Benchmarking **AND BEATING** Microsoft’s .NET 3.5 with WebSphere 7!

Performance Report

SWG CPO Performance Team

For questions on this document please contact:
Justo Guizan at guizan@us.ibm.com
Executive Summary

Recently, Microsoft published a paper claiming superior performance to WebSphere.

The test was based on a derivate of an IBM stock application called Trade. Several years ago, IBM released a sample application known as Trade. It has since gone through many changes and versions. The intent was to serve as a sample application illustrating the usage of the features and functions contained in WebSphere and how they related to application performance. In addition, the application served as a sample which allowed developers to explore the tuning capabilities of WebSphere.

The following diagram illustrates the long evolution of the Trade application:

IBM has released a version of Trade as an open source application known as DayTrader (the dark green box above). Microsoft then released an application know as .NET StockTrader (the red box above). Microsoft describes the .NET StockTrader application as follows:

“...an end-to-end application based on an online stock-trading scenario. ... As a benchmark sample and downloadable benchmark kit, the application also illustrates best-practice programming for building high-performance and scalable service-oriented applications...”

The application claims to be “functionally equivalent” to the IBM WebSphere Trade 6.1 sample application. It is not a “port” of the application in any sense. Little, if any, of the original application design was ported. Microsoft has made this an application that showcases the use of its proprietary technologies. A major indication of this is the fact that the .NET StockTrader application is not a universally accessible web application since it can only be accessed by using Internet Explorer, and not by other web browsers.
Microsoft created a completely new application and claimed functional equivalence at the application level. The reality is that the Microsoft version of the application used proprietary SQL statements to access the database, unlike the original version of Trade 6.1 which was designed to be a portable and generic application. They employed client-side scripting to shift some of the application function to the client. They tested Web Services capabilities by inserting an unnecessary HTTP server between the WebSphere server and the client. And if that was not enough, they failed to properly monitor and adjust the WebSphere application server to achieve peak performance.

The Competitive Project Office team did a reverse port of Microsoft’s StockTrader 2.0 product back to WebSphere 7 (the yellow box above). Please note that we did a port that faithfully reproduced Microsoft’s application design. The intent was to achieve an apples-to-apples comparison. We will refer to our reverse port as CPO StockTrader. CPO StockTrader beat Microsoft StockTrader 36% with no errors. .NET StockTrader is prone to errors which translate into lost business!
The Competitive Project Office has over the years done extensive testing of Microsoft’s offerings versus IBM products. We traditionally run a performance benchmark called Friendly Bank. Friendly Bank was designed from the ground-up to be performance oriented. The benchmark runs on both J2EE and .NET platforms. All caching is turned off in this benchmark to reveal the true underlying performance of the architecture.

The Friendly Bank results demonstrates an even bigger performance advantage for WebSphere. In this case, WebSphere runs from 2.8 to 4.5 times faster than .NET.

In the benchmark “war” Microsoft is waging against WebSphere, it’s important to keep one fact in mind. When the comparisons are based on apples-to-apples benchmark designs, WebSphere regularly outperforms .NET.

When caching is turned on, we observe a 1.3X advantage for WebSphere. When caching is turned off, we observe a 2.8 to 4.5X advantage for WebSphere. Depending on your workload, this can translate into significant savings in hardware processors and software licenses.

Now for the serious facts and figures that back up all of this high level information. In the rest of this document, we will discuss in detail both the Friendly Bank and the StockTrader benchmarks.
StockTrader

As mentioned in the introduction, the Competitive Project Office has made a port of Microsoft’s StockTrade 2.0

Application:
Based on the .NET StockTrader application, Microsoft has made claims of superior performance. It is around these claims on which this study was focused. The Competitive Project Office has taken the .NET StockTrader application version 2.0 and ported it to WebSphere. In order to preserve the original .NET StockTrader application as much as possible, the following items were kept without change:

- The database schema
- Static html pages
- Images
- Client side scripting

In addition to those elements which could be preserved without change, other elements were modified only where needed:

- ASPX pages were converted to JSP pages. Forms, scripting and static content was preserved nearly in its entirety, the handling of data objects was the major focus of change.
- Directory structure was kept nearly intact, with only the most minor changes.

The major changes to the application were in the data access layer, and in the implementation of a servlet to handle web requests. The data access layer, although written in Java, implemented all of the interfaces that the original .NET StockTrader data access layer implemented. The major differences in the data access are as follows:

- JDBC Datasources were used.
- The embedded SQL statements used SQL valid only with MSSQL Server and needed to be changed to allow them to work with DB2. In doing so, we were able to exploit certain capabilities of DB2.

Setup

For the lab study the applications were deployed on a front-end server, while the databases were deployed on a separate back-end server. For WebSphere, the application was deployed and tested on WebSphere version 7. The backend database used was DB2 8.2. Microsoft’s .NET StockTrader was deployed on .NET 3.5 with IIS 7.0. The backend database was SQL Server 2005. For both applications, the frontend server operating system was the 32 bit version of Windows Server 2008. The backend database servers were deployed on the 32 bit version of Windows Server 2003.

Details on the hardware can be found at the end of this document.
**Workload**

The applications were exercised using scripts which were written for each server to simulate a user login, getting quotes, stock buy, stock sell, viewing of the account portfolio, then a logoff. The user id and the stock symbol were generated randomly for each iteration of the script.

The script was run in stress mode, with no think times.

Each workload started with 1 user, and then an additional user was added every 10 seconds. A maximum of 36 users were used in both cases, which was enough to drive each server to maximum throughput and utilization.

The script validated the data on each returned page, discounting any transaction that produced errors.
WebSphere Results

The chart below shows the results of the workload running on WebSphere 7.

The graph above is from one of the runs using WebSphere version 7. CPU utilization on both application and database servers increases with the number of users and throughput.

No errors were recorded during any of these runs.
.NET Results

The chart below shows the results of the workload running on .NET.

The graph above is from one of the runs executed during testing. It can be seen that the CPU utilization on both application and database servers increases with the number of users and throughput.

It can be seen that after a small number of users logged on to the application, the application started having difficulties and was raising a small number of errors, 2-3 per second. This was typical of all of the .NET StockTrader runs. In the example run above, 34,477 Transactions were attempted during the run. Of them, 413 transactions were aborted due to errors. This resulted in a loss of 1.2% of the business that this application could have handled.
Tuning

Both versions of WebSphere were tuned using best practices documented on the IBM external websites. These included adjusting the following:

- Datasource connection pool sizes
- Web container threads
- SQL statement cache sizes
- JVM Heapsizes

In addition to the above, the following settings were used:

- Parallel garbage collection ( -Xgcthreads8 )
- DataSource custom property currentPackagePath set to NULLIDR1,NULLIDRA
  - This is in reference to using the DB2 REOPT feature and requires binding packages in DB2. It can improve performance of parameterized queries.
  - For more information see the DB2 documentation on REOPT.

Hardware Information

- IBM X366 Server, 4 x 3.66 GHz , 12 GB (Windows Server 2008)
- IBM X445 Server, 8 x 3.00 GHz, 16 GB (Windows Server 2003)
- Hyper threading was enabled for all the CPU’s
Friendly Bank

The purposes of this section is to outline the design details of Friendly Bank, a reference banking application implemented in WebSphere (J2EE) and .NET, and publish the performance results obtained in a comparison study of WebSphere and its competitors.

Customers need to have the most flexibility in determining how to deploy a web application. For performance, geography and security reasons it is sometimes advisable to split the elements that make up a Web application. Thus, Web Servers, Presentation Layers, Business Logic layers and databases must have the ability to be deployed in single servers (single tier), or in a number of separate physical servers each containing one or more of the elements of the Web application (multi-tier).

This study concentrates on the single tier versus multi-tier performance of the application servers. The following diagram depicts the cases tested.

![Diagram](image)

We built an enterprise level reference application in WebSphere and .NET which we use to compare the performance of WebSphere with the competitive platforms. The reference application supports or will support the following enterprise features:

- Server side Enterprise components (EJB 2.0 / Service Components in .NET)
- Distributed two-phase-commit transactions
• Asynchronous Messaging (JMS in WebSphere) (follow on)
• Web Services (follow on)
• Integration with legacy systems (e.g. CICS, follow on)

Friendly Bank
Friendly Bank is the name of a fictitious online bank application that supports the typical banking operations of a Teller and a Customer. The bank operations include:

• Login & Logout
• Deposit/Withdraw/Transfer funds between accounts
• View Transaction History (by Account/Customer)
• View & Update Customer Profile

Status
The current version is a base implementation and supports the above listed banking operations only in Synchronous mode. Additional functionality involving complex transactions for loan/mortgage applications with connectivity to back-end CICS system and support for JMS, Distributed transactions, Web Services will be added with later versions.
Description of the Performance Study

Every performance comparison, no matter how well or how poorly designed, models a very particular architecture, topology, usage pattern and workload. It is imperative to understand the importance of interpreting the results in relevant context.

Any variation of these factors would impact the performance results in the study:

- **Configuration** – The performance of an Application server is very closely tied to the deployment configuration. It varies for each scenario, single-tier or multi-tier, clustered or non-clustered etc. Even using an external web server with a plug-in or the internal web server impacts the results. This document covers single-tier and multi-tier scenarios.

- **Operating system & Hardware Platform** – The results may significantly differ by altering the operating system or the underlying hardware platforms used in the runs.

- **Workload** – The composition of the performance workload, percentage of read only operations compared to write operations will impact the results. All the performance runs in this study used workload with write intensive operations.

For this Friendly Bank performance comparison study, we have clearly outlined the details where needed covering all the above aspects.

**Configurations**

The following configurations were used:

1) **Single-Tier** – The presentation and component layers of the application deployed on a single server.

2) **Multi-Tier** - The presentation and component layers of the application deployed separately on two different servers.

For WebSphere we used the Internal Port to drive the workload.

**Workload**

None of the scenarios used caching. All cases were run on Intel Servers.

**Application** – The Friendly Bank performance comparison was run in the Synchronous-EJB mode with 50,000 customers and 150,000 accounts to measure the end-to-end application level performance and scalability. The test script imposed *database intensive update* operations doing the following banking operations:

- Login (*)
- Account Summary
- Deposit (*)
- Withdraw (*)
- Logout (*)
All (*) operations indicate database updates.

The workload as it is listed here is update-intensive and hence limits the impact of any caching. In all the performance runs caching was not enabled at any level in all Application servers. An update intensive stress load, as we run here, allows for not only comparing performance, but also the ability of the underlying application server to sustain the load, and thus assure quality of delivery.

Measurements

- The test script validated every response page by checking for valid user/request specific results and application errors. The user id and user names were saved at login time and verified to make sure that each returned screen was for that valid user.
- A transaction, as referred by the performance tool and used in this report is defined as completing all the steps of the test script. In the Friendly Bank workload outlined above this includes all the five operations, if any one of them fails, the entire transaction is not counted as throughput. Thus the metrics used are not only a measure of throughput, but also one of application quality.
- Transaction is a measure of throughput. Any error at any point in the script causes the request to be terminated and not counted toward the throughput number.
- All measurements were done in stress mode, with no think time, and the script looping continuously for the duration of the test with an objective to stress the CPU to highest possible utilization.
- Runs were started with one user, and then the next user added after 10 seconds and so on up to a number where the throughput would not go up any more. The best run of a group was chosen for all cases.
- To ensure that the application servers were running at their full capacity, CPU, disk and network utilizations were also monitored for all servers within a given run. In addition, certain database statistics (locks, lock escalations, etc.) were also tracked to eliminate concerns over database utilization.

Tuning and Configuration

For J2EE case measurement:

- An IBM DB2 database is used. The database was initially tuned using the DB2 wizards and advisors. Additional tuning and configuration was then done to further minimize CPU and disk utilization on the database machine. This primarily involved configuring DB2 to load the entire database into memory, and separating the data and log files onto separate disk drives.
- The application server configuration and tuning centered on configuring the datasource connection pool and setting JVM options for heap size and garbage collection.
- The JVM size was set to 3GB and GC policy was ‘OPTAVGPAUSE’.

For .NET case measurement:

- The .NET application server used a MS SQL Server database and built-in wizards were used to tune this database.
• The .NET application server also has minimal tuning capabilities. We configured three worker processes for the WEB Tier. In the split tier configuration we had five application pools configured on the server hosting serviced components.
Performance Results

The following cases were tested in the Non-Clustered scenarios:

1. Single Tier with both Web and Enterprise Components on same system
2. Multi-tier with Web and Enterprise Components on different systems

General Competitive Observations:

- WAS 7 outperformed Microsoft’s .Net 3.5 in both single server and split tier. In the single tier case WAS achieved 2.8 times the performance of Microsoft. In the split tier case WAS performed 4.5 times better than Microsoft.

- The application tested was originally developed for the .NET Framework version 1.1. It was recompiled for the version 3.5 Framework but is not using any of the new framework features.

- Microsoft performed degraded on splitting tiers. In the Split tier case the best performance we got was when 3 Web App Pools were configured from the front end and 5 App Pools for the Serviced Components at the back end.
APPENDIX A - Application Design Details

The following sections outline the design details of WebSphere and .NET implementation of the reference application, Friendly Bank.

Database Schema

The diagram below shows the tables defined and the relationships between them in the database schema.

<table>
<thead>
<tr>
<th>Table Type</th>
<th>Tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity</td>
<td>Customer, Account</td>
</tr>
<tr>
<td>Relationship</td>
<td>CustomerAccs, History</td>
</tr>
<tr>
<td>Reference</td>
<td>AccType, CustomerType, TransType, TransStatusType</td>
</tr>
</tbody>
</table>
WebSphere (J2EE) Implementation

- **MVC Pattern**
  Friendly Bank application is architected based on the Model-View-Controller design pattern. The JSP pages act as view components. The Controller, `CTLBankServlet` parses the requests and invokes the relevant handler code. The results encapsulated as Java bean is forwarded to the appropriate JSP for display.

- **EJB 2.0**
  Friendly Bank implements the EJB 2.0 specification. All the entity EJB’s implement the local interface.

  The application uses stateless session beans and CMP beans with Container Managed Relationships (CMR). It implements the session façade pattern where the session bean supports the business methods by invoking entity bean methods.

  The Session bean `CTLBankSession` implements a remote interface to allow for deploying the Web components and EJB components on different tiers.
EJB Class Diagram (This diagram is only visible if viewing the softcopy).
.NET Implementation

- **WebForms, WebUserControls, and Serviced Components**
  Friendly Bank application is architected using Webforms and User Controls for the front-end, user interface and display portion, and Serviced Components for the backend.

- **Webforms and WebUserControls**
  Webforms and WebUserControls are used to format and display end user data. Code-behind files encapsulate any additional logic necessary for data formatting and to invoke the back-end components.

- **Serviced Components**
  The business methods of the application are contained within a set of Serviced Components. Serviced Components allow for distributed transactions, easier management through the Component Services MMC and access via .NET Remoting and DCOM protocols.

- **Protocols**
  Communication between the User Interface section and the back-end components is configurable. .NET Remoting using Server Activated Objects and DCOM are both supported.

  The .NET remote objects were hosted in IIS, because it is believed that currently this is the most common deployment of Remotable objects because of ease of development, deployment and management Single Call Server Activated Objects were chosen because documentation was found that indicated that Client Activated Objects could not be reliably hosted in IIS, and Singletons were suspected to be a performance bottleneck.

.NET Model
Appendix B – IBM Results

Single Server

![Graph showing performance metrics for single server measurements.](image-url)

<table>
<thead>
<tr>
<th>Group</th>
<th>Name</th>
<th>Visible</th>
<th>Values</th>
<th>Type</th>
<th>Factor</th>
<th>Source</th>
<th>Unit</th>
<th>Count</th>
<th>Min</th>
<th>Avg</th>
<th>Max</th>
<th>StdDev</th>
<th>Interval</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary G.</td>
<td></td>
<td>Yes</td>
<td></td>
<td></td>
<td>Auto</td>
<td></td>
<td></td>
<td>100</td>
<td>50</td>
<td>9400</td>
<td>65000</td>
<td>0.000</td>
<td>10</td>
<td>m6-8S4v16G16S4Nd16Gfl1(C-38)ProachVxas</td>
</tr>
<tr>
<td>CustomProcessor_T.</td>
<td></td>
<td>Yes</td>
<td></td>
<td></td>
<td>Auto</td>
<td></td>
<td></td>
<td>100</td>
<td>50</td>
<td>11000</td>
<td>18000</td>
<td>0.000</td>
<td>10</td>
<td>m6-8S4v16G16S4Nd16Gfl1(C-38)ProachVxas</td>
</tr>
<tr>
<td>ReadTime</td>
<td>nStBf</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Auto</td>
<td></td>
<td></td>
<td>100</td>
<td>50</td>
<td>9416</td>
<td>8235</td>
<td>95.87</td>
<td>25965</td>
<td>10</td>
</tr>
<tr>
<td>ReadTime</td>
<td>nStCf</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Auto</td>
<td></td>
<td></td>
<td>100</td>
<td>50</td>
<td>8235</td>
<td>95.87</td>
<td>25965</td>
<td>10</td>
<td>m6-8S4v16G16S4Nd16Gfl1(C-38)ProachVxas</td>
</tr>
<tr>
<td>WriteTime</td>
<td>nStBf</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Auto</td>
<td></td>
<td></td>
<td>100</td>
<td>50</td>
<td>8235</td>
<td>95.87</td>
<td>25965</td>
<td>10</td>
<td>m6-8S4v16G16S4Nd16Gfl1(C-38)ProachVxas</td>
</tr>
<tr>
<td>WriteTime</td>
<td>nStCf</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Auto</td>
<td></td>
<td></td>
<td>100</td>
<td>50</td>
<td>8235</td>
<td>95.87</td>
<td>25965</td>
<td>10</td>
<td>m6-8S4v16G16S4Nd16Gfl1(C-38)ProachVxas</td>
</tr>
</tbody>
</table>
Separate Tiers:
APPENDIX C - Detailed Results Microsoft .NET 3.5

Single Server:

![Graph of detailed results for Microsoft .NET 3.5 single server performance metrics.](image-url)
Separate Tiers:
APPENDIX D – Configuration Details:

Hardware Configurations

**Performance Testing Tool Hardware**
X345 8676 Server
2 X 3.06 GHz Intel Processor with Hyper Thread Technology
8 GB RAM
18.2 GB 15K rpm SCSC Hard Disk Drive
1 GB Ethernet interface

**Application Server Hardware**
IBM X3950 Server, 8 x 3.50 Ghz, Intel Xeon Processors with Hyper Thread Technology, 64 GB RAM

**Database Server Hardware**
X445 8670 Server, 8x 3.0 Ghz. Intel Xeon Processors with Hyper Thread Technology, 16 GB RAM
UltraSCSI 320 Controller, EXP 300 SCSI Expansion Unit, 14x 18.2 GB 15K rpm Hard Disk Drive configured as 2 Raid Arrays.
One for Logs & One for Database, Each array is comprised of 7 hard disks in a Raid 0 configuration.

**The Ethernet Network Backbone**
The isolated network hardware is comprised of 3x 3Comm SuperStack 4950 switches and one 3 Comm SuperStack 4924 switch running at 1 GB.
Software Configurations

Application Server Core Software (*)
Windows Server 2008 Data Edition (64bit)
ASP.NET Framework Version v3.5 (64bit)
IBM WebSphere Application Server 7.0 (64bit)

Database Server Software (*)
Windows Server 2003
DB2 UDB EE 8.2.5
Microsoft SQL Server 2005
ServerRAID Manager 5.11.05

(*) Only the applicable server is running, the others are stopped
Physical Diagram:

Single Tier

- App Server
- Int HTTP Server
- Presentation and Component

- 8 way
- 3.5 Ghz
- xSeries

- Database

- 8 way
- 3Ghz
- xSeries

Multiple Tiers

- App Server
- Int HTTP Server
- Presentation and Component

- 8 way
- 3.5 GHz
- xSeries

- App Server
- Component

- 8 way
- 3.5 Ghz
- xSeries

- Database

- 8 way
- 3Ghz
- xSeries