Integrating Fault Tolerant CORBA and the CORBA Object Transaction Service

W. Zhao, L. E. Moser and P. M. Melliar-Smith
Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106
Telephone: (805) 893-7788 Fax: (805) 893-3262
wenbing@alpha.ece.ucsb.edu, moser@ece.ucsb.edu, pmms@ece.ucsb.edu

Abstract

The Common Object Request Broker Architecture (CORBA) is widely used in enterprise applications as the middleware bus. The Object Management Group (OMG) has specified two fault tolerance related specifications, namely, Fault Tolerant CORBA (FT CORBA) and the CORBA Object Transaction Service (OTS), with different aims in mind. FT CORBA enables high availability by replicating CORBA objects, i.e., if one or more replicas of a CORBA object fails, the surviving replicas protect the processing of the business logic and, thus, provide continuous service to the clients. The OTS provides high reliability of enterprise applications by coordinating distributed transactions and protecting the consistency of the data, even in the presence of faults.

In this paper we present an infrastructure that integrates Fault Tolerant CORBA and the CORBA Object Transaction Service, transparently, with commercial off the shelf (COTS) application servers and database systems. Our infrastructure replicates the application servers to protect the business logic processing. It also replicates the transaction coordinator, which renders the two-phase commit protocol non-blocking and, thus, avoids potentially long service disruptions caused by coordinator failure. The infrastructure provides client-side automatic failover by means of a thin interpositioning library, which guarantees that clients know the outcome of the requests that they have made. It handles the interactions between the application servers and the database servers through replicated gateways that prevent duplicate requests from reaching the database servers. It retries aborted transactions, caused by process or communication faults, automatically on a client's behalf.

The infrastructure assumes that clients are not directly involved in transactions; rather, it starts a transaction automatically at the application server as a result of a client's invocation. Furthermore, the infrastructure assumes that the application servers use the implicit model of programming defined by OTS. In addition, it assumes that the application servers store persistent state in the database servers and that the database servers participate in the transaction completion protocol using the X/Open XA interface.

This paper makes the following research contributions: (1) It introduces mechanisms for replication and recovery for transactional objects that interact with COTS database servers, with strong replica consistency guarantees. (2) It recognizes the existence of additional state introduced by the OTS service during a transaction. The need to handle this additional state has an important consequence for recovery of an application server replica, namely, the application server must be at a transactional quiescent point, i.e., it must not be involved in any transaction. (3) It optimizes the two-phase commit protocol by exploiting a reliable totally-ordered multicast protocol. The multiple transmissions of the prepare and commit/abort request messages from the transaction coordinator to the XA resource managers are converted into two broadcasts to the resource managers, one for the prepare message and one for the commit/abort message. This optimization can reduce the number of messages by almost 50% if the number of XA resource managers involved in a transaction is large. (4) It provides automatic transaction retry at the server if a transaction is aborted due to process or communication faults, so that the client sees maximum roll-forward service.

Keywords: Enterprise models integration, fault tolerance, transaction processing, distributed systems, object orientation, CORBA, COTS

1. Introduction

Enterprise applications typically use three-tier architectures that are composed of thin clients, application servers and database servers. The three-tier architecture matches the logical decomposition (presentation, logic, and data) of the application. The front-end clients provide a user interface for service access. The application servers implement the business logic. The database servers manage data and transactional operations at the back-end. Transaction processing is the programming model that the application servers use. Typically, an application server initiates one or more transactions when it receives a client's request. After processing the request, the application server stores the resulting state at the back-end database servers, and returns a result to the client.

The Common Object Request Broker Architecture (CORBA) is widely used in enterprise applications as the middleware bus. The Object Management Group (OMG) has specified two fault tolerance related specifications, namely, Fault Tolerant CORBA (FT CORBA) [10] and the CORBA Object Transaction Service (OTS) [11]. FT CORBA provides management interfaces and mechanisms, with processing protected by the replication and recovery of CORBA objects. The OTS provides interfaces for distributed transaction operation and completion, with data protected by the ACID (atomic, consistent, isolated, durable) properties of the transactions.

It is highly desirable to combine replication and transaction processing in an infrastructure that uses replication to protect the processing of the business logic and transaction processing to protect the data. However, a naive combination of the two approaches does not make sense, as pointed out in [2], because existing replication frameworks lack the necessary support for three-tier transaction processing systems. In this paper we investigate the necessary mechanisms to integrate replication and transaction processing in the scope of three-tier architectures. We describe an infrastructure that is based on the replication and recovery provided by FT CORBA and the transactions provided by the OTS.

The paper is structured as follows. Section 2 describes the mechanisms of the infrastructure that replicate and recover transactional objects, optimize the two-phase commit protocol, and automatically retry failed transactions. Section 3 presents the performance measurement results and analysis. Section 4 describes related work, and Section 5 concludes the paper.

2. Integrating Replication and Transaction Processing

Many mechanisms defined in FT CORBA are complimentary to transaction processing. In particular, the problem of transaction outcome determination (i.e., the client does not know whether its request has been serviced completely) can be solved by FT CORBA's client redirection and transparent reinvocation mechanisms. However, FT CORBA does not address issues relating to the potential interactions between CORBA objects and non-CORBA objects, such as the interactions between the application servers and the database servers. Additional mechanisms are needed to handle replica consistency and duplicate detection in the presence of such non-IIOP interactions. Moreover, in FT CORBA, the granularity level is the CORBA method invocation whereas, in transaction processing, the granularity level is the transaction, which is composed of several CORBA invocations/responses and several interactions with the database server. An infrastructure that integrates replication and transaction processing must maintain the association between transactions and messages so that the logging and recovery, including transaction retry, can be done atomically for each transaction.

Figure 1 shows the architecture of the integrated infrastructure, where the middle tier (i.e., the application server) is replicated. To enable unreplicated clients to use the middle-tier services, the infrastructure employs in-bound gateways between the clients and the application servers. The infrastructure also employs out-bound gateways between the application servers and the database servers for the purpose of duplicate detection and suppression. Both the in-bound gateways and the out-bound gateways are replicated using passive replication. The application servers store persistent state in the database servers and that the database servers participate in the transaction completion protocol using the X/Open XA interface [16]. Database replication, although an interesting topic, is beyond the scope of this paper.

2.1. Replication Mechanisms for Transactional Objects

The Replication Mechanisms monitor the creation/completion of each transaction, keep track of the request/reply messages that belong to the transaction, and parse all invocations on, and responses from, the OTS server. For each transaction, they maintain a transaction identifier (XID). For the OTS management objects and application-controlled management objects, they maintain a list of object keys together with their object group identifiers. The Replication Mechanisms recognize invocations that are part of a transaction by comparing the object key in

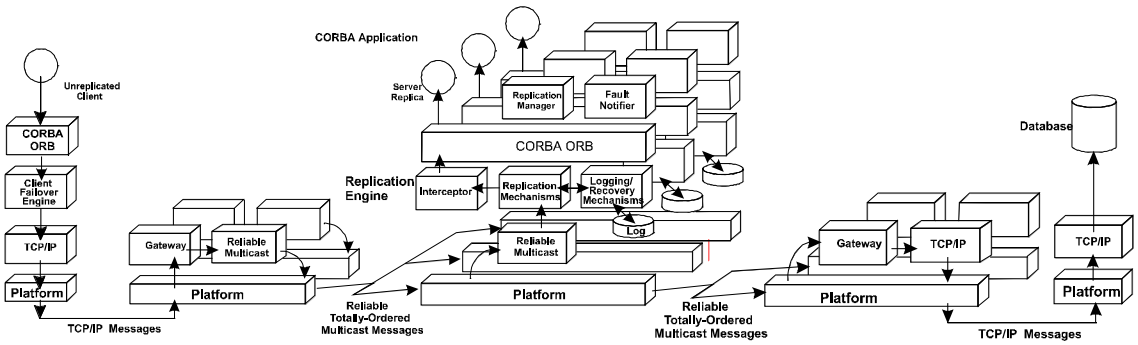


Figure 1. Architecture for access from an unreplicated client (left) to a replicated application server (middle) to a database management system (right).

the request message with the object keys in the object key list. To indicate the start or stop of a transaction, they multicast special messages to the transactional objects, and update their transaction tables accordingly. The Replication Mechanisms multiplex/demultiplex network messages and detect/suppress duplicate messages using a request/reply identifier in the message header.

Request messages from transactional objects or resource managers to database servers typically do not conform to the CORBA IIOP standard and are database vendor specific; we refer to such request messages as non-IIOP messages. The infrastructure employs an out-bound gateway, which interfaces between the transactional processes and the database server and acts as a proxy for the database server. The transactional processes and the out-bound gateway communicate via a multicast protocol, and the gateway and the database server communicate via TCP/IP. Using an operation identifier, the out-bound gateway detects and suppresses duplicate messages and sends non-duplicate request messages to the database server.

2.2. Logging/Recovery Mechanisms for Transactional Objects

The Logging/Recovery Mechanisms ensure that the state of a recovered object replica is consistent with that of the other object replicas. For non-transactional application objects, there are three kinds of state, application-level state, ORB-level state and infrastructure-level state [8]. For transactional objects, there is additional state introduced by the OTS service, which we refer to as *service-level state*. The service-level state in OTS applications comprises the state of the OTS management objects and threads created by the OTS runtime. For example, the OTS service-level state includes the association of threads and transaction contexts, the association of transactions and XA resource managers, and the status of ongoing transactions at the coordinator and the XA resource managers. For the implicit programming model, the OTS service-level state is hidden from the transactional object.

For transactional objects we consider two levels of quiescence: request/reply level, which we refer to as *operational quiescence,* and transaction level, which we refer to as *transactional quiescence.* The Logging/Recovery Mechanisms can checkpoint an object only if the object is transactional quiescent. When a transactional object completes all transactions in which it has been involved, it is transactional quiescent. Transactional quiescence naturally implies operational quiescence. A nice property of transactional quiescence is that there is no OTS service-level state associated with a transactional quiescent object.

The user can specify the checkpoint frequency as guidance for the Logging/Recovery Mechanisms; however, the Logging/Recovery Mechanisms might not be able to checkpoint the object at the exact specified frequency because they must wait or enforce a transactional quiescent state. Enforcing a transactional quiescent state means queuing requests that do not belong to the transactions in which the replica is currently involved, until all current transactions have completed. After the Logging/Recovery Mechanisms have taken a checkpoint of a replica, they can garbage collect all logged messages for that replica preceding the checkpoint.

2.3. Optimization of the Two-Phase Commit Protocol

To simplify the following discussion, we assume that duplicate messages are suppressed at the sender. In a replicated system each two-way invocation/response between two object groups is converted into (at least) two multicasts, one for the request and the other for the reply. In the 2PC protocol, assuming that the number of

participants is n, the total number of multicasts is reduced from 4n to 2n+1 by aggregating the multiple messages from the transaction coordinator to the participants.

The list of object group IDs and object keys for all of the Resource objects are piggybacked onto the multicast message. The infrastructure decides if the message should be delivered to the replica based on the object group ID information. Before delivering the message to a replica, it replaces the object key that is contained in the invocation message with the object key that corresponds to the object group ID of the receiver. The infrastructure that supports the transaction coordinator collects responses from these Resource objects and delivers them to the transaction coordinator when a request for the corresponding object is received.

2.4. Automatic Transaction Retry

Certain cases exist for which retrying an aborted transaction on behalf of the application is desirable. For example, the crash of the primary gateway facing the database server, or a communication failure between the gateway and the database server, might cause the database server to rollback all associated transactions. We mask this type of failure from clients by retrying aborted transactions.

The infrastructure restarts an aborted transaction by replaying the request that initiated the transaction. The replaying of a transaction is nothing different from processing a new transaction, except that it is initiated by the infrastructure, rather than by a client.

4. Implementation and Performance

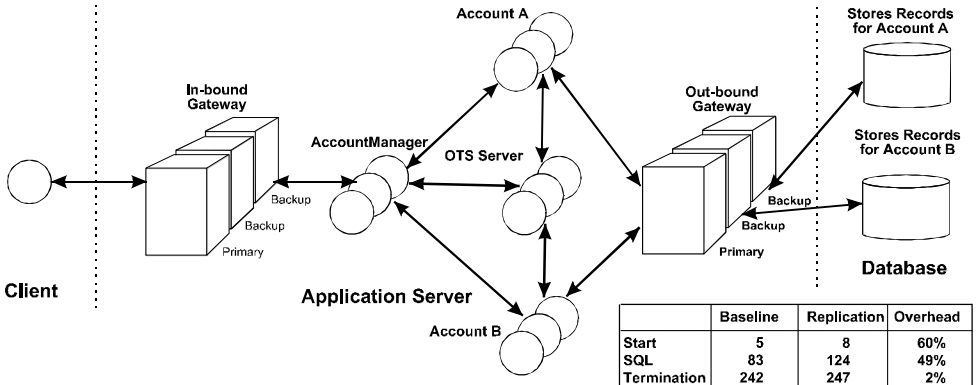
We have developed a prototype of the infrastructure that integrates replication and transaction processing, based on the OTS implementation (using ORBacus 4.0.4) from Object Oriented Concepts, Inc. [12] and the FT CORBA implementation from Eternal Systems, Inc. [7, 8]. We used the Oracle 8i Database Management System (DBMS) as the XA resource manager. Our experiments were carried out on four Pentium 3 PCs over a 100Mbps local-area network. Each PC is equipped with a single 1GHz CPU and 256MB of RAM, and runs the Mandrake Linux 7.2 operating system. Two copies of the Oracle DBMS were run on two of the PCs.

For our experimental measurements, we developed a simple application that performs a bank balance transfer from one account to another, where each account is stored in a distinct Oracle database. Figure 2 shows the experimental setup. Each transaction comprises two updates and two queries on the database server. An unreplicated client issues one remote invocation per second on an AccountManager object for a bank balance transfer. Upon receipt of a request from the client, the AccountManager object sends a request to the Account A object to debit some amount of money and sends a request to the Account B object to credit that same amount. The two Account objects then query and update their tables in the database. Each of the objects in the middle tier is three-way actively replicated. Both in-bound and out-bound gateways are three-way passively replicated.

The latencies for starting a new transaction (denoted by Start), for querying and updating the database (denoted by SQL), and for terminating the transaction (denoted by Termination) are measured at the front-end server object. The end-to-end latency (denoted by Total) is measured at the client. The difference between the Total latency and the sum of the Start, SQL and Termination latencies is denoted by Other. The latency for Other is due primarily to the round-trip communication between the client and the front-end server object. Most of the communication and processing overhead is due to the underlying reliable totally-ordered multicast protocol [15]. The integrated system incurs about 0.5ms round-trip overhead for synchronous remote invocations with messages of size less than 1KB.

It is somewhat surprising to see that the integrated system incurs virtually no overhead for the transaction termination stage. Analysis shows that the overhead from the integrated system is offset by eliminating the expensive TCP/IP connection set-up and tear-down for the 2PC protocol in the unreplicated system. The integrated system reuses the local IPC (i.e., Unix stream socket) connection between the replicated application and the Replication Engine.

The end-to-end overhead as seen by the client is about 15% for the integrated system, compared to the non-replicated case. Considering the increased reliability and availability provided by replication, this overhead is acceptable. The resource utilization of the integrated system is moderate. The integrated system takes about 10-20% CPU time on each node, and consumes about 4 Mbps network bandwidth for the rotating token of the reliable totally-ordered multicast protocol.



	Baseline	Replication	Overhead
Start	5	8	60%
SQL	83	124	49%
Termination	242	247	2%
Other	3	4	33%
Total	333	383	15%

Figure 2. Test three-tier application setup, and latency overhead due to replication.

4. Related Work

The Eternal system [7, 8] is a FT CORBA compliant implementation that uses an interception approach to provide transparent replication and recovery for CORBA applications. Eternal uses the Totem group communication system [6] to multicast messages reliably and in total order to all of the replicas of an object.

Several other researchers [5, 9] have developed partial or complete implementations of the Fault Tolerant CORBA standard that middle-tier application servers might use. To the best of our knowledge, they have not addressed the interaction with a database server in a three-tier architecture.

Frolund and Guerraoui [2] have pointed out the deficiencies of the OTS and FT CORBA as high availability and fault tolerance solutions for e-business applications. They have also recognized the difficulties of combining the two services to provide higher quality of service for distributed applications and have proposed a set of proprietary protocols as a solution [3]. Their protocols are based on passive replication and guarantee exactly once semantics for transactions (they coined the term e-transactions for this protocol). In e-transactions the client retries a request until it is eventually committed. The database servers are replicated using commercial clustering techniques. E-transactions include a 2PC protocol equivalent to distributed transaction completion, but with no-blocking guarantees rendered by passive replication of the transaction coordinator.

Little and Shrivastava [4] have proposed a high availability solution for CORBA application written in Java, where a transactional naming service manages the replication and persistent state. Their solution does not use group communication and does not use standard COTS database systems.

Much work has been done on improving the reliability of database systems, the last tier in the three-tier architecture [1, 13, 14]. We have not addressed that topic here.

5. Conclusion

In this paper we have described an infrastructure that transparently enables the integration of replication and

References

[1] D. Agrawal, G. Alonso A. El Abbadi and I. Stanoi. Exploiting atomic broadcast in replicated databases. In Proccedings of EuroPar, pages 496-503, Passau, Germany, September 1997.

[2] S. Frolund and R. Guerraoui. CORBA fault-tolerance: Why it does not add up. In Proceedings of the IEEE 7th Workshop on Future Trends of Distributed Systems, pages 229-234, Cape Town, South Africa, December 1999.

[3] S. Frolund and R. Guerraoui. Implementing e-transactions with asynchronous replication. In Proceedings of the IEEE 2000 International Conference on Dependable Systems and Networks, pages 449-458, New York, NY, June 2000.

[4] M. C. Little and S. K. Shrivastava. Implementing high availability CORBA applications with Java. In Proceedings of the IEEE Workshop on Internet Applications, pages 112-119, San Jose, CA, July 1999.

[5] C. Marchetti, M. Mecella, A. Virgillito, and R. Baldoni. An interoperable replication logic for CORBA systems. In Proceedings of the International Symposium on Distributed Objects and Applications, pages 7-16, Antwerp, Belgium, September 2000.

[6] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos. Totem: A fault-tolerant multicast group communication system. Communications of the ACM, 39(4):54-63, April 1996.

[7] L. E. Moser, P. M. Melliar-Smith, and P. Narasimhan. Consistent object replication in the Eternal system. Theory and Practice of Object Systems, 4(2):81-92, 1998.

[8] P. Narasimhan, L. E. Moser, and P. M. Melliar-Smith. State synchronization and recovery for strongly consistent replicated CORBA objects. In Proceedings of the IEEE 2001 International Conference on Dependable Systems and Networks, pages 261-270, Goteberg, Sweden, June/July 2001.

[9] B. Natarajan, A. Gokhale, S.Yajnik, and D. C. Schmidt. DOORS: Towards high-performance fault-tolerant CORBA. In Proceedings of the International Symposium on Distributed Objects and Applications, pages 39-48, Antwerp, Belgium, September 2000.

[10] Object Management Group. Fault Tolerant CORBA (final adopted specification). OMG Technical Committee Document ptc/2000-04-04, April 2000.

[11] Object Management Group. Transaction service specification v1.2 (final draft). OMG Technical Committee Document (ptc/2000-11-07), January 2000.

[12] Object Oriented Concepts, Inc. ORBacus OTS, 1.0 beta 2 edition, 2000.

[13] F. Pedone and S. Frolund. Pronto: A fast failover protocol for off-the-shelf commercial databases. In Proceedings of IEEE 19th Symposium on Reliable Distributed Systems, pages 176-185, Nuremberg, Germany, October 2000.

[14] A. Vaysburd. Fault tolerance in three-tier applications: Focusing on the database tier. In Proceedings of the IEEE 18th Symposium on Reliable Distributed Systems, pages 322-327, Lausanne, Switzerland, October 1999.

[15] W. Zhao, P. Narasimhan, L. E. Moser and P. M. Melliar-Smith. Experimental evaluation of a fault-tolerant CORBA system. In Proccedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, pages 390-396, Las Vegas, NV, June 2001.

[16] X/Open Company Ltd. Distributed Transaction Processing: The XA Specification. The Open Group, February 1992.