IJREAT International Journal of Research in Engineering & Advanced Technology, Volume 1, Issue 1, March, 2013
ISSN: 2320 - 8791
www.ijreat.org

An Evaluation of Fault Tolerant Web Service in Mobile Cloud

Abarna.P[1], Leena Mary.L[2], Ravimaran.S[3]
Department of Computer Science & Engineering
M.A.M College of Engineering
Siruganur, Trichy, Tamil Nadu, India

Abstract

In mobile cloud based services reliability for transaction and fault tolerance are key issues. Web service-based transactional business processes require a high degree of reliability to guarantee consistent results. Since webservices are called by websites in different platforms, client faults may occur randomly at any place either due to node failure, incorrect input or any data loss transmission etc. This is more prominent in mobile services because of fewer resources. This leaves the data and consequently the webservices in an unstable state. In case of failure, clients must take necessary actions to leave the process in a globally-correct state. Based on the Web Services Transactions specifications, a framework for fault tolerance to the mobile web services is proposed. A mechanism that combines exception handling and transaction techniques are used to devise fault tolerance in web service transaction. This framework provides fault tolerance to mobile webservices when consumed by the relevant application.

Keywords: Web Services, Fault tolerance, Transactions, Mobile cloud, Exceptions, Fault Handling, Recovery in web services

I.INTRODUCTION

Transactional workflow systems have been available for long time. However, their elevated interoperability costs have kept them limited to mission-critical enterprise systems. Thanks to Service Oriented Computing and in particular, Web services, the popularity of such systems promises to increase in the near future: XML is now a widely accepted standard to represent exchanged data, applications can easily expose select business logic as Web service operations, while the Internet provides an affordable yet pervasive communication link.

Construction of fault handling logic is a time consuming and error prone task, because constructs to handle the fault tolerance are located at low syntax level.This makes it difficult to develop, maintain and update both logic types. Also the system enforces the execution of compensation before terminating the whole process leading to unstable termination of services as it is at the mercy of the designer.

Second is the assumption that every service in a transactional web service is compensable, which under practical circumstances is not possible as the compensation is allowed only within a stipulated time with cost constraints.Fail to guarantee the correctness of the fault handling logic; hence the composite service terminates in an inconsistent state. Autonomy of transactional web services is neglected and as a result process consistency is violated and whenever a fault occurs. The cost is also very high in such a scenario.

A prototype implementation shows how this framework can be integrated into existing services by introducing minimal changes to their application code. The Web services targeted by our study are commonly the result of existing application business logic. As such, they generally access an underlying database management system (DBMS) which, despite providing local data consistency, is insufficient to guarantee the integrity of the distributed business processes they collaborate with.

In the presence of operational failures, they may loose all or part of their state, leaving parts of the process in an unknown status, which leads to process inconsistency. Hence, our approach to fault tolerance is primarily interested in allowing a running transactional process to resume normal execution past the point of failure, instead of resorting to complete rollback (i.e., backward recovery) or to take all the necessary actions to reach an acceptable state.

RELATED WORKS

Though there has been an elaborate study about fault tolerance there is no comprehensive study when it comes to web services systems in combination with mobile cloud the area seems to be vague with little research done in the preceding years.

Tai et al., 2004 suggests how BPEL, WS-C, WS-AT and WS-BA can be combined to provide coordinated Web processes. Sauter and Melzer, 2005 compares capabilities of BPEL vs. WS-BA, and advocates extending BPEL to provide distributed coordination features. Tartanoglu et al., 2003 also addresses service reliability using forward recovery, but their Web service composition actions are statically defined for every participant.

Mikalsen et al in 2002 proposes a framework of action ports (proxy services) that negotiate and enforce transactional behaviour in existing services.

Rusinkiewicz and Sheth, 1995 (and earlier work) discusses transactional workflows, their heterogeneous nature and the need for failure and execution atomicity. Tang and Veijalainen, 1995 introduces the concept of consistency

units (C-unit), transaction-style sections of a process that enforce data integrity constraints across participants.

Redo Recovery after System Crashes [Lomet and Tuttle, 1995]. An *installation graph* can explain the order of log writes to forward-recover the state of a database. Logical Logging [Lomet et al., 1999] in *Logical log* operations reduce the volume of log records.

Allows more generic logging schemas. *Durable Scripts Containing Database Transactions* [Salzberg et al., 1996 says faults can be used to record and replay short ACID transactions to restore an application's state.

The *Generic Log Service* [Wirz and Nett, 1993] can be used to log protocols represented by finite state machines. These include transactional protocols such as 2PC. In his work *Transparent Recovery in Distributed Systems*, Bacon, 1991 proposes that in order to avoid fault-tolerance logic in every application, transparent recovery solutions can transform non-resilient applications into resilient ones.

PROPOSED ARCHITECTURE

The paper proposes the FACTS framework that addresses the above shortcomings for fault tolerant composition of transactional web services. It is a hybrid fault tolerant mechanism is used which combines exceptional handling and transaction techniques to improve reliable composite services.

There is a STTP (Service transfer based termination protocol) that assists composite services to terminate in consistent state whenever an unrepairable fault occurs. An Event Condition Action [ECA] is adopted which has rules to describe when and how to use exception handling. An algorithm called taxonomy of transactional web services is then used to verify the correctness of the fault handling logic.The proposed architecture has the following advantages, in that it terminates in consistent state in case of a fault and also simplifies the development and maintenance of fault tolerant web services composition.

The verification algorithm ensures that the composite services end in a consistent state despite faults. The framework is applicable to any web service execution engine. In terms of reusability it can use both fault handling and business logic and when it comes to portability it can execute in any WS BPEL engine. And as for convenience it frees service designers from simulations and tests. Also in terms of economy cost constraints are taken into account hence more profit oriented systems can be built.

Transactions have always relied on logging and Web service-based transactions are no exception. While in centralized transactional systems logging is used to enforce data consistency, in distributed environments it can also help to guarantee the consistent execution of distributed protocols.

The importance of this role can be realized considering that a distributed protocol for participants that cannot recover from failures has the potential of leaving the system in an incoherent state. By leveraging existing techniques, logging

provides a protocol event history that enables participant recovery, allowing the protocol to be continued exactly at the state it was interrupted by a failure. An Event Condition Action [ECA] is adopted which has rules to describe when and how to use exception handling.

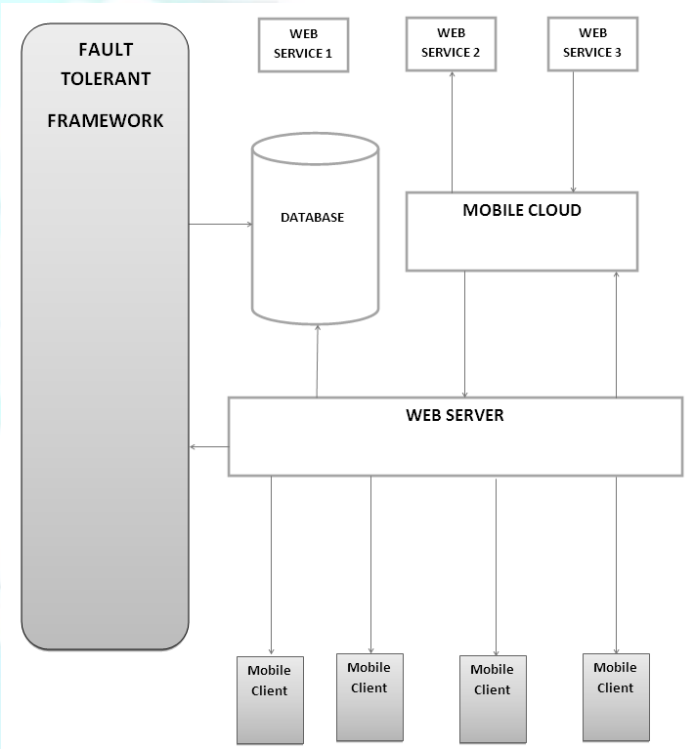

Fig 1. Fault Tolerant Framework

Since conventional Web services rarely provide the necessary fault-tolerance guarantees to take part in a distributed transaction, the reasons for using logging are twofold. This behaviour agrees with the open-nested transaction model, in which the effects of sub transactions are readily visible to others.

If this were not done, the concurrency penalty incurred by conventional data isolation could not be tolerated by most applications. The algorithm called taxonomy of transactional web services is then used to verify the correctness of the fault handling logic.

The Transactional WS composition problem has been extensively treated in the literature by using a predefined

control structure such as workflows called Advanced Transactional Models, which are based as follows:

A. Exception Handling and Transaction

A hybrid fault tolerant mechanism is used which combines exceptional handling and transaction techniques to improve reliable composite services. It employs 8 high level exception handling techniques to repair faults during execution of composite services. If there is no fault then the execution continues.

In case of a fault there is a STTP (Service transfer based termination protocol) that assists composite services to terminate in consistent state whenever such an unrepairable fault occurs .

B.WS BPEL designer

Here the business requirements are specified as processes. Transactions have always relied on logging and Web service-based transactions are no exception. While in centralized transactional systems logging is used to enforce data consistency, in distributed environments it can also help to guarantee the consistent execution of distributed protocols. The importance of this role can be realized considering that a distributed protocol for participants that cannot recover from failures has the potential of leaving the system in an incoherent state. By leveraging existing techniques, logging provides a protocol event history that enables participant recovery, allowing the protocol to be continued exactly at the state it was interrupted by a failure

C.Specification Module

An Event Condition Action [ECA] is adopted which has rules to describe when and how to use exception handling. Since conventional Web services rarely provide the necessary fault-tolerance guarantees to take part in a distributed transaction, the reasons for using logging are twofold: The need to deliver application state recovery, represented in terms of vital system variables (which collectively form the coordination context), and the need to provide transaction protocol recovery in order to guarantee its correct execution.

While this focuses on supporting the 2PC protocol, logging can be used just as effectively with other transaction protocols. For every registered participant, the participants relation stores the endpoint URL where the service can be reached, allowing the coordinator to contact active participants during crash recovery. Records in this relation are uniquely identified by combining the transaction's unique identifier and the ordinal sequence in which the participant was registered

D.Verification Module

An algorithm called taxonomy of transactional web services is then used to verify the correctness of the fault handling logic by following the ECA logic. In order to cope with failures, participants must provide pre-commit behavior: Once an operation is invoked and its positive vote is communicated, the participant has tacitly promised its ability to finalize the operation at a later time. A frequent problem, however, is that most commercially-available database systems either do not provide this behavior or provide it only as part of separate programming interfaces 4 (e.g., as part of an XA interface). Instead, the default action in case of site failures is to undo all uncommitted operations. Therefore, fault tolerant services must also provide a mechanism to restore provisional database changes present as of the time of failure.

The long-running nature of this kind of transactions requires participants to persist individual database operations as soon as they are performed, even though they are logically tentative. This behavior agrees with the open-nested transaction model, in which the effects of sub transactions are readily visible to others. If this were not done, the

concurrency penalty incurred by conventional data isolation could not be tolerated by most applications.

E. Implementation module

The output of the implementation module is fault tolerant composite services which can be deployed and executed. Since committed operations by definition cannot be rolled back, compensating operations are required to undo partial work. Compensation consists of semantically canceling the effects of an operation, as opposed to physically restoring data back to a previous state (i.e., by means of a rollback).

Unfortunately, due to the effects of open-nested transactions, compensation may lead to the undesirable need to issue cascading aborts : a chain of related compensating actions required to fix side effects created by the lack of isolation. Given these implications and the nature of some application domains, it is important to note that compensation may not always be a feasible option.

II.CONCLUSION

Thus an integrated framework for specification, verification and execution of fault tolerant composite services is achieved at a minimal cost in mobile computing for transactional web services. The entire structure is simple and easy to maintain in even large environments.

The fundamental properties of transactional business processes cannot be guaranteed unless the intervening participants offer minimum guarantees about their reliability. Our work analyzed the fundamental fault tolerance requirements for Web services in such a transactional environment. Based on increasingly popular specifications, we proposed a framework that leverages existing logging and recovery techniques to enable failure recovery. In this way, services become capable of restoring critical state information and pending database activity, allowing business processes to resume normal operation past the point of failure.

III.FUTURE WORK

In the immediate future the system can be extended to conform to additional reliability requirements that transactions resulting from more complex business processes impose on our framework. Such transactions are likely to involve advanced transaction models, whose implications might not yet be explicitly stated in the existing specifications. Transactional business processes could also improve their resilience by approaching failures using autonomic recovery. By enabling transaction coordinators to discover alternative participants, this option would greatly expand the options currently available to the framework, especially under permanent failure scenarios.

REFERENCES

[1] A. Liu, Q. Li, L. Huang, M. Xiao, FACTS: A Framework for Fault Tolerant Composition of Transactional Web Services, IEEE Trans. on Services Computing (PrePrints) (2009) 1–14.

[2] S. Bhiri, O. Perrin, C. Godart, Extending workflow patterns with transactional dependencies to define reliable composite web services, in: Proc. of the Advanced Int. Conf. on Telecomm. and Int. Conf. on Internet and Web Appl. and Services (AICT-ICIW), Washington, DC, USA, 2006, p. 145.

[3] J. El Haddad, M. Manouvrier, M. Rukoz, TQoS: Transactional and QoS-aware selection algorithm for automatic Web service composition, IEEE Trans. on Services Comp.To appear. Note de recherche LAMSADE ParisDauphine Univ. N

[4] K. Vidyasankar, G. Vossen, A multilevel model for web service composition, in: Proc. of the IEEE Int. Conf. on Web Services (ICWS), 2004, pp. 462–469.

[5] N. B. Lakhal, T. Kobayashi, H. Yokota, FENECIA: failure endurable nested-transaction based execution of composite Web services with incorporated state analysis, in: VLDB Journal, Vol. 18, 2009, pp. 1–56.

[6] J. E. Haddad, M. Manouvrier, M. Rukoz, A Hierarchical Model for Transactional Web Service Composition in P2P Networks, in: Proc. of the IEEE Int. Conf. on Web Service (ICWS), 2007, pp. 346–353.

[7] A. Brogi, S. Corfini, R. Popescu, Semantics-based composition-oriented discovery of web services, ACM Trans. on Internet Technology 8 (4) (2008) 1–39.

[8] E. Blanco, Y. Cardinale, M.-E. Vidal, Aggregating Functional and Non-Functional Properties to Identify Service Compositions - In: IGI BOOK, Vol. 53, 2010, accepted to be published.

[9] R. Hamadi, B. Benatallah, A petri net-based model for web service composition, in: Proc. of the 14th Australasian database Conf. (ADC), Darlinghurst, Australia, 2003, pp. 191–200.

[10] X. Deng, Z. Lin, W. Chen, R. Xiao, L. Fang, L. Li, Modeling web service choreography and orchestration with colored petri nets, in: Proc. of the 8th ACIS Int. Conf. on Soft. Eng., Art. Intell., Networking and Parallel/Distributed Comp., SNPD, Qingdao, China, 2007, pp. 838–843.