DISTRIBUTED DATABASE TRANSACTIONS USING THE HP INTEGRITY NONSTOP SERVER

A BUSINESS WHITE PAPER FROM CREATIVE SYSTEM SOFTWARE, INC.

Executive Summary 1
How XA works 2
Transaction Flow – Single Phase Commit 3
Transaction Flow – Two Phase Commit 4
Case Study 1 6
Case Study 2 7
Summary 8

EXECUTIVE SUMMARY

HP's Integrity Nonstop (NonStop) servers fully support open access to the data that is housed on them. This open access to data combined with the ability to use modern programming languages, web-centric tools and industry standard software means that the same development and support resources can be shared across all of the open platforms (UNIX/Linux/Windows) in the IT enterprise. This ability to leverage resources brings the cost of development and support down since fewer resources are now required; all while retaining NonStop's industry leading scalability, reliability and TCO.

One of the biggest challenges facing IT organizations today is how to provide their users access to the data that is stored in a variety of different database managers (DBM). Legacy applications typically use a single DBM (NonStop SQL, Oracle, Informix, SQL Server, MySQL, …) to store their data whereas in today's SOA (Service Oriented Architecture) world applications are being developed that access data that is stored in a variety of DBM. In order to protect the integrity of the databases that are being used in these applications an architecture was developed by The Open Group consortium: XA (eXtended Architecture) which specifies the rules for executing a transaction that accesses more than a single data source. The XA standard defines how a transaction manager coordinates database operation made by different DBMs using a two-phase commit.

Of course NonStop being the industry-standards based server that it is supports XA in a variety of ways including: JDBC, CORBA, and Oracle Tuxedo (Tuxedo). All of these products work in a bi-directional manner allowing either the NonStop to control the transaction or to participate in a transaction that is under the control of another environment. While it is strongly recommended to convert legacy Enscribe files to NonStop SQL it is possible to include them in a XA protected transaction by using either CORBA or Tuxedo. This allows all of the data that resides on a NonStop system to be accessible.

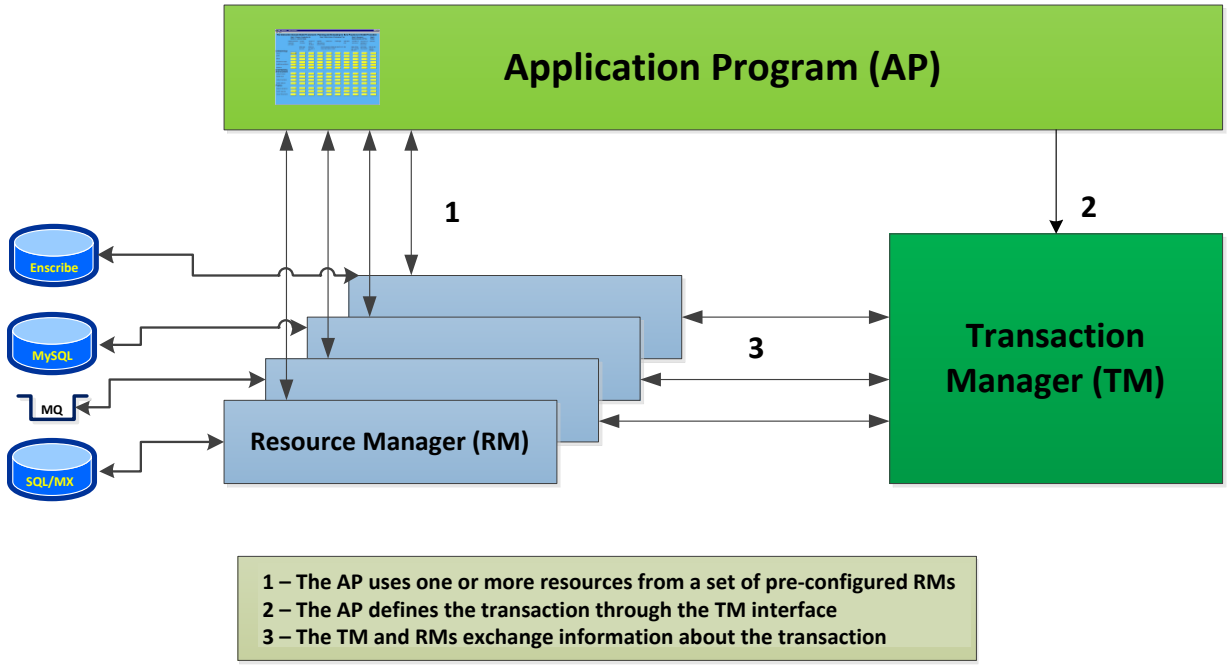
By leveraging the data that resides on disparate platforms in a safe and secure way more corporations can leverage one of their most valuable assets, their data.

While working at Tandem Computers in the late 1970s Jim Gray defined the properties of a reliable transaction system which today is known as ACID:

- ***ATOMICITY*** *– requires that each transaction is all or nothing if any step within the transaction fails, the entire transaction fails, and the database state is left unchanged.*
- ***CONSISTENCY*** *– ensures that any transaction will bring the database from one valid state to another.*
- ***ISOLATION*** *– ensures that no transaction can interfere with another transaction.*
- ***DURABILITY*** *– means that once a transaction has been committed, it will remain so, regardless of outside influences.*

Database Management (DBM) systems are very good at providing ACID properties for activities that are under its control. XA allows many DBMs to participate in a single, coordinated, atomic fashion.

How XA works



1 – The AP uses one or more resources from a set of pre-configured RMs
2 – The AP defines the transaction through the TM interface
3 – The TM and RMs exchange information about the transaction

Application Program

The Application Program (AP) defines transaction boundaries and the business logic that make up the transaction. Each request is passed to the appropriate resource manager through function calls specific to that resource manager; this is typically done through an interface such as JDBC, Tuxedo or CORBA.

Resource Manager

The Resource Manager (RM) provides an application program with access to resources such as NonStop SQL, Enscribe, and Websphere MQ. There is a separate resource manager for each type of resource that is accessed.

Transaction Manager

The Transaction Manager (TM) assigns identifiers to transactions, monitors their progress, and takes responsibility for transaction completion or failure.

Global Transactions

Global transactions involve two or more resource managers that are defined to the transaction manager, and are under the TM's two-phase commit control.

Transaction Flow – Single Phase Commit

When a Tuxedo service running on the NonStop needs to update data that is on another server it can use a single phase commit. The application only has to be concerned with a few verbs that control the transaction. The code snippet below shows how to start a transaction and to commit or abort it.

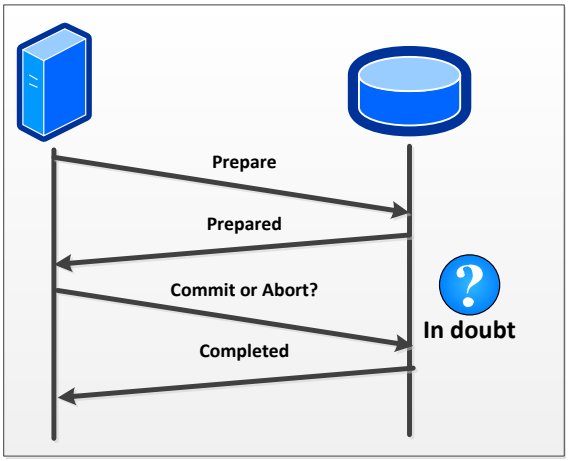
```
if (tpinit((TPINIT *) NULL) == -1)
{ // Join the Tuxedo application
        printf("Failed to join the application\n");
        exit(1);
}

if (tpbegin(30, 0) == -1) // Start a global transaction
{
        printf("Failed to start a transaction\n");
        exit(1);
}

// Send something to another service that will do some I/Os
if (retc = tpcall(service_name, audv, sizeof(struct aud), audv, audrl, 0) == -1)
{
        printf("%s service failed with an error %s\n", svc_name, audv -> ermsg);
        retc = -1;
}

/* Commit global transaction */
if (retc < 0) // We had an error so abort the global transaction
        tpabort(aflgs);
else {
        if (tpcommit(cflgs) == -1) // Otherwise commit it
        { // Commit failed :(
                printf("%s: failed to commit transaction\n", pgmname);
                exit(1);
        }
}
```

A single phase commit transaction looks and behaves like a NonStop TM/MP (TMF) transaction. There are three verbs that the developer must be familiar with (Begin, End, and Abort). Before any database operations are done the developer issues a Begin command, at the end of the transaction the End command is used, and an Abort command is used if the developer wants to return all of the data sources to the state they were in before any operations were done.

Transaction Flow – Two Phase Commit

Consider a transaction that transfers money from an account in one bank (running NonStop SQL) to an account in another bank (running MySQL). It would not want to delete the money from the NonStop SQL table until it was sure that the MySQL table had updated the account information.

The flow of such a transaction would be similar to the following:
Connect to the NonStop SQL resource manager
Connect to the MySQL resource manager
Create a unique identifier to identify the transaction
Start a global transaction via the transaction manager
Delete the money from the NonStop SQL bank account
Update the MySQL bank account with the transferred money
End the resource manager's work for NonStop SQL
End the resource manager's work for MySQL
Tell the NonStop SQL resource manager to prepare to end the transactions – If it can't it will abort the global transaction
Tell the MySQL resource manager to prepare to end the transactions – If it can't it will abort the global transaction
Commit the global transaction



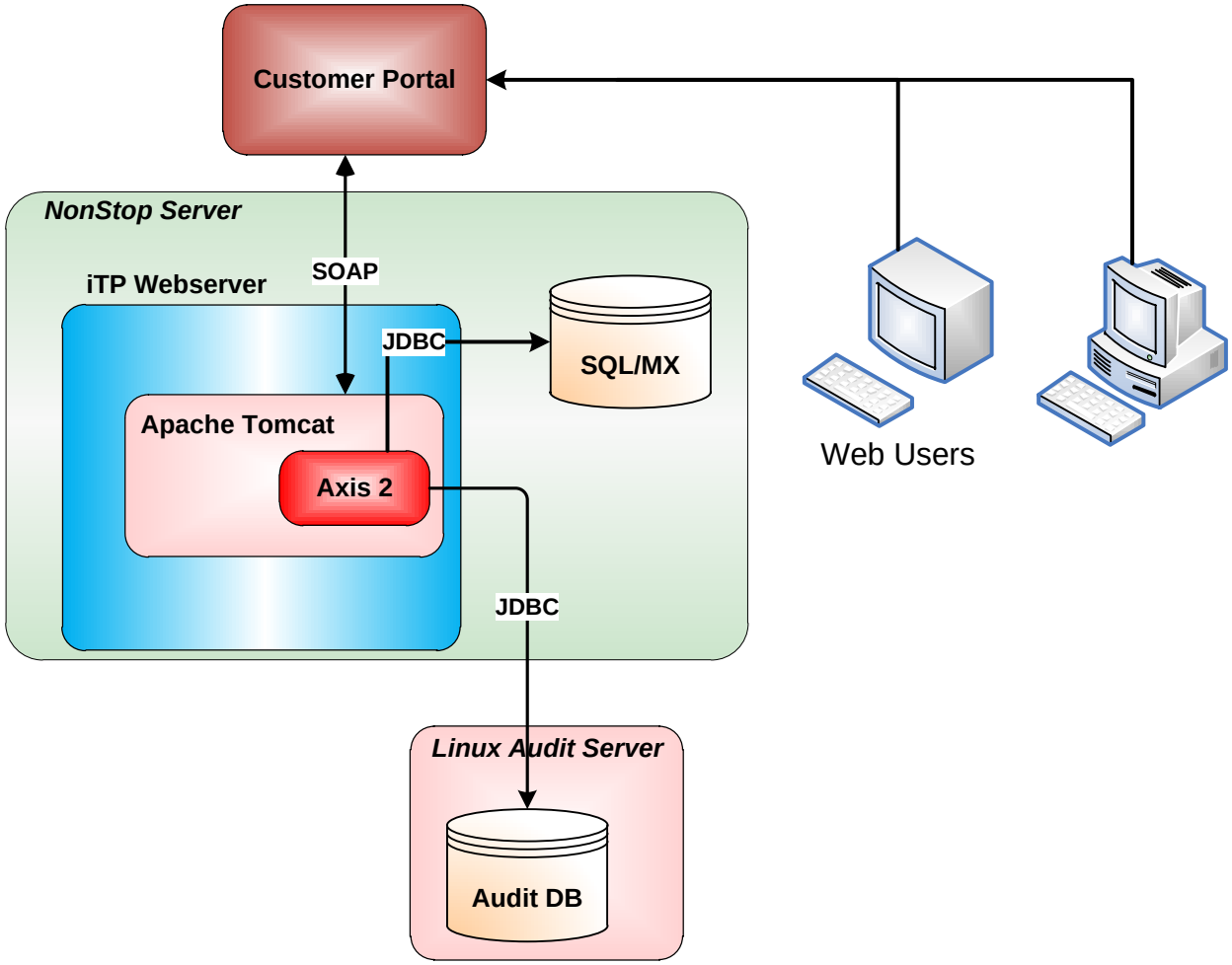
The Java code to implement the above would be similar to the example on the next page which uses JDBC to talk to the two databases. This code can be run on NonStop/UNIX/Linux/Windows or any other platform that supports Java and JDBC.

The coding of a two phase commit looks a bit more complicated due to the addition of the Prepare verb but the general flow is the same as in a single phase commit.

```
// Connect to the data source (NonStop SQL, MySQl, MQ, ...). This just gets the connection to the RM
XAConnection ds1 = getXAConnection();
XAConnection ds2 = getXAConnection();

// Get the XA Resources
XAResource xar1 = ds1.getXAResource();
XAResource xar2 = ds2.getXAResource();

// Get the Physical Connections - Open the database or queue that we will perform I/Os against
Connection conn1 = ds1.getConnection();
Connection conn2 = ds2.getConnection();

// Create the Xids With the Same Global Ids
Xid xid1 = createXid(1);
Xid xid2 = createXid(1);

// Start the branch transactions
xar1.start(xid1, XAResource.TMNOFLAGS);
xar2.start(xid2, XAResource.TMNOFLAGS);

//
// This is where the business logic goes (Create, Read, Update, Delete)
//

// End both of the branch transactions
xar1.end(xid1, XAResource.TMSUCCESS);
xar2.end(xid2, XAResource.TMSUCCESS);

// Prepare the RMs
int prp1 = xar1.prepare (xid1);
int prp2 = xar2.prepare (xid2);

boolean do_commit = true;

if (!((prp1 == XAResource.XA_OK) || (prp1 == XAResource.XA_RDONLY)))
   do_commit = false;      // Something went wrong, abort the operation

if (!((prp2 == XAResource.XA_OK) || (prp2 == XAResource.XA_RDONLY)))
   do_commit = false;      // Something went wrong, abort the operation

if (prp1 == XAResource.XA_OK)      // If everything is OK, commit the transaction
   if (do_commit)
      xar1.commit (xid1, false);
   else
      xar1.rollback (xid1);

if (prp2 == XAResource.XA_OK)      // If everything is OK, commit the transaction
   if (do_commit)
      xar2.commit (xid2, false);
   else
      xar2.rollback (xid2);

// Close the Physical connection to the database or queue
conn1.close();
conn2.close();

// Close the connection to the data source (RM)
ds1.close();
ds2.close();
```

CASE STUDY 1

In order to comply with a new requirement from their internal audit department, a NonStop customer had to add auditing capabilities to one of their existing systems. Their corporate standard for audit tables is to use a hardened DBMS running on a Linux system. Since the customer had previously converted their system to a SOA this was not a hard request to comply with.

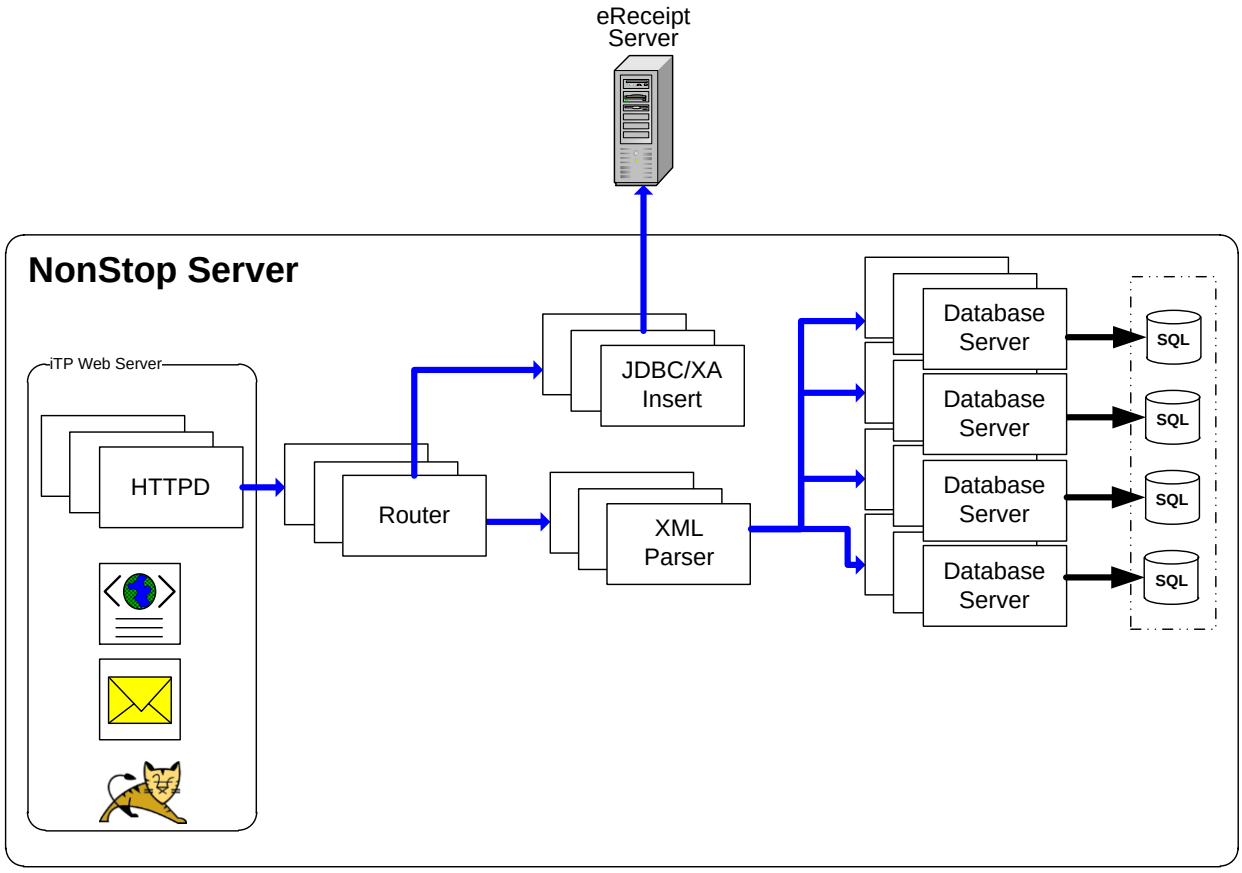


They modified the existing Java services to also do an insert into the Audit database. Since they were now updating both the SQL/MX database and the new Audit database they used XA to ensure that the databases were synchronized. They were able to make the changes in a few days by simply having the Axis 2 router send a JDBC Insert to the Audit database. Since the router was also doing JDBC to the SQL/MX database they used XA to ensure that both databases stayed synchronized.

CASE STUDY 2

A large retailer had to modify one of their NonStop systems to support eReceipts. A new Java server was created that used JDBC to insert data into the eReceipt database running on a UNIX server. The Java server receives an XML document that contains the details of the transaction and writes it to the eReceipt database running on a remote server. The Router, also written in Java, controls the XA transaction to ensure that both database stay perfectly synchronized.

The enhancement to add this service to the existing application was completed within a matter of weeks by a junior programmer with very little experience due to the use of standards based software.



SUMMARY

XA is a critical component for every IT organization to understand as it allows the NonStop to participate in a variety of transactions and different architecture models. In the SOA world that most organizations exist in today it allows components to be deployed on the right computing platform regardless of the underlying database technology. The addition of XA to an existing application isn't that difficult due to how well the HP NonStop software development team implemented it. XA can be viewed as a logical extension to TM/MP (TMF). Developers already familiar with TMF only need to add a few lines of code to their application to utilize this very powerful functionality. XA has been available on NonStop for over ten years so it is well understood and tested. Additionally, XA gives the NonStop unparalleled abilities to participate in network transactions.

As more IT organizations build converged infrastructures they will find that XA is an invaluable way to leverage the NonStop and the data that it houses. It will allow them to utilize the right platform for each application in their portfolio therefore providing a much better product to their users.

About The Author

Marty Edelman has assisted some of the world's most respected companies in architecting, designing, and implementing their information technology infrastructure. He is frequently sought for his insight and comments within the technology industry and has been a frequent speaker at numerous industry events, such as the Standish Groups Ciaos University, Hewlett Packard's NonStop users group, and the HP Executive Council. Additionally, he has been quoted and interviewed for numerous industry publications.

About Creative System Software

For over 25 years Creative System Software has been providing IT consultancy to Fortune 500 companies. Our teams of professionals have built some of the largest IT systems in the world including The Home Depot's return system, the UPS Tracking System, and the S.W.I.F.T. next generation computing platform. With an emphasis on modernization our clients have been able to increase reliability and uptime while decreasing costs and time-to-market for their IT initiatives.

For more information on the HP Integrity NonStop family of servers or any of the products mentioned in this paper please visit www.hp.com/go/nonstop

CREATIVE
SYSTEM SOFTWARE

© Copyright 2012 Creative System Software (CSS), Inc. The information contained herein is subject to change without notice. The only warranties for CSS products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty.
CSS shall not be liable for technical or editorial errors or omissions contained herein. Linux is a U.S. registered trademark of Linus Torvalds. Microsoft, Windows is a U.S. registered trademarks of Microsoft Corporation. UNIX, XA, and CORBA are registered trademarks of The Open Group. NonStop is a registered trademark of the Hewlett-Packard Development Company, L.P