Web Services Performance: Comparing Java 2^{TM} Enterprise Edition (J2EE^{TM} platform) and the Microsoft® .NET Framework

A Response to Sun Microsystem's Benchmark

Microsoft Corporation
July 2004

Introduction

In June 2004, Sun Microsystems published a benchmark showing the relative performance of Microsoft .NET vs. Sun's Java Web Service Developer Pack on a series of simple Web Service tests. The original Sun benchmark report and results can be found at http://java.sun.com/performance/reference/whitepapers/WS_Test-1_0.pdf . Sun's report shows Java outperforming .NET in their WSTest benchmark suite by a wide margin. Sun did not publish the source code used for the tests when they published the paper (as of July 14th, 2004) for customers to examine the test suite, or to replicate and verify the results at that time. In their paper, however, they describe the functional specification for the Web Services tested, the benchmark driver program, the tuning used for both Java and .NET, and the software/hardware platform used in their testing. This information provided enough detail for Microsoft to create an implementation of both the Java and .NET benchmark suites, and to verify the test results using the latest Java platform components including Sun's J2SDK 1.4.2.05, Tomcat 5, and JWSDP 1.4. In this paper we describe the results of testing these re-created implementations.

Microsoft is making all source code available for both our Sun Java JWSDP and .NET implementations as well as our driver program so that customers and other vendors can download the code and verify the results on their own hardware platforms (to download the code, visit http://www.theserverside.net/articles/showarticle.tss?id=SunBenchmarkResponse). In our tests, we used the recently released Tomcat 5.0 and Sun's JWSDP version 1.4, slightly newer releases of these software components than Sun used. We have also created several additional tests of Web Services on each platform to illustrate the relative performance when the backend Web Services are required to do more work. These additional tests are more realistic than Sun's tests, and show the relative performance when the Web Service message payload is increased.

Summary of Results

In the four tests that replicate the original Sun tests, we got similar results for the performance of the Java Web Services as reported by Sun. Our results were slightly better than Sun's results for Java, a difference that can be attributed to the fact our backend server had CPUs that were slightly faster (3 GHz vs. 2.6 GHz) than the CPUs Sun used. However, we also found that Sun seriously misrepresented .NET performance, with our results significantly outperforming the .NET results as reported by Sun. In short, the .NET results are actually more than ***two to three times better than Sun reported***. In our tests, .NET roughly matched or slightly exceeded J2EE performance for Sun's four original tests. Furthermore, in the additional more realistic tests involving higher Web Service message payloads we found .NET to significantly outperform Java.

Issues with Sun's Benchmark

Most benchmarks are simplifications of real-world scenarios designed at showing the performance of one part of an overall system. Sun's WSTest is such a simplification. However, the tests performed in Sun's WSTest suite are so simplistic that we do not believe they represent enough information for customers to make balanced decisions on the relative performance of .NET and Java for Web Services. For example, Sun's tests do not include any database interactions, nor do they include the use of core features of J2EE such as Enterprise Java Beans. Furthermore, Sun did not use a major commercial application server platform such as IBM WebSphere or BEA WebLogic in their testing, choosing instead to use Tomcat, a lightweight application server that does not support core J2EE features such as EJBs. For these reasons, we encourage customers to also read the more thorough Doculabs Web Service benchmark report, which can be found at http://www.theserverside.net/articles/content/DoculabsWebServiceScalability/DoculabsWebServiceScalability.pdf . This benchmark reports the relative performance of .NET and J2EE Web Services when used against a variety of backend databases, is inclusive of EJB and Java Servlet testing, and includes results for J2EE on three different application servers, including JBOSS and two major commercial J2EE application servers.

Notwithstanding these issues, customers should examine the results for .NET in this report as they are significantly different than Sun's findings, and also examine the results for tests involving Web Services that perform more realistic work on the backend, where .NET significantly outperforms Sun's JWSDP. Finally, customers can download, examine and execute the code for the Java implementation and .NET implementation, as well as the common Java driver program we used in the benchmark to fully replicate the results and perform further testing.

Test Description

As described by Sun, WSTest "simulates a multi-threaded server program that makes multiple Web Service calls in parallel. To avoid the effects of other platform components, the Web Service methods perform no business logic but simply return the parameters that were passed in." Sun's WSTest benchmark and Microsoft's implementation of WSTest include the following four Web Services, as described in Sun's benchmark paper:

- **echoVoid** – Sends and receives and empty message.
- **echoStruct** – Sends and receives an array of size listsize, each element is a structure composed of one element each of an integer, float and string datatype.
- **echoList** – Sends and receives a linked list of size listsize – each element is a Struct as defined in echoStruct.
- **echoSynthetic** – Sends and receives multiple parameters of different types – string, struct and byte array of size 1K.

Microsoft also extended WSTest to additionally test the following new Web Services that perform more work on the backend and are hence more realistic:

- **echoList with list size 100** – Sends and receives a linked list of 100 elements, each element is a Struct as defined in echoStruct.
- **echoList with list size 200 --** Sends and receives a linked list of 200 elements, each element is a Struct as defined in echoStruct.

- **echoStruct with list size 100 --** Sends and receives an array of size 100, each element is a structure composed of one element each of an integer, float and string datatype.
- **echoStruct with list size 200 --** Sends and receives an array of size 200, each element is a structure composed of one element each of an integer, float and string datatype.
- **getOrder** – accepts two integers as parameters and creates a customer order object and returns it to the client. The customer order object simulates a real order with XML/SOAP data types including structures representing customer information such as bill to and ship to addresses, an order header, and a randomly generated number of line items from 1 to 100 per order.

Like the Sun benchmark, the Microsoft implementation of WSTest reports throughput as the average number of Web Service operations executed per second, as well as response times as measured at peak throughput. In all cases care was taken to ensure the client driver was not the bottleneck, and the number of agent threads was enough to saturate each server so that the numbers reported accurately reflect peak throughput on the hardware used. We found that a single 2-proc client machine was unable to completely saturate Web Service host machine for several of the Web Services tested. Without saturating the server, peak throughput is not achieved, so it is critical to run enough clients to ensure full saturation of the server. Hence, we used two client machines in our tests, and properly ensured that the Web Service host machine was just saturated to 100% CPU load for both the .NET and Java implementations tested. It should be noted that with two client driver machines in use, we were able to saturate both the Java and .NET implementations properly to above 97% CPU load.

Test Details

As described by Sun, our implementation of WSTest can be configured with the following parameters, specified in an initialization file:

Agents – This is the number of client threads and is set to maximize CPU utilization and system throughput.
RampUp – Time allotted to system warmup.
SteadyState – Time allotted to collecting data.
RampDown – Time allotted for rampdown.
EchoVoidMix - % of operations that are echoVoid
EchoStructMix - % of operations that are echoStruct
EchoListMix - % of operations that are echoList
GetOrderMix - % of operations that are getOrder
EchoSyntheticMix - % of operations that are echoSynthetic
ListSize – size of the list for echoList and echoStruct
NumBytes – size of byte array for echoSynthetic.

To replicate Sun's testing, no think time is used in the client driver. The throughput is reported by the clients at the end of the run. When running two clients, throughput is aggregated across the client machines, and response times if different are averaged across the clients.

System Configuration

WSTest was run on the following system configuration, with the same hardware and Windows Server™ 2003 software used for both J2EE and .NET. The common Java client driver was run on two separate machines than the Web Service. Systems were connected via a 1GB Ethernet link.

Web Service host machine
Dell Power Edge 2650 2 x 3.06GHz w/ 2GB RAM
Windows Server 2003 Enterprise Edition with .NET 1.1 enabled
J2SE 1.4.2.0.5 SDK (uses 1GB heap)
JWSDP 1.4 with Tomcat 5

Benchmark client driver machines (2)
Unisys 4 x 3.0GHz w/ 8GB RAM
Windows Server 2003 Enterprise Edition
J2SE 1.4.2.05 SDK (uses 1GB heap)
JWSDP 1.4

The Java Web Services Developer Pack Version 1.4 was used for testing the JAX-RPC implementation. This pack includes Tomcat 5.0 as the Web server. Windows Server 2003 includes .NET 1.1 and IIS 6.0 as the Web server.

WSTest Configuration

For the results reported, WSTest was run with the following parameters set, with the mix changed to 100% for each of the Web Services tested:

Machines client driver run on
2

Config for each client machine
Agents = 8
RampUp = 300
SteadyState = 300
RampDown = 10
NumBytes = 1024
ListSize = 20, 100 and 200 to test three different message sizes

Windows Tuning

We did not find any of the following tuning parameters to have a material impact on either the Java or the .NET results. Instead we found the Windows® tuning performed by Sun below to be completely unnecessary, but we replicated it anyway to be consistent with Sun's tests:

Disabling the following services:
Alerter
ClipBook
Computer Browser
DHCP Client
DHCP Server
Fax Service
File Replication
Infrared Monitor
Internet Connection Sharing
Messenger
NetMeeting Remote Desktop Sharing
Network DDE
Network DDE DSDM
NWLink NetBIOS
NWLink IPX/SPX
Print Spooler
TCP/IP NetBIOS Helper Service
Telephony
Telnet
Uninterruptible Power Supply

Setting registry keys
\\HKEY_LOCAL_MACHINE\System\Current_Control_Set\Services\Tcp\parameters:

TcpWindowSize = 0xFFFF
TcpMaxSendFree = 64240

.NET Tuning

The table below shows the .NET tuning applied by Sun and the .NET tuning applied by Microsoft in our replicated tests.

Key differences:

- We find no impact in changing the Garbage Collection mode for .NET and chose to leave it at its default setting
- It is unnecessary in this test on a 2-proc machine to increase the number of worker processes
- If Sun used a .NET client driver program vs. a Java driver program for their tests (their report does not say), they should have increased MaxNetworkConnections in machine.config on their client machine. The default setting of 2 throttles the number of outbound network connections for the client to make Web Service calls, and should have been set to at least equal the number of client threads being run. Since we used a common Java driver program for the benchmark, this client-side setting did not need to be applied in our tests.

Element Tuned	Sun’s Setting	Microsoft’s Setting
.NET Garbage Collection Mode	Server	Default (User)
IIS Logging	Disabled	Disabled
Worker Process Recycling	Disabled	Disabled
Pinging and Rapid Fail Protection	Disabled	Disabled
Number of Kernel Requests	Unlimited	Unlimited
Number of Worker Processes	4	1 (default value)
ASP.NET Session State	Disabled	Disabled
ASP.NET Authentication Mode	None	None
Debug Compilation	False	False
IIS Web Server Authentication Mode	Not specified (presumably default of integrated Windows Authentication)	Anonymous (integrated authentication disabled to match Java Tomcat authentication behavior)

Java Tuning

We precisely matched Sun’s Java tuning as follows:

1. The following parameters were changed from their default value for Tomcat in <JWSDP_HOME>\conf\server.xml :

```
<!-- Disable access log writing
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
prefix="access_log." suffix=".txt" resolveHosts="false"/>
-->

<!-- Disable Connector lookups of Non-SSL connections
<enableLookups="false"/>
-->

<!-- Set the minimium processors of Non-SSL connections
<minProcessors="8"/>
```

2. The JVM options for Catalina server in <JWSDP_HOME\jwsdpshared\bin\launcher.xml were set as follows :

```
<launch classname="org.apache.catalina.startup.Bootstrap" ... >
<jvmarg value="-server" />
<jvmarg value="-Xms256m" />
```

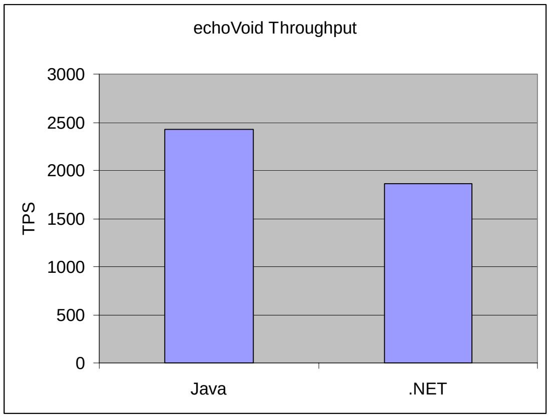
```
<jvmarg value="-Xmx256m" />
```

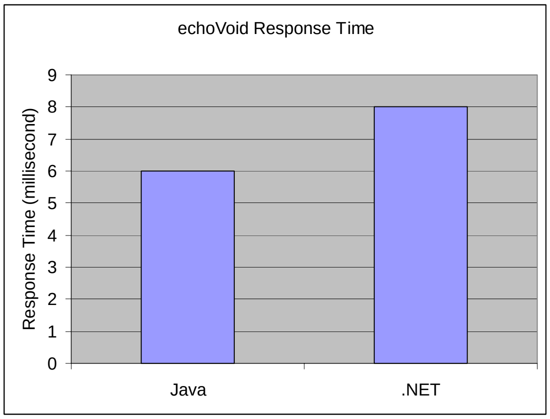
3. On the client side, the JVM options were changed in <WSTest_HOME>\Java\src\build.xml :

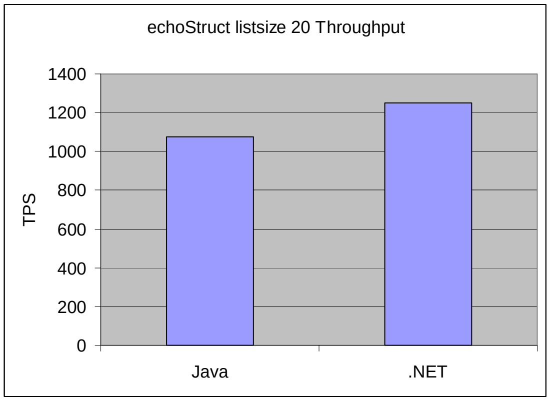
```
-Dhttp.maxConnections=256 -Xms300M -Xmx300M
```

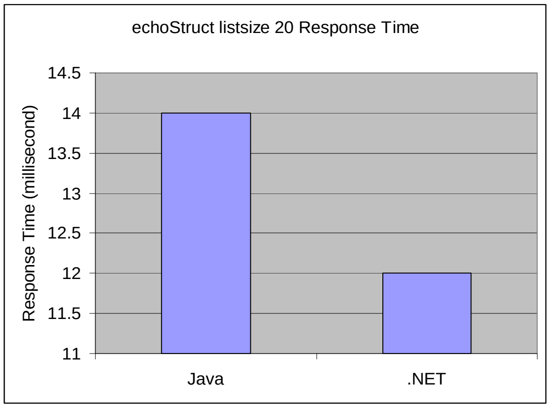
Performance Results

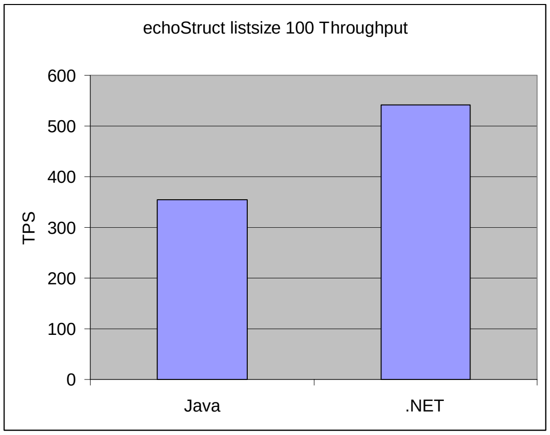
The measured throughput and response times obtained are shown graphically below, with throughput measured in transactions per second (higher is better) and response times measured in millseconds (lower is better). In the four simple tests Sun performed, .NET performed 2-3 times better than Sun reported across the board, beating the Java results in both the echoStruct and echoList tests even with a small list size of 20 (as Sun tested). As the list size was increased in these tests, hence increasing the XML SOAP message size, .NET exceeded Java performance by a wider and wider margin. For example, with a list size of 100 .NET performed 72% better in the echoList test, while at a list size of 200 .NET performed 82% better than Java. .NET also exceeded Java performance in the new test GetOrder.

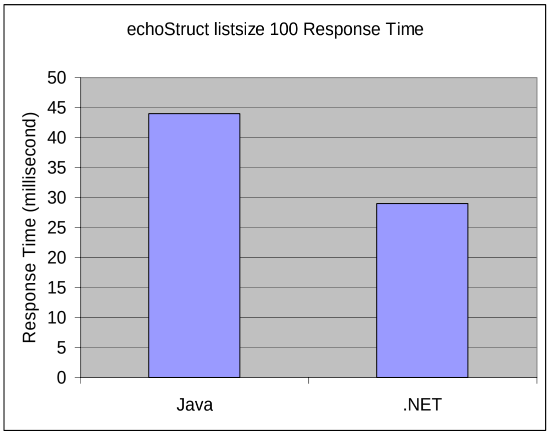

echoVoid Throughput

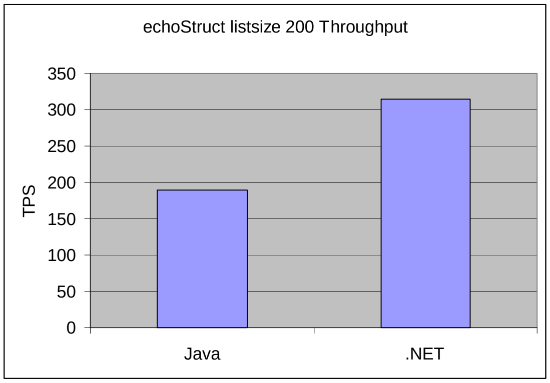

echoVoid Response Time

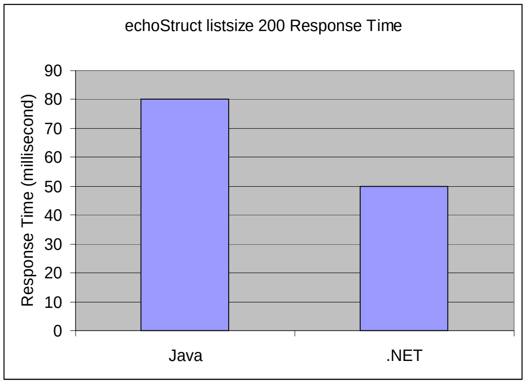

echoStruct listsize 20 Throughput

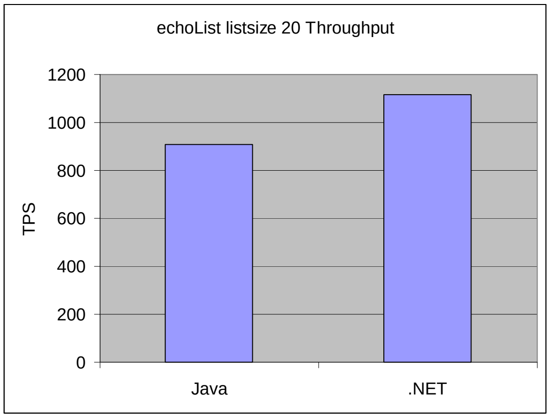

echoStruct listsize 20 Response Time

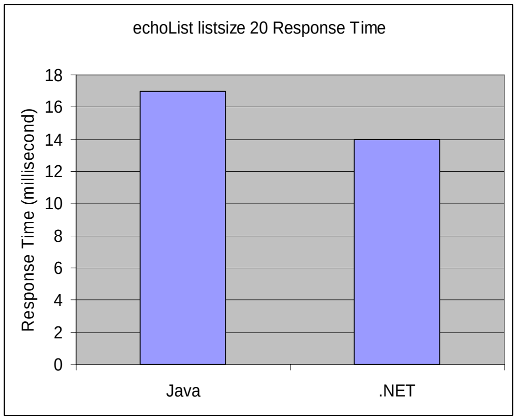

echoStruct listsize 100 Throughput

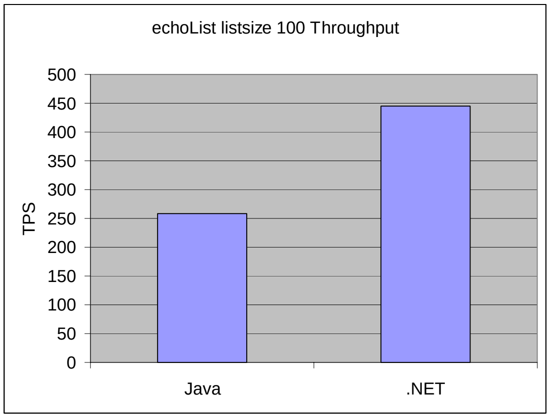

echoStruct listsize 100 Response Time

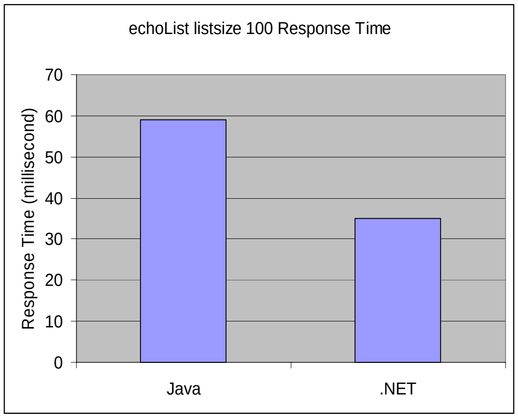

echoStruct listsize 200 Throughput
TPS
350
300
250
200
150
100
50
0
Java
.NET

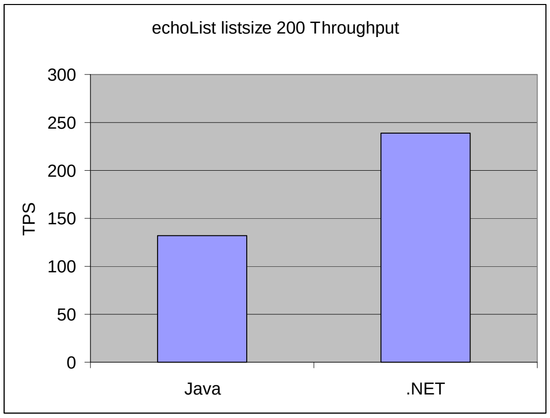

echoStruct listsize 200 Response Time
Response Time (millisecond)
90
80
70
60
50
40
30
20
10
0
Java
.NET

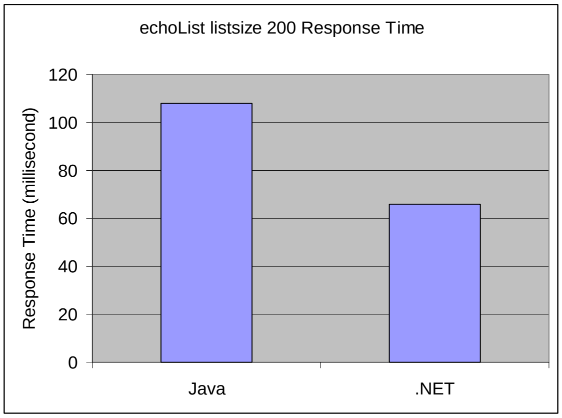

echoList listsize 20 Throughput
TPS
1200
1000
800
600
400
200
0
Java
.NET

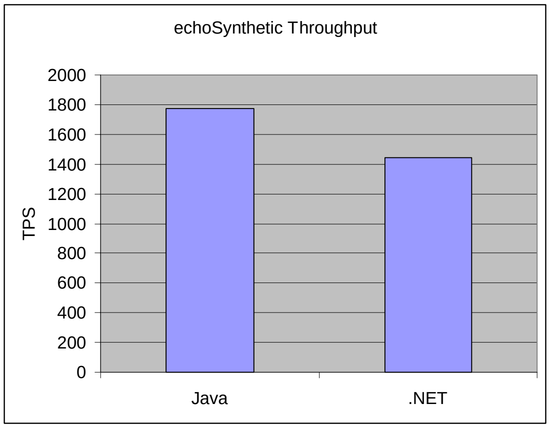

echoList listsize 20 Response Time
Response Time (millisecond)
18
16
14
12
10
8
6
4
2
0
Java
.NET

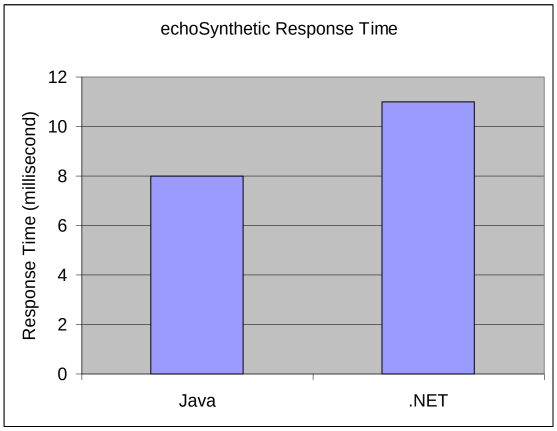

echoList listsize 100 Throughput
TPS
500
450
400
350
300
250
200
150
100
50
0
Java
.NET

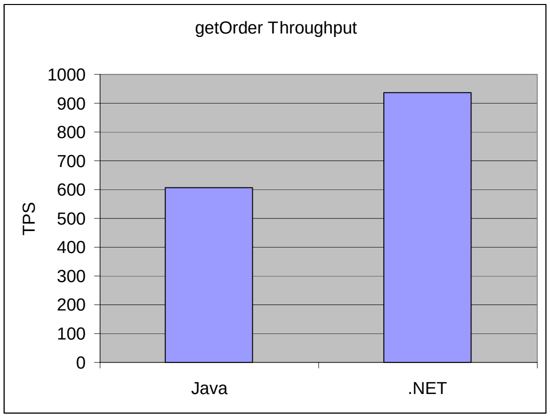

echoList listsize 100 Response Time
Response Time (millisecond)
70
60
50
40
30
20
10
0
Java
.NET

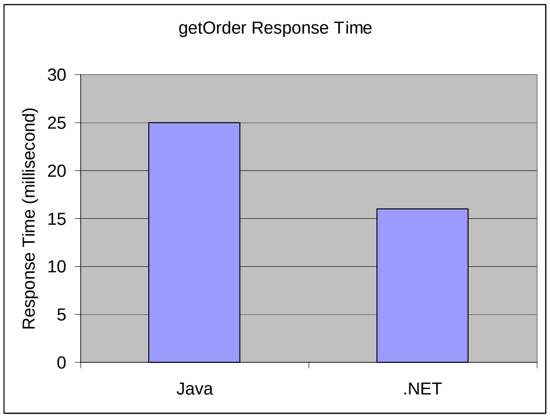

echoList listsize 200 Throughput
TPS
300
250
200
150
100
50
0
Java
.NET


echoList listsize 200 Response Time
Response Time (millisecond)
120
100
80
60
40
20
0
Java
.NET


echoSynthetic Throughput
TPS
2000
1800
1600
1400
1200
1000
800
600
400
200
0
Java
.NET


echoSynthetic Response Time
Response Time (millisecond)
12
10
8
6
4
2
0
Java
.NET


getOrder Throughput
TPS
1000
900
800
700
600
500
400
300
200
100
0
Java
.NET


getOrder Response Time
Response Time (millisecond)
30
25
20
15
10
5
0
Java
.NET

Conclusion

In this paper, we respond to the Sun representation of .NET vs. Java relative Web Service performance with corrected results for .NET, and expand on Sun's tests to show relative .NET vs. J2EE performance for more realistic Web Services that do more work. We provide all the source code including the .NET implementation, the Java implementation, and the Java driver program so customers can replicate the results. We believe we have accurately re-created Sun's original tests given that the Java results we achieve for Sun's four original tests very closely match Sun's reported results when taking into account the slightly faster server hardware we used in our testing (3 GHz vs. 2.6 GHz CPUs). However, we find the .NET results to be 2-3 times better than Sun reports. Finally, we find .NET to significantly exceed the performance of Sun's JWSDP 1.4 in tests involving larger message sizes. We encourage customers to download our benchmark kit and test the platforms for themselves; and also to examine the more comprehensive Doculabs @Bench Web Services benchmark which can be downloaded from http://www.theserverside.net/articles/content/DoculabsWebServiceScalability/DoculabsWebServiceScalability.pdf.