Analyzing the Scalability of Transactional CORBA Applications
Phong Tran (phong.tran@cmis.csiro.au)
Ian Gorton (ian.gorton@cmis.csiro.au)
Software Architectures and Component Technology (SACT) Group,
CSIRO Mathematical and Information Sciences (CMIS),
Building E6B Macquarie Uni.
North Ryde NSW 2113 Australia

Abstract:
The middleware technology used as the foundation of Internet-enabled enterprise information systems is becoming increasingly complex. In addition, the various technologies offer a number of standard components architectures that can be used by designers as templates to build applications. However, there is little understanding in the software industry on the strengths and weaknesses of competing technologies, and the different trade-offs that various component architectures impose. This paper describes the approach being taken in CSIRO’s Middleware Technology Evaluation (MTE) project to attempt to alleviate some of these problems. Specifically, this paper focuses on the scalability of transactional CORBA applications using the Visibroker Integrated Transaction Service v1.2. It discusses the methodology that has been developed, the application and environment used, and presents some performance results regarding the scalability of the Visibroker ITS.

1. Introduction

The Internet has forever changed the requirements for enterprise software systems. The very nature of the Internet brings to bear new pressures on applications that are not commonly experienced by traditional networked information systems. The impact of well-known non-functional requirements such as manageability, scalability, security, reliability and transactions is dramatically increased when applications are opened up to potentially limitless numbers of concurrent users.

In response to the demands of the increased scope of enterprise systems across the Internet, a new genre of software product has evolved. These are known as Application Server (AS) technologies. These products provide a ready-made distributed system infrastructure for building high-performance, enterprise-scale information systems. They provide an abstract model that hides the low-level complexity of programming interfaces for transport layer mechanisms such as TCP/IP sockets.

AS products are typically built on top of some existing middleware infrastructures, such as CORBA, Java RMI or COM+. AS technology leverages the underlying middleware infrastructure, and incorporates higher level services such as transaction and database connection management, security and directory services.

A number of competing software vendors provide AS products that implement CORBA specification. Consequently, when enterprises commit to a CORBA strategy, they are faced with choosing a specific product to utilize in their applications. This is challenging, because on the surface all the products provide much the same set of services and features, as they implement the CORBA specification. However, the quality and degree of support for these services does in fact vary considerably, and the success of mission-critical applications often depends on choosing a
technology which fits the specific application needs. This is a complex, time-consuming and usually expensive evaluation process that most organizations are ill equipped to perform.

In response to these challenges faced by the software industry, we have instigated the Middleware Technology Evaluation (MTE) project. The major aim of this project is to rigorously and independently evaluate competing AS products and disseminate the results to industry through publications and seminars. Our approach leads us to study and analyze each product, build real applications with them, and carry out extensive performance testing of these products. In order to compare the products' behavior, we impose a fixed application architecture, database structure and defined client transaction load, and observe how the system behaves. We then use the results from the performance tests to analyze in depth the behavior of the products, and give us insights into what each product does well, and which features are problematic, or simply do not work.

The MTE project has so far applied this evaluation approach to CORBA-based products such as Inprise's Visibroker ITS, Iona's OrbixOTM, various EJB-based technologies, as well as proprietary products such as Sun Microsystems's Forte 4GL. Comprehensive reports for each technology containing independent and in-depth technology evaluations and performance test results are available through www.cutter.com.

This paper focuses on the scalability of the Visibroker Integrated Transaction Service (ITS) version 1.2. The ITS component is integrated with the Visibroker ORB version 3.4. The paper starts with a brief overview of the Visibroker ITS. It then describes the test application and environment used for the performance analysis. Results are then presented to show the scalability of the Visibroker ITS system.

2. Overview of Visibroker ITS

The Visibroker Integrated Transaction Service (ITS) is a CORBA-based product for distributed

![Visibroker ITS architecture](image)

Fig. 1: Visibroker ITS architecture

transactional CORBA applications [1]. Its architecture is shown in Figure 1. Implemented on top of the Visibroker ORB, the ITS supports the CORBA Object Transaction Service specification. Distributed transaction processing capabilities are provided by a set of services for:

- Distributed transaction coordination
- Transaction state logging and recovery
• Integration with databases and legacy systems
• Administration or running systems

The ITS is the core component that manages transaction commencement and completion. The ITS functionality can be incorporated into an application as a stand-alone executable or as a shared library. This architecture allows the ITS Transaction Service to be deployed as a standalone process or be linked to an application server process.

The Database Integration Components includes
• JDBC DirectConnect: This enables Java ITS-based applications to use JDBC for database connectivity.
• ITS Session Manager XA Implementation: This allows an application to obtain an ITS-enabled connection to a database, handles XA calls, and performs a 2-phase commit for transaction resolution.
• ITS XA Resource Director: his handles all transactions with a specific Resource Manager (for example a database) on the network. It bridges the ITS and X/Open transaction environments.
• ITS Session Manager DirectConnect Implementation: This provides non-XA database connections and utilizes a single-phase commit protocol.

The ITS Administrator is a graphical tool used for managing distributed transactions over the network and for configuring connection profiles for use with specific databases.

The Visibroker ORB provides the underlying infrastructure for ITS-based applications. This including thread-pooling, multiplexed and recycled connections, load-balancing, and fault tolerance. An in-depth comparison of the ITS architecture with other CORBA OTS is in [3].

3. Test Application

In order to explore the performance of the Visibroker ITS, we use a sample application known as Stock-OnLine. This section describes the application design. A more complete description is available in [2].

3.1. Requirements

Stock-OnLine is a simulation of a simple on-line stock broking system. It enables subscribers to buy and sell stocks, inquire about the up-to-date prices of particular stocks, and get a holding statement detailing the stocks they currently own. From a subscriber’s perspective, the following services are offered:
• Create Account: A person wishing to enroll with Stock-OnLine can create himself a subscriber account with the service provider.
• Update Account: A subscriber can modify their allocated credit limit.
• Query Stock Value: A subscriber can query the current price for a given stock. A unique identifier code, or a mnemonic code can be used to identify the stock value to be retrieved.
• Buy Stock: A subscriber can place an order to buy a given number of a specified stock. If successful, a transaction record is created for later processing.
• **Sell Stock:** A subscriber can place a request to sell a specified number of any stock item they have purchased through the Stock-OnLine. If successful, a transaction record is created for later processing.

• **Get Holding Statement:** A subscriber can request a statement of all the stocks they have purchased through Stock-OnLine and still retain ownership of.

The *Buy* and *Sell* transactions are designed to be relatively demanding in terms of the number of database operations they perform. Both update a number of tables and will encounter lock contention in their operation. *CreateAccount* and *Update* perform database modifications, but are simple and fast. The remaining three transactions are all read-only, and will execute quickly as they only perform a single database operation and should encounter little contention for data.

### 3.2. Database Design

In the application used for the performance tests, a simple database design is used that contains the minimum tables and fields to allow the system to sensibly operate. The database is physically implemented using Oracle version 7.3.3. A brief description of the tables is as follows:

• The *SubAccount* table holds basic information on a subscriber, including name, address and credit limit. When a new account is created, the system needs to allocate a new, unique account number. This is achieved using Oracle *Sequences*.

• The *StockHolding* table contains information on the amount of every stock item that a subscriber holds.

• The *StockItem* table holds price information about each stock that a subscriber can trade through the Stock-OnLine system.

• The *StockTransaction* table contains information on each buy or sell transaction that a subscriber performs.

![Diagram](image)

**Fig.2: Application Architecture**

### 3.3. Application Architecture
The basic 3-tier architecture of the application is shown in Figure 2. The Visibroker Server (VS) node, which contains the application server processes, uses the Oracle SQL*Net to communicate with the Oracle database on a remote machine.

3.4. Client and Server Configurations

The configurations for the application client and server processes used for the performance tests are described as follows:

- Multi-threaded clients: A thread in a client process represents a single application client. Each client thread executes a specified mix of transactions as described in the following section. There are typically 10 threads per client process.

- Multi-threaded Server Processes (SP): Each application server process creates and registers one server object (SO) with the ORB. Each process has a fixed size thread pool to manage client requests. The maximum number of threads in the ThreadPool is set by function thread_max_value). Each SP also embeds an ITS Transaction Service instance.

- Server-managed ITS Transactions: A Current transaction object is associated with each method of the server object. This means each server method calls the Current object’s begin(), commit() or rollback() methods in order to manage the database operations within an ITS transaction.

- ITS DirectConnect Session Pool Manager: The DirectConnect database connection pool manager is used to allocate connections to an Oracle resource manager.

3.5. Client Test Behaviour

Each client thread performs a number of iterations of a fixed transaction mix. The transaction mix represents the concept of one complete business cycle at the client side. It comprises 1 CreateAccount, 3 BuyStock, 3 SellStock, 1 Update, 15 QueryById, 15 QueryByCode and 5 GetHoldingStatement transactions, totaling 43 individual transactions. Of these transactions 81% are read-only and 19% are read-write. The aim of this transaction mix is to represent a reasonably typical WWW site transaction load, where over 80% of transactions are read-only (browse) operations.

Essentially, the client loops through the transaction mix, selects a transaction type at random and performs the selected transaction. At the end of the process, when all iterations are completed, the client logs all the performance measurements to a tab-separated text file for future processing.

The client doesn’t wait or pause between each transaction invocations. Once a transaction is complete, every client immediately starts the next selected transaction. In this sense, the clients in these tests are more characteristic of servlets in a WWW-based application. A wait period is often known as a think time, and is meant to simulate the delay between transaction submissions from a single user. A think time effectively lowers the transaction rate that an application must process from a given number of clients. With think times, many more clients would need to be run to produce an equivalent transaction load without think time. Also, servlets don’t think!

4. Test Environment

In order to analyze the scalability of the Visibroker ITS, a number of tests were carried out using either one VS node with multiple replicated SPs or multiple VS nodes, each with equal number of replicated SPs.
4.1. Hardware

The machines used in the tests are high-end PC's running Windows NT 4.0. Each machine is a Dell Precision 410 with dual Pentium 450Mhz processors, 0.5GB of main memory and 20GB of disk. A switched 10Mbit Ethernet connects the test machines. To ensure the environment's consistency, all processes that are not related to the Visibroker ORB runtime and services, the operating system, the database and the application server are not activated.

4.2. Database Population

Prior to each performance test, the database is populated with initial test data. The SubAccount and StockItem table each contains 3000 rows. The StockHolding table contains 10 rows for each row in the SubAccount table, totaling 30000 rows. Because the database's content and the number of rows change during each test run, it is re-initialized to the initial test data after each test.

4.3. Performance Measurement

The application performance is expressed in terms of three measures for a given client load. These measures are:

- Average client response time, measured from the time the client invokes a remote method to the time the method returns. It includes the network transmission time, the queuing time at the server and the server-side processing time.
- Average server-side time, measured from the time a server object’s method starts to the time the method completes at the server. It includes the time for getting a database connection from the connection pool, getting an ITS transaction context, accessing the database and committing the ITS transaction.
- Average number of Transactions Per Second (TPS), calculated by dividing the total number of completed transactions of a given client load by the average client thread time. This represents the overall system throughput.

Before a test begins, the ITS Session Manager's database connection pool is empty. In initial tests, we observed that there is a delay of several seconds in getting a new connection at test startup. Therefore a warm-up run is performed that causes the connection pool to be filled before each test to avoid these delays, which can skew overall results. The warm-up run involves running a number of clients that execute a number of simple read-only transactions at each of the server processes.

5. Test Results

In each test, we use multiple replicated 5-threaded server processes (5T SPs). The client loads are 200, 300, 500 and 800 clients.

5.1. Results based on one VS node.

Figure 3 shows the client response times against the number of clients using six 5T SPs (6x5T SPs). It shows that the client response times of all transactions increase linearly with the number of clients. Figure 4 shows the application throughput for each test run with 200, 300 and 500 clients. It shows the TPS only fluctuates minimally with the increasing number of clients.
5.2. Results based on multiple VS nodes

Figure 5 and 6 show the client response times and the server-side times respectively of all transactions with one and two VS nodes. They show that while the server-side times are very similar, the client response times decrease to about half when two VS nodes are used.

Figure 7 shows the behavior of the client response times against the number of server processes running on two VS nodes. It shows that increasing the number of SPs does not change very much the client response times. Similar behavior for the client response times is also seen when one VS node is used.
5.3. Performance Comparison

Figure 8 compares the performance in terms of TPS values of one, two and three VS nodes. It shows that the use of multiple VS nodes significantly improves the overall system performance, achieving a maximum throughput of around 400 TPS.

During the tests, the CPU usage of the VS nodes, the database machine and the client machine is monitored using the Windows NT Task Manager. With one VS node in use, the CPU usage of the VS node is at a consistent 100% level for the whole duration of each test as seen in Figure 9. The CPU usage of the database server is on average about 25% as seen in Figure 10. The CPU usage on the client machines is minimal for all tests, normally less than 5%.
With two server nodes in use, the level of CPU usage of the database server, as seen in Figure 11, is twice as high compared with when one server node is used. The CPU usage of each of the two VS nodes remains consistently at 100%.

These performance results and the CPU usage of the database machine clearly show that the increased processing capacity of the VS nodes can be effectively utilized by the ITS server processes. With more CPU cycles available, the server threads are able to drive the database more effectively, and consequently improve overall throughput. This is an excellent result, demonstrating good scalability of the technology.

![CPU usage of DB server with 2 VS nodes for 6x5T SOs & 300 clients](image)

Still, the 100% level of CPU usage of each of the 2 server nodes indicates that each VS node remains overloaded. When tests run with 3 VS nodes, we find the CPU usage of each of the three VS nodes is on average 50%. This indicates that the load from 300 clients is not sufficient to fully utilize the capacity of the three VS nodes.

### 6. Conclusion

Visibroker ITS 1.2 has been evaluated with the Stock-Online application to test its capacity for scalability. The performance test results indicate that it has demonstrated good scalability. The throughput rate increases from 160 TPS with one VS node in use to about 400 TPS with three VS nodes. While two VS nodes are still fully loaded with 300 clients, the use of three VS nodes indicates that each node still has extra capacity to handle more clients. This indicates that the higher number of server nodes increases the transaction throughput with a higher client load.

### References