**ONE MILLION FINANCIAL TRANSACTIONS PER HOUR USING ORACLE DATABASE 10G AND XA**

Bipul Sinha, Amit Ganesh, Lilian Hobbs, Oracle Corp.
Dingbo Zhou, Basavaraj Hubli, Manohar Malayanur, Fannie Mae

**INTRODUCTION**
With the proliferation of 3 tier web architecture and easy to program and deploy languages like Java, XA technology is once again in the front and center of business computing. XA is very important in these environments because it ensures integrity of complex distributed business transactions, which use multiple computing environments like databases, message servers, queues etc. With the built-in support for atomicity across asynchronous data processing models, it is hard to do web commerce without touching on some aspects of XA transaction. This paper shows at Fannie Mae how they managed to achieve a throughput of over 1 million financial transactions per hour using XA.

**ORACLE XA TECHNOLOGY OVERVIEW**
Oracle participated in the XA specification committee and introduced XA protocol support in Oracle 7 database release. Today, Oracle Database 10g supports a comprehensive XA technology stack which provides a highly scaleable and highly available XA transaction processing environment.
With the proliferation of application servers, it is critical for a database server to continue to provide for high performance transaction processing in a wide variety of system configurations. Figure 1 shows some of the mid-tier mid-tier application server environments that Oracle’s XA implementation supports. Notice that the same database XA infrastructure is leveraged across these different mid-tier environments. In this paper, we will later describe how Fannie Mae built its high throughput financial transaction processing system on Oracle XA to achieve its objectives.

**SCALEABLE TRANSACTION PROCESSING: ROW LEVEL IN-DOUBT TRANSACTION**

One of the benefits of using XA is that in the unlikely event that a transaction doesn’t complete and there is some doubt as to what state it is in, Oracle Database 10g is capable of managing this situation. The transaction prepare information is persisted to the disk and it survives database shutdown and system failures. When prepared transactions remain in the system for a long time it automatically releases precious system resources while keeping the required locks by making the transaction “in-doubt”. These in-doubt transactions prevent a new transaction from writing to the locked data while allowing the reads to go through, just like any other regular transaction. This row level in-doubt transaction lock makes the system more scaleable by supporting very high transaction concurrency.

**MULTI-DATABASE TRANSACTION COORDINATION**

Oracle XA technology is very tightly integrated with the distributed transaction infrastructure. Oracle XA may be used to coordinate a transaction with an Oracle database that internally is a distributed transaction that spans multiple Oracle databases. When used in this fashion, Oracle’s distributed transaction processing engine becomes a subordinate of the external transaction coordinator through XA. This is done automatically and without any changes or support required from either the application server or the application. The application server continues to make XA calls to the database using the appropriate libraries. The database XA infrastructure internally coordinates with the distributed transaction processing component to execute this multi-level distributed transaction.

It can be used when a transaction has to write to more than one database, as illustrated in Figure 2. A two-phase commit engine built-into the Oracle database supports XA interface initiated distributed transactions to remote databases over database links. It essentially acts as the sub coordinator to ensure atomicity of the whole transaction.
MULTI-PROCESS/MULTI-LANGUAGE TRANSACTION PROCESSING: DISTRIBUTED COMPONENT MODEL

Oracle Database 10g allows multiple client sessions to work on the same transaction if they are participating in the same global transaction. This essentially enables multiple client application components to atomically write all the changes to the database. These components may be written in one of many languages that Oracle database supports. This processing model is also very resource efficient as multiple sessions may share the same transaction branch at different times.

FAULT-TOLERANT TRANSACTION PROCESSING: XA TRANSACTION SERVICE

The external XA transaction manager initiates two-phase commit protocol to ensure atomicity of the business transaction across multiple databases. The in-doubt transaction locks are held at the databases till the transaction manager recovers from a failure and communicates the final status of the transaction to all the participants. This may cause the database locks to be held for a long while, thereby, serious affecting availability of critical data. Oracle XA Transaction Service essentially solves this problem by allowing external transaction manager to transfer the two-phase commit and recovery responsibility to the transaction manager of the database. All the two-phase protocol logs are kept in the database along with the regular transactional log. It ensures the availability of the locked data to be as good as the availability of the database. Note that this also prevents the mid-tier from being a point of failure for data in the database. Figure 3 describes a traditional XA based coordination from the mid-tier. In this model, the transaction logs need to be maintained with the transaction coordinator in the mid-tier. This adds the mid-tier as an additional point of failure and recovery.

---

Fig. 2. Regular XA Processing with multiple databases
An environment using Oracle’s XA Transaction Service is illustrated in Figure 4. With Oracle’s XA Transaction Service, delegating transaction coordination to one of the Oracle databases creates a higher availability system – one without a mid-tier dependency for transaction recovery and failures.

![Diagram of XA Transaction Service: Oracle Coordinated Two-Phase Commit](image)

Also, delegating the responsibility of transaction coordination to the database creates opportunity for performance optimization of the protocol itself. XA Transaction Service automatically converts two-phase commit to a single-phase commit if all the XA transaction branches are co-located with the database transaction manager. This intelligence ensures the atomicity of business transaction without the expensive two-phase protocol. Such an optimization is particularly useful in mixed application environments – where some of the transactions are performing XA on multiple databases while others are using multiple XA branches on a single database.

**IMPLEMENTING XA AT FANNIE MAE**

**BUSINESS REQUIREMENTS**

The requirement was to be able to process one million financial transactions per hour. Each transaction is a complex set of database operations based on the type of financial transaction.

The proof-of-concept (POC) exercise was intended to measure the overhead of XA protocol on the database, CPU utilization and to help make recommendations on the choice of JDBC drivers. The overall goal was to minimize the database CPU utilization while using the XA protocol.
WHAT WAS MEASURED

We measured the CPU utilization on the database server host to achieve about 1 million transactions per hour (about 278 transactions/second). The actual run rates varied, but the results were normalized to project the CPU utilization that would be seen if run at a rate of 1 million transactions per hour (about 278 TPS).

DATABASE TRANSACTIONS

Business transactions were divided into two categories: small - with 2 inserts, 8 updates and 10 select statements and large - with 11 inserts, 12 updates and 25 select statements.

TEST SCENARIOS

In order to measure the performance overhead associated with various components we categorized the tests as follows:

- **Scenario 1** (baseline): Java client program executed small or large transactions. XA and middle tier/application server were not used. Baseline tests were intended to measure the best possible performance under ideal condition with type-II and type-IV Oracle JDBC drivers. This scenario also includes Java client executing small or large types of transactions by calling PL/SQL procedures.

- **Scenario 2** (XA, but no middle tier): Java client program executed small and large transactions using Oracle XA drivers. Neither middle tier nor application servers were used. The intent of this test was to measure the XA overhead in its simplest form.

- **Scenario 3** (full XA): Stateless session beans executed small and large transactions in BEA Weblogic application server container. Session beans were driven by message driven beans (MDBs). BEA Weblogic cluster was configured on four nodes. A Java client pre-populated an inbound JMS message queue, which triggered the stateless session bean to execute a Container Managed Transaction. The bean executed small or large Oracle transaction and then wrote the confirmation message to JMS outbound queue, all in a single XA transaction.
CONFIGURATION OF VARIOUS SYSTEMS

This section describes the configuration and tuning for various components of the system under test.

DATABASE SYSTEM CONFIGURATION

The database server was a SUN E15K with 48 CPUs (1.2 GHz) with 96 GB of memory running Solaris 8. Storage was an EMC DMX array. Oracle Enterprise Edition 9.2.0.4 was used. The database was 800 GB configured based on Oracle recommendations as follows:

- Locally Managed Tablespaces (LMT): In locally managed tablespaces, Oracle moves the tablespace information out of the data dictionary tablespace and stores it directly within the tablespace itself and it relieves data dictionary contention.
- Automatically Segment Space Management (ASSM): With ASSM, the linked-list freelists are replaced with bitmaps, a binary array that turns out to be very fast and improves segment storage internals.
- Table and Index Partitioning: Partitioning was used for many large tables and indexes to reduce and balance IO and reduce database contention.
- Automatic Undo Management (AUM): AUM completely automates the management of undo data. A database running in automatic undo management mode transparently creates and manages undo segments. It significantly simplifies database management and removes the need for any manual tuning of undo (rollback) segments.

APPLICATION SERVER CONFIGURATION

We used BEA Weblogic application server version 7.0 SP2. Up to 8 clustered instances of BEA Weblogic servers were employed and were distributed on 4 E480R Unix hosts. The load balancer evenly distributed JMS messages to managed servers.

JDBC CONFIGURATION

Oracle JDBC Client Version: 10.0.1 class-IV (Thin) client only, and 9.2.0.3 both class-IV (Thin), and class-II (OCI) drivers were used during the POC.

UNIX SYSTEM TUNING

UNIX and database statistics were collected and analyzed for each run. Initially, database showed significant waits on buffer busy, latch free and enqueue. Oracle recommended converting these hot spots to warm spots by rebuilding these tables with 64-way hash partitions or sub partitions. Hash partitioning or sub partitioning is very effective in reducing or eliminating waits.

APPLICATION SERVER TUNING

Here are the parameters we tuned to obtain optimal performance.

A. XA Prepared Statement Caching
By far, enabling this feature had the best performance impact.

**WHAT IS THIS PARAMETER?**

When you enable prepared statement caching, Application server caches a set number of prepared and callable statements used in applications and EJBs. When an application or EJB calls any of the prepared or callable statements stored in the cache, WebLogic Server reuses the statement stored in the cache. Reusing statements eliminates the need for parsing statements in the database, which reduces CPU usage on the database machine, improving performance for the current statement and leaving CPU cycles for other tasks.

**HOW TO SET THIS PARAMETER?**

- Use the Administration Console to set the XA Prepared Statement Cache Size attribute for a connection pool OR
- Use the WebLogic management API to set the XAPreparedStatementCacheSize attribute OR
- Set the attribute directly in the configuration file (when Weblogic Server is not running).

**IS THERE ANY DIFFERENCE BETWEEN XA and Non-XA PREPARED STATEMENT CACHE?**

Yes, for non-XA Prepared Statement Cache, WebLogic uses a fixed algorithm to store in cache for each connection in the connection pool where as for XA Prepared Statement Cache, Weblogic uses Least Recently Used (LRU) algorithm to determine which statement to store in the cache for each connection in the connection pool.

**B. WLS Execution Threads and JDBC Connections**

- 25 Threads and 25 Connections
- 25 Threads and 20 Connections
- 30 Threads and 12 Connections

**Observations**

Ideally, 25 WLS threads and 20 connections yielded the best results.

**C. Instance level Configurations**

- Message Driven Bean Pool:
  - 10 per WLS instance
  - 15 per WLS instance
  - 30 per WLS instance
- EJB Pool Size:
  - 10 per WLS instance
  - 15 per WLS instance
- EJB types
  - EJB executes transactions within EJB via JDBC calls
  - EJB executes transactions via PL/SQL procedure

**Observations**
XA Prepared statement caching helps bring down the database hits significantly. When transactions were executed via PL/SQL procedures, we noticed a substantial improvement in performance.

RESULTS AND ANALYSIS

A. Initial Configuration (using 9i JDBC driver)

Following graphs show comparisons between scenario (1) no XA and (3) full XA.

As we see, using the XA protocol and the middle tier (Weblogic server (WLS) and JMS) added a considerable overhead to Oracle server CPU usage when compared to stand-alone Java program making no XA calls; The overhead is larger for the small transactions.

Break down of CPU overhead:
Part of the overhead is due to the use of XA protocol itself, and part of the overhead is due to the use of the middle tier, which includes BEA Weblogic application server and JMS, and part of the overhead is due to JDBC driver, which affects both middle tier and back end database; Overhead is higher for small transactions.

**B. Improvements with Oracle Database 10g JDBC drivers**

Using the Oracle Database 10g thin JDBC driver resulted in substantial improvements in performance. Oracle re-architected the Oracle Database 10g “thin” JDBC driver to reduce the code path, reduced round trips between client and server, and improved scalability.

Breakdown of CPU overhead:
When CPU breakdown between section (A) and (B) are compared, overhead of both the XA layer and the middle tier are much smaller than with 9i drivers, and overhead of the XA layer is relatively higher for the small transactions.

Further improvements with Weblogic server tuning:

Overhead of XA end to end is reduced to 11%, CPU required is 56% of that without Weblogic tuning, and enabling XA “Prepared Statement Cache” appears to be making the difference. We did not attempt to study the breakdowns of CPU usage in this case.

RESULTS AND ANALYSIS

Performance and scalability of XA
Given the complexity inherent in the XA protocol it may be natural to question whether database transactions using XA perform and scale compared to regular non-XA transactions. Our POC showed that over 1 million XA
transactions an hour could be processed on a large SMP server using the Oracle 10g JDBC driver. The POC also showed that the performance overhead of XA (measured in terms of CPU use on the database server) could be small.

Use of Oracle Database 10g Thin JDBC Driver
Prior to Oracle 10g JDBC driver, the OCI driver offered performance benefits in some circumstances (whether or not XA was used). Our POC showed that in the 10g version, the performance benefits of the OCI driver are now available in the thin driver as well. Users now have the option of the use of the thin driver for simplicity of administration and purity of architecture without giving up performance. The OCI driver continues to provide features that the thin driver does not (such as Transparent Application Failover) and may be appropriate in some cases. In summary, Oracle 10g JDBC drivers expand the choices available to the users by enhancing the scalability and performance of the thin drivers.

**CONCLUSION**
Oracle’s XA technology provides a highly available and a high performance infrastructure for distributed transaction processing. Using Oracle Database 10g JDBC drivers, Fannie Mae demonstrated that they could scale to a throughput rate of one million XA transactions an hour on a 48-processor system while incurring a small overhead from using XA.