Windows Server 2008/.NET Framework 3.5 and IBM WebSphere 6.1 Service-Oriented Performance and Scalability Benchmark

.NET StockTrader vs. IBM WebSphere Trade 6.1 Benchmark Results for Transactions, Web Services, and Messaging Workloads

2/24/2008

Microsoft .netFramework

This document supports the release of Windows Server® 2008 and the Microsoft .NET Framework 3.5.

The information contained in this document represents the current view of Microsoft Corp. on the issues disclosed as of the date of publication. Because Microsoft must respond to changing market conditions, this document should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented. This document is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Microsoft grants you the right to reproduce this guide, in whole or in part.

Microsoft may have patents, patent applications, trademarks, copyrights or other intellectual property rights covering subject matter in this document, except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights or other intellectual property.

Microsoft, Windows Server, the Windows logo, Windows, Active Directory, Windows Vista, Visual Studio, Internet Explorer, Windows Server System, Windows NT, Windows Mobile, Windows Media, Win32, WinFX, Windows PowerShell, Hyper-V, and MSDN are trademarks of the Microsoft group of companies.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Contents

Introduction

This paper presents detailed benchmark results based on extensive performance and scalability testing of 64-bit IBM WebSphere 6.1.0.13 Network Deployment Edition running on 64-bit Red Hat Linux Advanced Platform 5, and 64-bit Windows Server 2008/64-bit NET 3.5 with the Windows Communication Foundation (WCF). The benchmark focuses on three core workloads:

- Web Services
- Message-oriented transaction processing (via message queuing)
- Data-driven Web application with middle-tier transaction services and data access

The benchmark tests focus on comparing an end-to-end solution based on the IBM Trade 6.1 performance application. Trade 6.1 is a J2EE-based application developed by IBM as a best-practice performance sample application 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 most of their core enterprise Redbooks for WebSphere. Since the application was designed specifically as a performance-driven application, it presents a good opportunity to compare the performance of IBM WebSphere to the performance of .NET 3.5/Windows Server 2008 running an application server workload.

.NET StockTrader Sample Application and Performance Kit

For the benchmark comparison, Microsoft created an application that is precisely functionally equivalent to the Trade 6.1 application, both in terms of user functionality and middle-tier database access, transactional and messaging behavior. This application was created using best-practice programming techniques for .NET and the Microsoft Application Development platform. The resulting application, the ***.NET StockTrader***, is now published on MSDN as a best-practice .NET enterprise application. The .NET StockTrader is a service-oriented application that utilizes Windows Communication Foundation (WCF) for its underlying remoting and messaging architecture. The user interface is an ASP.NET/C# Web application that is equivalent to the Trade 6.1 Java Server Pages (JSP) application. Additionally, the middle-tier services, written in C#, mirror the functionality and transactional characteristics of the backend Trade 6.1 services which are based on J2EE and the IBM WebSphere application server.

Multiple Clients with Open Integration to Middle Tier via WCF

As a service-oriented application based on WCF, multiple interoperability scenarios are enabled with the .NET StockTrader. Since both the J2EE Trade 6.1 and .NET StockTrader applications expose their middle-tier services as industry-standard Web Services, the two applications can be seamlessly integrated with no code changes required. The JSP Trade 6.1 front-end application can fully utilize the .NET middle tier services and messaging capabilities of the .NET StockTrader; and the ASP.NET StockTrader front-end application can fully utilize the EJB-based WebSphere Trade 6.1 middle tier services and messaging capabilities. This interoperability is possible with the .NET StockTrader since WCF, Microsoft's new component remoting and distributed application programming model, is fundamentally based on open Web Service standards including SOAP, XML

and the latest WS-* industry standards. In addition to the ASP.NET Web based application that integrates via services with the middle tier, the sample also includes a Windows Presentation Foundation (WPF) desktop client (also developed in C#), that provides a smart-client interface to the middle tier. The WPF client can also seamlessly connect to either .NET middle tier services, or J2EE Trade 6.1 middle tier services simply by changing the services URL in the configuration page—no code changes are required to achieve this interoperability.

Full Disclosure Notice

The complete source code, all test scripts and all testing methodology for this benchmark are available online. Any reader may download and view the actual code for all implementations tested, and may further perform the benchmark for themselves to verify the results. The benchmark kit can be downloaded from http://msdn.microsoft.com/stocktrader. Extensive time was spent to generate results that represent optimal tuning for the platforms tested, and we are quite confident in the results. We encourage customers to download each kit and perform their own comparative testing and functional and technical reviews of each application.

Tests Performed and Testing Details

Three core benchmark tests were performed:

1. Web Services: Remote activation of backend services from the front-end Web application.
2. Durable Messaging: Orders placed via transacted/durable queue in loosely-coupled architecture.
3. Web Application: All elements of application run in a single JVM or CLR instance, no Web Services or messaging. In this mode, orders were set to be placed synchronously.

All tests were performed on the same Hewlett Packard DL380 G5 server with dual quad-core Intel EMT-64 CPUs (8 cores total); operating at 2.66 GHz. The machine was configured with dual gigabit NICs and 16GB RAM.

Both the WebSphere Trade 6.1 and .NET StockTrader can easily be configured to run in each of the three modes above. For each result, we report a peak sustained transaction per second (TPS) throughput rate as averaged over a 30 minute measurement period as tracked by Mercury LoadRunner. Distributed LoadRunner agents drive load against the system tested via simulated Web users running the test script across 40 different distributed client test PCs. Users are added to the system until peak throughput is obtained. Extensive iterative benchmark runs (as required) were done prior to measurement runs to ensure proper tuning of the middle tier systems. For each of the four tests above, we report the peak sustained ***TPS rate***, and also a ***calculated dollar cost per TPS*** so customers can better understand what that performance costs in normalized measurement across systems. The cost calculations are based solely on measuring the middle tier application server software costs (all tests are conducted on the exact same hardware setup). Notes and details on the pricing of the middle tier application servers is included in the Appendix, and based on basic published pricing from each vendor.

Customers should understand that full .NET capabilities are included in every edition of Windows Server, and upgraded versions of .NET are made available for free download from MSDN (for example, .NET 3.5 as tested here). Hence, there is no additional or separate application server cost associated with a .NET application; while commercial J2EE application servers such as WebSphere are separately licensed (and typically quite expensive) products. The dollar cost per TPS rates might, therefore, surprise some readers.

It is also important to remember that this is a test of specific workloads based on the Trade 6.1 and equivalent .NET StockTrader application. Each application, created by the respective vendors for their platform, however, utilizes most (if not all) of the commonly deployed architectural building blocks used in almost all enterprise applications.

Fair Benchmark Comparisons Between .NET StockTrader and IBM WebSphere Trade 6.1

Since each application supports many different configurations, it is very important to understand what constitutes a fair comparison. First, the configurations compared must be equivalent. This means that the applications must produce the exact same functionality and processing behavior in the configurations compared. You cannot, for example, compare one application running with one-phase transactions between the message queue and the database, while running the other application with a two-phase commit across these distributed resources. The .NET StockTrader, while based on .NET and not J2EE, was designed to mirror most of the Trade 6.1 configurations possible with this testing goal in mind. The key configuration modes Trade 6.1 and .NET StockTrader support, in any combination, are discussed below.

Database Access Technology/Programming Model:

WebSphere Trade 6.1:

- EJB (default)
- Direct

The EJB mode employs a standard IBM/J2EE recommended development paradigm: JSPs invoke stateless session beans, which in turn front-end Entity Beans that use Container Managed Persistence (CMP). The Direct mode eliminates the Entity Beans and CMP, and instead uses direct JDBC calls to the database. Both modes use Java model classes to pass data information between tiers.

.NET StockTrader:

Microsoft has a single data access strategy based on ADO.NET, so there is no mode that equates to this Trade 6.1-configurable setting in the .NET StockTrader application. The ADO.NET implementation uses C# model classes to pass data between tiers. It uses ADO.NET DataReaders isolated in a separate data access layer (DAL) as a best-practice performance programming practice, and to maintain clean separation of database logic from the other tiers of the application.

Interface from Web Application to Backend Business Services:

WebSphere Trade 6.1:

- Standard (via local invocation of EJB Session Beans from JSP pages, not remote/RMI-based invocations)
- Web Services (JSP pages make remote Web calls to the backend services, which do all database/transaction work)

Trade 6.1 Web Services are based on the IBM Web Services/SOAP implementation, which use an Http transport and XML encoding. These are the only possible transport and encodings supported by the IBM Web Services programming model. In Web Services mode, you can optionally direct the endpoint (via their configuration page) to point to any of the .NET Web Service endpoints for StockTrader (discussed below), and the JSP application will seamlessly work with the .NET middle tier services and data access layers.

.NET StockTrader:

- Http_WebService. In this mode, the ASP.NET application makes calls to a Windows-hosted WCF service called .NET StockTrader Business Services. This is a self-host application—self hosting services is a core new concept introduced with WCF, and allows services to be hosted in any application, not just IIS. In this mode, calls are made from the client over HTTP with XML encoding.
- Tcp_WebService. In this mode, the ASP.NET application makes calls to the same self-host WCF Windows application as with Http_WebService. However, WCF enables this same client and same host to work with different "bindings"—in this case TCP as opposed to Http. So in this mode, the application is using **TCP and binary** encoding between the Web application and Business Services. WCF unifies the programming model for *all* remoting in .NET, and separates transport/encoding standards out from the programming logic. So the self-host Business Services program is actually simultaneously supporting Http/XML and Tcp/Binary modes of operation with no extra programming. The WCF clients work the same way, and different clients can simultaneously use different bindings.
- IISHost_WebService. In this mode, the WCF Business Services are hosted in IIS, as opposed to a self-host program. IIS 6.0 hosted services always use Http and XML encoding. However, this is not a restriction for IIS 7.0. For this benchmark, all testing was done on IIS 6.0 however.

Order Processing Mode

Trade 6.1

- Synchronous (default). In this mode, orders are simply processed as they come in from the Web tier-- either by way of JDBC logic (Direct mode) or EJB/Entity Bean logic (EJB mode).
- Async TwoPhase. In this mode, JMS is used to integrate with an IBM Service Integration Bus (SIB) message queue. In this mode a full two phase distributed transaction takes place when processing orders, so they are not lost if the database transaction fails. Hence, in this mode it

makes sense to configure the message queue for persistent storage and "assured message delivery." The transaction is coordinated by WebSphere JTA transaction facilities.

.NET StockTrader

- Sync_InProcess. This mode equates to the Synchronous mode for IBM Trade 6.1. There is no messaging interface, orders are simply processed by Business Services as they come in from the Web application.
- ASync_Msmq. This mode equates to running Trade 6.1 with the ASync_TwoPhase configuration, with the SIB message queue configured for persistent storage. In this mode, the WCF service is bound to a transacted (durable) MSMQ message queue, and thus presents assured message delivery, since a two phase distributed transaction is always used when processing off the message queue into the database. The transaction is coordinated by the MS Distributed Transaction Coordinator (MS DTC).

Caching

Trade 6.1

- No Caching (default). The name of this mode is a bit misleading. As long as WebSphere is configured for Servlet Caching, the Market Summary will be cached based on a 3 minute cycle, as defined in the WebSphere Cachespec.xml file. All other elements of the application are not cached, database interactions occur on every request. This is necessary, since the Market Summary query is very heavyweight, and with any real data load, would quickly bring the database and the app to a crawl if run on every visit to the home page. Since the query is the same in the .NET StockTrader application, the same is true for .NET StockTrader. For benchmark runs, we tuned to cache to a 1 minute expiration in both cases since user's would likely want market updates a bit more frequently than every three minutes.
- Distributed Map Caching. This uses IBM's Dynamic Cache Service, based on settings in the Cachespec.xml file, to perform a fairly complex series of caching steps for the application. This enables the application to reduce database calls.
- Command Caching. This is provided for backwards compatibility since the Distributed Map Caching replaces Command Caching as IBM's primary cache technology/recommended approach to caching.

.NET StockTrader

The .NET StockTrader uses the .NET cache API (output caching specifically) to cache the Market Summary page for 60 seconds. We chose not to implement further caching in the application because it is simply not realistic. While Trade 6.1 can cache stock quotes, account data, portfolio data and the like (and .NET StockTrader could too if implemented), the simple fact is that this is not a realistic approach or "cache policy" for this application. Consider that the IBM cache, while distributed (it can keep cached items in sync across clustered servers), is not invalidated by the data source itself. Hence, an update to a database table by any other application using the same database would result in possibly corrupt data, or at least presenting

incorrect "stale" data to users in the application. Unless a customer is willing to direct ***all*** database updates/deletes/inserts through the Trade 6.1 entity beans, data on the middle tier would quickly become out of sync with the actual database. In other words, such a strategy keeps the organization from building new applications against a common database. Market Summary information is fine to cache, since its read-only and can stand to be 60 seconds stale; user account balances and stock price information used on trades cannot. Hence, the .NET StockTrader does not implement further caching beyond Market Summary.

It should be noted, however, that Microsoft introduced a new SQL Cache Dependency feature for the .NET cache with .NET 2.0; and unlike IBM's Distributed Cache technology, this is directly invalidated by the data source itself. In other words, a completely separate application (or even a manual update to a single row of data) will cause and application cache that contains that row to be invalidated, such that data will remain in sync with the database itself. While this is a great feature, it is not appropriate for constantly changing data (since the subscriptions generate network traffic). Such constantly changing data should likely not be cached to begin with. Given the nature of the benchmark, almost all data is constantly changing in the benchmark runs, so again, for realism we do not use this .NET feature.

Enable Long Run Support

Trade 6.1

- On (default)
- Off

This setting was introduced to Trade with WebSphere Trade 6.1, apparently because customers running the benchmark for long periods of time saw steady degradation of performance as user accounts started to contain more and more orders. This resulted in heavier queries, and large and steadily increasing amounts of data being passed between tiers and formatted into the account summary page. Hence, IBM created this setting. All it does is completely eliminate the recent order query and order display on the account page. It is understandable that in a benchmark setting, you cannot control the amount of data as you would in a production application (typically, for example, by querying based on order date, etc. for a restricted set of data). However, there are better approaches to solve this problem. For example, capping the maximum orders returned by a given query at a reasonable number (like last 100 orders). EJBSQL, however, does not support this capability, so IBM chose this option instead.

Despite this discussion, there is a much better way to ensure benchmarks can be run for long periods of time using Trade 6.1 and StockTrader---and a solution that makes the benchmark more realistic at the same time. That solution is to use a much larger data load in the database. For example, with 500 users, and a benchmark script creating 100 orders per second, it would take just 5000 seconds (83 minutes) for each visit to the account page to cause a 1000 record SQL query to be run on each request, followed by passing 1000 records over the network for each user on each request. This is the default load IBM uses for the database. We chose simply to increase the default data load, which also makes

the benchmark more realistic---very few enterprise production databases would be so small. We used 500,000 accounts, each with 5 existing orders (2.5 million orders). Now, at 100/orders per second during a benchmark, it would take 58 days to get to 1000 orders per user.

.NET StockTrader

As mentioned, there is no equivalent setting for StockTrader, instead we run the benchmark for both applications against a larger, much more realistic data load (500,000 accounts, 5 order per account, 100,000 quotes). We provide a database loader (written in .NET Windows Forms) to load both SQL Server 2005 and DB2 V9. This loader runs significantly faster than the Trade 6.1 JSP Data Load Page. This loader program will also reset the database between benchmark runs to the same starting state by deleting added data records.

In summary, you should make sure to de-select "Enable Long Run Support" in Trade 6.1 before doing benchmark comparisons involving the account page, since this only disables the account page functionality which is not realistic.

Database Load

As discussed, we used a default load for all benchmark runs (reset between runs) of 500,000 accounts, 5 orders/holdings per account, and 100,000 quotes. This is much more realistic than the IBM default settings.

Database Configuration

The IBM Trade 6.1 application was tested against an all-IBM configuration, using IBM DB2 V9.5 (Enterprise Edition) as the backend database, and the latest IBM DB2 V9.5 JDBC drivers for data access. The .NET StockTrader was tested against an all-Microsoft setup, with a backend SQL Server 2005 database (Enterprise Edition). The benchmark is not a database benchmark: enough capacity was employed for the database hardware to ensure it was not a bottleneck in any benchmark run. Each database was deployed to the same 64-bit Windows Server 2008 machine, running on a dual quad-core Hewlett Packard DL380 G5 @ 2.66 GHz. This server uses Intel EMT-64 processors, with 8 cores total. The database was configured with two fast RAID arrays (10 ms disk access times, 14 drives each in a RAID 10 configuration); logging was directed to one array, the primary database files were stored on the second array. Each array was configured with its own dedicated controller. The 64-bit editions of DB2 and SQL Server were installed. All tuning steps were followed for DB2 according to the Trade 6.1 documentation, however, additional logging space was configured given the larger data load. The equivalent drive space was configured for SQL Server logging. The database disk usage was closely monitored to ensure each run could complete without requiring the database to extend the logging or data file space during a benchmark run. This is an important consideration for custom tests.

Test Scripts

Mercury LoadRunner was used to record test scripts—browser interactions that exercise most of the functionality in the application. These were run across 40 client machines (500 MHz Windows XP desktops with 512 MB RAM). User agents were configured to run with a one second think time between each request. Each benchmark run included a warm up run to get to steady state, and a 30 minute

measurement period. TPS rates were determined by LoadRunner by averaging across the 30 minutes. Error rates were monitored to ensure they remained at less than .01% during the measurement period. Some dropped connections and or database deadlock conditions do result from running very large user loads (great than 1,000 concurrent users at a one second think time) against the applications. Dropped connections, if any, occurred only during the warm-up period as larger user loads were ramped against the applications. User loads were run for each application up to a number that represented peak throughput for that configuration, as determined in many iterative runs (literally hundreds) during the tuning stages. Extensive time was spent tuning IBM WebSphere (see the appendix) to achieve peak throughput for the software/hardware configuration tested. IBM does not publish pre-set tuning guides for Trade 6.1, and developers must iteratively test and tune the various knobs in WebSphere (there are many) to get to an optimal setup for a given hardware configuration and software workload. .NET does not require nearly as much tuning, and in general will scale quite well out of the box, given a properly coded application. Some tuning was applied, however, and this is documented in detail along with the WebSphere and Linux tuning in the appendix.

In general, it is fairly straightforward to recognize an in-properly tuned system for both platforms. Given enough database capacity, each application server running under load should be able to reach full saturation (~100% CPU saturation) when properly tuned, assuming good vertical scaling of the technology/OS across processors, and no external bottlenecks. It is a requirement to iteratively test after adjusting, individually, the core tuning settings for the application, to determine peak throughput and user loads that achieve peak throughput (neither oversaturating or under saturating the middle tier server being tested). With Trade 6.1, these tuning settings are extensive, including several different thread pools, connection pools for databases and queue connection factories, Java heap sizes, and the like. We are quite confident in the results and the system tuning applied, however, should IBM recommend different (specific) settings, we will be happy to re-run the benchmark and re-publish new results. Customers can also run the benchmark—it is a great way to really judge the capacity of the two platforms for a realistic workload, and then to judge the cost of each platform and compare the cost to the results achieved.

Simulated User Settings

Mercury agents were set to not download images (this is not a web server/network I/O benchmark) during runs. They make requests to the application server, and all processing is completed and just the HTML returned to the agent. This reduces the overhead on the agent machines, and helps ensure the 40 client machines used in the testing never become an artificial bottleneck. Just as importantly, the agents were **configured to reset connections between iterations**, to simulate constantly new users logging into the application, and more fully exercise the underlying networking stacks and HTTP keep alive system that the application servers and Web servers use to support large concurrent user bases. Too many benchmarks are simply run with 10-15 threads, no think times and no network resets between script iterations. These types of benchmarks often produce very different (often over-inflated) results than real world usage conditions. Our settings are meant to much more closely mimic the real world. Think times (even if just one second) and connection resets between iterations make all the difference here.

Changes to the IBM Downloadable Version of Trade 6.1 as Used for Testing

We wanted to avoid making changes to IBM's code; after all, it was developed by IBM for their own platform as a best-practice performance application for performance testing and capacity planning. The only setting we changed, therefore, besides ensuring the cachespec.xml file was only caching Market Summary as did the .NET application (as previously discussed), was to ensure the application did not make requests to the Stock Streamer sample Java Client application published with Trade 6.1. This application uses a Topic-based pub/sub mechanism to show a subset of stock trades at periodic intervals in a Java client application. Since we did not implement (at least not yet) this equivalent functionality for .NET StockTrader (we instead did a full blown WPF client); we needed to make sure the Trade 6.1 application was not making JMS calls for each stock trade. This is accomplished easily by merely changing environment entries (PublishQuotePriceChanges) in the deployment descriptors that disable this functionality specifically, or by merely commenting out the call to publish to the JMS topic for the Streamer application. Correct settings can be ensured by turning on Trade 6.1 detailed logging, and ensuring these calls are not being made (which we did). Of course logging should be turned off for the actual benchmark. Note however, that updates to stock prices and volumes remain on (UpdateQuotePriceVolume), as the .NET StockTrader also updates stock prices and trading volumes in the equivalent fashion for each order placed as part of the buy and sell transactions.

Security Settings

Trade 6.1 does not implement any security between the Web application and the JSP engine—for example, no encryption of passwords takes place. We made sure the .NET StockTrader similarly did not implement any extra security, although this would be appropriate for a Web-based deployment. ASP.NET Forms Authentication was used as the authentication mechanism (with anonymous Internet access ala Trade 6.1), and ASP.NET Forms Auth makes it extremely easy to implement any number of encryption algorithms simply via a configuration setting. The application, if ever deployed on the Web vs. an Intranet, would require SSL/HTTPS as the primary security mechanism. Customers can configure IBM HTTP server or IIS for SSL optionally for additional testing if they desire. The Web Services in the application as implemented by IBM are simply SOAP 1.1 based. They do not employ WS-*. Therefore, neither does the 1.0 implementation of StockTrader, but the applications present a possible way for benchmarking of various WS-* standards in the future, such as WS-Atomic transactions and WS-Reliable Messaging which WCF and .NET fully support. It should be noted, however, as the applications do not require federated security, in the real world neither would likely implement WS-Security for an actual deployment, considering the overhead and lack of need for it in this specific application as designed.

IBM HTTP Server vs. Port 9080

When benchmarking IBM WebSphere, it is important to understand that while WebSphere provides an in-process HTTP listener service (port 9080, by default), IBM best practice recommended deployments are in conjunction with the full-blown IBM HTTP Server (a repackaged version of Apache). Hence, for all configurations, we used IBM HTTP Server as packaged with WebSphere, configured with the WebSphere Plugin. This includes the distributed Web Service benchmark runs, where four client application servers execute the JSP Web application (co-located with the IBM HTTP Server); and make requests to the Web Service Host application server. These requests are made on port 80 to the IBM HTTP Server that is

installed on the WebSphere Web Service Host application server. This is the recommended IBM configuration for deploying high-load application servers hosting Web Services. IBM HTTP Server tuning details for Windows and Linux are included in the Appendix.

Web Application Pages Exercised by the Test Scripts

The test scripts were designed to drive load on the system in a way that exercises most functionality in the application, and puts a heavier emphasis on transactions; such as adding new registered users and buying stocks. The precise flow of the test scripts (exactly the same for all test runs on all platforms) is listed below. A one second think time (smaller than real-world, to driving more load per simulated user) was placed between all URL requests, and in the results, a 'transaction' is defined as the successful completion of the URL request to the server, with valid response/HTML returned.

- 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 all the logic to login and display home page)
- Request four random quotes (1 to 100,000 distinct quotes loaded in database; one 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 Account Page (no account update performed)
- Visit home Page
- Logout the Registered user via logout page
- 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

32-Bit versus 64-Bit Testing

Middle Tier

For this benchmark, we tested only 64-bit platforms as follows:

- Windows Server 2008 64-bit
- Red Hat Linux Advanced Platform 5 (x86-x64: there is only one edition, it is the 64-bit OS and also runs 32-bit Linux programs)
- IBM WebSphere ND 6.1.0.13 64-bit for Windows X64 and Linux X64.
- .NET 3.5 64-bit

Database

The database was run on 64-bit Windows Server 2008 running the 64-bit versions of DB2 V9.5 and SQL Server 2005. This computer was configured with 16GB of RAM and two separate SCSI RAID arrays (each with their own controller) in a RAID 10 configuration. Each RAID array consists of 14 separate drives with 10ms access time; one array was used for logging; the other for primary database data storage, for both databases. All tests were performed on the same Hewlett Packard DL380 G5 database server with dual quad-core Intel EMT-64 CPUs (8 cores total); operating at 2.66 GHz. The machine was configured with a gigabit NIC and 16GB RAM.

As published benchmark kits, the two applications do present the chance for customers to run their own tests, which we highly encourage.

Benchmark Results

Web Service Benchmark

With Trade 6.1 in Default EJB Mode (Entity Beans/CMP Data Access)

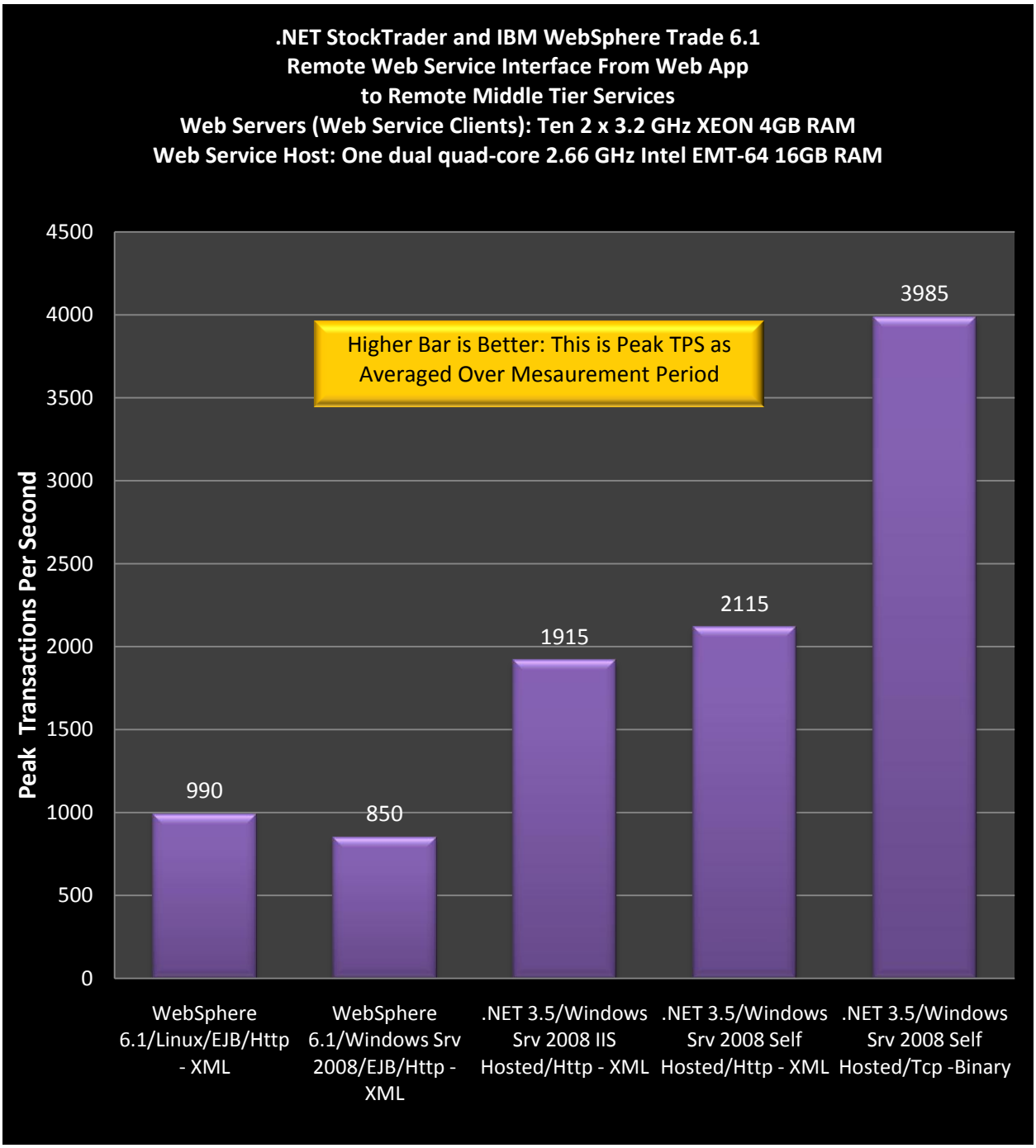

Figure 1: Peak TPS Rates for the Web Service Test. IBM WebSphere Trade 6.1 is running in its default EJB/Entity Beans with Container Managed Persistence (CMP).

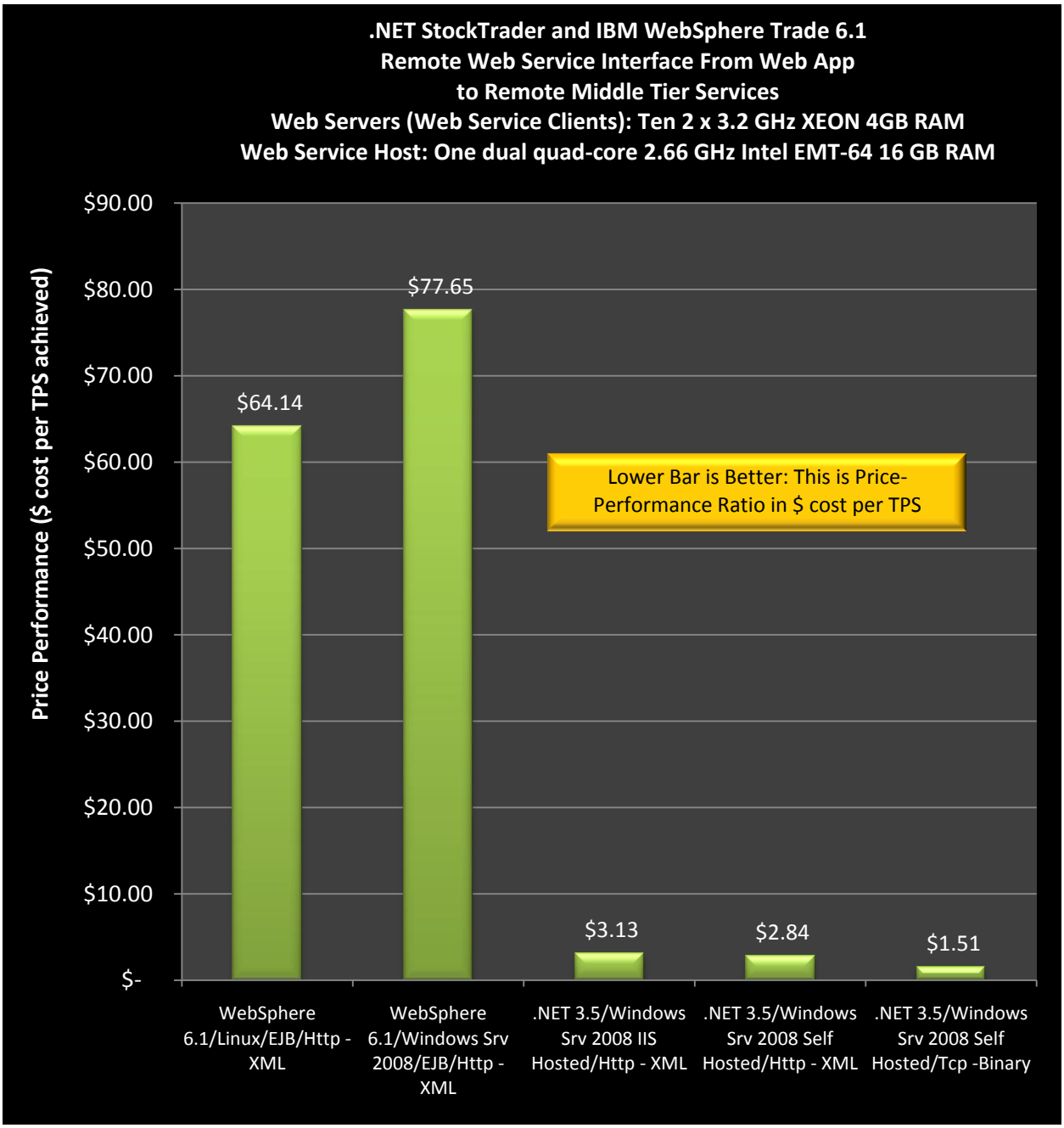

Figure 2: Price Performance Ratio for the Web Service Test. IBM WebSphere Trade 6.1 is running in its default EJB/Entity Beans with Container Managed Persistence (CMP). Lower bar indicates better value for performance delivered. See appendix for pricing calculations, which includes the software cost of the middle tier OS and application server as tested.

With Trade 6.1 in JDBC Direct Data Access Mode (No Entity Beans or CMP)

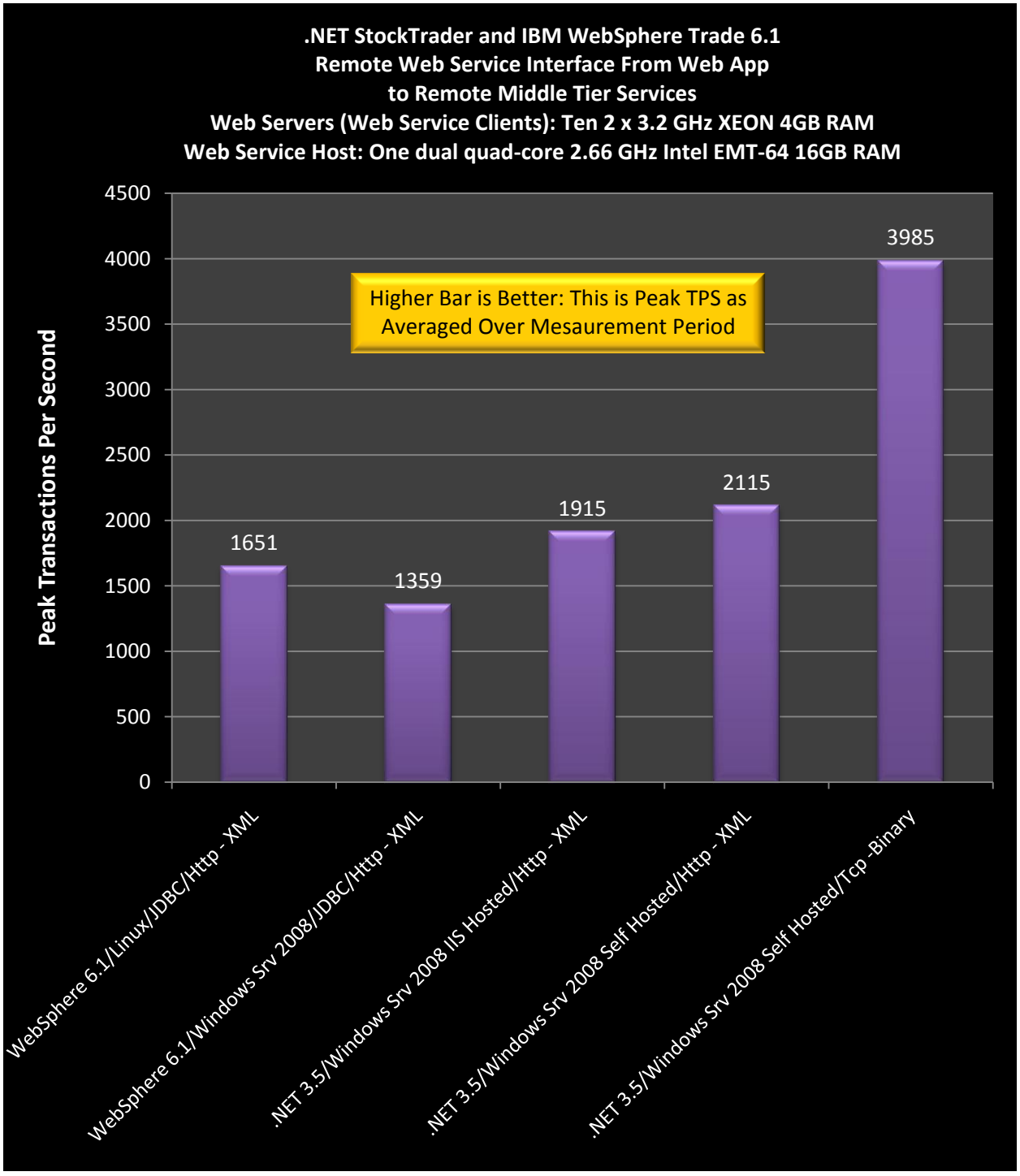

Figure 3: Peak TPS Rates for the Web Service Test. IBM WebSphere Trade 6.1 is running in JDBC mode with no Entity Beans, instead using JDBC API with SQL statements to access the database.

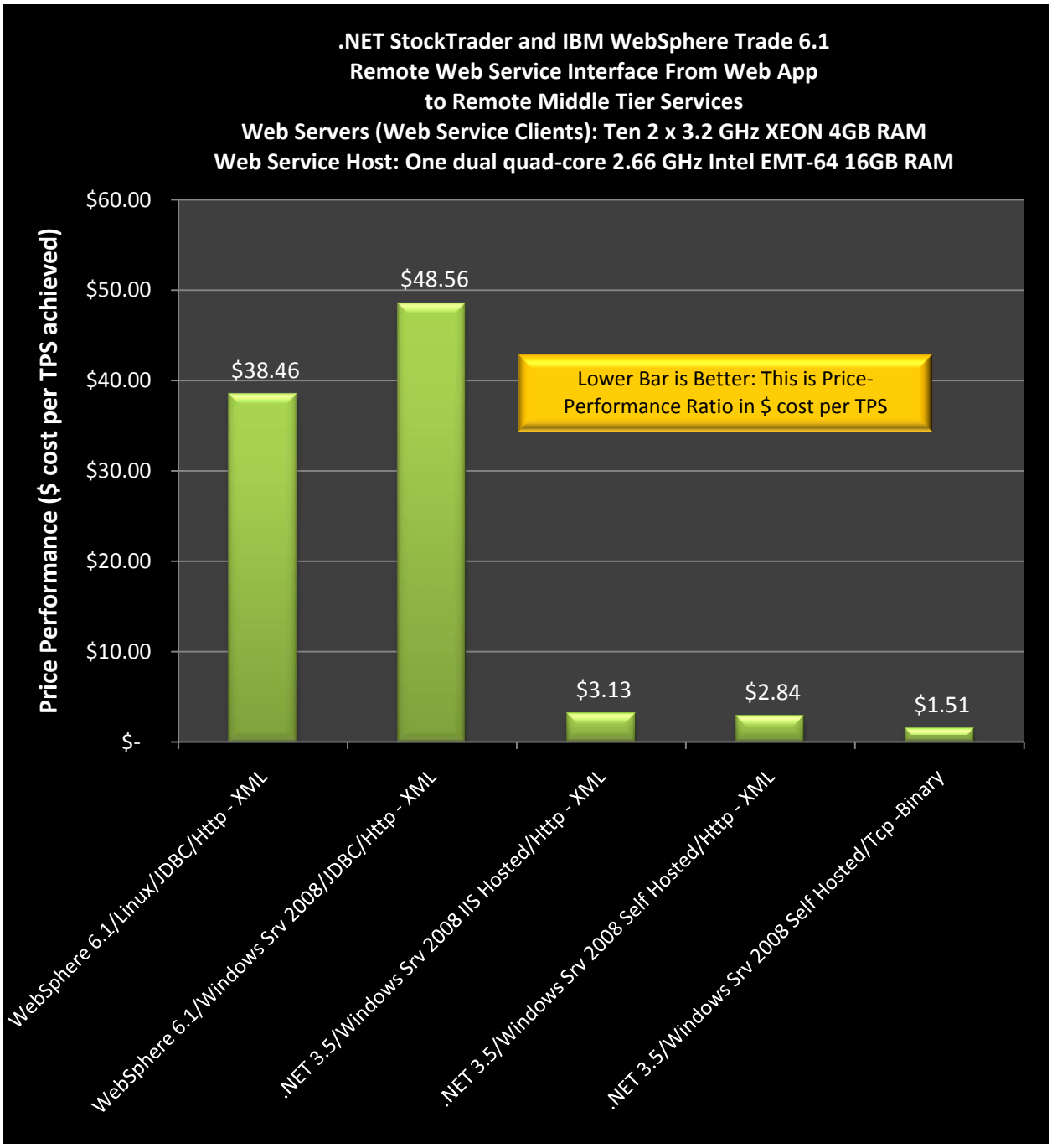

Figure 4: Price Performance Ratio for the Web Service Test. IBM WebSphere Trade 6.1 is running in JDBC mode with no Entity Beans, instead using JDBC API with SQL statements to access the database. Lower bar indicates better value for performance delivered. See appendix for pricing calculations, which includes the software cost of the middle tier OS and application server as tested.

The Web Services Benchmark Discussion

In this test, the Web Application servers (running the JSP or ASP.NET front ends) are the Web Service client machines, using Web Services to remotely invoke the service layer, hosted on the Web Service Host application server. Both the clients and the host are performing XML serialization and de-

serialization. The Web Application server clients are accessing the services via a SOAP Proxy (Trade 6.1), or the WCF client (.NET StockTrader). We set the test up with ten distributed Web Application Servers, with round-robin load balancing performed from the Mercury Controller as iterations are performed, such that all four Web Application Servers get equal load. Each in turn makes remote network requests to the Web Service Host, which is servicing all ten Web Application Servers. The use of 10 Web Application Servers ensures we do not have a bottleneck on the Web Tier, and we are accurately comparing just the performance of the Web Service Host computer for all benchmark runs. Some important conclusions can be drawn from this test:

1. The .NET 3.5 SOAP/HTTP (text-XML encoding) configurations are significantly faster than IBM WebSphere; especially when comparing to the EJB/entity bean configuration for Trade 6.1, which is IBM's recommended architecture. .NET 3.5 (WCF) hosted in IIS offers 94.4% better throughput than the EJB WebSphere configuration on Linux in this test.
2. Self-hosting WCF services can lead to performance advantages over hosting .NET Web Services in IIS—even when operating over an HTTP-XML basicHttpBinding.
3. The self-hosted WCF services can also support, simultaneously, the netTcp WCF binding, with binary encoding. This can lead to significant performance boosts for remote calls. The WCF netTcpBinding replaces .NET Binary Remoting (used with .NET 1.1 and 2.0) as the preferred way for remote calls between .NET clients and remote .NET services. Supporting both HTTP/XML and TCP/Binary requires no extra development, as WCF unifies the programming model for HTTP-based Web Services and .NET Binary-remoted components, and service hosts will listen simultaneously on all configured endpoints to support any different type of client on any platform.
4. The Tcp-Binary binding (netTcpBinding) between the ASP.NET clients and the Web Service host offer 108% better throughput than the WCF basicHttpBinding used in the IIS-hosted .NET service tier. The Tcp-Binary remote mode offers 141% better throughput than the fastest WebSphere 6.1 Web Service configuration (JDBC data access, WebSphere on Linux).

Refer to the appendix for the performance monitor captures for this test taken after initial warm-up at steady-state throughput rates for each configuration.

Messaging Benchmark - Durable Queue with Two Phase/Distributed Transactions

With Trade 6.1 in Default EJB Mode (Entity Beans/CMP Data Access)

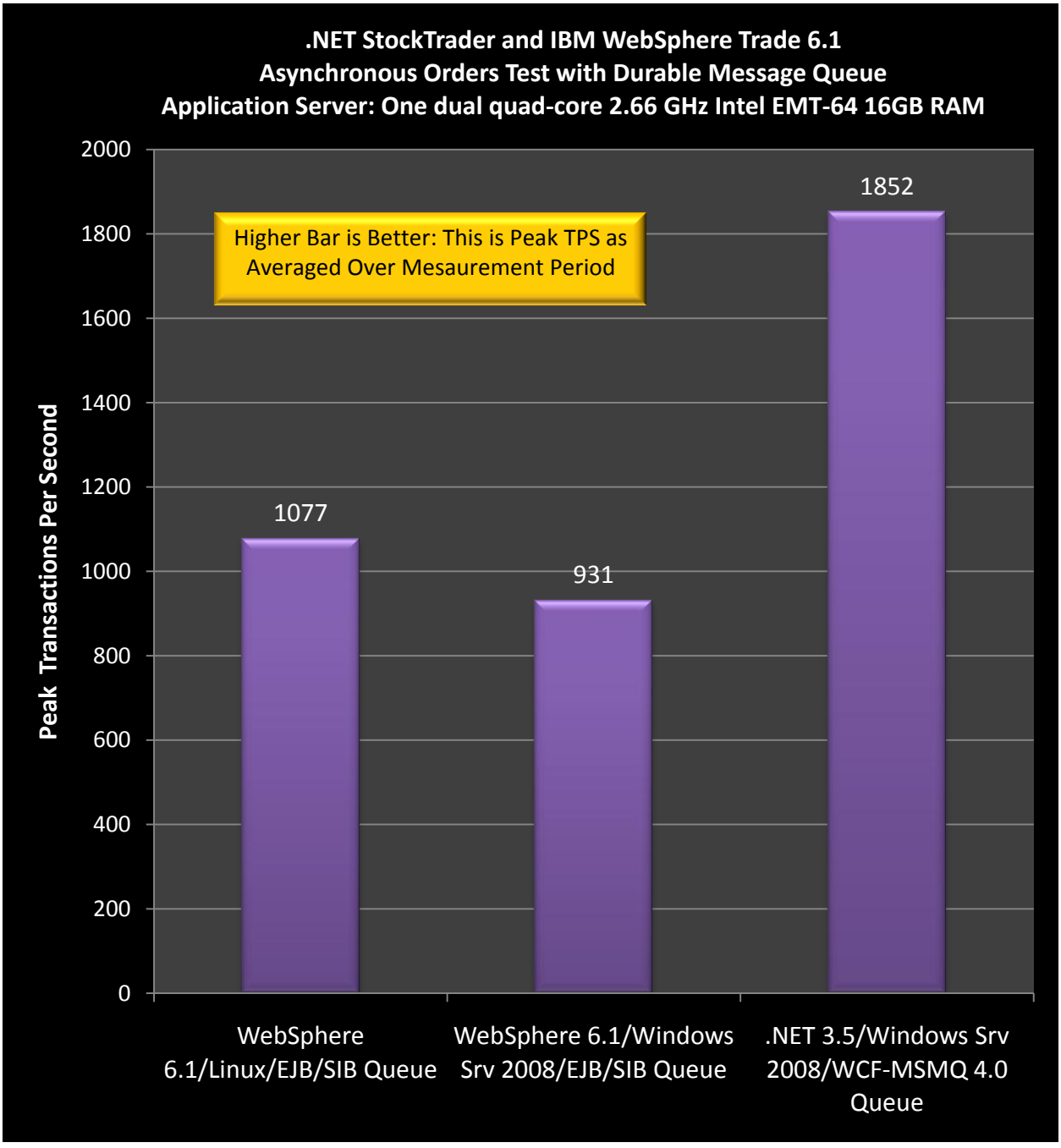

Figure 5: Benchmark TPS rates for durable messaging tests, with IBM WebSphere Trade 6.1 running in its default EJB data access mode using entity beans and CMP. The Messaging engine (MSMQ or IBM's Service Integration Bus Message Queue/JMS messaging engine) are co-located on the same application server as the other parts of the application in this test.

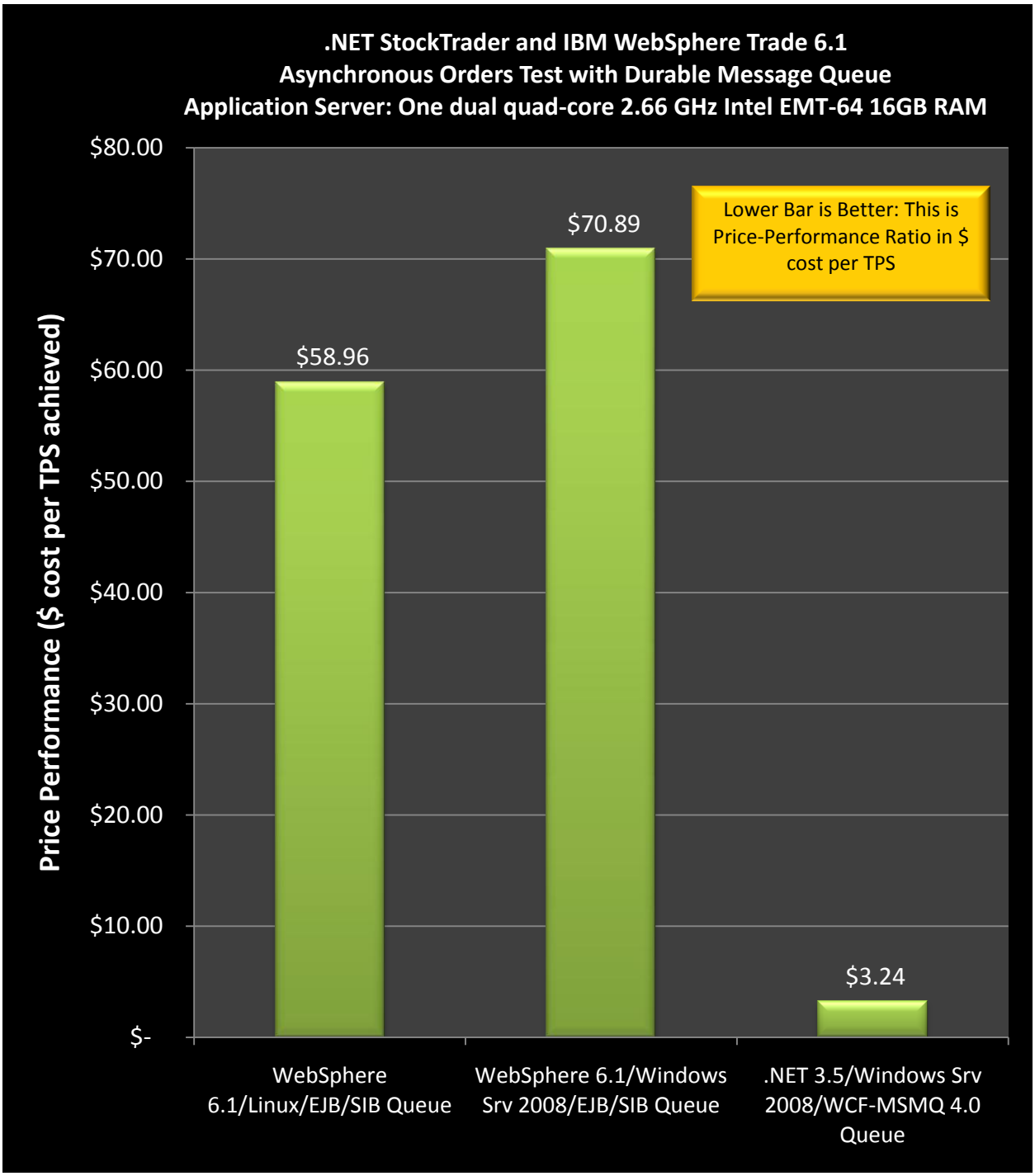

Figure 6: Price performance ratios for the durable message queue order processing test, with IBM WebSphere Trade 6.1 running in its default EJB data access mode using entity beans and CMP, set to Async Orders/Two Phase. The messaging engine (MSMQ or IBM's Service Integration Bus Message Queue/JMS messaging engine) are co-located on the same application server as the other parts of the application in this test. See appendix for pricing calculations/detail.

With Trade 6.1 in JDBC Data Access Mode (No Entity Beans or CMP)

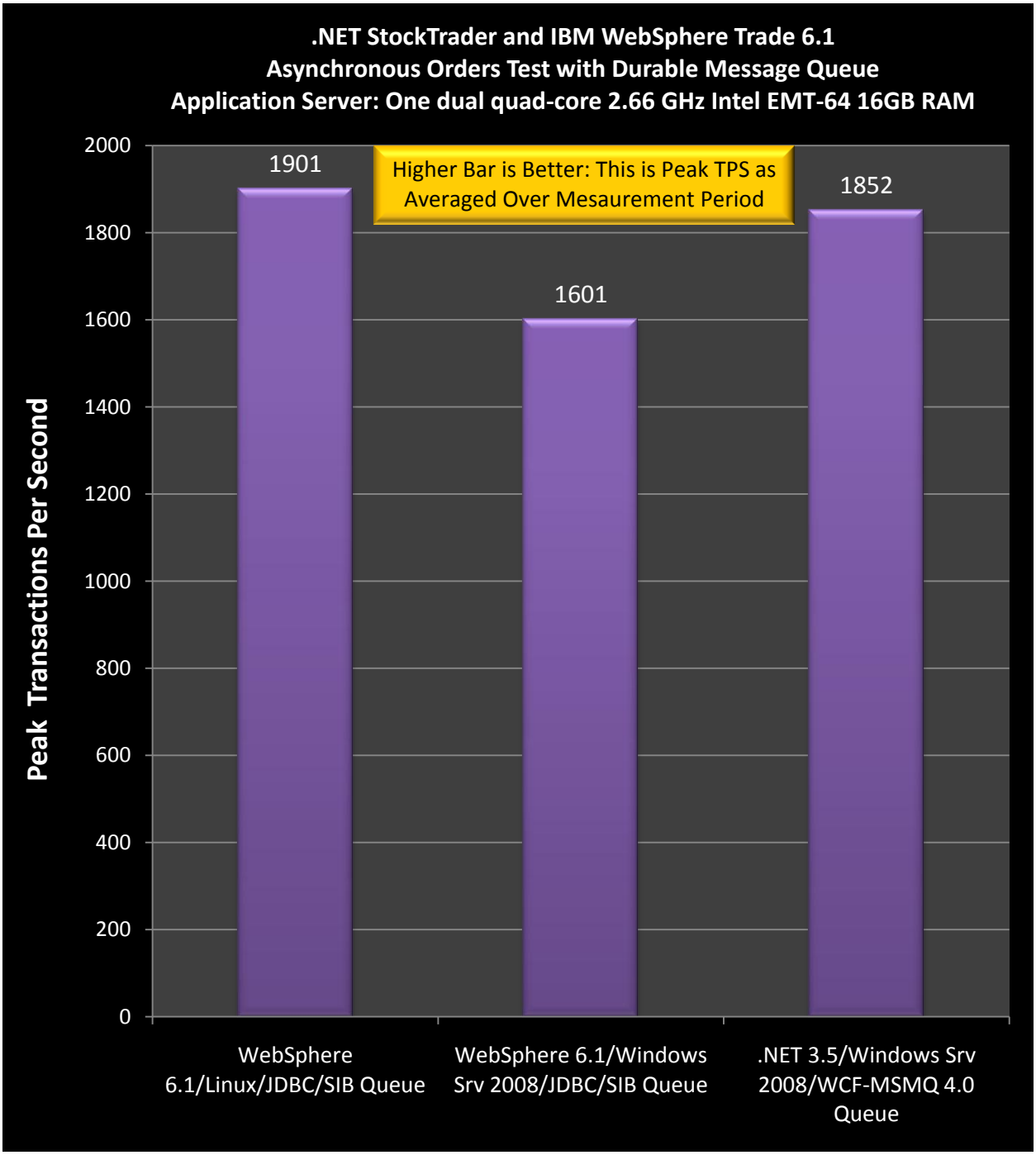

Figure 7: Peak TPS/throughput data for the durable message queue order processing test, with IBM WebSphere Trade 6.1 running in direct JDBC mode with no Entity Beans, and not using Container Managed Persistence (CMP). Trade 6.1 is set to Async Orders/Two Phase. The messaging engine (MSMQ or IBM's Service Integration Bus Message Queue/JMS messaging engine) are co-located on the same application server as the other parts of the application in this test.

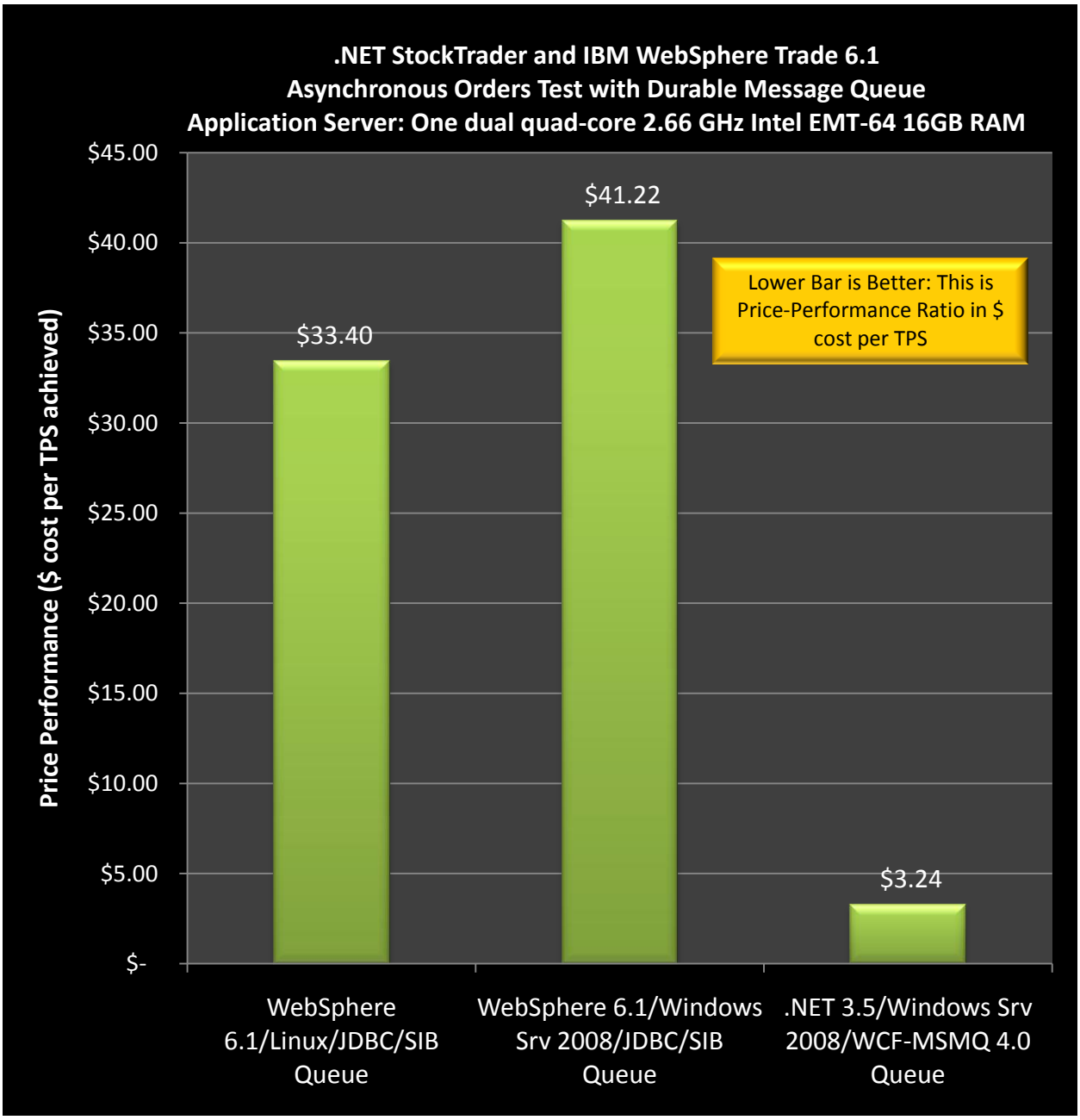

Figure 8: Price performance ratios for the durable message queue order processing test, with IBM WebSphere Trade 6.1 running in direct JDBC data access mode with no Entity Beans, and not using Container Managed Persistence (CMP). Trade 6.1 is set to Async Orders/Two Phase. The messaging engine (MSMQ or IBM's Service Integration Bus Message Queue/JMS messaging engine) are co-located on the same application server as the other parts of the application in this test. See appendix for pricing calculations/detail.

Messaging Benchmark Discussion – Durable/Persistent Message Queue

In this test, the OrderMode is set to Asynchronous-TwoPhase for Trade 6.1, with the SIB queue configured for persistent storage and Assured Message Delivery. This is the configuration that would be used in a production application, with the reads from the queue and the corresponding database

inserts/updates occurring as part of a single atomic, distributed transaction. This ensures that messages are not lost if a database processing failure occurs. For .NET StockTrader, the OrderMode is set to ASync_Msmq. In this mode, the WCF Service is bound to a transacted (durable/persisted) message queue (MSMQ 4.0), with a similar two phase distributed transaction when processing orders off the queue. Again, this mode ensures messages are not lost if a database processing error occurs. Some conclusions that can be drawn from this test:

1. Again, as in all tests, the JDBC "Direct" mode for WebSphere offers better performance than the use of EJB entity beans.
2. .NET WCF outperforms the WebSphere EJB configuration (Linux) in this test by 72%; however in JDBC data access mode with no entity beans, WebSphere/Linux slightly outperforms in raw throughput .NET 3.5/WCF/MSMQ by 2.6%.

Data-Driven Web Application Benchmark

With Trade 6.1 in Default EJB Mode (Entity Beans/CMP Data Access)

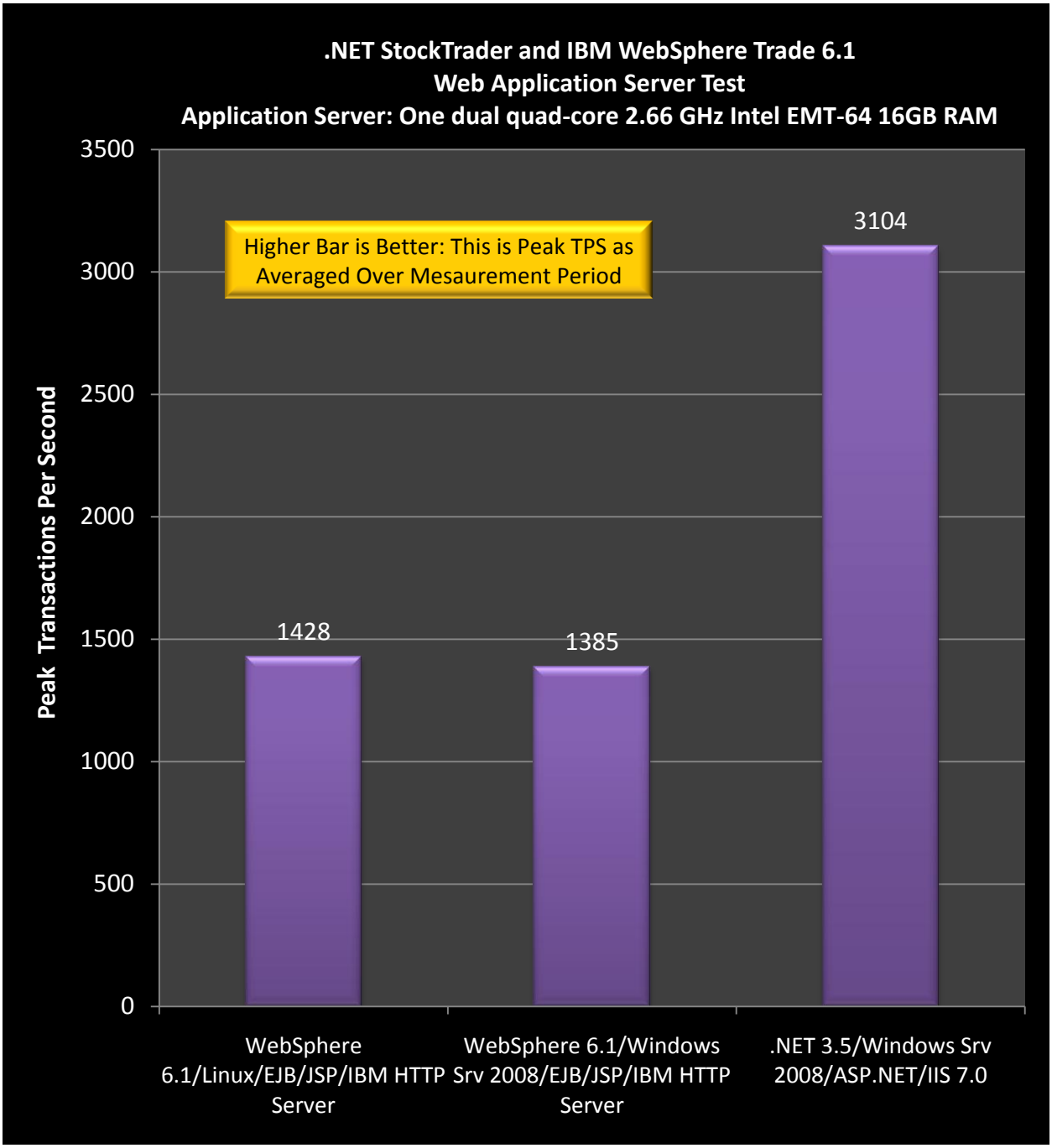

Figure 9: Peak TPS Rates for the Web Application Test. IBM WebSphere Trade 6.1 is running in default EJB mode with entity beans and CMP for data access.

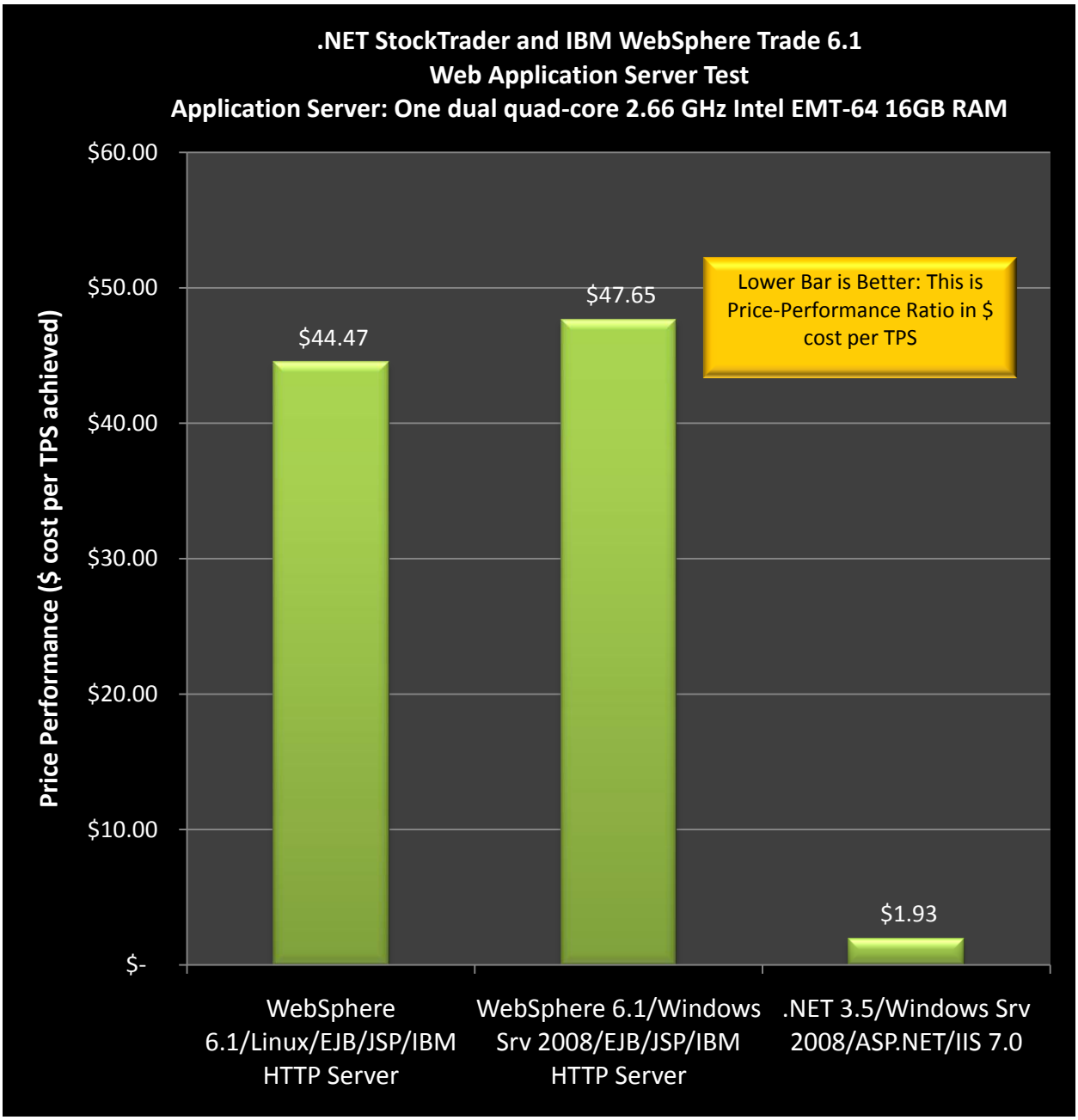

Figure 10: Price/Performance Chart for the Web Application Test. IBM WebSphere Trade 6.1 is running in default EJB mode with Entity Beans and Container Managed Persistence (CMP). Refer to appendix for pricing calculations for the middle tier software.

With Trade 6.1 in JDBC Data Access Mode (No Entity Beans or CMP)

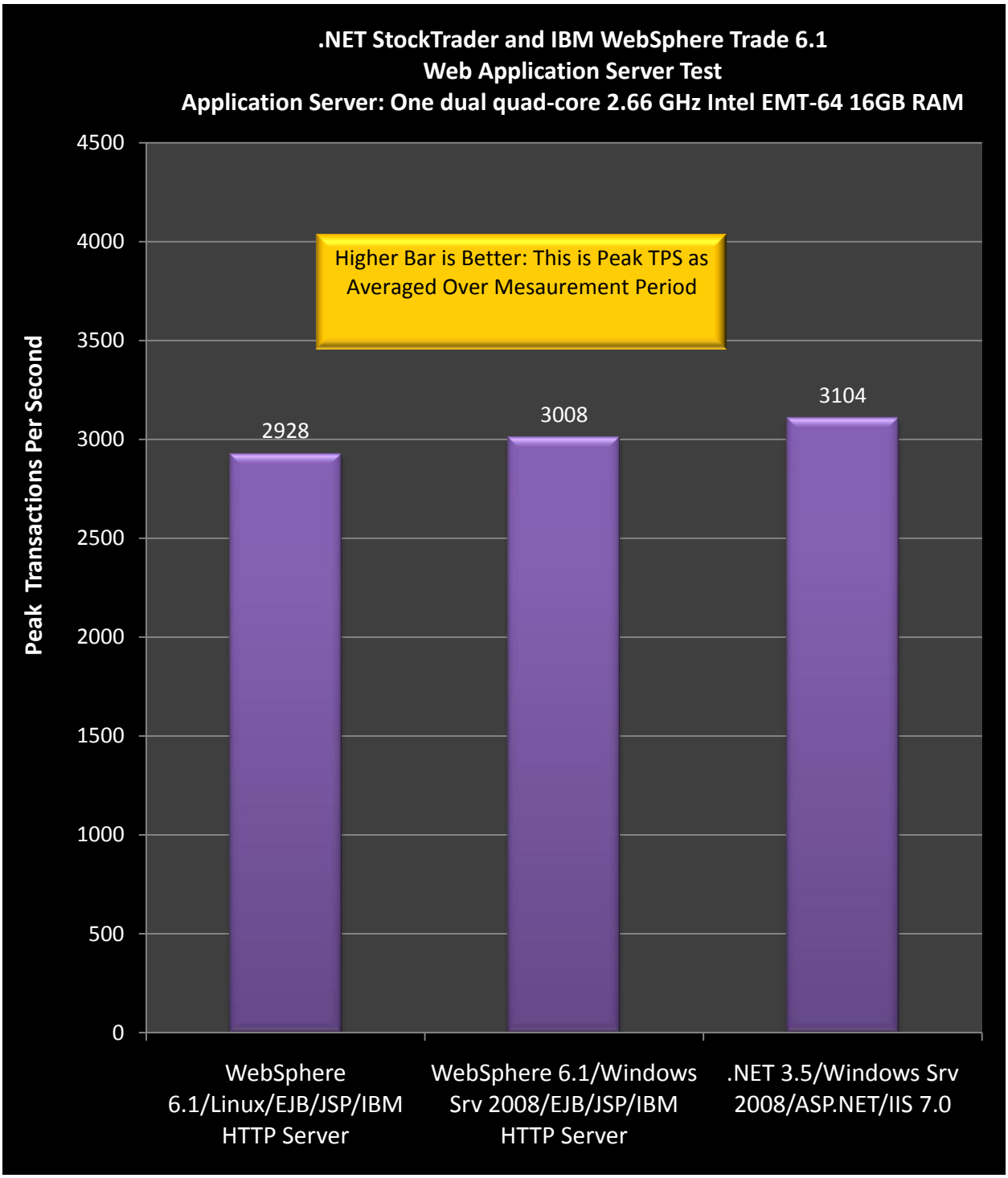

Figure 11: Peak TPS Rates for the Web Application Test. IBM WebSphere Trade 6.1 is running in JDBC data access mode with no Entity Beans or CMP.

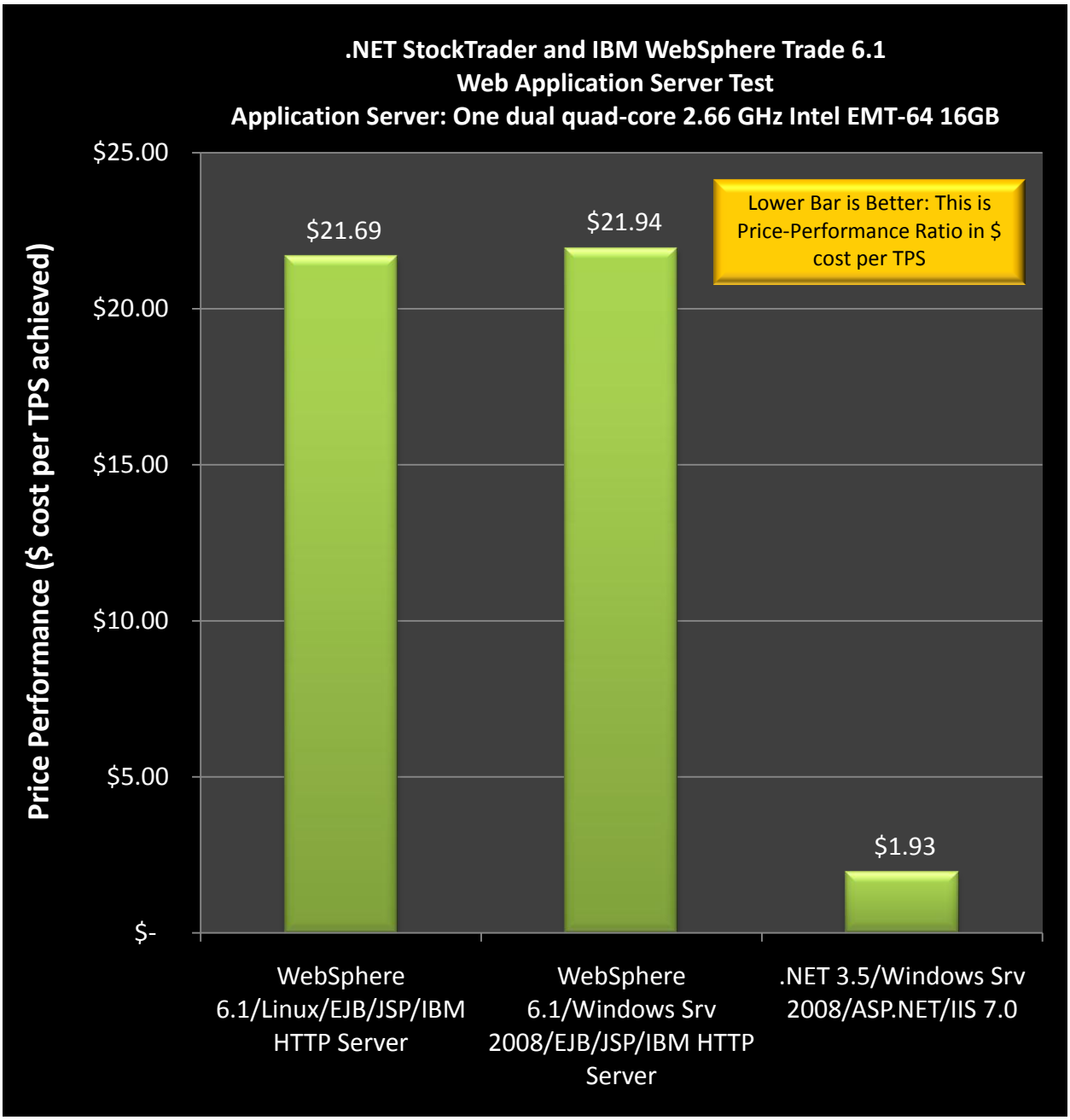

Figure 12: Price-Performance Ratio for the Web Application Test. IBM WebSphere Trade 6.1 is running in JDBC data access mode with no entity beans or CMP. Refer to appendix for pricing details.

Data-Driven Web Application Benchmark Discussion

This mode of operation represents a non-service oriented, monolithic Web application. The performance is quite good, considering there are no distributed transactions, no messaging/queuing, and no remote calls between the Web Application and the middle tier processing components. All elements of the application must be deployed in unison, and hence versioned/updated in unison. Nevertheless, for applications that do not require Web Services, remote calls, or messaging, this mode of operation, for both applications, is a viable choice for a deployment, and one which can provide very

fast performance. We see again in these results a marked difference between IBM WebSphere Trade 6.1 in EJB mode, IBM's recommended J2EE architecture, vs. JDBC mode with straight JDBC calls for data access.

Application Architecture Diagrams

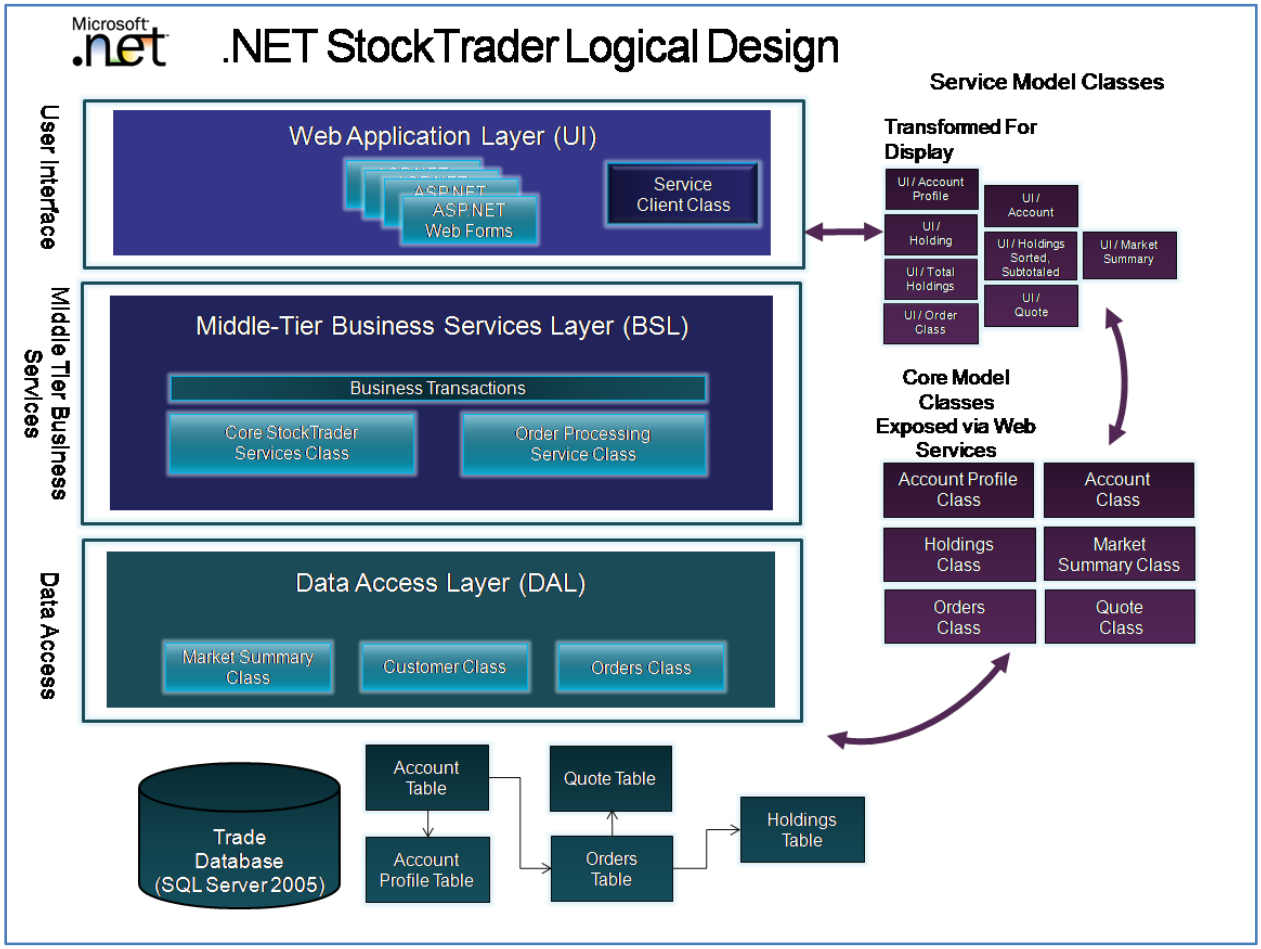

Figure13: Logical Design of .NET StockTrader

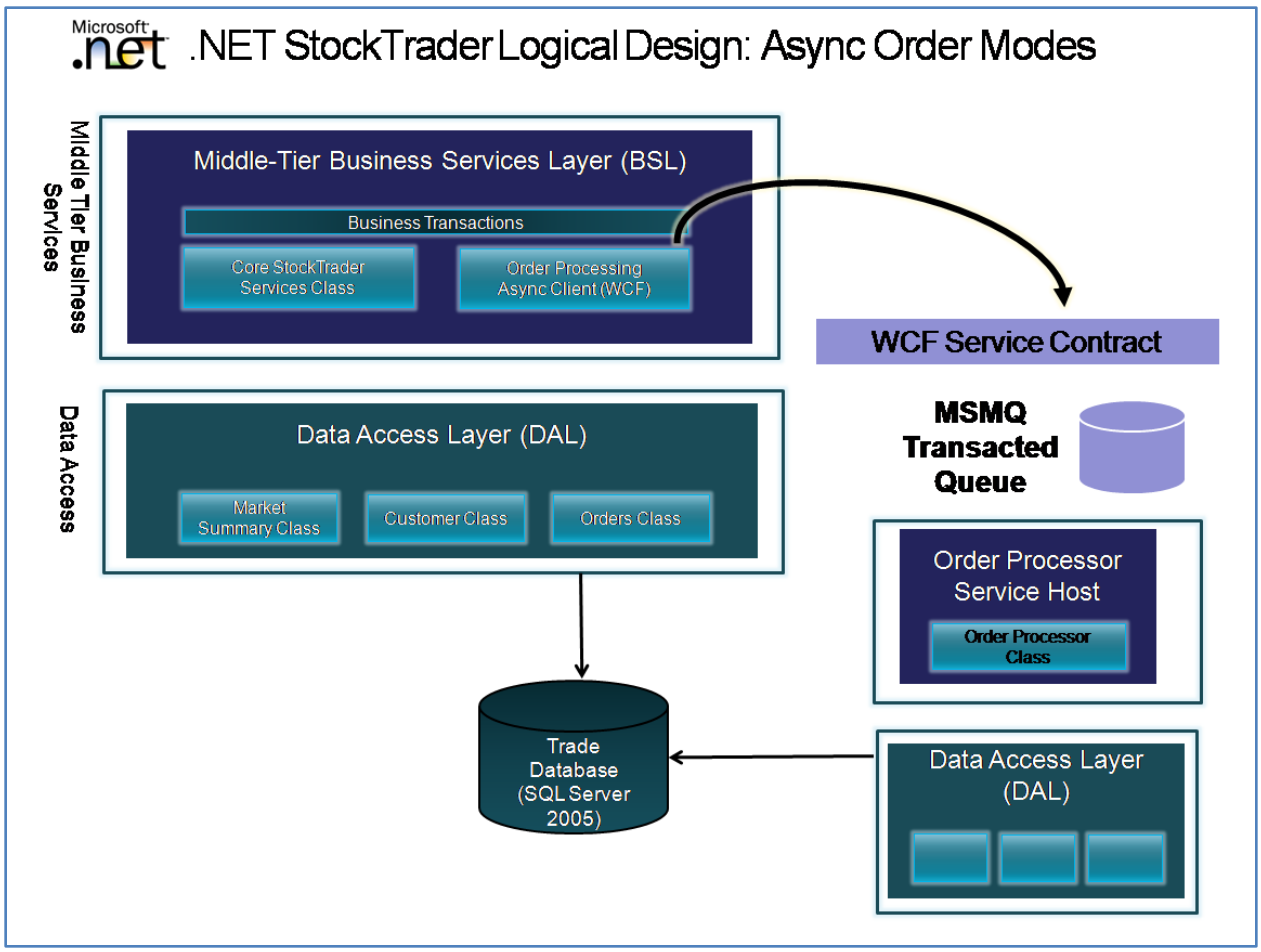

Figure 14: Asynchronous, Message-Oriented Processing of Orders

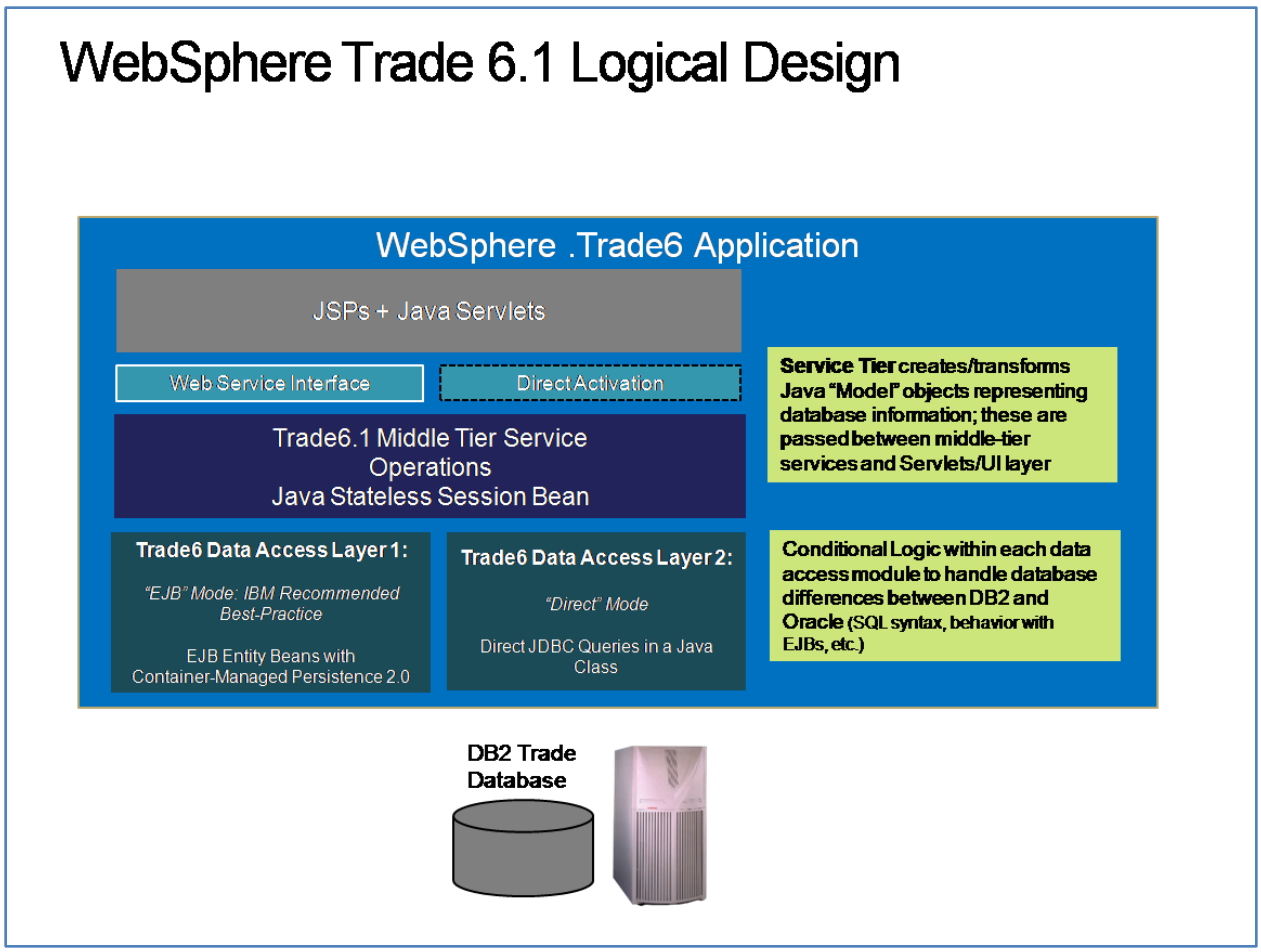

Figure 15: Trade 6.1 Logical Design

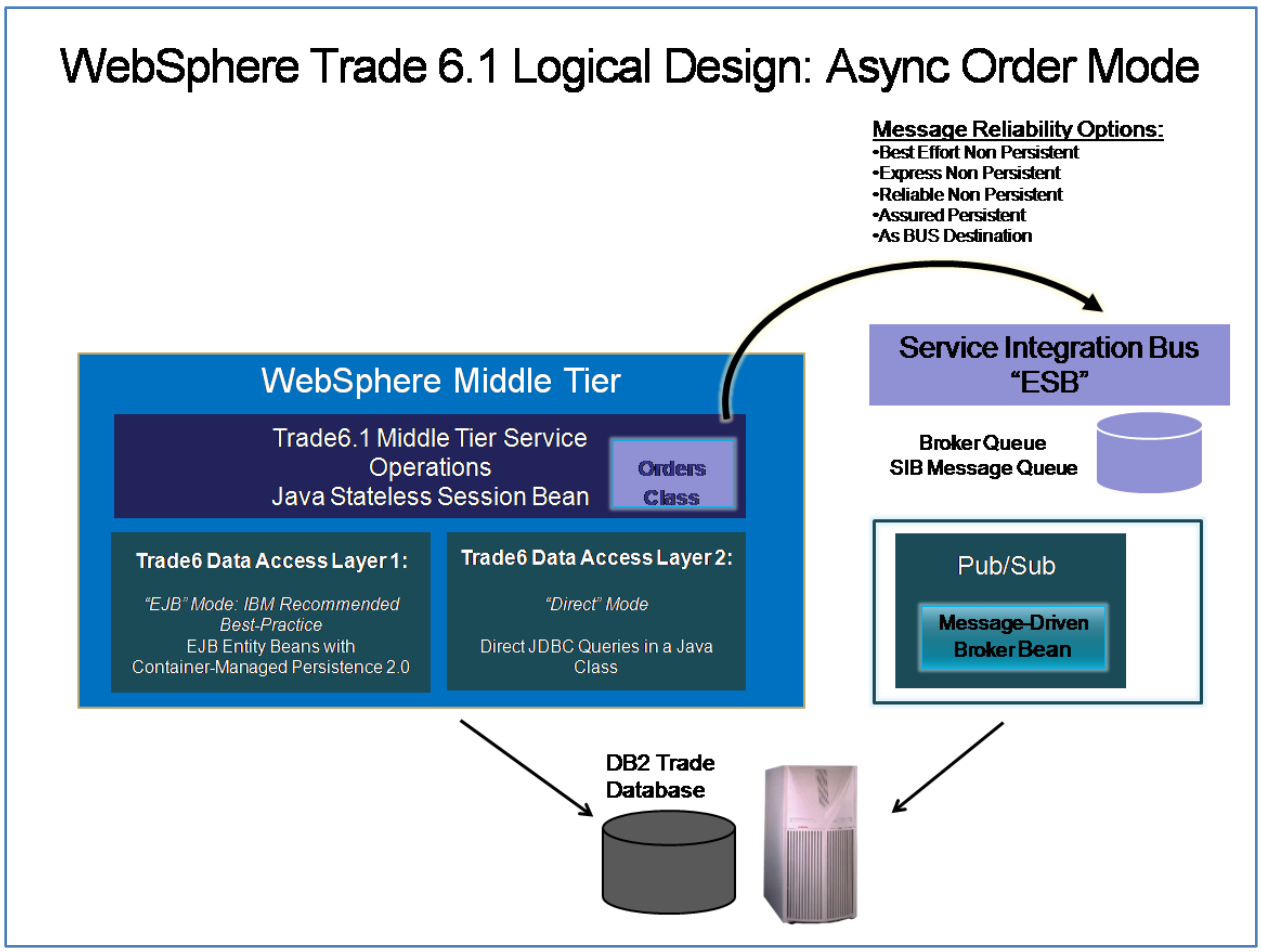

Figure 16: Asynchronous, Message-Oriented Processing of Orders

Conclusion

This paper presents an extensive array of benchmark comparisons between IBM WebSphere 6.1.0.13 Network Deployment Edition and .NET 3.5/Windows Server 2008 running an application server workload. The benchmark is based on the functional specification of IBM WebSphere Trade 6.1, as defined and developed by IBM for the WebSphere 6.1 platform. The .NET results are based on a migration of this application to .NET 3.5 with the use of Windows Communication Foundation for the service layers, and running on Windows Server 2008 Enterprise Edition. The .NET StockTrader is a best-practice performance implementation for the .NET platform, and is functionally and behaviorally equivalent to the tested Trade 6.1 application in the configurations tested. The benchmark results show the two platforms running in a variety of different configurations. With published source code for both implementations, we encourage customers to perform their own comparative testing; and also to use the .NET StockTrader application as a learning sample for various features of .NET 3.5, WCF and the Microsoft enterprise development technologies.

Appendix A: Pricing

The following pricing was used for the $/TPS calculations. Pricing is based on published list pricing for the products.

Pricing for the Application Server + OS Used in the Tests

Pricing includes middle tier software licensing costs (OS + Application Server) for the primary application server/Web Service Host used in the remote tests. Database software costs and middle tier/database hardware costs were not included. We priced IBM WebSphere 6.1 Network Deployment Edition for the application server, running on both Red Hat Linux Advanced Platform 5, and Windows Server 2008. Network Deployment Edition is IBM's recommended enterprise application server, and includes its core enterprise features. Note that for the .NET 3.5/Microsoft Windows Server 2008 configuration, no separate application server is necessary: .NET is integrated into Windows Server and new versions are made available as free downloads on MSDN. There is also no redistribution license fee to redistribute the full .NET Framework runtime. Red Hat Advanced Platform 5 was priced for the Linux application server tested. Windows Server 2008 Enterprise was priced for the application server.

For the .NET configurations, the Windows Server 2008 External Connector License, which allows unlimited Anonymous Web access (as used in the StockTrader Application) without CALs was added to the total cost.

WebSphere Pricing Windows

1 x WAS 6.1 Network Deployment Edition:
$155.00 per Value Unit (VU); 50 value units for quad-core Intel CPUs, 8 cores =
155.00 x 50 x 8 = $62,000.00

1 x Windows Server 2008 Enterprise Edition:
$3,999.00

Total: $65,999.00

WebSphere Pricing Red Hat Linux

1 x WAS 6.1 Network Deployment Edition:
$155.00 per Value Unit (VU); 50 value units for quad-core Intel CPUs, 8 cores =
155.00 x 50 x 8 = $62,000.00

1 x Red Hat Linux Advanced Platform 5:
$1,499.00

Total: $63,499.00

.NET Pricing (Windows Server 2008)

1 x Windows Server 2008 Enterprise Edition:
$3,999.00

1 x External Connector License (@$1,999.00 per copy)
$1,999.00

Total: $5,998.00

Appendix B: Tuning Parameters

Linux OS Tuning

net.ipv4.tcp_max_syn_backlog=1024

kernel.msgmni=1024

kernel.sem=1000 32000 32 512

fs.file-max=65535

kernel.shmmax =4294967295

net.core.netdev_max_backlog = 20000

net.core.somaxconn = 20000

net.ipv4.tcp_fin_timeout = 30

net.ipv4.tcp_syn_retries = 20

net.ipv4.tcp_synack_retries = 20

net.ipv4.tcp_sack = 0

net.ipv4.tcp_timestamps = 0

net.ipv4.conf.all.arp_ignore = 3

net.ipv4.conf.all.arp_announce = 2

Open File Handle limit (soft) increased to 20000

Windows OS Tuning

No tuning was required on the core Windows Server OS for either application server platform.

WebSphere Tuning – All Linux EJB Modes (Entity Beans/CMP for Data Access – Default Trade 6.1 Config)

Servlet Caching turned on in Web Container

Session State set to 5 minute expiration (in-process session state)

Access Log Turned Off

Performance Monitor Infrastructure Turned Off

App Profile Service Off

Diagnostic Trace Turned Off

System Out Off

Trade 6.1 Configured not to write System.Out messages

EJB Cache Size = 20000

HTTP Channel maximum persistent requests = -1

Minimum Web Container threads = 40

Maximum Web Container threads = 40

Minimum ORB threads = 24

Maximum ORB threads = 24

Minimum Default threads = 20

Maximum Default threads = 20

Minimum Message Listener Service Threads = 30

Maximum Message Listener Service Threads = 30

Minimum SIBInBound Thread = 30

Maximum SIBInbound Thread = 30

Minimum SIBFAPThread = 30

Maximum SIBFAPThread = 30

Custom JavaEnvironment Variable: com.ibm.websphere.ejbcontainer.poolsize value = "*=75,750"

SIB Bus Security = Disabled

Discard Messages = on

Hi Message Threshold = 50000

Quality of Service/Persistent = Assured Reliable

Quality of Service/Non Persistent = Express/Non Persistent

ReadAhead for Queue enabled

MaxConcurrency/Max Endpoints for Queue = 15

MaxBatchSize for JMS/Messaging = 5

Minimum JDBC Connections in Pool = 25

Maximum JDBC Connections in Pool = 25

Minimum Queue Connection Factory Connections in Pool = 30

Maximum Queue Connection Factory Connections in Pool = 30

Java Heap Size: 3000 min/3100 MB max

All runs: Trade 6.1 configured with "Enable Long Run Support" off to ensure it properly displays orders on the Account Page. With our large database load, we did not notice any perf degradation over a 30 minute measurement interval.

IBM HTTP Server Tuning Linux

Access Log Off

Max KeepAlive Requests unlimited

ServerLimit 100

StartServers 80

MaxClients 4000

MinSpareThreads 100

MaxSpareThreads 100

Threads/Child 40

MaxRequests/Child 0 (unlimited)

WebSphere Tuning – All Linux JDBC Modes (No Entity Beans/CMP for Data Access – Direct JDBC Config)

Servlet Caching turned on in Web Container

Session State set to 5 minute expiration (in-process session state)

Access Log Turned Off

Performance Monitor Infrastructure Turned Off

App Profile Service Off

Diagnostic Trace Turned Off

System Out Off

Trade 6.1 Configured not to write System.Out messages

EJB Cache Size = 20000

HTTP Channel maximum persistent requests = -1

Minimum Web Container threads = 40

Maximum Web Container threads = 40

Minimum ORB threads = 24

Maximum ORB threads = 24

Minimum Default threads = 20

Maximum Default threads = 20

Minimum Message Listener Service Threads = 30

Maximum Message Listener Service Threads = 30

Minimum SIBInBound Thread = 30

Maximum SIBInbound Thread = 30

Minimum SIBFAPThread = 30

Maximum SIBFAPThread = 30

Custom JavaEnvironment Variable: com.ibm.websphere.ejbcontainer.poolsize value = "*=75,750"

SIB Bus Security = Disabled

Discard Messages = on

Hi Message Threshold = 50000

Quality of Service/Persistent = Assured Reliable

Quality of Service/Non Persistent = Express/Non Persistent

ReadAhead for Queue enabled

MaxConcurrency/Max Endpoints for Queue = 15

MaxBatchSize for JMS/Messaging = 5

Minimum JDBC Connections in Pool = 35

Maximum JDBC Connections in Pool = 35

Minimum Queue Connection Factory Connections in Pool = 35

Maximum Queue Connection Factory Connections in Pool = 35

Java Heap Size: 3000 min/3100 MB max

All runs: Trade 6.1 configured with "Enable Long Run Support" off to ensure it properly displays orders on the Account Page. With our large database load, we did not notice any perf degradation over a 30 minute measurement interval.

IBM HTTP Server Tuning Linux

Access Log Off

Max KeepAlive Requests unlimited

ServerLimit 100

StartServers 80

MaxClients 4000

MinSpareThreads 100

MaxSpareThreads 100

Threads/Child 40

MaxRequests/Child 0 (unlimited)

WebSphere Tuning – All Windows Server 2008 EJB Modes (Entity Beans/CMP for Data Access – Default Trade 6.1 Config)

Servlet Caching turned on in Web Container

Session State set to 5 minute expiration (in-process session state)

Access Log Turned Off

Performance Monitor Infrastructure Turned Off

App Profile Service Off

Diagnostic Trace Turned Off

System Out Off

Trade 6.1 Configured not to write System.Out messages

EJB Cache Size = 20000

HTTP Channel maximum persistent requests = -1

Minimum Web Container threads = 50

Maximum Web Container threads = 50

Minimum ORB threads = 18

Maximum ORB threads = 18

Minimum Default threads = 20

Maximum Default threads = 20

Minimum Message Listener Service Threads = 30

Maximum Message Listener Service Threads = 30

Minimum SIBInBound Thread = 30

Maximum SIBInbound Thread = 30

Minimum SIBFAPThread = 30

Maximum SIBFAPThread = 30

Custom JavaEnvironment Variable: com.ibm.websphere.ejbcontainer.poolsize value = "*=75,750"

SIB Bus Security = Disabled

Discard Messages = on

Hi Message Threshold = 50000

Quality of Service/Persistent = Assured Reliable

Quality of Service/Non Persistent = Express/Non Persistent

ReadAhead for Queue enabled

MaxConcurrency/Max Endpoints for Queue = 15

MaxBatchSize for JMS/Messaging = 5

Minimum JDBC Connections in Pool = 19

Maximum JDBC Connections in Pool = 19

Minimum Queue Connection Factory Connections in Pool = 30

Maximum Queue Connection Factory Connections in Pool = 30

Java Heap Size: 3000 min/3100 MB max

All runs: Trade 6.1 configured with "Enable Long Run Support" off to ensure it properly displays orders on the Account Page. With our large database load, we did not notice any perf degradation over a 30 minute measurement interval.

IBM HTTP Server Tuning Windows Server 2008

Access Log Off

Max KeepAlive Requests unlimited

Default Threads 3200

Threads/Child 3200

WebSphere Tuning – All Windows Server 2008 JDBC Data Access Modes (No Entity Beans/CMP for Data Access)

Servlet Caching turned on in Web Container

Session State set to 5 minute expiration (in-process session state)

Access Log Turned Off

Performance Monitor Infrastructure Turned Off

App Profile Service Off

Diagnostic Trace Turned Off

System Out Off

Trade 6.1 Configured not to write System.Out messages

EJB Cache Size = 20000

HTTP Channel maximum persistent requests = -1

Minimum Web Container threads = 50

Maximum Web Container threads = 50

Minimum ORB threads = 24

Maximum ORB threads = 24

Minimum Default threads = 20

Maximum Default threads = 20

Minimum Message Listener Service Threads = 30

Maximum Message Listener Service Threads = 30

Minimum SIBInBound Thread = 30

Maximum SIBInbound Thread = 30

Minimum SIBFAPThread = 30

Maximum SIBFAPThread = 30

Custom JavaEnvironment Variable: com.ibm.websphere.ejbcontainer.poolsize value = "*=75,750"

SIB Bus Security = Disabled

Discard Messages = on

Hi Message Threshold = 50000

Quality of Service/Persistent = Assured Reliable

Quality of Service/Non Persistent = Express/Non Persistent

ReadAhead for Queue enabled

MaxConcurrency/Max Endpoints for Queue = 15

MaxBatchSize for JMS/Messaging = 5

Minimum JDBC Connections in Pool = 50

Maximum JDBC Connections in Pool = 50

Minimum Queue Connection Factory Connections in Pool = 30

Maximum Queue Connection Factory Connections in Pool = 30

Java Heap Size: 3000 min/3100 MB max

All runs: Trade 6.1 configured with "Enable Long Run Support" off to ensure it properly displays orders on the Account Page. With our large database load, we did not notice any perf degradation over a 30 minute measurement interval.

IBM HTTP Server Tuning Windows Server 2008

Access Log Off

Max KeepAlive Requests unlimited

Default Threads 3200

Threads/Child 3200

.NET 2.0/3.0 Tuning

.NET Worker Process

Rapid Fail Protection off

Pinging off

Recycle Worker Process off

ASP.NET

Authentication set to "None" to match anonymous access of IBM WebSphere Trade 6.1

Forms Authentication Timeout=5 minutes

IIS 7.0 Virtual Directory

Authentication Basic Only

Access Logging Off

ServicePointManager.DefaultConnectionLimit = 64

Note, this is a key setting for Web Service clients running under load. Without this setting, Web Service clients (our ten ASP.NET App Servers) will be throttled to 2 network connections per outbound IP Address. This is set programmatically, although it can also be set in web.config.

WCF basicHttp, nNetTcp and Msmq bindings: Security = "None" (no transport security for Web services or the Service Integration Bus is configured for Trade 6.1 as well, see tuning for WebSphere.

Service Behavior for Business Services and Order Processor Service:

<behavior name="TradeServiceBehaviors">
<serviceDebug httpHelpPageEnabled="true" includeExceptionDetailInFaults="true"/>
<serviceMetadata httpGetEnabled="true" httpGetUrl=""/>
<serviceThrottling maxConcurrentInstances="400" maxConcurrentCalls="400"/>
</behavior>

Order Processor Service: maxBatchSize = 5 (set programmatically in the Host)

.NET StockTrader

Max DB Connections = 90

Min DB Connections = 90

MSMQ

Connection Caching turned on

MSTDC

Transaction Timeout = 15 seconds

Network DTC Access Turned on (inbound and outbound allowed)

DB2

Logging files expanded to 15 GB

Logging Set to One Drive Array (array a)

Database file on Second Drive Array (array b)

Max Application Connections = 150

SQL/Server

Logging files expanded to 15 GB

Logging Set to One Drive Array (array a)

Database file on Second Drive Array (array b)

Surface Area Configuration Allow Remote Connections (Named Pipes and TCP/IP)