.NET 3.0 vs. IBM WebSphere 6.1 Benchmark Results

Microsoft .NET StockTrader and IBM WebSphere Trade 6.1 Benchmark

Introduction

This paper is a summary of extensive benchmark testing of two functionally and behaviorally equivalent performance-driven sample applications: IBM WebSphere Trade 6.1 (developed by IBM) and the Microsoft .NET StockTrader (developed by Microsoft). While the paper specifically discusses the benchmark testing results, readers should also be aware that .NET StockTrader can be seamlessly integrated with the J2EE-based Trade 6.1 application, providing bi-directional interoperability via Web Services with no code changes required to either application. The IBM Trade 6.1 front-end JSP application can seamlessly utilize the Windows Communication Foundation (WCF) C# services exposed by the .NET middle tier; and the ASP.NET Web client and the Windows Presentation Foundation (WPF) smart client can both seamlessly utilize the J2EE/Java middle-tier services within IBM WebSphere 6.1.

Background

IBM's Trade 6.1 performance application is a J2EE-based application developed entirely by IBM as a benchmark application, best-practice performance sample, and capacity testing tool for IBM WebSphere 6.1. The application is available for free download from the IBM WebSphere performance site, and is used extensively by IBM throughout many of the IBM enterprise Redbooks for WebSphere. Since the application was designed specifically as a performance-driven application by IBM for WebSphere, it presents a good opportunity to compare the performance of IBM WebSphere to the performance of .NET 3.0/Windows Server 2003 running an equivalent application server workload.

Towards that end, Microsoft migrated the IBM application to .NET using best-practice performance practices for the .NET platform, and performed extensive benchmark testing and comparative analysis on Windows Server 2003 and Red Hat Linux. The complete testing details, tuning parameters for all platforms and configurations compared are detailed in the paper ***.NET 3.0/Windows Communication Foundation and IBM WebSphere 6.1 Service-Oriented Performance and Scalability Benchmark,*** available for download at http://msdn.microsoft.com/stocktrader. Full source code and Visual Studio solutions for the .NET StockTrader benchmark application are also available at this site.

Trade 6.1 and .NET StockTrader Configurations

Hardware Configurations

Complete hardware configurations are detailed in the detailed benchmark paper referenced above. For every test, the hardware used for Trade 6.1 and .NET StockTrader is the exact same hardware. The primary test server is a 4-way AMD Opteron system with 16GB RAM and 1.8 GHz processors. For the Web Service tests, additionally four 2-proc blades (2.2 Ghz Intel XEONs w/ 4 GB RAM) were used as the Web Service clients. The backend database for all tests was a 64-bit dual-core (4 CPUs total) AMD Opteron server with 16GB RAM and 2.2 GHz CPUs.

Data Access Technology

IBM WebSphere Trade 6.1 has two modes of operation that equate to two ways to access the backend DB/2 database: EJB mode using a stateless session bean front-ending entity beans using container-managed persistence (CMP); and a JDBC-direct mode, with JSP/Servlets driving data access without the use of entity beans. The test results include performance data for both of these Trade 6.1 configurations. For .NET StockTrader, database access is always performed using ADO.NET and a data access layer, as entity-beans and CMP are a Java-only construct. The EJB/entity bean design remains IBM's recommended design pattern for enterprise WebSphere applications.

The Trade 6.1 application is tested as an 'all-IBM' configuration with DB/2 V9 as the backend database; the .NET StockTrader is tested as an 'all-Microsoft' configuration using SQL Server 2005 Enterprise edition as the backend database.

Web Service Modes

IBM Trade 6.1 uses the latest IBM WebSphere 6.1 Web Services SOAP engine, based on Apache/Axis. Microsoft WCF, a core component of .NET 3.0, offers a variety of different configurations for service hosting, and several different Web service configurations were compared. WCF consolidates all remoting technologies (ASMX, WSE, SOAP Web Services, WS-*, .NET binary remoting) into a single new service-oriented programming model and runtime, based on open industry standards. While the WCF modes tested are not comprehensive, they do provide a good picture of relative performance available with IIS-hosted services, self-hosted services, and both HTTP/Text-XML encoding and TCP/Binary encoding using the pluggable binding architecture of WCF.

Test Software and Script Flow

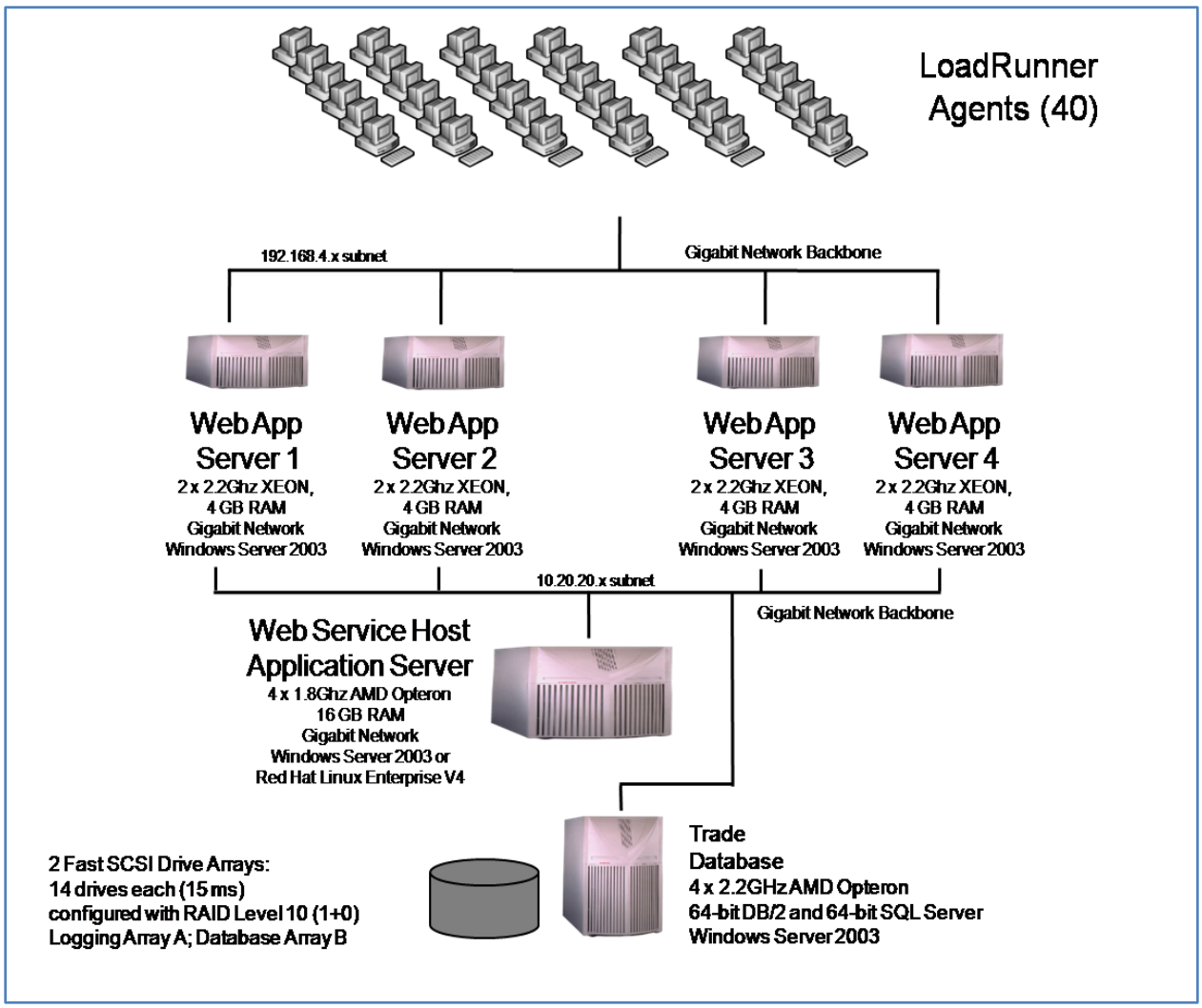
For all tests, Mercury LoadRunner was used to generate load, with a one-second think time between requests. Clients simulate real users logging into the application, with client connections closed between iterations. 40 distributed client machines were used to generate high concurrent loads, and each platform was tuned to achieve peak throughput and full server CPU saturation. Once peak throughput concurrent user loads were determined based on iterative testing and tuning of each configuration, Mercury LoadRunner captured and average sustained peak TPS over a 30 minute test interval after a 15 minute warmup period.

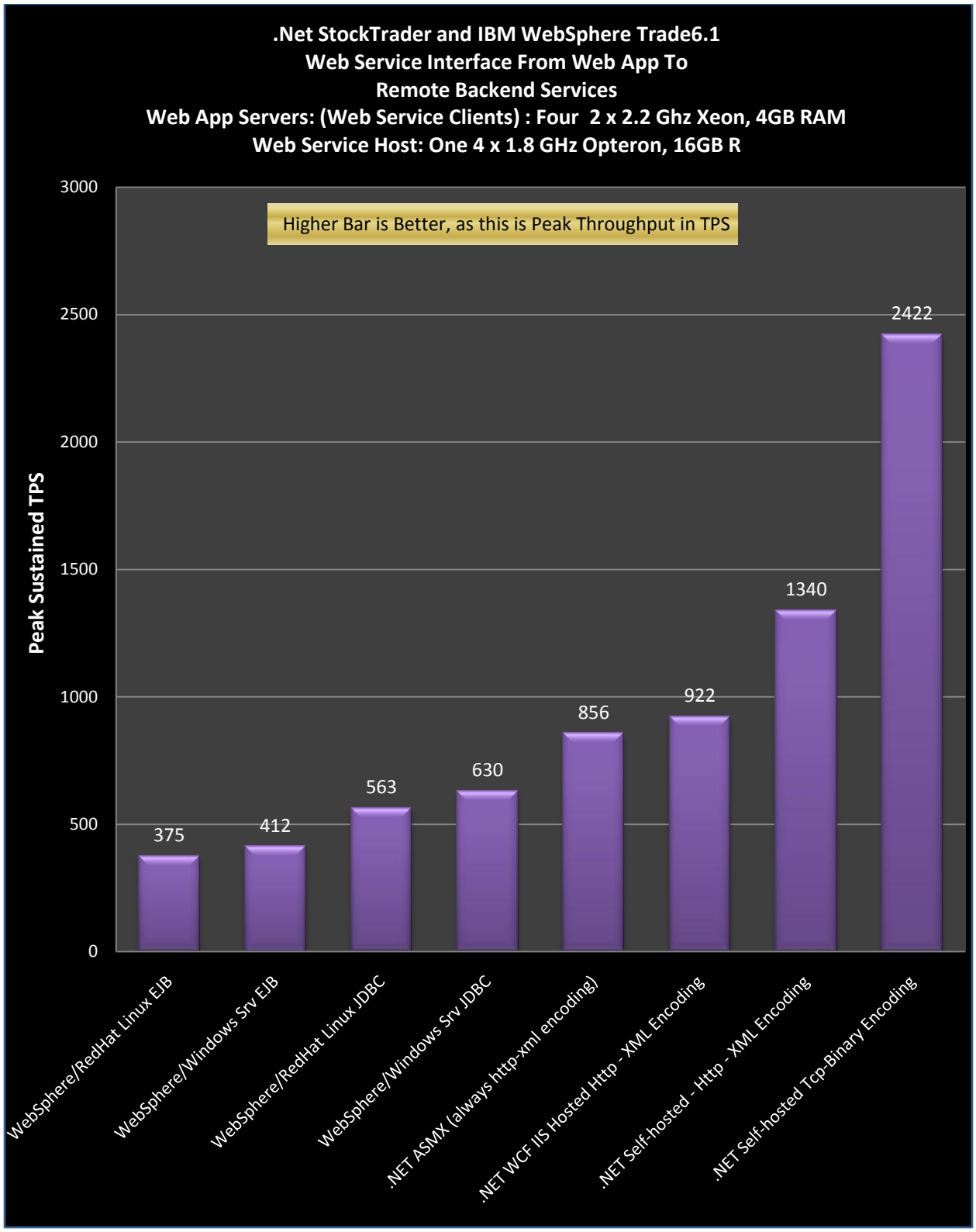
The test script defined the following operations and flow of the simulated users for all configurations tested:

- Login random registered user (1 to 500,000 users loaded in database; the login includes in both apps a redirect to the Home page, and hence all the logic to login and display the Home page/market summary)
- Request four random quotes (1 to 100,000 distinct quotes loaded in database; one html form post performed with 4 stocks requested)
- Request four random quotes (1 to 100,000 distinct quotes loaded in database; one post performed with 4 stocks requested)
- Visit Portfolio page
- Visit View Account page
- Visit Home page
- Logout the registered user via logout URL
- Register a new user/submit registration form (this also logs new user in with redirect/display of Home page)
- Visit Portfolio page
- Buy a random stock symbol (1 to 100,000 stock symbols in database; buy operation involves a direct post/submit to the order submission pages, which submit the order for all backend processing)
- Visit Home page
- Buy a random stock
- Visit Account page
- Get quotes for 4 random stocks (one post performed with 4 stocks requested)
- Buy a random stock
- Buy a random stock
- Visit Portfolio page
- Visit Home page
- Logout

Web Service Benchmark Results

This configuration uses four Web Service client machines hosting the JSP and ASP.NET front end applications, each configured to make requests to the single 4-Proc Web Service host machine. The test is designed to provide enough front-end capacity (4 x 2-proc Web application servers) to ensure the system under test (limiting resource) remains the 4-proc Web Service host application server. In this configuration, WebSphere 6.1 Web service performance across a realistic workload is compared to ASMX 2.0, and the various WCF/.NET 3.0 configurations supported by .NET StockTrader middle tier business services.


LoadRunner
Agents (40)
192.168.4.x subnet
Gigabit Network Backbone
Web App
Server 1
2 x 2.2Ghz XEON,
4 GB RAM
Gigabit Network
Windows Server 2003
Web App
Server 2
2 x 2.2Ghz XEON,
4 GB RAM
Gigabit Network
Windows Server 2003
Web App
Server 3
2 x 2.2Ghz XEON,
4 GB RAM
Gigabit Network
Windows Server 2003
Web App
Server 4
2 x 2.2Ghz XEON,
4 GB RAM
Gigabit Network
Windows Server 2003
10.20.20.x subnet
Gigabit Network Backbone
Web Service Host
Application Server
4 x 1.8Ghz AMD Opteron
16 GB RAM
Gigabit Network
Windows Server 2003 or
Red Hat Linux Enterprise V4
2 Fast SCSI Drive Arrays:
14 drives each (15 ms)
configured with RAID Level 10 (1+0)
Logging Array A; Database Array B
Trade
Database
4 x 2.2GHz AMD Opteron
64-bit DB/2 and 64-bit SQL Server
Windows Server 2003


.Net StockTrader and IBM WebSphere Trade6.1
Web Service Interface From Web App To
Remote Backend Services
Web App Servers: (Web Service Clients) : Four 2 x 2.2 Ghz Xeon, 4GB RAM
Web Service Host: One 4 x 1.8 GHz Opteron, 16GB R
Higher Bar is Better, as this is Peak Throughput in TPS
Peak Sustained TPS
3000
2500
2000
1500
1000
500
0
375
412
563
630
856
922
1340
2422
WebSphere/RedHat Linux EJB
WebSphere/Windows Srv EJB
WebSphere/RedHat Linux JDBC
WebSphere/Windows Srv JDBC
.NET ASMX (always http-xml encoding)
.NET WCF IIS Hosted Http - XML Encoding
.NET Self-hosted - Http - XML Encoding
.NET Self-hosted Tcp-Binary Encoding

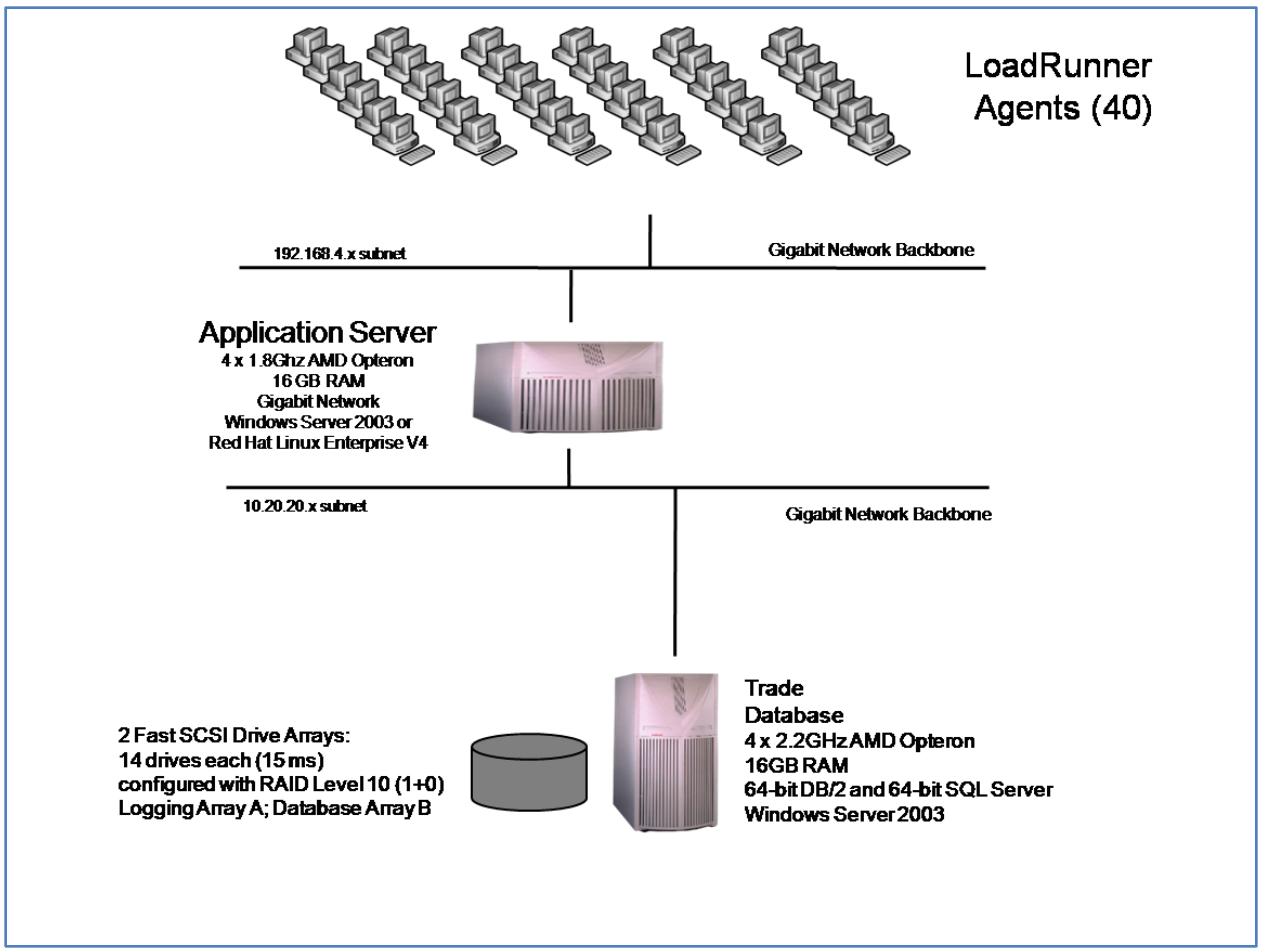
Summary

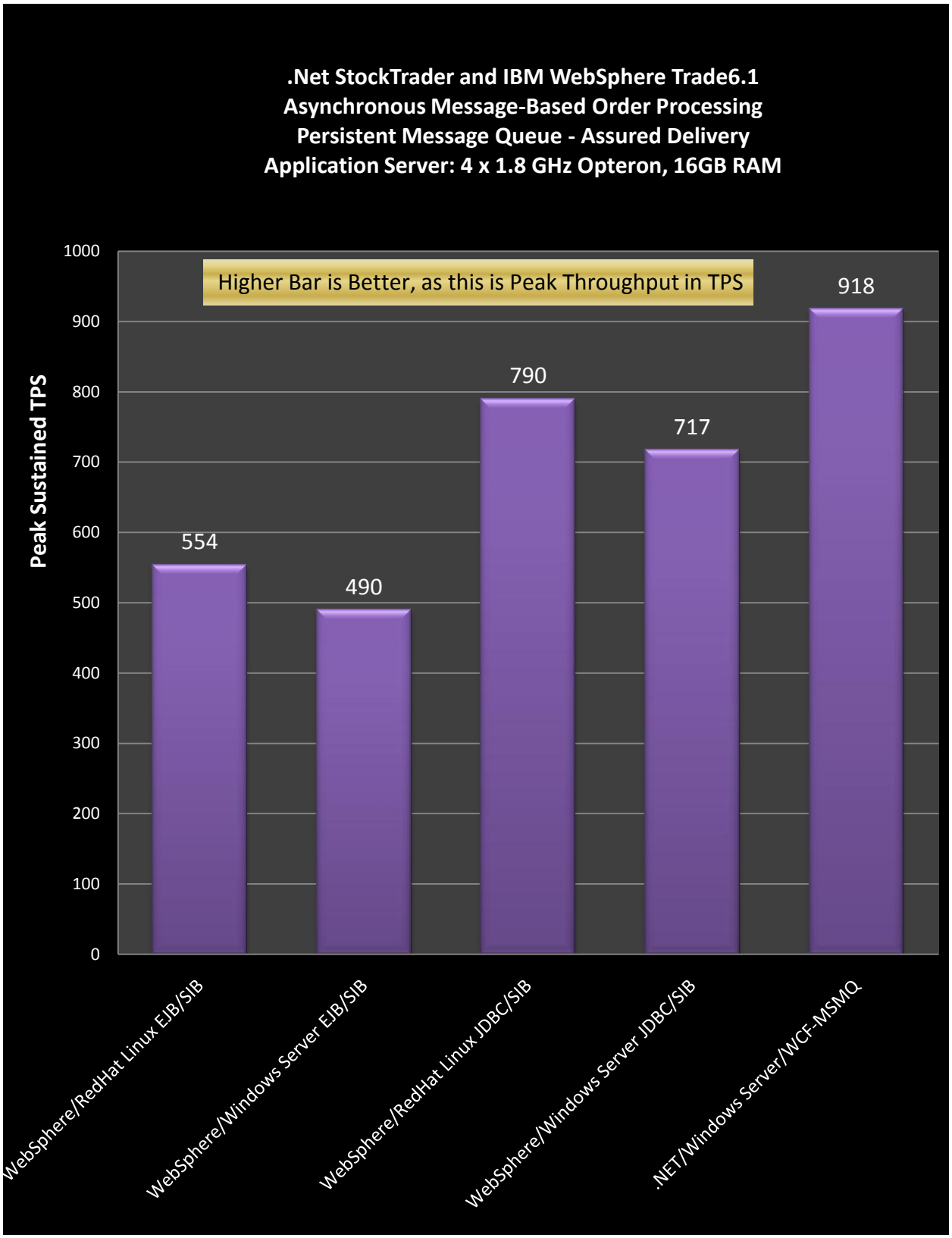
- .NET 3.0 hosted on IIS with an Http binding and XML encoding offers **124% better throughput** than the fastest WebSphere/EJB Web Service implementation tested; and 46% better throughput than the JDBC (no entity beans) WebSphere implementation tested.
- .NET 3.0 self-hosted over Http/XML **offers 225% better throughput** than the fastest WebSphere/EJB Web Service implementation tested; and 113% better throughput than the JDBC WebSphere implementation tested.
- .NET 3.0 with binary encoding over a TCP binding **offers 488% better throughput** than the fastest WebSphere EJB Web service implementation tested; and 284% better throughput than the JDBC WebSphere implementation tested. Using Tcp/binary encoding with WCF is an interesting option, as (per the design of StockTrader) the service host can listen on both http/xml and Tcp/binary endpoints simultaneously, still supporting any platform as a client with no extra programming required for the service. Significant performance gains are possible with .NET 3.0 and binary encoding, with the same Web Service programming model utilized. The WCF Tcp/Binary binding functionally replaces the previous .NET 2.0 binary remoting technology.

Asynchronous Order Processing/Assured Message Delivery Benchmark Results

This configuration uses a single server hosting both the Web application and the asynchronous order processing services. For Trade 6.1, asynchronous order mode utilized JMS/MDBs and an IBM Service Integration Bus (SIB) durable message queue, with a two-phase commit distributed transaction between the message queue and the DB/2 database. This configuration tests a basic IBM “Enterprise Service Bus” configuration, although other remote configurations are possible.

On the .NET side, the equivalent functionality for assured message delivery is accomplished using a WCF service-oriented design with an MSMQ binding to a transacted (durable) MSMQ, and a two-phase commit performed by Microsoft Transaction Coordinator (MS DTC) across the queue and the database. This configuration tests a basic loosely-coupled message-oriented architecture for .NET StockTrader, although other remote configurations with replicated MSMQ messaging engines are possible with .NET StockTrader. Again, the exact same hardware and hardware configuration is compared.


LoadRunner
Agents (40)
192.168.4.x subnet
Gigabit Network Backbone
Application Server
4 x 1.8Ghz AMD Opteron
16 GB RAM
Gigabit Network
Windows Server 2003 or
Red Hat Linux Enterprise V4
10.20.20.x subnet
Gigabit Network Backbone
2 Fast SCSI Drive Arrays:
14 drives each (15 ms)
configured with RAID Level 10 (1+0)
Logging Array A; Database Array B
Trade
Database
4 x 2.2GHz AMD Opteron
16GB RAM
64-bit DB/2 and 64-bit SQL Server
Windows Server 2003


.Net StockTrader and IBM WebSphere Trade6.1
Asynchronous Message-Based Order Processing
Persistent Message Queue - Assured Delivery
Application Server: 4 x 1.8 GHz Opteron, 16GB RAM
Higher Bar is Better, as this is Peak Throughput in TPS
Peak Sustained TPS
1000
900
800
700
600
500
400
300
200
100
0
554
490
790
717
918
WebSphere/RedHat Linux EJB/SIB
WebSphere/Windows Server EJB/SIB
WebSphere/RedHat Linux JDBC/SIB
WebSphere/Windows Server JDBC/SIB
.NET/Windows Server/WCF-MSMQ

Summary

WCF over a durable MSMQ message **queue offers 67% better throughput** than the IBM WebSphere JMS/SIB message queue configuration using EJB/entity beans; and 16.2% better throughput than the JMS/SIB message queue configuration using JDBC.

More Information

The complete benchmark paper with all tuning details and results for other configurations tested is available at http://msdn.microsoft.com/stocktrader, along with downloadable code for the .NET StockTrader service-oriented application so that customers can perform their own testing and capacity comparisons on the hardware of their choosing.