A B2B Distributed Replication Service

J.J. Astrain, A. Córdoba, J. Villadangos
Dpt. Matemática e Informática
Universidad Pública de Navarra
Campus de Arrosadía, 31006 Pamplona (SPAIN)
{josej.astrain,alberto.cordoba,JesusV}@unavarra.es

Abstract

A deadlock free distributed replication service for B2B CORBA based applications is presented. This service provides persistent storage for commercial transactions performed by B2B applications and ensures asynchronous passive document replication in terms of XML documents over a set of geographically distributed repositories. In order to support fault-tolerance of the centralized replication manager, a leader election algorithm is introduced to select a new manager if a failure occurs. The service provides high available document retrieval against possible server failures, granting the access to the last version of the documents. It supports failures at the replication manager level and at the repositories, and provides load balancing.

1 Introduction

Business-to-business (B2B) exchanges provide dramatic opportunities to automate collaborative business processes with customers and suppliers, generate internal efficiencies, and reach new markets at minimal cost. B2B applications extend market possibilities for companies offering such service and provide online marketplaces for businesses. B2B transactions typically involve long, complex processes including searching for vendors, requests for quotation, evaluating different proposals, negotiation, supply chain planning, shared product design, document exchange, billing, payment and extensive data analysis.

Users perform commercial transactions with B2B applications using web interfaces. Interfaces interpret XML documents that contain the structure and data of commercial transactions. B2B applications track transactions and store persistently their data on databases. Moreover, B2B applications require a high degree of fault tolerance in order to maintain high available all the information stored in the database. One of the problems that often appear is server failure and the chance the network brings, as a big resources repository, to recover the information that has been lost.

An increasing number of works are devoted to study object replication in a great number of application scenarios. Some of them concern replication in Content Distributed Networks (CDN) [4], adaptive data replication algorithms [5] and replica management algorithms [1, 3]. Other works are focused on the development of replication systems in the context of CORBA. We propose a distributed service as an interface to be used for replication, that ensures a transparent storage of data concerning commercial transactions (CTs) in Relational Database Management Systems (RDBMS) located on a wide area network. When working over WANs, broadcast communications cannot be used, and multicast communications can force to use grouping communication mechanisms. CORBA provides error free communication channels among objects.

In [2] we proposed a centralized replication service for B2B environments. We now propose a distributed replication service where the centralized control performed by the replication manager (RM) can be located on anyone of the system nodes. We preserve the same system architecture and include a new software layer in charge of the concurrency management and the data replication. It also provides a mechanism, based on an optimal message complexity election algorithm [6], to select a new manager whenever the previous manager is not available. The RM centralizes all the storage and replication operations of the system ensuring a deadlock free system [2] using an optimistic replication strategy performed on demand of the replication nodes (RNs). In comparison to other proposals, our work reduces the time wasted due to abortions, and allows load balancing.

The rest of the document is organized as follows: section 2 is devoted to introduce the system model and to explain the system operation; section 3 deals with the failure detector system; and finally conclusions, acknowledgements and references end the paper.
2 Distributed replication service

The B2B system is composed by a set of replication nodes and a set of web servers located around a wide area network. Servers and nodes can be located on a same site or not. An RN, as it is depicted on figure 1, is composed by a replication service (RS), an RDBMS and a communication module. The replication module (RMod) is in charge of the information replication. It includes the failure detector module (FDM) and the replication manager module (RMM). The RMod coordinates the replication process, establishing a communication with the replication manager whenever a web service requires the RS to store and replicate a CT. The FDM allows the RS to detect the failure of the centralized replication manager and to initiate the selection of a new one if that occurs. Since the RM is unique, it is not possible to elect a new RM if a previous one exists. In the same way, as all the replication and storage processes are always managed in a centralized way by a unique instance of the RM, the RS cannot operate if the RM crashes. All the replication nodes include in their replication services a RMM. The RMM contains all the functionalities needed by a replication service to become a replication manager if the previous RM disappears or breaks down. Only one NR of the system can have activated this functionality on a certain instant. The RMM is activated by the FDM when an RM failure is detected and the current replication service has been elected to become the new RM. Communication among nodes and sites is powered by CORBA, that ensures a reliable error-free communication channel.

![Figure 1. Replication node structure.](image)

B2B applications request the replication of XML documents to the replication service located in the replication node where they are connected to. The RS queues the requests and contacts the replication manager. In the same way, the replication manager queues the requests received from the replication nodes of the system and deals with them one by one avoiding conflicts on the remote replications services. It is an asynchronous passive document replication coordinated by the replication manager. The choice of a centralized solution grants a deadlock free service, but increases the risk of a system failure if the RM crashes. So a failure detection mechanism is introduced to detect the failure of the RS, and to select a new one among all the replication nodes candidates. When an RS requests a document replication, the manager queries the performance table [2] and selects the node where to place the master copy of the document. The replication service where a master copy is persistently stored is called the RN owner of the document. Normally, the RN that requests the replication of a certain document becomes the owner of this document, but when a node is near its saturation, the RN owner can be another one. The replication node owner of a certain document always has its latest version.

Replication is always performed on demand. The RS that requires the replication of a certain document makes a temporal copy of the CT in his RDBMS by means of the LI. Then, the replication module of the RS sends a rep_req message to the RM requesting the persistent storage of the related CT and its replication on other nodes of the system. This message includes the SQL sentences needed to store the document into the RDBMSs, some information concerning the current resources and performances of the local machine and the CT identifier. This identifier is unique for each document in the system and includes the identifier of the replication node where the CT replication was requested. This message is queued with the other messages generated in other nodes at the RM. When the RM handles this message, it verifies that the document was not previously stored and sends a rep_ack message to the remote RS. This message includes the performance table of the replication manager. This table contains the RNs owners of all the documents stored in the system. In order to decrease the amount of information that must be sent on this message, the RM only sends the registers that have been modified since the last update of the remote RS. If no failures have occur at the remote RS, the only information sent is the notification that the RN is the owner of the current document. In other case, it sends to the RS all the modifications performed over the performance table of the master during its failure period. Then the remote RS stores persistently the CT information, modifies its performance table entry for this document, notifies the correct storage to the replication manager with the rep_commit message and deletes the temporal copy of the document. After receiving the commit message, the RM starts the replication of the document in all the nodes of the system (starting by itself) sending the forced_rep message to the rest of available nodes. The message includes the previous stored SQL sentences, the performance table updated after the commit and the document identifier. At the end of the replication process, each remote RS sends a (forced_rep_ack) message to the replication manager indicating the correct storage of the document.
The manager waits for a certain time the acknowledgement message, and if it does not arrive, it takes note of the failure, sets the remote NR as unavailable and finish the replication process for this node. If less nodes than the minimum previously established commit the replication, the manager performs a second replication effort. If the error persists, the replication manager announce this situation to the rest of nodes and ends the service.

When a web service requires the retrieval of a commercial transaction previously stored, the replication node where the request is performed looks into its performance table for the RN owner of the document. The local RN contacts the RN owner and compares the versions of the document stored in both sites. If they are the same, the local RN provides the local copy of the document to the web service. In other case, the RN owner sends the latest copy to the local NR, and this one stores in the RDBMS the copy, updates the version of the document and transfers the copy to the web service. Recovering requests do not introduce deadlock problems, since all the messages are queued at the destination NR (so serialized) and read-write conflicts are directly managed by the local interface and the RDBMS of the destination NR. The messages used in this operation are `data_req` and `data_rep`.

The update or modification of a commercial transaction implies write operations over the RDBMS that can introduce conflicts or deadlock problems. In order to avoid this situation, these operations are always requested to the replication manager. When the update request is performed over the RN owner of the document, the NR performs the update over its RDBMS, modifies its performance table and sends an `update_req` message to the replication manager requiring the replication of the update. These three operations are performed in a transactional way. The failure of anyone of them implies the rollback of the transaction, that must be repeated until a commit. Then, the RM stores the changes, modifies its performance table and forces a replication sending the `forced_update` message to the rest of available replication nodes. If the replication is performed correctly, each NR sends to the RM a `forced_update_ack` message. When the update request is performed over a RN that is not the owner of the document, the RN requests the document to its owner with the `data_req` message, and the owner node sends it with the message `data_reply`. The rest of the update process is the same as previously described.

Concurrency is established exclusively at RS level. When a new query or storage request arrives to the RS, a new execution thread is generated. This thread performs the replication task in the corresponding LI or queries the RDBMS to recover a document. Thus, the more requests appear, the more threads are initiated (one thread per request). Each LI has associated a queuing system where it introduces the storage requests. Each LI interacts with its RDBMS in a transactional way. All transactions share one table of the RDBMS (where metadata is stored), transactions are serialized to access this table, avoiding deadlock situations. Figure 2 illustrates the replication service operation.

An RN detects the failure of the RS when it requests the replication or the update of a CT and the replication manager does not answer this request in a certain time interval. In this case, the replication node performs a second try, and if the failure persists, it triggers a new replication manager election process. The RM detects the failure of a certain replication node whenever it does not acknowledge a replication or query request. The RS try twice to perform the action, and if the RN does not respond, the RM sets as unavailable this RN in its performance table. This state can be changed by the RM if a new registry process is initiated by the RN. The manager pings periodically the RNs when no activity is detected during a long time. That allows the manager to verify that everything performs well on the system. When a failure of the RM occurs, the performance tables of the RNs have a significant relevance, since they allow to determine which nodes have the latest document versions, and the nodes candidates to become the new replication manager.

![Figure 2. Comercial transactions replication schema.](image-url)
3 Fault tolerance

In the previous work [2] we considered stop and crash failures of the LI and the RDBMS. In this work we take into account that the RM can stop or crash, too. Failures in the RM leads to the increase in the response time of the operations required to realize client requests. In general, fault-tolerant replication architectures increase the response time in the presence of failures. In our case and in order to offer a high available replication service, we provide a RM distributed selection mechanism based on a leader election algorithm [6] which ensures using the lower number of messages that there always exists an RM in the system. In case we adapt the algorithm for a cluster architecture, such algorithm could be reduced to a two message passing system in order to select a new RM independently of the number of nodes in the system. This could be done taking advantage of the implementation of the communication module and the characteristic of cluster architectures.

Nodes use the information contained in the performance table during algorithm execution. \( PT = (DocId, DocIdOwner, DocVer, time, load) \). \( DocId \) is the document identity, \( DocIdOwner \) contains the identity of the node responsible of the above document, \( DocVer \) is the version of the document, time indicates the time stamp when the table entry was created at the RM and load indicates the processing level at the document owner site when it updated the document. Nodes identities are unique and a total order relation is defined between them. When a node suspects that the RM is faulty, it initiates the algorithm. The algorithm consists in the forwarding of one message between the nodes of the system to select one of the active nodes of the system based on the data contained in the performance table. The algorithm uses an optimal number of messages to select this node, \( O(n) \), despite of multiple nodes can initiate the algorithm. The performance table is used by each node to select the next node in a logical ring between the nodes of the system. Each node knows a set of nodes present in the system because of the error free channel assumption and the ordered client request processing at the RM. Each node begins the algorithm by sending an ALG message to its first lower identity node in the performance table. When this message is received, the node forwards the message to its first lower identity node in the performance table, except if it is the lowest identity node in the performance table. In such case, the node forwards the message to the highest identity node. In case that different nodes initiates the algorithm, instances of each node are differentiated by the node identity. As there is a total order relation defined for nodes identities, nodes can allow to progress only one instance, the one initiated by the highest identity node. In case a node fails during the algorithm operation, that is, the sender of a message receives an unreachable exception. The sender will initiate the algorithm again taking into account this fact. Once the RM is selected, all the nodes are informed about its identity by sending an RM message toward the neighbors of each node.

4 Conclusions

This work presents a distributed replication service that allows the persistent storage of information concerning commercial transactions performed in XML format by B2B applications over web services. B2B clients access the web services following proximity criteria. Web servers are bound to replication nodes, that are in charge of the commercial transactions replication. Replication requests are managed by a distributed replication manager that ensures a correct concurrency control and provides a deadlock free service by centralizing and serializing the replication operations. Fault-tolerance of the replication manager is granted introducing a leader election algorithm to select a new manager if a failure occurs. The replication manager provides load balancing for the replication requests.

5 Acknowledgments

This work is supported by the Spanish Government under research grants TIC2003-09420-C02 and TIS2004–02940.

References


