Journal of Computer Science 8 (3): 382-388, 2012
ISSN 1549-3636

Aspect Oriented Decision Making Model for Byzantine Agreement

[1]Murugan, S. and [2]V. Ramachandran
[1]Faculty of Computer Science and Engineering,
Sathyabama University, Jeppiaar Nagar, Rajiv Gandhi Salai,
Chennai-600 119 Tamilnadu, India
[2]Department of Information Science and Technology,
Anna University, College of Engineering, Guindy,
Chennai-600 025 Tamilnadu, India

Abstract: Problem statement: The main aim of this research study is to develop an enhanced strategy for decision making whether to commit or rollback a request to a Web service in the presence of Byzantine faults using aspects. The proposed study extends the Lamport's algorithm for Byzantine agreement to have an effective decision while handling the service request. When the service is initiated based on the request, its execution behaviour is being monitored before, after and at the time of execution and being handled with aspect concerns to provide the corresponding responses as input to the Lamport's Byzantine agreement algorithm. The decision on the client requests is based on the outcome of the aspect advices before, after and around the execution of the services. **Approach:** This approach identifies the inception of the process which starts to exhibit abnormal behaviour before, after or during the execution of the service and instigates that helps in resolving the faulty service and identifies the root cause of the problem to rectify it. Aspect oriented advices do not require any external invocation as it executes with the service and hence no additional overhead involved in processing the service request. **Results:** In the existing methodology only the around advice decision is considered for reaching an agreement in the presence of Byzantine faults. An enhancement in the decision making process is proposed by including the state of the services: before, after and around advices of the aspects. **Conclusion:** The obtained experimental results based on the proposed methodology depict that the performance measure, Round Trip Time is slightly increased when compared with the existing Lamport's algorithm for Byzantine Agreement and this performance overhead is not a major concern as the proposed approach produces an enhanced decision by considering aspect concerns and also determines the origin of the fault. The change in execution behaviour of Byzantine algorithm when aspects are introduced is compared with the implementation of the algorithm without aspects in various distributed environments.

Key words: Agreement algorithm, various distributed environments, computing environment, Aspect Oriented Programming (AOP), Round Trip Time (RTT), Model View Controller Framework (MVC)

INTRODUCTION

In today's complex distributed business system, reliability of individual element is of major concern which is to be maintained at higher level to keep the other components in the system intact and hence to make the entire system available. Identifying and elimination of faults in the complex business system is a major challenging task. Sometimes the identified faults behave abnormally by exhibiting Byzantine behaviour, which may go undetected often, as it continues to study and produces results which are illegitimate that causes business loss, customer dissatisfaction, loss of reputation and various other factors. Fault handling is the major issue in the distributed computing environment. The fault detection and elimination is a straight forward process when the service stops its execution or throws exception or when the hardware begins to malfunction. But when the system executes with faults without any notification or

Corresponding Author: Murugan, S., Faculty of Computer Science and Engineering Sathyabama University Jeppiaar Nagar, Rajiv Gandhi Salai Chennai, 600 119 Tamilnadu, India

not revealing any symptom and produces inappropriate results, it becomes more complicated to identify and fix those faults. This kind of uncharacteristic behaviour of faults is referred to as Byzantine fault or Byzantine behaviour which is more common in this age of Internet, where systems are infused with faults that are very difficult to identify, locate and eliminate.

The Byzantine Generals problem (Lamport *et al*., 1982) is built around an imaginary General in defense who makes a decision to attack or retreat and must communicate the decision to his lieutenants. The general and some of the lieutenants may be traitors. Traitors cannot be relied for proper communication of orders; worse yet, they may actively alter messages in an attempt to subvert the process. The generals are collectively known as processes. The general who initiates the order is the source process. The orders sent to the other processes are messages. The general and lieutenants those send faulty messages are traitorous and termed as faulty processes. Loyal general and loyal lieutenants are correct processes. The order to retreat or attack is a message with a single bit of information: 1 or 0. Lamport *et al*. (1982) proposed an algorithm to eliminate the Byzantine fault in which an agreement is arrived based on the messages that are exchanged among the processes.

In order to identify and handle the faults effectively, the proposed model allows the system to maintain the state and values of the parameters before, after and during execution of the services that are invoked for accomplishing a task. The current object oriented programming paradigms are capable of handling exceptions effectively but explicit instructions are to be written to handle abnormalities, which is a tedious process. The method of capturing the state of the service for fault handling is referred to as cross-cutting concerns. Fault tolerance mechanisms that are implemented using software are capable of handling both hardware and software failures. Software based fault tolerant techniques have been developed using reflection and meta-programming. Aspect Oriented Programming (AOP) is an extension of meta-programming that offers a provision for handling cross-cutting concerns that can be plugged into any of the widely adopted programming languages. AOP improves the software quality by reducing code tangling and separating the concerns. By using AOP, fault handling code is separated from the actual implementation of the business logic and the aspect's advices do not require any explicit invocation as they get triggered along with the service and does the appropriate process of collecting the required information. As the aspects are modularized, the fault handling mechanism using aspects doesn't require any modification even when the application modules are either extended or altered and this nature makes aspects more flexible and extensible.

Many aspect oriented application programming interfaces are available as open source for different programming paradigms. AspectJ, 2011 is used for implementation of the proposed decision making model for Byzantine agreement. In AOP, Joinpoints are well defined check points in the flow of the application, which may be (i) method call or return, (ii) bean operations (set and get) and (iii) exception handler entry point. A collection of joinpoints is termed as pointcuts. Advices are codes that will execute on some conditions like before, after or around the joinpoint. Aspect is like a class which includes pointcuts and advices for implementing the cross-cutting concerns. Concern refers to a specific purpose i.e., a portion of code for which the aspect is introduced. Weaver combines the classes and aspects for constructing the actual application.

Ji-De and Ying (2010) have developed an exception softening methodology to handle the exception faults effectively in AspectJ environment. They have analyzed and summarized several exception fault types of AspectJ and illustrated the way with appropriate examples to analyze the impact of exception faults on program control flow. Sevilla *et al*. (2007) envisaged the role of Aspect Oriented Programming in distributed component services with respect to distribution, fault tolerance and load balancing. Usually the code for providing QoS parameters (both functional and non-functional) is merged with the business logic and hence it is harder to develop, maintain and reuse the code. In the proposed aspects based model for Byzantine agreement, the Byzantine behaviour identification module is completely decoupled from the Web service, which is meant for its intended task (Domokos and Majzik, 2005) have modelled the fault tolerant structures using aspects and this framework is extended for automatic construction of an analysis model, which is a dependability model that is used to determine the non-functional properties of the system. In order to improve the reliability and availability of distributed object oriented systems, Herrero *et al*. (2001) have introduced object replication mechanisms and presented a replication model, JReplica, which is a Java fault tolerance language based on Aspect Oriented Programming. This replication model separates the specification of the replication code from the functional behaviour of objects by providing a high degree of transparency. JReplica provides facilities to the programmers to introduce new behaviors for specifying different fault tolerant requirements. To enhance the

reliability of the Web services, it is not only necessary to handle the crash faults but also efforts should be taken to monitor and to handle the Byzantine faults due to the untrusted communication environment in which they operate. Zhao (2007) had developed a Byzantine Fault Tolerance framework for Web services, which operates on top of the standard SOAP messaging framework with minimum changes in the Web applications. The Byzantine Fault Tolerance framework is implemented as a pluggable module and hence this model also supports inclusion of new fault tolerance requirements.

Byzantine agreement using aspect oriented programming: The proposed model is an extension to the existing Lamport's algorithm for elimination of Byzantine behaviour and this approach uses inherent aspects for tolerating Byzantine type of faults in the Web services application environment. The proposed aspects oriented model for elimination of Byzantine behaviour is shown in Fig. 1, in which there are 'n' Web services coordinating with each other, against which the Aspect Advices (a(α, β, γ)) are defined and are labeled as α (before advice), β (around advice) and γ (after advice). These advices are weaved together and provided as input to Lamport's algorithm for eliminating Byzantine faults and the final decision taken is based on the response $\{\alpha_d, \beta_d, \gamma_d\}$. In the response set obtained using Lamport's algorithm, each entity refers to the "commit" or "rollback" decision based on the collection of 'before advice (α)', 'after advice (β)' and 'around advice (γ)' values separately as provided by the Web services in association with aspects those have been weaved inherently. The entity which occurs many times in the response set will lead to the final decision, i.e., the output of the Lamport's algorithm, which is having a set of three values, the maximum occurrence of the value either 'commit' or 'rollback' is considered as the final decision. The response due to an advice which is different from other advices is the one that behaves abnormally and the service, for which this advice belongs to, is the faulty service.

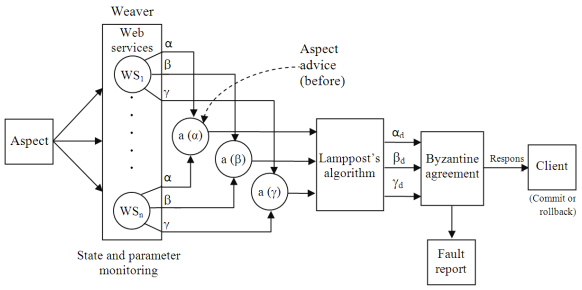

Fig. 1: Aspect oriented model for byzantine agreement

The instance at which the service exhibits the faulty value or behaves abnormally i.e., before, after and around the execution of the service is determined in a straight forward manner.

An on-line debit transaction from a customer account in a bank is considered as a case study to analyze and to test the aspects oriented decision making model for Byzantine agreement. It is assumed that four services namely primary service (receives and processes the request from the client), authentication service, transaction service and balance verification service are involved for processing the transaction request. When all the services agreed mutually, the amount is debited from the customer account. When any one of the services provides a negative response, the transaction is declined. When there is a malicious service which exhibits Byzantine behaviour then the genuine transactions are also declined.

In order to apply Lamport's algorithm for Byzantine Agreement in distributed services environment, the minimum requirement is 3f+1 (f ≥1) processes (Lamport *et al*., 1982) and hence in the case study, a total of four processes (one primary service + three services) are considered. In this Web application, the client provides the initial set of input for transaction like authentication (customer and pin verification), selection of account type, transaction type (only withdrawal is considered) and amount to be withdrawn. Once the request for the transaction is initiated, the "before advice" of the primary service gets triggered and also the "before advice" of the coordinating services. The messages that are exchanged during "before advice" among the services (including primary service) have been fetched at this stage and weaved together. The Lamport's Byzantine Agreement Algorithm is invoked using the set of "before advice" responses to reach possible agreement. The decision taken in this phase is labeled as "α_d". The next process starts with verifying the account balance and at this point of time the "around advice" of all the involved services get activated and the messages that are collected are weaved together. The set of "around advice" responses is fed into the Byzantine Agreement Algorithm and the result obtained is identified as "β_d". When all the processes have completed the request, post commit transactions are performed with which the "after advice" messages are collected and weaved to pass them to Byzantine Agreement Algorithm and "after advice" result is available in "γ_d".

In this case study, each response α_d, β_d, or γ_d represents a value "commit" or "rollback" and the final decision whether to debit or not is arrived based on the value which occurs more times in the set. The

checkpoint at which the system starts violating or exhibiting abnormal behaviour is identified and appropriate action is taken against the traitorous service by reporting it to the concerned authority by the way of generating fault reports.

Implementation: A set of interfaces has been defined while implementing the proposed aspect oriented decision making model for Byzantine agreement. The interfaces used for implementing the Byzantine algorithm are:

Service interface	- For maintaining the list of participant services
NodeValue interface	- For retrieving current message value that is passed from one service to another service
MapRepository interface	- For defining the services hierarchy, for identifying the path in which the message communication takes place, for specifying the number of messages exchanged between the participant services
Broadcast interface	- For maintaining the number of repetitions and the message transfer details
AspectValue interface	- For storing the advices
message value	- Vector based parameters are used for before, after and around advices

The above interfaces except AspectValue interface are implemented to establish the Lamport's algorithm as a Web service in the distributed environments such as RMI, Servlets and AXIS2 based SOAP communication environment. The Web service designed for Lamport's algorithm is further extended and deployed in the public cloud environment. Aspects are introduced to enhance the decision arrived by the Lamport's algorithm for Byzantine agreement by the way of injecting before, after and around advices to the developed Web service.

To elucidate aspects with Web service for Byzantine agreement, "AuthenticationService" for verifying the user credentials is considered. The "AuthenticationService" declares a method "verifyUser ()" for validating the user credential and its signature is given below:

```
Public boolean verifyUser(String username, String password)
```

The aspect and the "AuthenticationService" are weaved together through "pointcuts". The pointcut processMessage () inherently monitors the state and parameters of the method "verifyUser()" as defined in the service "AuthenticationService". The method signature of "verifyUser ()" specifies two parameters both of type String, hence pointcut is also defined with the same number and type of parameters which is given below:

```
public pointcut processMessage(String uname, String pwd): execution(public
boolean verifyUser(String, String)) and args (uname, pwd);
```

The advices before, after and around are associated with the pointcut. The following code fragment shows the around advice of the pointcut "processMessage()". The state of the parameters are logged and depends on the service the logging variable is different. In case of AuthenticationService the parameter "username" is logged. The response generated by the "AuthenticationService" is captured by the around advice and passed to Lamport's algorithm. The message from all the other services of around advice is transferred in this manner. A similar kind of approach is adopted for other aspects with before and after advices. The around advice of the service "AuthenticationService" is given below:

```
boolean around(String uname, String pwd):processMessage(uname, pwd) {
    // write the String variable uname (user name) into the log
    // obtain the response created by the service
    // pass the message to the Lamport's Algorithm for further processing
 }
```

The values received from the aspect advices are processed separately to arrive at an agreement and for reporting the faults. The aspect code is shown in Fig. 2, which consists of two pointcuts for processing the messages and for identifying the faulty node. Three advices before, after and around are defined for message processing pointcut and the after advice is defined for fault node identification. The pointcut processMessage() is defined and it is associated with the method "BFTSolution ()", which is the implementation of the Byzantine Agreement Algorithm.

```
Public aspect ByzantineAOP  {
 Public pointcut processMessages (): execution (public boolean BFTSolution ());
 Public pointcut after Decision (): execution (public Vector identify faulty node ());
 Object around (): processMessages ()
 {
          AspectValues.around = getValues ();
        Boolean around = LamportBAA (AspectValues.around);
        return around;
 }
 ……..
 ……..
 Boolean [] after (): after Decision () {
      Vector  serviceids = new Vector (Service.getNoServices);
      Enumeration enumService = serviceids.elements ();
      Boolean [] faultyServices = new Boolean [serviceids.size];
     While (enumService. has More Elements ())
           FaultyServices [i] = (boolean) fault Detection (getServices().elementAt (i));
     return faultyServices;
 }
 Public static Vector getValues() {
    Vector inter Values=new Vector (Service.getServices());
    For (int i=0;i<Service.getServices();i++) {
          InterValues[i].addElement (NodeValue.getSourceValue ());
   Return interValues;
}
Public static Boolean fault Detection (int serviceid) {
    Boolean isfaulty=getFaultyByzantineNode(serviced, before, around, after);
    Return isFaulty;
 }
}
```

Fig. 2: Pointcut and advices for byzantine agreement algorithm

For the advices before, after and around, the respective methods are invoked for taking the decision at that instance and the results are retained for taking the final decision. Based on the values collected for identifying and reporting the faulty node, a pointcut called "afterDecision ()" is used. The functionality of this aspect is to identify the service which behaves abnormally.

The anomalies are analyzed for abnormal behavior with respect to any external force, which is responsible for the fault or due to human intervention or due to any other factors. The information gathered helps in eliminating the Byzantine behaviour of the faulty service and thereby extended to prevent other services from being affected by similar kind of behaviour.

Performance analysis: The performance measures namely Round Trip Time (RTT) and throughput has been computed for execution of the proposed Byzantine Agreement algorithm with advices (before, around and after). Round trip time is the time that elapses between the initiation and obtaining the response of the service by the client. Throughput is estimated as the total size of the data transferred divided by the duration of the test run. In the proposed study, RTT gradually decreases in the order of the advices (before, around and after) associated with the Lamport's algorithm and throughput increases in the order after, around and before advices, which is shown in Fig. 3.

The average round trip time and throughput for Lamport's algorithm for Byzantine Agreement is 0.081 ms and 12.53 kbps respectively. Introducing aspects into Lamport's algorithm doesn't show any major impact in terms of execution.

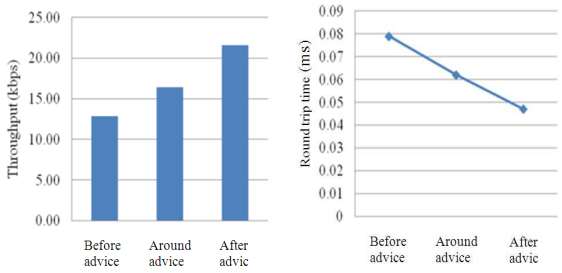

Fig. 3: Throughput and RTT of aspect advices for byzantine agreement algorithm

RTT is decreased on subsequent invocation of Byzantine algorithm for aspect advices before, around and after. This analysis reveals that introducing aspect advices for tolerating Byzantine behavior doesn't show any increase in RTT or decrease in throughput. Since there is no external invocation and hence no additional overhead in processing these advices are required.

A comparison study has been carried out on the performance of Byzantine Agreement algorithm with injection of around advice aspects and without involving any aspect in various distributed environment paradigms. Figure 4 shows the execution behavior of Byzantine Agreement Algorithm in Remote Method Invocation (RMI) with and without aspects. The average round trip time (log based) is moderately low if the algorithm is implemented with aspects when compared with the implementation without aspects. The testing is done by varying the number of processes and number of messages.

The performance measure, round trip time for the Byzantine agreement algorithm using Web servlets by varying the number of messages is shown in Fig. 5. RMI exhibits a better execution time when compared with servlets.

While the Byzantine Agreement algorithm is implemented as a Web service using AXIS2 Apache the Fig. 6 shows that the average round trip time is very less when compared with implementations using RMI and Web servlets. Involving aspects in Web services reduces the execution time considerably than any other environment.

The execution times of the Byzantine Agreement algorithm with and without aspects while implementing the same in the Model View Controller Framework (MVC) using Struts and in the public Cloud environment using Google App Engine are shown in Fig. 7 and 8 respectively. These two environments exhibit a higher execution time when compared with RMI, Web servlets and Web services.

Fig. 4: Round Trip Time for Byzantine Agreement with and without Aspects in Remote Method Invocation (RMI) Environment

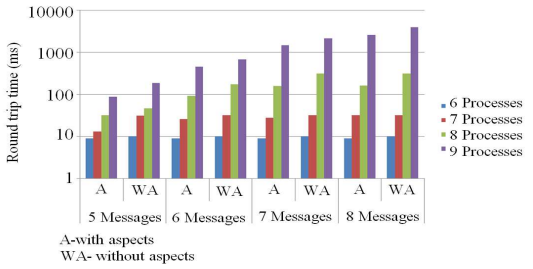

Fig. 5: Log Based Round Trip Time for Byzantine Agreement with and without Aspects in Web Servlets Environment

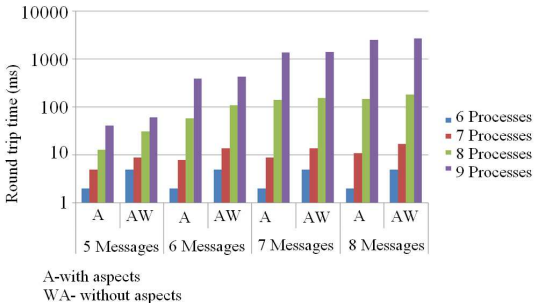

Fig. 6: Round Trip Time for Byzantine Agreement with and without Aspects in Web Services Environment

Inducing aspects into Model View Controller Framework and in public cloud environment consumes higher execution time than the Lamport's algorithm for Byzantine agreement is implemented without aspects.

Table 1 shows the average throughput for the Byzantine agreement algorithm implemented using various distributed environment paradigms. Among the tested distributed environments, it is observed that the Web service implementation provides a better throughput which is followed by Java Remote Method Invocation when aspects are introduced for achieving Byzantine agreement whereas cloud platform generates low throughput.

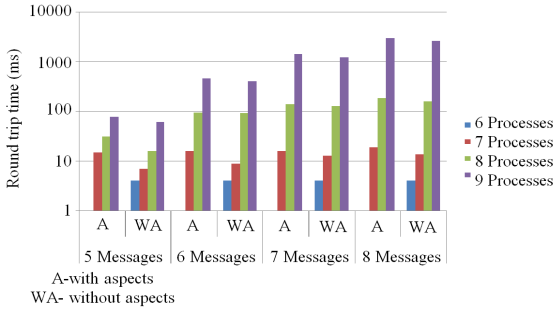

Fig. 7: Round Trip Time for Byzantine Agreement with and without Aspects in Model View Controller Framework (using Struts)

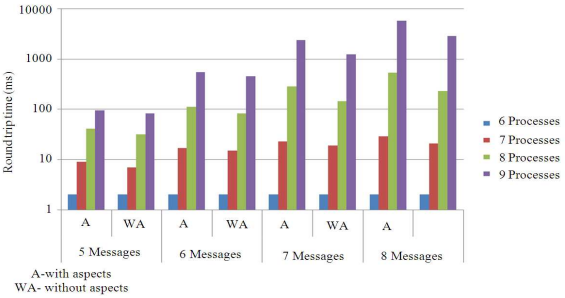

Fig. 8: Round Trip Time for Byzantine Agreement with and without Aspects in Public Cloud Environment

Table 1: QoS measure-throughput of the proposed byzantine fault tolerance model in distributed environment paradigms

	Number of services (n_s)							
	Throughput with aspects (kbps)				Throughput thou aspects (sec)			
Platform	n_s = 6	n_s = 7	n_s = 8	n_s = 9	n_s = 6	n_s = 7	n_s = 8	n_s = 9
RMI	2.0313	0.3963	0.0301	0.0034	0.3533	0.1195	0.0261	0.0032
Web container	0.4514	0.1641	0.0366	0.0035	0.4063	0.1280	0.0193	0.0023
Web service	2.0313	0.4924	0.0453	0.0038	0.8125	0.3009	0.0341	0.0035
MVC	4.0625	0.2462	0.0356	0.0032	1.0156	0.3779	0.0409	0.0037
Cloud platform	2.0313	0.2083	0.0166	0.0018	2.0313	0.2621	0.0330	0.0035

The case is reverse when the agreement algorithm is implemented without aspects i.e., the throughput of Struts based implementation is high which is followed by the cloud environment. Even though cloud environment yields a low throughput, the aspects can still be introduced because of the features supported by the cloud platform are not available in the other distributed paradigms.

CONCLUSION

An enhanced model for identifying Byzantine behaviour of Web services using aspects is proposed in this study. The proposed approach is an extension to the existing Lamport's algorithm for Byzantine Agreement, in which the state and parameters before, after and at the time of execution of the services are considered for decision making process. The proposed model enhances the decision making in the presence of Byzantine faults and also determines the checkpoint where the fault transpires. The origin of the fault location is identified and appropriate action is taken against the faulty service by the way of generating the fault reports and notifying to the authorities. By introducing aspects into the Byzantine agreement algorithm, performance issues do not arise, as the advices are not invoked by any other external resources, as they are being triggered along with the service itself. The methodology adopts an enhanced decision making approach for tolerating Byzantine faults using aspects by considering the advices before, after and around the execution of the service and therefore an accurate decision is arrived rather than the decision taken considering only the around advice as is the case in the existing Lamport's algorithm.

REFERENCES

Domokos, P. and I. Majzik, 2005. Design and Analysis of Fault Tolerant Architectures by Model Weaving. Proceedings of the 9th IEEE International Symposium on High-Assurance Systems Engineering, Oct. 12-14, IEEE Xplore Press, Germany, pp: 15-24. DOI: 10.1109/HASE.2005.8

Herrero, J.L., F. Sanchez, O. Sanchez and M. Toro, 2001. Fault Tolerance AOP Approach. The Pennsylvania State University.

Ji-De, Z. and Y. Ying, 2010. Analysis of exception fault types based on AspectJ. Proceedings of the International Conference on Computer Application and System Modeling, Oct. 22-24, IEEE Xplore Press, Taiyuan, pp: 287-289. IEEE. DOI: 10.1109/ICCASM.2010.5619408

Lamport, L., R. Shostak and M. Pease, 1982. The byzantine generals problem. ACM Trans. Programm. Languages Syst., 4: 382-401. DOI: 10.1145/357172.357176

Sevilla, D., J.M. Garcia and A. Gomez, 2007. Aspect-oriented programing techniques to support distribution, fault tolerance and load balancing in the CORBA-LC component model. Proceedings of the 6th IEEE International Symposium on Network Computing and Applications, Jul. 12-14, IEEE Xplore Press, Cambridge MA, pp: 195-204. DOI: 10.1109/NCA.2007.8

Zhao, W., 2007. BFT-WS: A byzantine fault tolerance framework for web services. Proceedings of the 11th International IEEE EDOC Conference Workshop, Oct. 15-16, IEEE Xplore Press, Annapolis, Maryland, pp: 89-96. DOI: 10.1109/EDOCW.2007.6