JAVA™ WEB SERVICES PERFORMANCE ANALYSIS AND BENEFITS OF FAST INFOSET

White Paper
October 2005

Table of Contents

Glossary of Terms

ANS.1: American National Standards Institute (also ANSI)

ASCII: American Standard Code for Information Interchange

DOM: Document Object Model

EAR: Enterprise Archive

EJB™: Enterprise JavaBeans™

JAR: Java™ Application Resource

JAX: Java API for XML

JAXB: Java Architecture for XML Binding

JVM: Java Virtual Machine

J2EE™: Java Enterprise Edition (Java EE) formerly called Java 2 Platform, Enterprise Edition

OAGIS: Open Applications Group Integration Specification

RPC: Remote Procedure Call

SOA: Service Oriented Architecture

SOAP: Simple Object Access Protocol

STaX: Streaming API for XML

TPS: Transactions Per Second

WSDL: Web Services Description Language

XML: eXtensible Markup Language

Overview

Service Oriented Architecture (SOA) has become the industry's de facto standard to build and deploy complex composite applications with greater business agility and improved efficiencies for trading partners and customers. Web services, the predominant underlying technology of SOA, have continued to evolve in areas of security, interoperability, reliability, and performance. This paper provides detailed analysis and realities of newly available technologies, such as Fast Infoset (FI), which provide dramatically improved Web services performance and scalability.

To access how the Java™ EE compatible, Sun Java System Application Server 8.1 leverages Fast Infoset and its performance impact, the publicly available Web Services performance test kit from PushToTest (the SOA Kit) was used. These tests were conducted on similar environments as were used in past Web services published benchmarks on products such as BEA WebLogic Server 8.1, Oracle® Application Server 10g, IBM WebSphere Application Server V6.0, JBoss Application Server 4.0, and Apache Geronimo. These elected to perform benchmark comparisons without the benefit of Fast Infoset.

Sun Java™ System Application Server

The Java System Application Server provides a Java EE 1.4 compatible platform for developing and delivering server-side Java applications and Web services. This application server is the industry's first, and most popular, production Java EE 1.4 application server. Focused mainly on developer productivity, the full-featured, high-performance, small-footprint container is free for development, deployment, and redistribution. This edition is ideal for embedding and bundling, and is already included with NetBeans™, Java Studio Creator, Java Studio Enterprise, and the Solaris™ OS.

In addition, Java System Application Server is the core run-time engine for the Java Enterprise System, a revolutionary, subscription-based approach to infrastructure software that reduces cost and complexity throughout data centers by providing fully integrated, end-to-end infrastructure software suites.

Fast Infoset in Java Web Services Developer Pack

In addition to the Web services technologies provided out of the box in the Java System Application Server 8.1, the add-on Java Web Services Developer Pack (WDSDP) provides developers and customers with quick, incremental updates that can be used to build, test, and deploy XML applications, Web services, and Web applications with the latest Web services technologies and standard implementations.

The Java WSDP is also a completely free, integrated toolkit. With the new 1.6 release, developers are now able to:

1. Develop and deploy applications using the latest XML and Web services technologies slated for inclusion in Sun's deployment platforms
2. Enhance Web services performance without revising WSDL files or application code
3. Create XML and Web services-enabled applications that exploit enhanced security features
4. Continue to enjoy Java interoperability and portability across different platforms and devices
5. Simplify and lower the cost of legacy application integration, data interchange, and publishing in Web environments

The Java WSDP 1.6 release contains Fast Infoset technology that can increase Web services performance two to four times by using ANS.1-based binary encoding that decreases transmission and processing times for messages, compared to XML and ASCII messages with zero code changes[5]. The Java WSDP also includes a preview of next-generation, XML Web services security, a preview of the Service Registry for SOA applications, updates to existing Web services technologies previously released in the Java WSDP, and guidelines for developing client-side Web services.

PushToTest SOA Kit Analysis of Fast Infoset

The PushToTest SOA Kit[4] is a performance workload consisting of a set of Web services and components that leverage the "Patterns and Strategies for Building Document-Based Web Services"[4] developed at Sun. The workload has three main modules called SushiBoats, TheBuffet, and TVDinner, which invoke Web services implemented as EJB end points. The service implementation uses three main parsing technologies: StAX, DOM, and JAXB.

During the course of analyzing this workload with Fast Infoset and porting it to the Java System Application Server, several issues and disparities were uncovered in past Web services SOA Kit benchmark publications for:

- BEA WebLogic Server (WebLogic) 8.1
- Oracle Application Server (Oracle) 10g
- IBM WebSphere Application Server (IBM) V6.0
- JBoss Application Server (JBoss) 4.0
- Apache Geronimo

These include but are not limited to:

1. Irregularities of the SOA Kit when applied across Java EE application server vendors:
   - The TVDinner modules in WebLogic, IBM, and JBoss can parse XML only when passed as a string in the WSDL. TVDinner module has disparate WSDL files. Oracle uses `Element`; IBM, JBoss, and Weblogic use strings.
   - debug= "on" is set for Oracle and JBoss builds in javac targets.
   - All WSDL files are document literal for Weblogic, but Oracle is set as RPC encoded.
   - It's unclear if client and server were on the same or different machines. In looking at the build scripts and code, it appears they were local (URLs appear to be hard-coded as http://localhost).
   - Several WSDLs use `xsd:anyType` to pass data. Use of `anyType` is not standard. `anyType` JAX-RPC 4.2.1. The JAX-RPC specification does not define the standard Java mapping for the `xsd:anyType`. The preferred mapping is `xsd:any` [4].
   - Methods throw a generic java.lang.Exception, rather than application-specific faults.
   - The BEA code is built to run on Weblogic 9.0 Beta. The current version is Weblogic 9.0 (GA). The code did not build and deploy out of the box on this version, and had to be debugged.
   - The benchmark uses OAGIS schemas, and the version used is stated as being as 8.1. However, from the OAGIS Web site[1], only version 9.0 and 8.0SP3 were available for download.
   - `TVDinnerDPL.createPPO_XML` uses XMLBeans, irrespective of which application server was being used and tested.
   - EARs created are missing `ejb-jar.xml` files, and must be added manually.

- The code is not clean and had several issues. Some examples include:
  - The same value is returned irrespective of the load test being run.

```
// Loads in the corresponding string:
        switch (pPayload){
              case 1 :
                    sOAGIS_P1 = poDoc.xmlText();
                    break;
              case 2 :
                    sOAGIS_P2 = poDoc.xmlText();
                    break;
              case 3 :
                    sOAGIS_P3 = poDoc.xmlText();
                    break;
              }
```

  - Code contains System.outs in the critical code path. For example, `StaxService.java)`

```
System.out.print("getRequestValues \n");
```

2) The code seems to work differently for different application servers, so it is not possible to make a fair, apples-to-apples comparison. For example, in the TVDinner server (XMLBeansService.java) for BEA, part of the payload passes as a primitive long:

```
//Set the value of this instance to the result
 xmlint_usrarea.setLongValue(crcOrderQty.getValue());
```

Whereas for other servers, it creates a memory-intensive DOM object for the same functionality:

```
DocumentBuilderFactory dbf =
                         DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder( );
Document domdocObject =
db.getDOMImplementation().createDocument( NS,"CRC32",null);
      // The element CRC32 is created and appended
domdocObject.getDocumentElement().appendChild
(domdocObject.createTextNode(Long.toString
(crcOrderQty.getValue())));
```

3) The bundle released by PushToTest appears to be incomplete and is missing several pieces, including:

- Failure of the DPL package to compile. The source is either out of sync or incomplete. `TVDinnerDPL` uses classes that are not included with the download. For example: `ProcessPurchaseOrderDocument`, `ProcessPurchaseOrderDataArea`, `PurchaseOrderSubLine`, and so on.
- The original schemas used to define all the documents and generate the JAXB and XMLBeans classes are missing (for example, the OAGIS schema). Because of this, we could not generate the appropriate JAXB code.

NOTE: The build target/tree for generating the XMLBeans and JAXB classes is not included in any of the files. This appears to have been done manually, then imported into Eclipse and a JAR file created. It is unclear why a simple ant target was not written instead.

4) There are fundamental issues with SOA Kit benchmarking methodology in past WS benchmark publications for BEA WebLogic Server 8.1, Oracle Application Server 10g, IBM WebSphere Application Server V6.0, JBoss Application Server 4.0, and Apache Geronimo.

- The tests were run from a single JVM client and requests sent to the server. However, the number of client threads was fixed and performance was measured by comparing the magnitude of CPU utilization. We believe that the real test of an application server's capability to perform under load can only be measured by saturating the CPU until no further load can be processed by the server. Several scalability studies that were completed previously and recommendations for tuning, including those at BEA[2], have taken this approach. Measuring or reaching a peak throughput without full saturation of the CPU is an indicator of a performance bottleneck such as I/O contention, thread-synchronization, or incorrect configuration.
- The testing tool employed is TestMaker which, upon inspection and use, appears to more of a unit testing tool and less of a load generator. Using the documentation and code provided by PushToTest, we could not get the results validated or duplicated on comparative or better hardware for BEA or any of the other application servers in question, using their client drivers and harness TestMaker. The TPS was severely limited, possibly due to threading or socket-handing issues in the framework, as well as its use of Jython (a derivative of the interpreted language Python, which was employed to script all the test cases). It is unclear how the harness was run, or if specific, undocumented tunings were applied to make it run during previous publications.
- The load from the clients is generated in a convoluted manner. Rather than having well-defined XML request documents, the schema is first compiled externally into Java using JAXB. Then, a sample XML document is unmarshalled, modified in memory, and marshalled out. Simpler and more efficient approaches could have been utilized, such as reading the requests from XML or generating them in their entirety.
- The client JVM is limited by the number of network sockets to five. By default, the JVM uses http.keepAlive=true and limits the number of sockets to five. All the PushToTest load tests were run with this default setting. It is unclear if this was, in fact, a throttling bottleneck.
- The application design also seems to have certain issues dictated by the initial design choices that were made. The majority of the WSDLs pass the XML document as a xsd:String or xsd:anyType. While these are both valid strategies, they are clearly not the most common use case employed in building document-based Web services. The common customer use case involves binding the schemas directly in the WSDL (known as the *XML in the Body Strategy*), or utilizing xsd:any to build schema-independent, polymorphic processors.

We believe that with the advent of standardized and advanced APIs, like JAX-Web Services (WS) 2.0 and JAXB 2.0, the embedded XML in the Body Strategy will become even more prevalent. Table 1 summarizes a comparison of some possible strategies, as well as their advantages and disadvantages.

Table 1. Comparison of Possible Strategies

Strategy	Advantages	Disadvantages
Using string	• Simple, same as writing a "hello world" application.	• Schema validation offered by the runtime cannot be used, and errors with the document will not be picked up until the service has read the document in memory and attempted to process it. • Service interface is not descriptive because the document type is just a general string. • Schemas must be negotiated out of band. Both service provider and consumer need *a priori* knowledge of the contents of the payload, because the WSDL file does not describe the schema of the expected documents.
Using `xsd:any`	• The mapping of the xsd:any has been standardized to map to SOAPElement with JAX-RPC 1.1. • Even though an element is named in the WSDL (for example, Business-DocumentRequest) and the business document passed appears inside these elements on the wire, the Web services client can still work with complete XML documents and maintain schema integrity without having to include document content under these elements (this is not the case with the anyType strategy).	• Requires developers to work at the lower levels of XML, because they now have to work with creating and manipulating SOAPElement objects. • There is no cohesiveness between the WSDL and the documents, because the schemas defining the documents are not referenced directly. • Schemas need to be negotiated out of band. Both service provider and consumer need *a priori* knowledge of the contents of the payload, because the WSDL file does not describe the schema of the expected documents.

Strategy	Advantages	Disadvantages
Using `xsd:anyType`	• Allows the action and payload to be passed together. This can be useful when creating a polymorphic processor that accepts multiple document types with the same actions, for example, a single service that performs a search action on a purchase order and an invoice, both of which conform to different schemas.	• JAX-RPC specification does not define standard Java mapping for the xsd:anyType, so not all imple mentations will behave like the Java WSDP and map to a SOAP-Element. In fact, support for the xsd:anyType is optional for an implementation. • Because the anyType actually defines the data type for a named element in the WSDL, the business document being passed in the SOAP body is located inside the element identified in the WSDL. For example, the PurchaseOrder is inside the BusinessDocumentRequest element. This means that the document being passed must either: • Have its root element identified in the WSDL • Be constructed appropriately or wrapped in the element on the fly
Using schema-defined types *(XML in the Body)*	• Interoperability. • Validate against schema if XML docs are used. • Better performance than encoded formatting styles. • Service interface clearly describes the types of documents expected. This makes the WSDL file easier to understand for clients	• Cannot use custom bindings or binding frameworks directly.

Test Environment

The software leveraged the Java System Application Server 8.1 UR2 with Java WSDP 1.6, which was Fast Infoset enabled and disabled to quantify performance gains of Fast Infoset in a standard manner. We employed comparable Windows 2003 Server hardware with different CPU speeds on the Sun™ machines. The major emphasis was to measure the performance gains of Fast Infoset, not to directly compare it to PushToTest past benchmark publications[5], mainly because of issues around reproducibility.

The hardware configuration used was similar to the configuration used by the original publications, which was an HP ProLiant DL560R01 Server with 2 x Xeon MP, 3.0-GHz, 1-GB DDR SDRAM, and Compaq Smart Array 5i Plus dual-channel 64 MB, plus:

- Ultra 160 SCSI Integrated RAID controller supporting 0, 1, 1+0, 5 RAID Level bus speed 400-MHz FSB
- Operating System: Windows 2003 Server

WSTest Driver

Since the client TestMaker driver provided in the PushToTest SOA Kit distribution had many issues and was nonfunctional, the client-side WSTest Driver[2] was used to drive the load on the end points. WSTest is an open source, Web services benchmark published by Sun in 2004, and subsequently used by other vendors to test Web services performance.

WSTest simulates a multithreaded server program that makes multiple Web services calls in parallel. WSTest measures the throughput of a system handling multiple types of Web service requests. This notion of a Web services operation corresponds to a request/response cycle. WSTest reports the Throughput-Average number of Web services operations executed per second and the Response Time-Average time taken to process a request. These metrics are reported for all tested operations.

WSTest reads these properties at initialization into an in-memory structure that is then accessed by each thread to initiate an operation as per the defined mix. A new operation is started as soon as a prior operation is completed (there is no think time). The number of operations executed and the response times are accumulated during the SteadyState period, and reported at the end of the run.

Methodology

1. Most use cases in the SOA Kit revolve around Web services that accept XML as string in the payload, and parse it using StAX, DOM, or JAXB protocols. Such strategies, though simple to design, suffer from the issues previously discussed. Additionally, these strategies are not optimal for high-performance encodings like Fast Infoset and others, even though a performance improvement is seen when Fast Infoset is enabled (see subsequent sections). For example, when an entire XML document is passed as a string, high-performance encodings like Fast Infoset cannot compute the structure of the XML document, and cannot optimize performance, which means they cannot demonstrate their true potential and benefits. For this purpose, we added an additional operation to the WSDL for the Java System Application Server that binds the payload to specific elements in the schema using the *XML in the Body Strategy* with document-literal formatting.

2. Tunings applied which adhered to those published by PushToTest were reapplied to the Java System Application Server with Fast Infoset enabled and disabled.

Summary of Results

When comparing performance and benefits, the Java System Application Server 8.1 UR2 with Fast Infoset provides dramatically enhanced Web services performance for both small and large data sets when tested on the PushToTest SOA Kit, across JAXB, DOM and StAX parsing technologies, and an incremental number of concurrent users. Details of the runs can be found in the following sections.

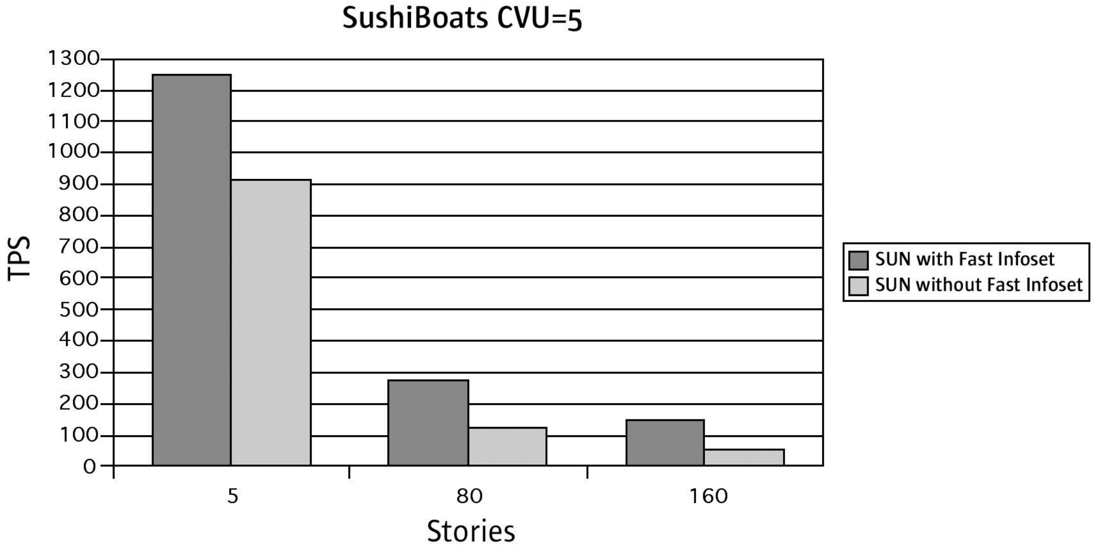
Fast Infoset use is completely transparent to end users. Fast Infoset can be enabled dynamically only when it detects Fast Infoset support in both client and server, otherwise the standard SOAP is used. The Java WSDP and future versions will continue to provide performance optimization as well as greater flexibility and interoperability across heterogeneous Java EE application server vendors.

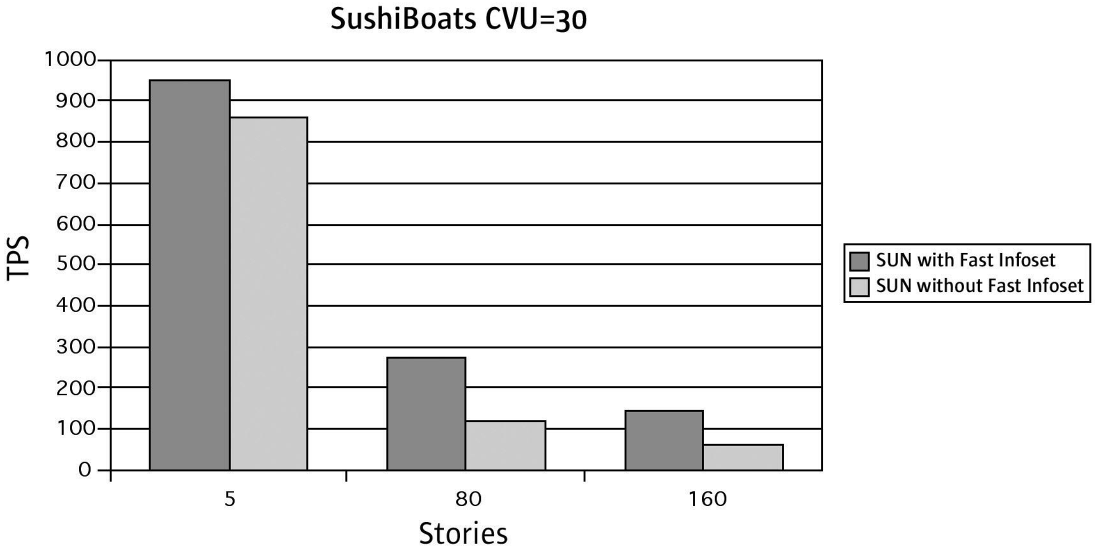
These results are not directly comparable to past PushToTest benchmark publications[5] due to differences in machine clock speeds, load generation clients, benchmarking methodologies, code irregularities, and other SOA Kit reproducibility issues outlined in this paper.

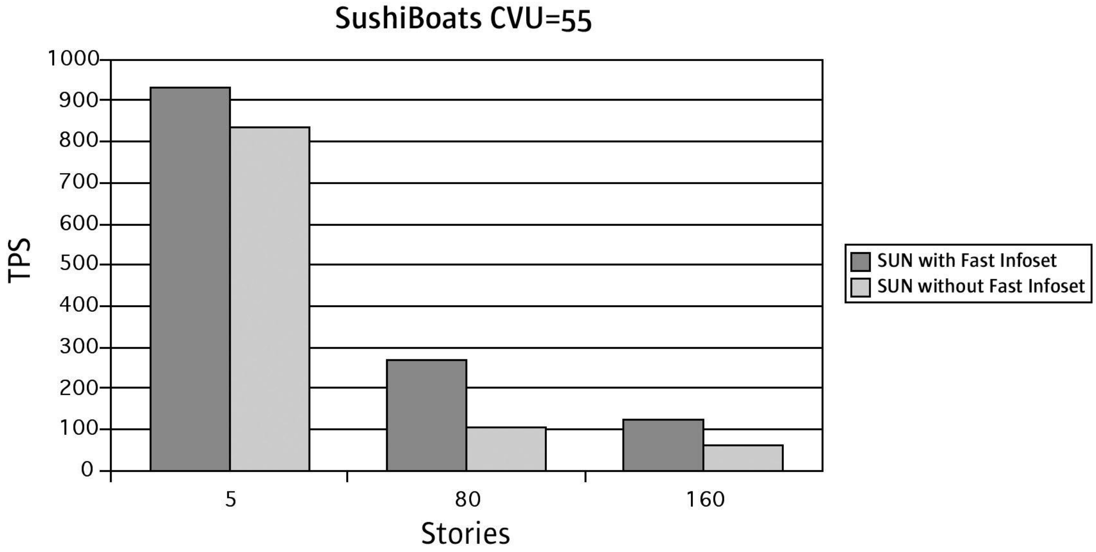
Graphic Summary of Results

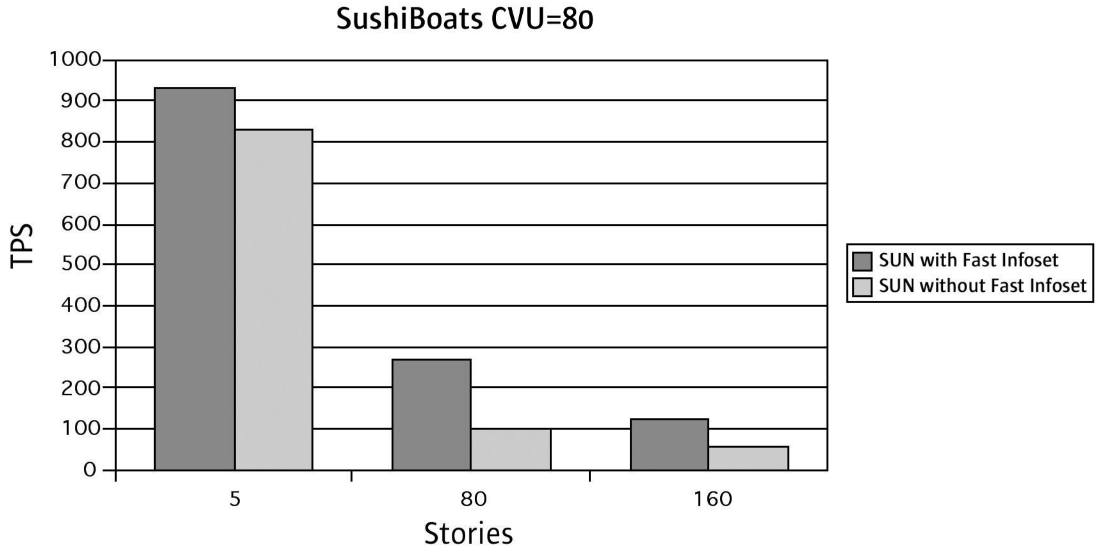
We ran two benchmarks on similar hardware in two sets. The first set was run with two CPUs on the server side and hyper threading turned on for the Xeon processors. The second set was run with four CPUs on the server side with hyper threading turned off for the CPUs. In both cases, results were almost identical. The diagrams show the results of the dual CPU runs, with varied Concurrent Virtual Users (CVUs).

SushiBoats Module — Endpoint does StAX Processing

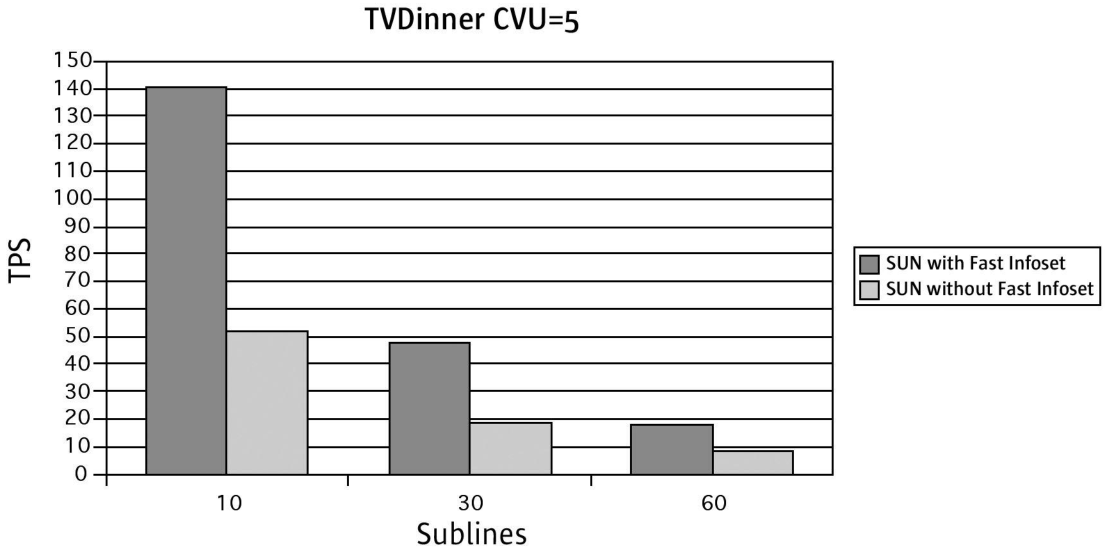


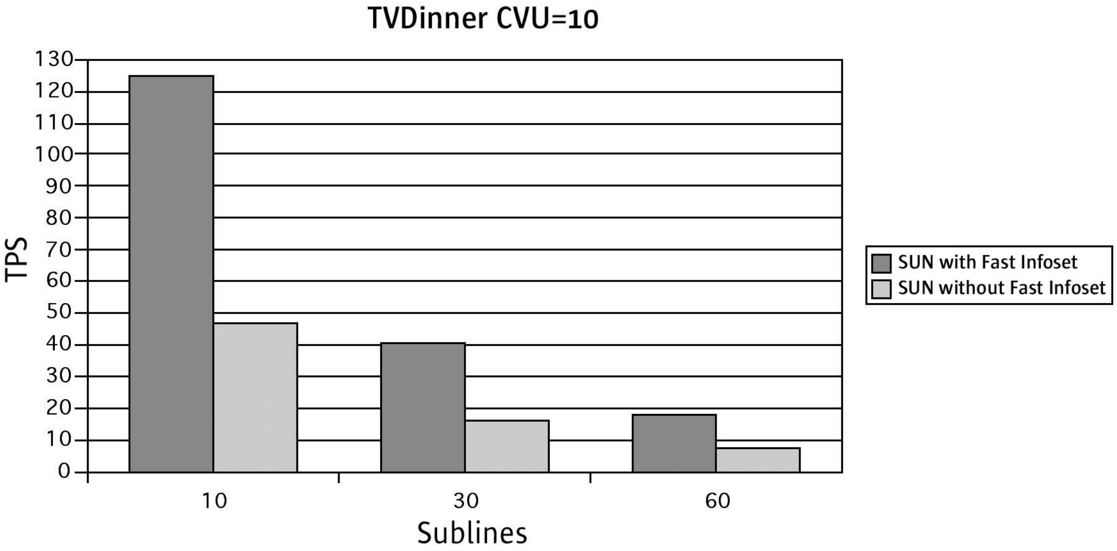

SushiBoats CVU=30
TPS
1000
900
800
700
600
500
400
300
200
100
0
5
80
160
Stories
SUN with Fast Infoset
SUN without Fast Infoset

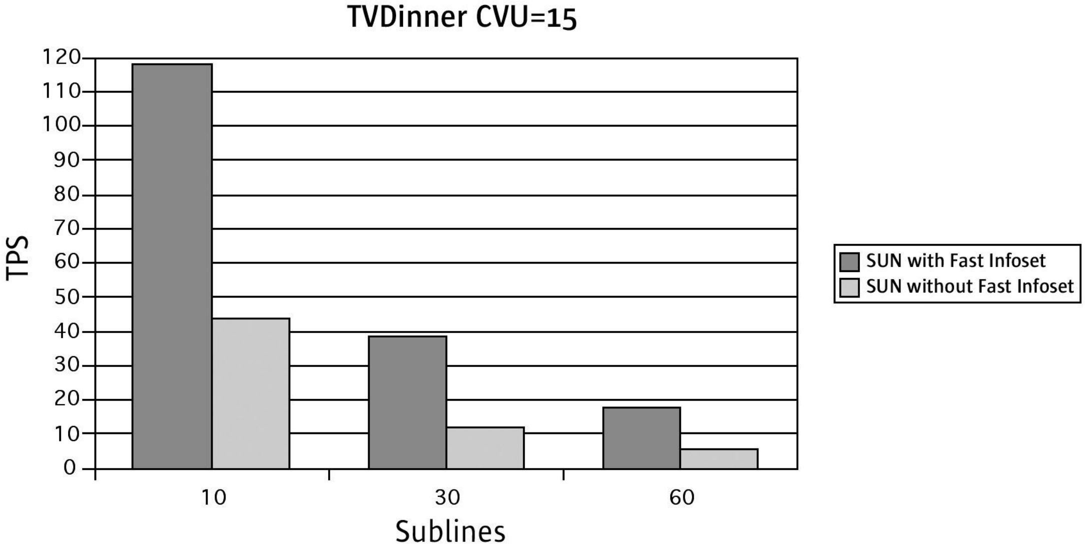

SushiBoats CVU=55
TPS
1000
900
800
700
600
500
400
300
200
100
0
5
80
160
Stories
SUN with Fast Infoset
SUN without Fast Infoset

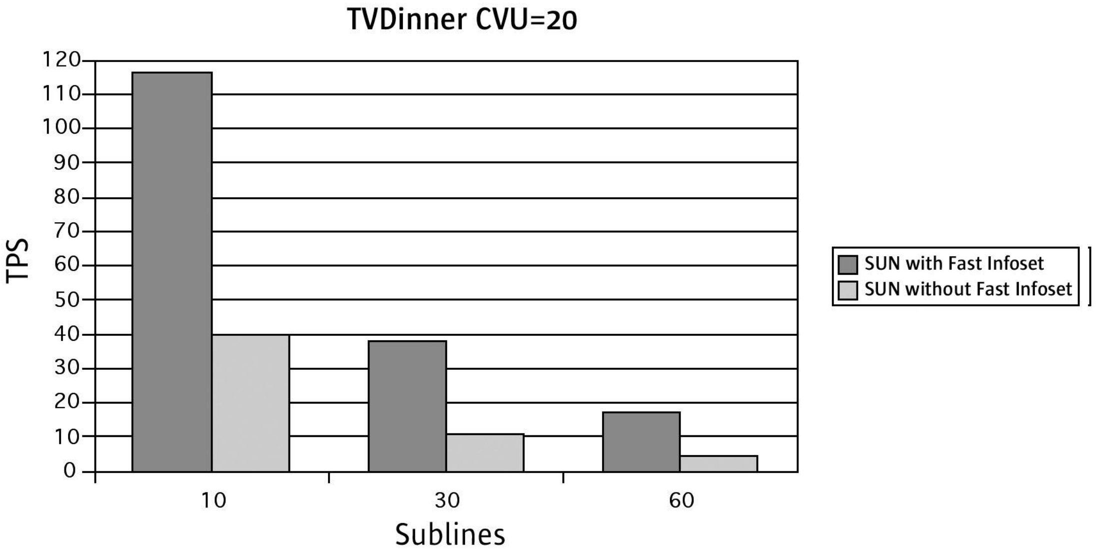


TVDinner Module — Endpoint does JAXB Processing

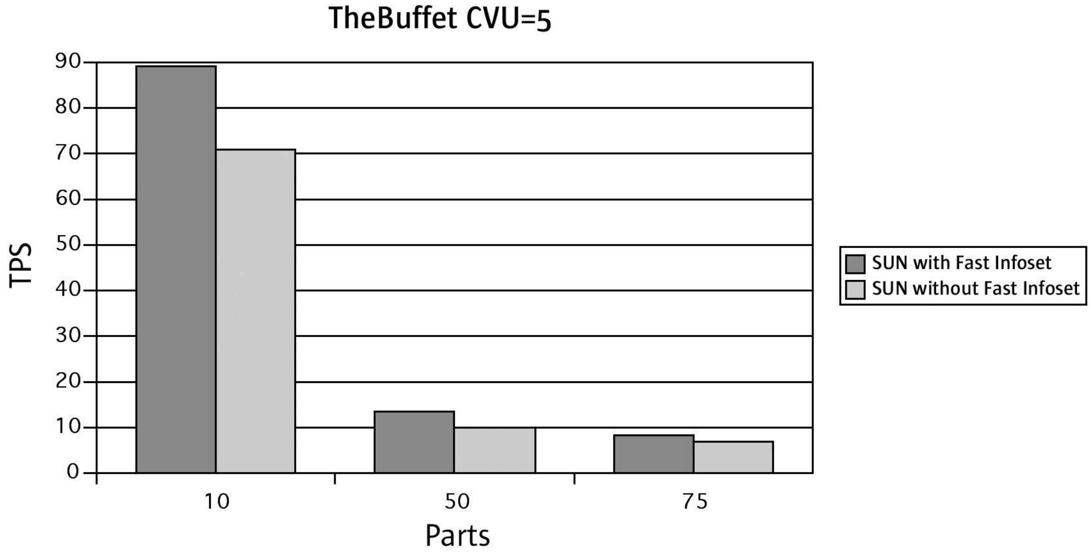


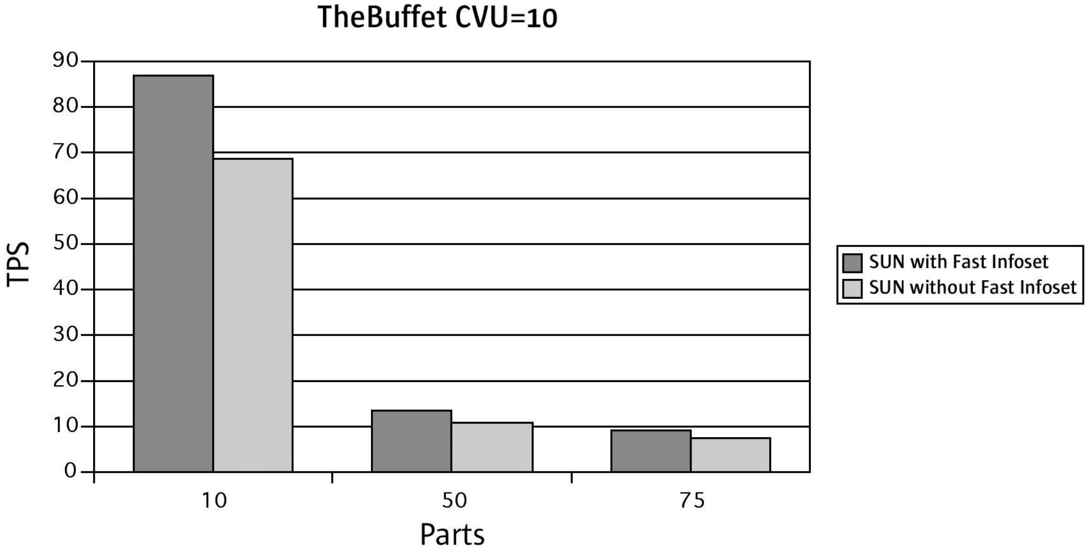

TVDinner CVU=10
130
120
110
100
90
80
70
60
50
40
30
20
10
0
TPS
10
30
60
Sublines
SUN with Fast Infoset
SUN without Fast Infoset

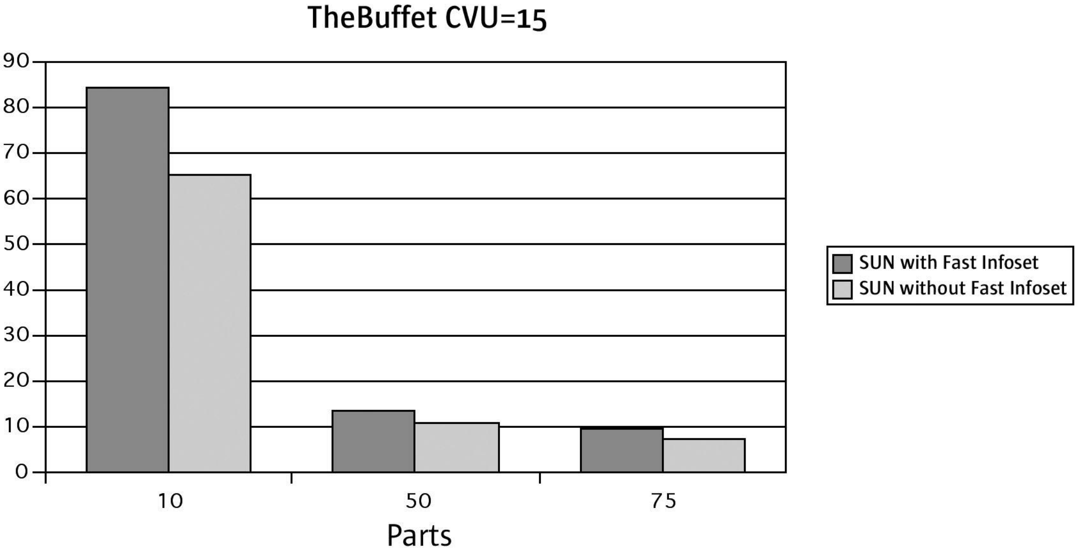

TVDinner CVU=15
120
110
100
90
80
70
60
50
40
30
20
10
0
TPS
10
30
60
Sublines
SUN with Fast Infoset
SUN without Fast Infoset

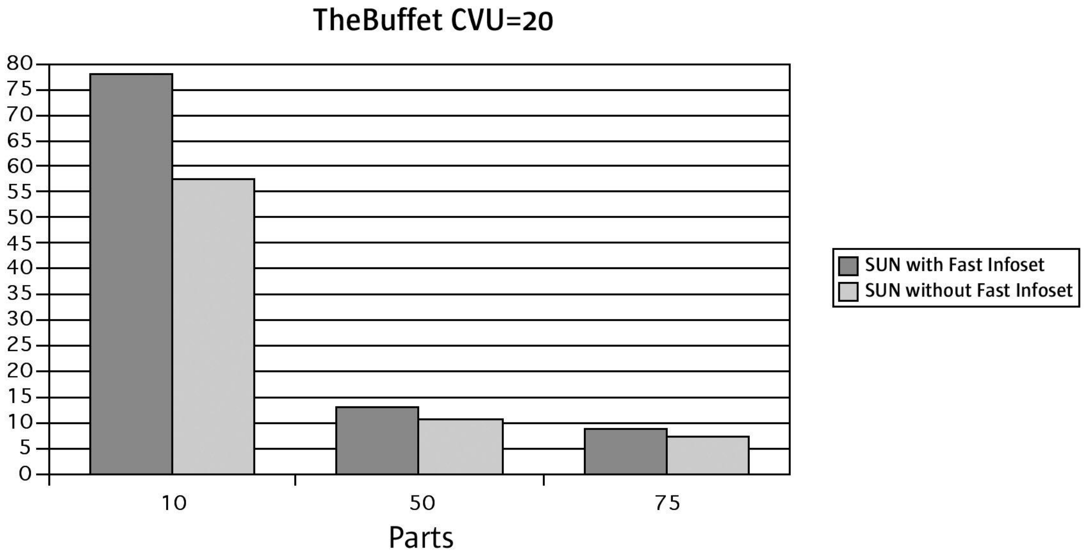


TheBuffet Module — Endpoint does DOM Processing (with String param)

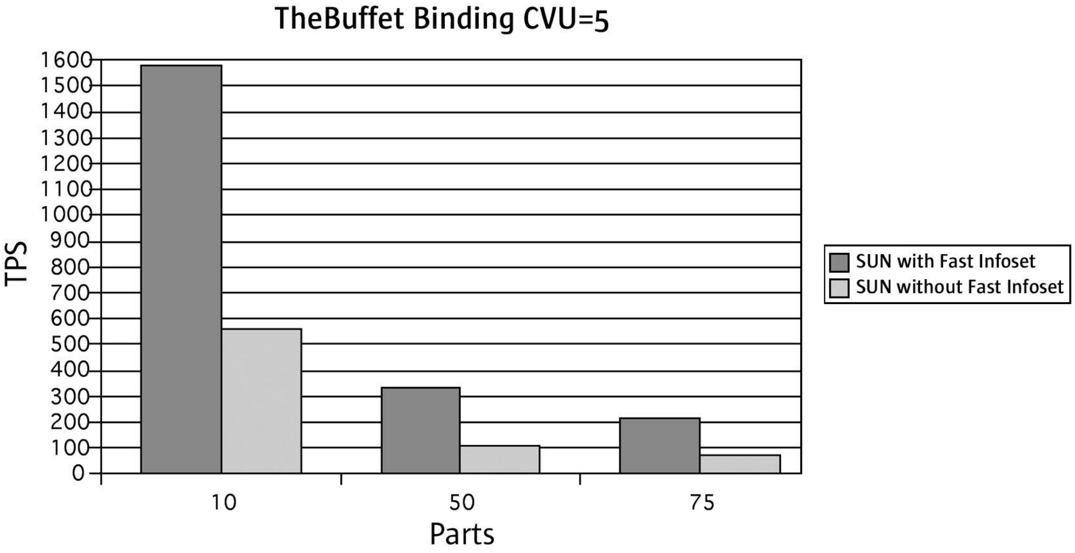


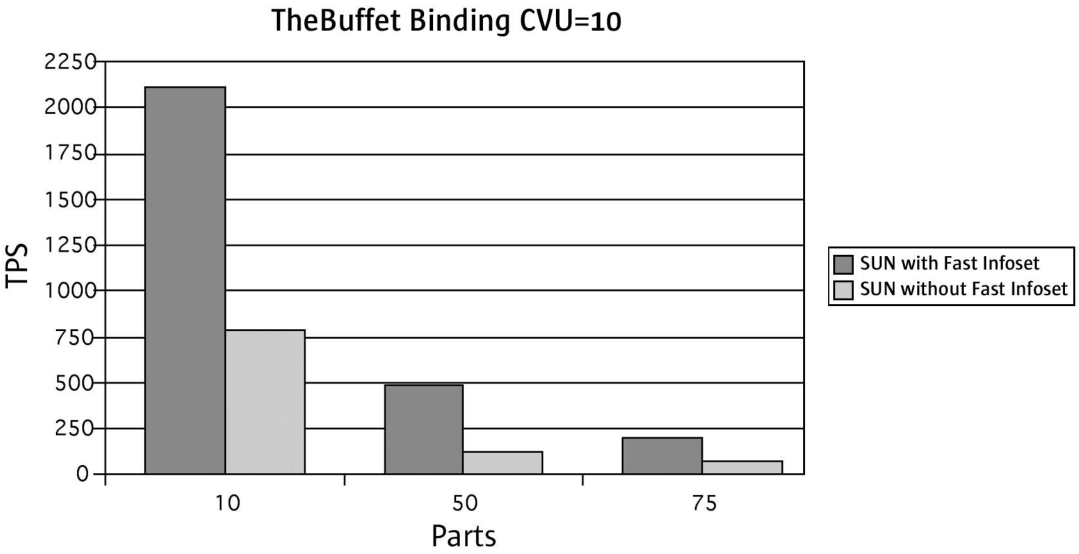

TheBuffet CVU=10
TPS
90
80
70
60
50
40
30
20
10
0
10
50
75
Parts
SUN with Fast Infoset
SUN without Fast Infoset

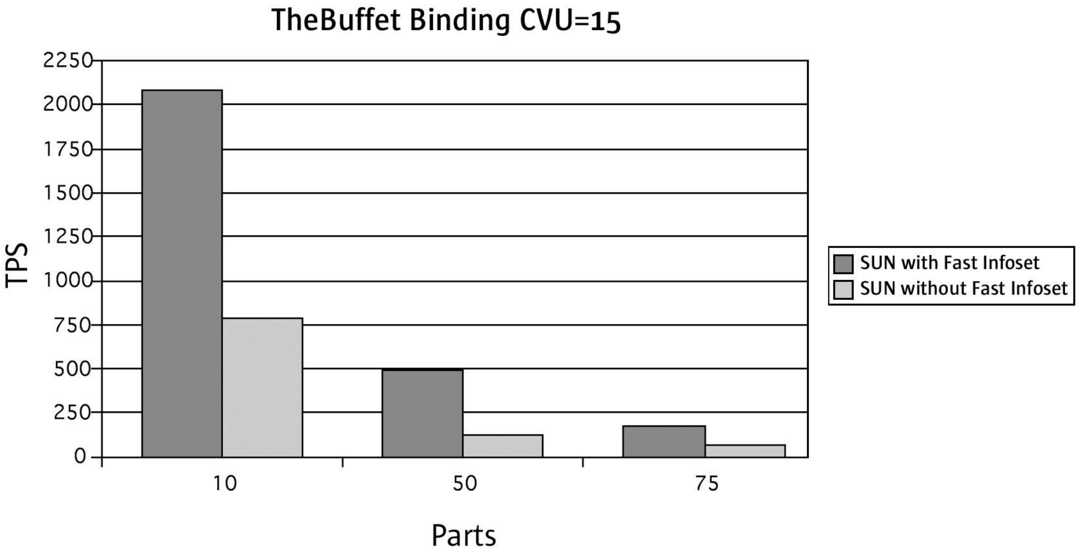

TheBuffet CVU=15
90
80
70
60
50
40
30
20
10
0
10
50
75
Parts
SUN with Fast Infoset
SUN without Fast Infoset

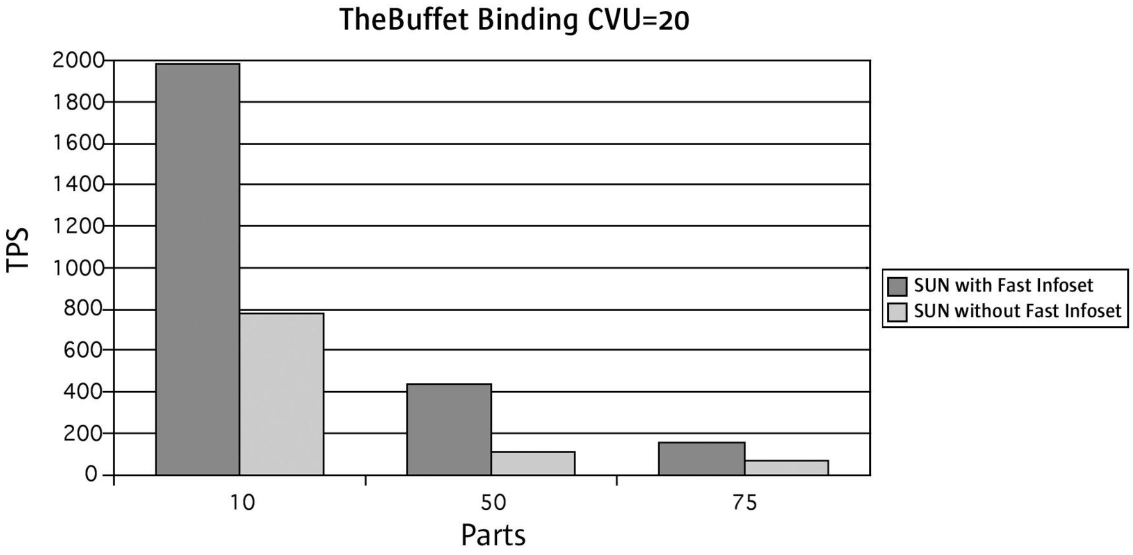


TheBuffet Module with Binding — Endpoint does DOM Processing (with doc/literal schema binding)




TheBuffet Binding CVU=10
TPS
2250
2000
1750
1500
1250
1000
750
500
250
0
10
50
75
Parts
SUN with Fast Infoset
SUN without Fast Infoset


TheBuffet Binding CVU=15
TPS
2250
2000
1750
1500
1250
1000
750
500
250
0
10
50
75
Parts
SUN with Fast Infoset
SUN without Fast Infoset


TheBuffet Binding CVU=20
TPS
2000
1800
1600
1400
1200
1000
800
600
400
200
0
10
50
75
Parts
SUN with Fast Infoset
SUN without Fast Infoset

References

1. OAGIS
   www.openapplications.org/
2. WSTest Web Services Benchmark
   wstest.dev.java.net/
3. SPECjAppServer2002 Performance Tuning
   dev2dev.bea.com/pub/a/2004/01/chow_deisher.html
4. Patterns and Strategies for Building Document-Based Web Services
   java.sun.com/developer/technicalArticles/xml/jaxrpcpatterns/index.html
5. Using Fast Infoset
   java.sun.com/webservices/docs/1.6/jaxrpc/fastinfoset/manual.html#Using-FI
6. PushToTest Benchmark
   www.pushtotest.com

Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA **Phone** 1-650-960-1300 or 1-800-555-9SUN **Web** sun.com