Volume 3, Issue 3, March 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

# Core Services in Web Service Atomic Transaction Avoids Byzantine Faults

**P. Jamuna***
Department of CSE &
Pondicherry University, India.

**V. Kavitha Chandrakanth**
Assistant Professor, Department of CSE &
Pondicherry University, India.

***Abstract- In this paper we have described the core services provided by the coordinator over in distributed transaction and also with our proposed Efficient Enhanced three phase commit protocol we can render the issue of reliable coordinator problem in distributed transaction. Our proposed protocol and core services can overcome the Byzantine fault i.e. The system may get failure over in an arbitrary way it may be a hardware failure, software failure or failure due to some third party intrusion. In this paper we have used the Byzantine Agreement only in coordinator side thus we have reduced the usage of Byzantine Agreement so it is less expensive. Thus with our proposed technique we can achieve trust, dependency, security and also less communication overhead.***

***Keywords- Distributed transaction, Byzantine fault, Byzantine Agreement, Web Service Atomic Transaction, Coordination***

## I. Introduction

Web services are the software components so as to communicate with pervasive, standards-based Web technologies includes HTTP and XML-based messaging. Web services which comprise a set of standards that makes machine to machine interaction. They are designed to be access by additional applications and differ in complexity from easy operations, such as inspecting a banking account balance through online, to multifaceted processes running CRM (customer relationship management) or enterprise resource planning (ERP) systems. While they are based on open principles such as HTTP and XML-based protocols including WSDL and SOAP, Web services are programming language, hardware and operating system independent. It means the applications have written in diverse programming languages and running on diverse platforms can switch over the data over Internet using the Web services technology.

Web services include three core technologies are WSDL, SOAP, and UDDI. Before constructing the Web service, developers are needed to create its description in the form of WSDL document which describes the service's site on the Web and also service provides. Information about the service is then entered in a UDDI registry that allows Web service consumers to look for and place the services they need. Based on information in the UDDI registry, the Web services developer uses directives in the WSDL to create SOAP [8] messages for data exchange with the service over HTTP. Extra about these core technologies in Web services are detailed below.

When multiple providers, requesters and intermediaries are participating in a Web service distributed transaction, it might be essential to coordinate them. There are two different types of mechanisms for coordinating Web services are Web services choreography and Web services orchestration. The coordinator is like a third party will keep track of transactional information over the internet and he should be trustworthy entity always and should be able to tolerate the byzantine fault i.e. The system may get failure over in an arbitrary way it may be a hardware failure, software failure or failure due to some third party intrusion. The coordinator provides some core services in order to overcome those Byzantine faults. In order to achieve high availability of data we are replicating the server at the coordinator side and shared a Byzantine agreement with each other to avoid Byzantine fault [9]. In the Byzantine agreement phase we can run algorithm like Practical byzantine fault tolerance algorithm [1] or with some algorithm [7] [6] [5] [4] [3] which can render the Byzantine fault. Primarily the coordinator provides some four core services [2] are Activation service, Registration service, Completion and Coordination services.

## II. CORE SERVICES

### *A. Activation Services*

Activation Service creates a new co-ordination object for each new transaction where as transaction id is a part of co-ordination object and transaction id should be unique. For security reason, transaction id is derived from a random number so that no one can be predicted. Each non faulty Activation replica server should have same transaction id. However one should not trust the primary Activation replica to create a transaction id because Universal Unique Identification (UUID) is random, hence Back up replica is not in a position to verify that random UUID . Hence the use of Random Transaction id is a good defense-in-depth strategy.

 

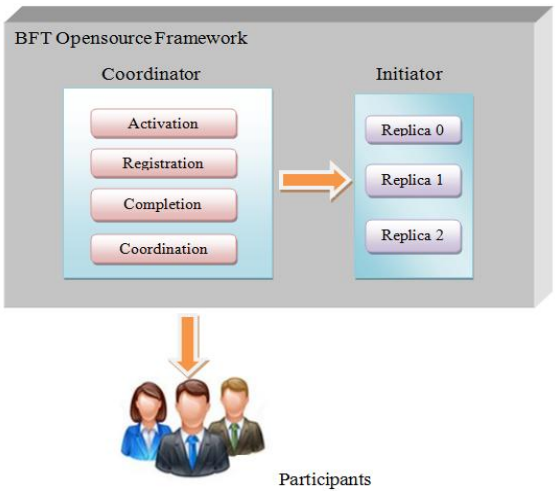


**Figure1. System Architecture**

***Operations involve in Activation Services***

1. Initially the non faulty client sends a request message to all Initiator replicas where as the request message is in the form of < CREQ, T, O, C> τC where as O denotes operations to be executed by Initiator. T denotes timestamp which is monotically increasing. C denotes the client id and τC denotes the client's security token.
2. Initiator replica Accepts those request message only if it is properly protected by security token and also the time stamp should not greater than or equal to T from the same client.
3. On receiving the request message from client starts the distributed transaction and multicast the Activation request message to all Activation replicas. The Activation request message is in the form of <ACTIVATION, C, T, K, V> τK where as V denotes current view number. T denotes the timestamp. C denotes the client id. K denotes the Initiator replica id and τK denote the security token.
4. The Activation replica logs and accepts the request message if and only if messages are properly protected by security token and request message with the timestamp should not greater than or equal to T from the Initiator replica.
5. Activation replica receives those request message with matching T and C from f+1 distinct Initiator replica. This ensures that the Activation Request message come from at least one non faulty Initiator replica.
6. Activation replica which can generate the UUID and multicast it to all other replicas. It allows the inter replica communication. Whereas the UUID exchange message is in the form of <UUID, ID, V, D, I, $UUID_I$> τI where v denotes the current view number. ID denotes the tuple <T, C> which identify the activation instance. D represents the Activation request digest. $UUID_{I\ denotes}$ replica's I's proposed UUID. I denotes replica id and τI is the security token.
7. Activation replica accepts UUID messages if and only if the messages are properly protected by security token and d should matches with the digest of Activation request. Initially the Activation primary replica generate its own UUID message and it has accepted by other 2f backup replicas which finally combines 2f+1 UUID by performing bit-wise XOR , which is a provable secure combination method. After initiates the Byzantine Agreement phase to avoid byzantine fault.
8. At the end of the Byzantine Agreement phase, replicas can derive a transaction id and create the co-ordination object for transaction id and sends the activation response message to all Initiator replicas. The Activation request response message has the form of <ACTIVATION-REPLY, C, T, c, V, I> τI where T denotes the timestamp. C denotes the client id. V denotes the view number. C denotes the transaction context. I denote the replica id and τI denotes the security token.
9. Finally Initiator replica accept those Activation Response message if and only if
   The messages are properly protected by security token and also tuple C and T should match with those Activation Request message. Whereas Initiator Replica accept the Activation response if it matches the Activation response from f+1 distinct Activation replica which ensure it has come from at least non faulty replica.

***B. Registration service***

Registration service which allows the participant to register with the coordinator replicas and it has no inter-replica communication.

***Operations involve in Registration Services***

1. Initially the participants accept those request message from Initiator replica when it has collected matching request message from f+1 replica in order to ensure that the message has come from non faulty Initiator replica.

 

2. Participants multicast their request message to all coordinator replica and wait to receive Acknowledgement from 2f+1 coordinator replica because at most f coordinator replica are faulty at least f+1 non faulty coordinator replicas will accept a registration.
3. If the Participant registers successfully and complete the execution of Initiator request and it will multicast the response message to Initiator replicas otherwise it multicast the exception message.
4. If the Initiator receives exception message it aborts the transaction. Initiator also registers with the Registration service provided by the coordinator as like Participants.

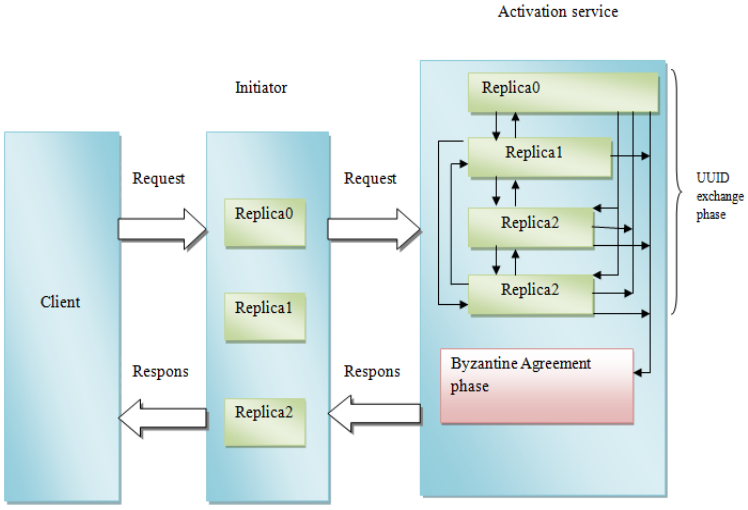


**Figure2. Activation service**

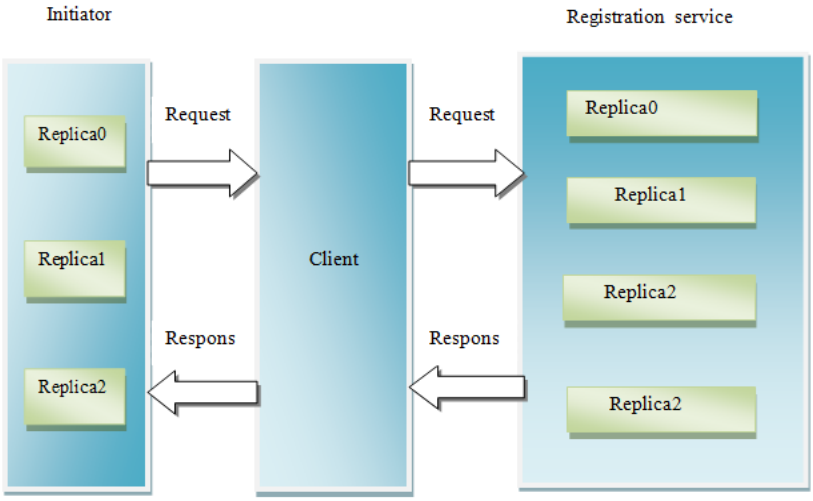


**Figure3. Registration service**

### *C. Completion and Coordination service*

The Completion service is to activate commits operation only. Coordination service provides the interoperability among multiple web service. Certainly if any service is fault means it will redirect to another replica service and it contain all information about the transaction and services. The Completion and Coordination service which includes Registration Update phase its purpose is to check whether all non faulty participants has registered successfully or not.

***Operations involve in Completion and Coordination Services***

1. When the Initiator replica has successfully completes all operations within transaction before timeout it will multicast the commit request to all coordinator replica otherwise it multicast the rollback request message. Initially the coordinator replica accept commit request when it has received matching request message from f+1 initiator replicas in order to ensure that it has come from at least one non faulty replica.
2. Accepting those commit request message, the coordinator replica starts the Registration update phase by multicast to all coordinator replicas. The Registration Update phase includes the registration records of participants. Its purpose

 

is to check whether all non faulty participants have registered successfully or not. The non faulty coordinator replicas have a record of its registration. The Registration Update message has the form <REGISTRATION UPDATE, RS, I, TID> τI where TID denotes the transaction id. RS denotes the set of Registration records. I denotes replica's id and τI denotes the security token. Where each record in set RS is registration record Rf = (TID, f) τf where f represent participant id.

3. When the coordinator replica has collected Registration Update message from 2f other back up replicas it checks if any of the registration record is missed if it so it finds and add those missing registration record if it is present in RS set.
4. Next the coordinator replica starts the some protocols after that make run Byzantine agreement so that non faulty coordinator replica agrees on same set of transaction and same set of participants. However Byzantine agreement phase is still needed to ensure that all the non faulty replicas agree same transaction outcomes.
5. In Byzantine Agreement, the primary coordinator replica must include its decision certificate c as evidence for the decision on all transaction outcomes where c may contains set of records for each participant. Records for participant f contains Registration record and Vote records. The registration records Rf = (TID, f) τf and Vote record Vf = (TID, vote) τf. In this TID is used in both records so that faulty primary coordinator replica won't reuse this registration and vote obsolete.
6. The backup coordinator replica trust the primary coordinator replica and begins the view change only if the registration record in c are either identical.
7. At the end of the Byzantine Agreement phase decision outcome is sent to participants.

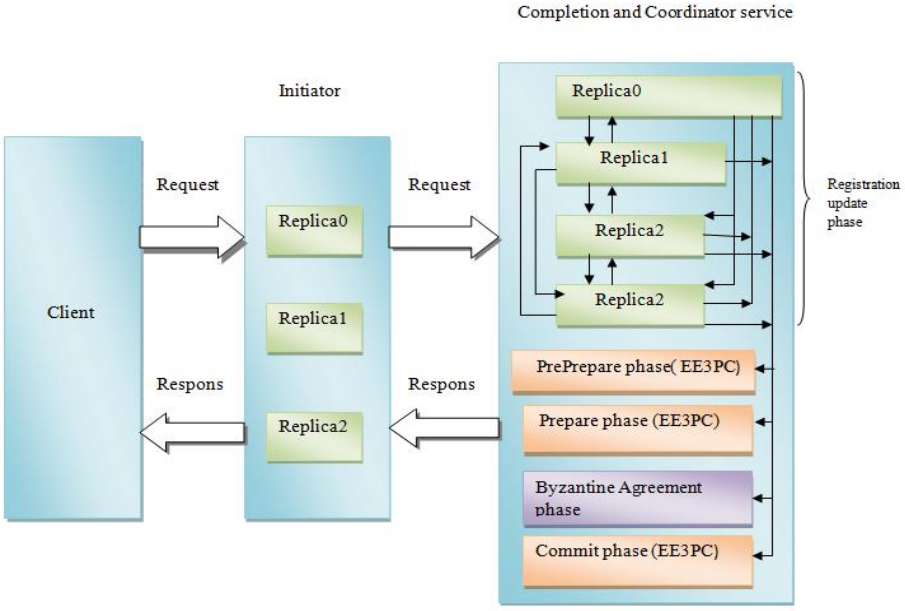


**Figure4. Completion and Coordination service**

## III. Efficient Enhanced Three Phase Commit Protocol (EE3PC)

The Efficient Enhanced Three Phase Commit Protocol [2] is a modified version of quorum based 3PC which overcome some of the drawback such as blocking problem, communication overhead, inconsistency over communication network .EE3PC recovery phase includes, at each transaction invocation the site try to elect new primary coordinator if the previous site is suspect to be byzantine fault. In order to achieve high availability of data we are maintaining 2 extra counters are

P_Elect:- It calculates the number of election take place in that site and the variable value is updated when a new coordinator is selected. Initialize its value as 0.

P_Attempt:-. It calculates the Election number in previous election. Initialize its value as 0

***Operations involved in EE3PC***

Select a new coordinator (R) when the previous is subject to byzantine fault. When the R gather the value of P_Elect and P_Attempt from all sites with that value R calculates the total elected value and total attempt value as T_Elect and T_Attempt. R sets P_ Elect→T_Elect+1 and report the value T_Elect to all participants(P).Participant(P) sets P_ Elect→T_Elect+1 and report that value to coordinator R.

Coordinator (R) gathers those values of various states from all sites P and makes the decision whether to commit or abort the transaction. Upon the decision the coordinator (R) sets P_Attempt→P_Elect. After receiving those preliminary decisions of pre-commit or Pre-abort from R, P sets P_Attempt→P_Elect and sends Acknowledgement to R. After receiving ACK, R will broadcast its decision and process the transaction.

 

**Decision Table**

| GATHERED VALUES | Decision |
|---|---|
| ∋ABORTED | ABORT |
| ∋COMMITTED | COMMIT |
| T_Attempt_Committable (true) | PRE-COMMIT |
| T_Attempt_Committable (False) | PRE-ABORT |
| Otherwise | BLOCK |

Where T_Attempt_commitable is true only if all the members are in non-final state and also P_Attempt should be equal to T_Attempt.

## IV. Conclusion

With our proposed Efficient Enhanced Three Phase Commit Protocol we can overcome the issue of trustworthy coordinator and also performs the job of electing a new coordinator when the existing coordinator is subject to be Byzantine fault. The core services provided by coordinator further overcome the Byzantine fault. In this paper we have also reduced the usage of byzantine agreement. Thus with our proposed technique we can achieve trust, dependency, security and also less communication overhead.

## References

[1] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance and Proactive Recovery,"ACM Trans. Computer Systems,vol. 20, no. 4, pp. 398-461, Nov. 2002.

[2] Honglei Zhang, Hua Chai, Wenbing Zhao, P.M.Melliar-Smith, L.E. Moser, "Trustworthy Coordination of Web Services Atomic Transaction", IEEE Transactions on Volume: 23 , Issue: 8

[3] G.S. Veronese, M. Correia, A.B. Bessani, and L.C. Lung, "Spin One's Wheels: Byzantine FaultTolerancewith aSpinning Primary,"Proc. IEEE 28th Int'l Symp. Reliable Distributed Systems, pp. 135-144, Sept. 2009.

[4] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong "Zyzzyva: Speculative Byzantine Fault Tolerance," Proc. 21st ACM Symp. Operating Systems Principles,pp. 45-58, Oct. 2007.

[5] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shri, "HQ Replication: A Hybrid Quorum Protocol for Byzantine Fault Tolerance," Proc. Seventh Symp. Operating Systems Design and Implementations,pp. 177-190, Nov. 2006.

[6] Y. Amir, B.A. Coan, J. Kirsch, and J. Lane, "Byzantine Replication under Attack," Proc. IEEE Int'l Conf. Dependable Systems and Networks,pp. 105-114, June 2008.

[7] M. Abd-El-Malek, G.R. Ganger, G.R. Goodson, M.K. Reiter, and J.J. Wylie, "Fault-Scalable Byzantine Fault-Tolerant Services," Proc. 20th ACM Symp. Operating Systems Principles,pp. 59-74, Oct. 2005.

[8] M. Gudgin et al., World Wide Web Consortium, "Simple Object Access Protocol (SOAP)," Version 1.2, Apr. 2007.

[9] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem,"ACM Trans. Programming Languages and Systems,vol. 4, no. 3, pp. 382-401, July 1982.

 