
WS3	0.755sec	15	85	0.15
WS4	2.116sec	37	53	0.37
WS5	0.232sec	11	89	0.11

Table 3 (a) & (b), presents aggregated data for 100 requests for five web services before and after applying the proposed methodology respectively. It can be observed that there is an increase in the number of successful execution of the web service. Hence there is a significant reduction in failure probability of web services by increasing the availability of services in the event of failure.

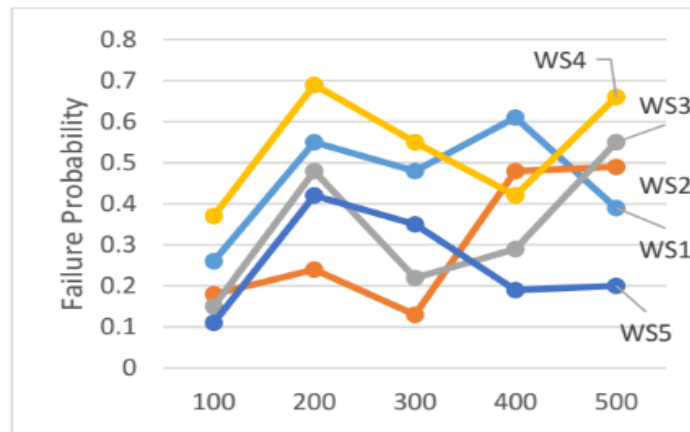
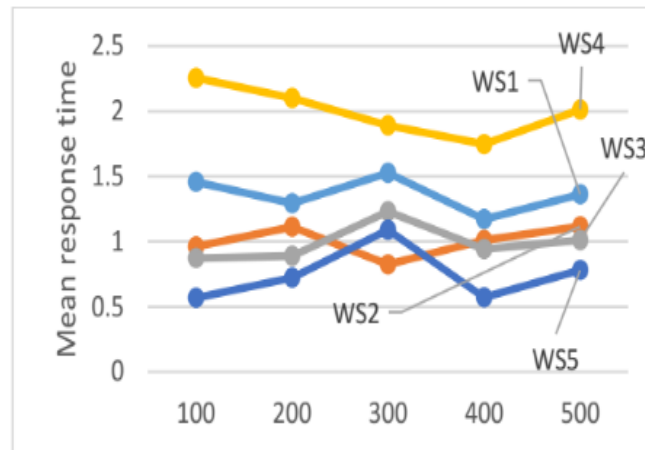
Figure 4 Failure Probability versus number of requests.*(a) Before implementing failure reduction.**(b) After implementing failure reduction.*

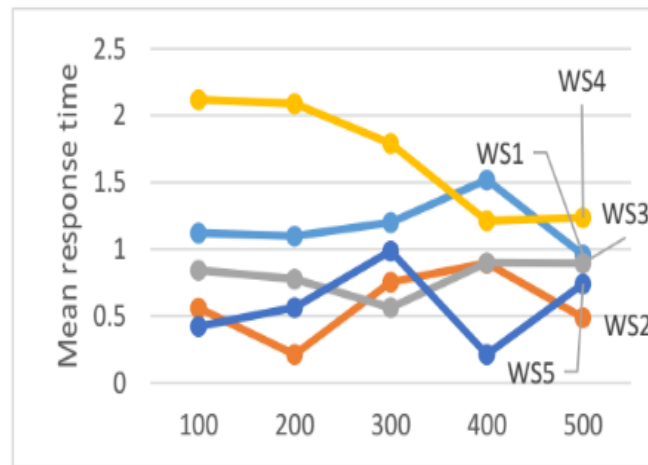
Figure 4 (a), (b) & Figure 4 (a), (b) presents the result of experiments conducted for 100, 200, 300, 400, 500 requests for five web services. Fig. 4 (a), (b) compare the failure probability and Fig. 5 (a), (b) compares the response time before and after applying the proposed methodology.

Hence it can be clearly observed from the graph that the failure probability is getting reduced with better response time. Thus the web services are being executed efficiently with optimum execution duration.

Figure 5 Response Time versus number of requests



(a) Before implementing failure reduction.



(b) After implementing failure reduction.

VI. CONCLUSION AND FUTURE WORK

We have given a streamlined approach for the structure of an optimized approach for the composition of web services by dynamically selecting the web services which allows specifying constraint on quality prerequisites. We have investigated the problem of the increasing risk of failure of web services by reducing the failure probability as well as significant improvement in the execution duration of web services, or the composition is observed with the help of the tabulations and the graphical analysis done during the experiment. Thus a composition approach has been implemented for fault detection, optimization, and reduction of failure probability.

In future, more exhaustive and extensive research will be done on other QoS values for efficient execution of web services. This work can be extended to more complex web services dealing with sensitive data like in medical diagnosis, bank transaction, etc

VII. REFERENCES

1. Zibin Zheng, Member, and Michael R. Lyu, "Selecting an Optimal Fault-Tolerance Strategy for Reliable Service-Oriented Systems with Local and Global Constraints," IEEE Transactions on Computers vol. 64, no. 1, pp. 219-231, Jan. 2015
2. D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," IEEE Trans. Softw. Eng., vol. 33, no. 6, pp. 369-384, Jun. 2007.
3. D. A. Menasce, "QoS issues in web services," IEEE Internet Computer., vol. 6, no. 6, pp. 72-75, Nov. /Dec. 2002.
4. L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware middleware for web services composition," IEEE Trans. Softw. Eng., vol. 30, no. 5, pp. 311-327, May 2004.

5. K. Nahrstedt, D. Xu, D. Wichadakul, and B. Li, "QoS-Aware Middleware for Heterogeneous Environments," IEEE Comm. Magazine, vol. 39, no. 11, pp. 2-10, 2001.
6. M. Gillmann, G. Weikum, and W. Wonner, "Workflow Management with Service Quality Guarantees," Proc. ACM SIGMOD Int'l Conf. Management of Data, pp. 228-239, June 2002.
7. C. Aurrecochea, A.T. Campbell, and L. Hauw, "A Survey of QoS Architectures," Multimedia Systems, vol. 6, no. 3, pp. 138-151, 1998.
8. Giuliana Teixeira Santos, Lau Cheuk Lung, and Carlos Montez discuss, "the fault tolerant infrastructure for web services," Proceedings of the 2005 Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05) IEEE.
9. C Jaya Prakash, P Maruthurkarasi, R Balaji Ganesh, V Maheswari, "Hybrid Reliability Model to Enhance the Efficiency of Composite Web Services," 2013 IEEE International Conference on Emerging Trends in Computing, Communication and Nanotechnology (ICECCN 2013)