Sun Open Net Environment (Sun ONE) Application Server 7 Performance

A Technical White Paper

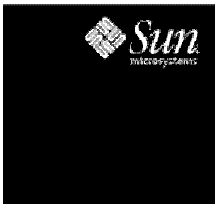

Table of Contents

Executive Summary

This release of the Sun™ ONE Application Server is based on the reference implementation of the Java™ 2 Platform, Enterprise Edition (J2EE™) 1.3 version with significant usability, performance, and scalability improvements to deliver a best-in-class development and deployment platform. By delivering an implementation of the application server that is built upon the very definition of the J2EE 1.3 specification, developers, integrators, and enterprise operators are assured of an environment that maximizes productivity and performance, while capable of scaling to meet virtually any user load requirements.

The Sun ONE Application Server 7 is based on a new architecture that is a significant enhancement to previous Sun application server products. The Sun ONE Application Server 7 offers a server platform to rapidly and cost-effectively deliver Web services and other business-critical applications.

Performance improvements that are highlighted in this paper:

- Benchmark testing results from PushToTest show that the Sun ONE Application Server 7 is nearly twice as fast as leading competitors in transactions per second.
- *Network Computing* Magazine says: “Its ability to process requests as load increased made it stand out from the pack.”
- In a continuing effort to improve performance, business benchmark testing results show that the Sun ONE Application Server 7 PE Edition Update 1 is up to 90 percent faster than the 7.0 release.
- The Sun ONE Application Server outperformed major competitors in an Enterprise JavaBeans™ (EJB™) component application environment where a Data Access Objects (DAO) pattern is used to manage data.

In addition, standards are a major part of the story. The Sun ONE Application Server 7 is in full compliance with key specifications such as JAX RPC 1.03 and other components of the Java Web Services Developer Pack 1.1. This helps to ensure portability and protects application and infrastructure investment.

This paper provides details on performance characteristics of the Sun ONE Application Server 7, including the underlying technologies that help create across-the-board improvements. Because enterprise applications vary greatly, tuning tips, which can also help optimize performance, are included. In this release, the developer experience is streamlined and improved— a chapter discusses the highlights and benefits.

Architectural Overview

The Sun ONE Application Server 7 provides a robust, J2EE technology-based platform for the development, deployment, and management of e-commerce application services to a broad range of servers, clients, and devices. The Sun ONE Application Server 7 is compliant with J2EE 1.3 technology. Scalability (horizontal and vertical), high availability, reliability, performance, and standards compliance are the key goals of this architecture. The software is also a significant architectural departure from previous generations of the Sun ONE Application Server. Because it combines existing Sun ONE products and technologies with the J2EE 1.3 Reference Implementation (J2EE 1.3 RI), the Sun ONE Application Server 7 architecture is built upon proven technologies.

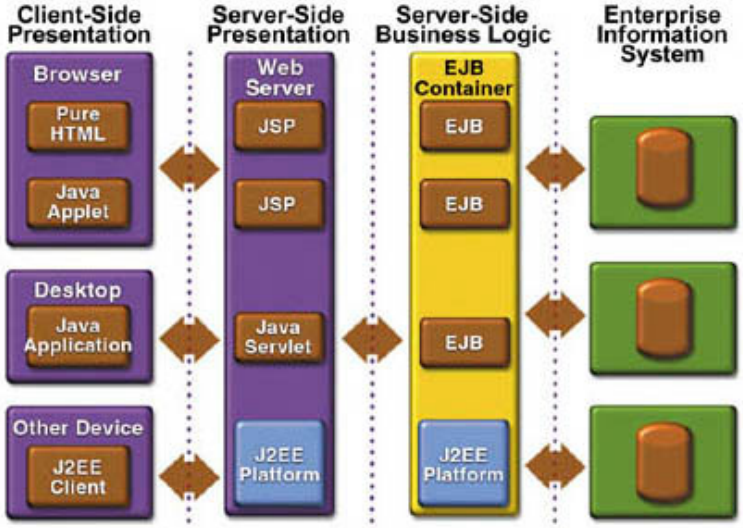

Figure 1: J2EE Application Architecture

As shown in Figure 1, the J2EE application model is very flexible, allowing the application architect to split application logic functionally into many tiers. The presentation layer is typically implemented using servlets and JavaServer Pages™ (JSP™) components, and executes in the Web container.

The Sun ONE Application Server 7 architecture illustrated in Figure 2 shows the component architecture, subsystems, access paths, and external entities interfacing with the core server.

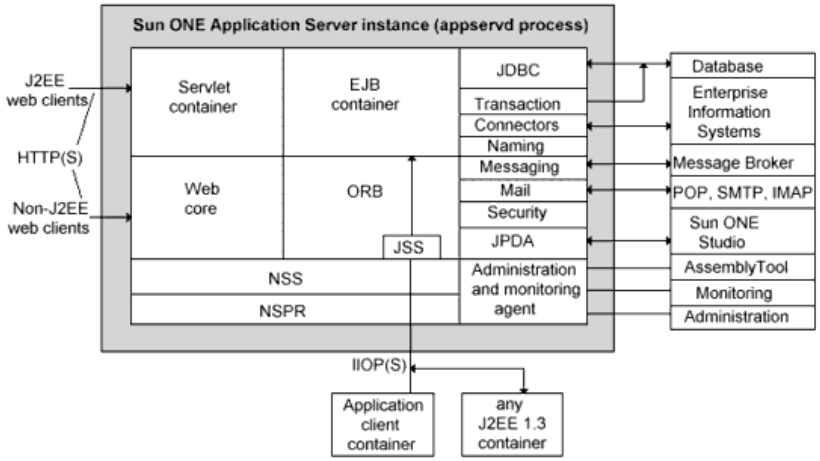

Figure 2: Sun ONE Application Server 7 Components

As Figure 2 illustrates, the Sun ONE Application Server 7 architecture is componentized, which results in a highly manageable architecture. All the services required by the J2EE specification are present, with well-defined standard interfaces to invoke them from within applications.

Application server instances form the basis of an application server deployment. The J2EE 1.3 Web and EJB containers are included in each application server instance. A proven, high-performance HTTP server is positioned in front of the Web container, while a built-in Object Request Broker (ORB) forms the underpinning of the EJB container. In support of access to backend systems, applications can leverage:

- J2EE Connector Architecture (JCA) support and third-party resource adapters
- Java Message Service (JMS) with either its built-in provider or third-party providers
- Any combination of popular third-party drivers supporting the JDBC™ API

Access to backend systems can be managed within the scope of distributed transactions using the built-in Java Transaction Manager.

The system is managed through the Administration Server. The Administrative Server houses the core administrative application and an SNMP agent. All remote management activity flows through the administrative server. Both command-line and Web browser-based administrative clients access the administrative server directly through HTTP, or securely through HTTP/S. The Web based administrative interface, new in the Sun ONE Application Server 7, provides an easily manageable server from remote locations, as well. For example, the server is designed such that an administration domain comprising of one administration server can administer multiple numbers of application servers.

In addition, a facility is available to configure a Web server installed on a separate system to act as a proxy and forward the requests to the application server instance. Web Server Proxy Plug-ins enable application server deployments behind one or more Web servers, which are housed in a demilitarized zone (DMZ) that is bracketed by one or more layers of firewalls. The plug-ins provide a means for the front-end, Web server tier to

direct incoming HTTP/S traffic received from the Internet to one or more application servers located in a backend application server tier.

A variety of client applications can access business services deployed to the application server. Web services and browser-based clients can use either HTTP or HTTP/S to access the Java Web services and J2EE Web applications. Java application clients can be deployed in a standalone mode or within a standard Application Client Container. They may use Java Remote Method Invocation over Internet Inter-ORB Protocol Technology (Java RMI-IIOP Technology) to access EJB components deployed to the application server. C++ language clients can use Java IDL™/IIOP API to access EJB components, as well.

The basic component of the Sun ONE Application Server is the `appservd` process, which is managed by a watchdog process. The application code runs in a multithreaded process created by the `appservd` process. The Java™ Virtual Machine (VM) is also started within `appservd`. (The terms *Java virtual machine* and *Java VM* mean a virtual machine for the Java™ platform.)

By employing the Java 2 Platform, Standard Edition (J2SE™) 1.4 technology for the server operation, the Sun ONE Application Server utilizes the enhanced abilities of this newer version of the Java Developer Kit (JDK™) to its advantage.

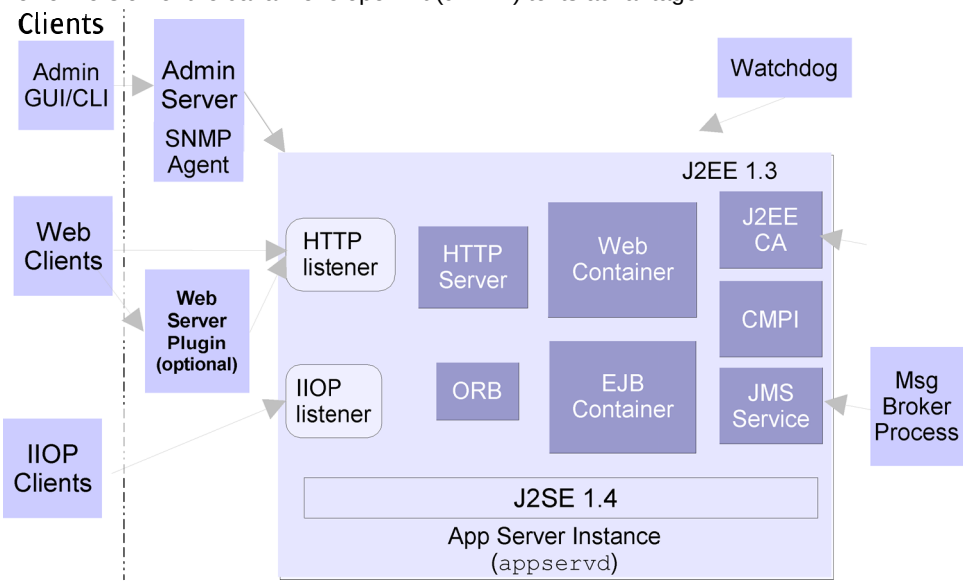

Figure 3: Sun ONE Application Server 7 Offers a Streamlined Runtime Environment

New Performance Technology in the Sun™ ONE Application Server 7

The Sun ONE Application Server 7 is a significant rewrite from previous versions of the application server. In addition to improvements in core application server components, high-performance subsystems are also included, such as the Sun ONE Web Server 6, the Sun ONE Message Queue software, and more.

Areas of improvement include:

Web Container

- The Java call stack is reduced, improving scalability.
- The number of times a Web container makes JNI calls is reduced — The JNI connects the Web container to the native Web server engine.
- `ThreadLocal` variables ware eliminated in the org.apache (Tomcat) code.
- Using information from various profiling tests, unnecessary string operations are identified and then reduced or eliminated.
- For JSP components, the number of system calls is reduced. In addition, support for precompiled JSP components is added. In certain benchmarks, this almost doubles the performance.

Java Technology

The Sun ONE Application Server 7 provides a J2EE certified platform, and its functionality and capabilities are based on Java technology. The Sun ONE Application Server 7 has been optimized to leverage the performance gains mentioned in this section. Additional information on general Java technology performance can be found at: java.sun.com/docs/performance.

Java technology improvements include:

- Reflective Method Invocation: The Java programming language can dynamically look up and call methods. In J2SE 1.4 technology, the mechanism that implements these features is reimplemented for dramatically improved performance. Figure 4 shows the relative performance on a simple benchmark that repeatedly calls small methods — performance is improved by a factor of 20 over J2SE 1.3.1 environments.

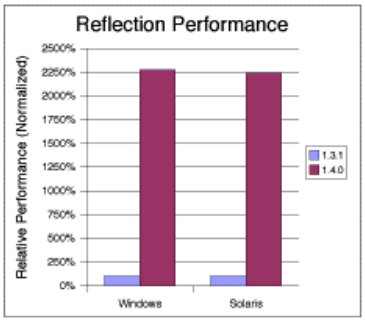

Figure 4: Reflective Method Invocation Performance Improvements

- JNI Method Invocation: In addition to the JNI features added to support the non-blocking I/O (NIO) framework, method calling through JNI is also improved when calling C-language methods from Java language methods, as well as in the reverse direction. NIO is an outgrowth of the Java Community Process, specifically JSR 051. The goal of this specification request is to define a set of new and improved I/O APIs for the Java platform. Many of these goals relate to improving the I/O performance of Java APIs. (For more information see java.sun.com/j2se/1.4/performance.guide.html.) The performance for calling simple native methods in J2SE version 1.4 improved by 74 percent over J2SE version 1.3.1. When calling Java language virtual methods from C, the performance improved by 38 percent.
- Garbage Collectors: New garbage collector (GC) algorithms have resulted in performance improvements, as well. In internal testing with middle-tier Java applications, throughput improved from 14 percent to 93 percent in the Java VM 1.4.2 over the Java VM 1.4.1, depending on the platform.
  In addition, two new GCs are available within the Sun ONE Application Server 7. The Java HotSpot™ VMs are generational. The generational algorithm provides efficient memory recycling and object aging. The Java VM heap is split into a *young generation* and an *old generation,* according to object age. The young generation is further split into one Eden and two Survivor spaces. For most applications, two-thirds of allocated objects die very young, are considered short-term objects, and can be collected in the young generation. Typically, the young generation is much smaller in size relative to the total heap size. This leads to frequent but short pauses in the young generation, but more memory is recovered per unit of collection work. However, objects that survive a sufficiently large number of young generation collections are considered old or long-term objects and are promoted or tenured to the old generation. Even though the old generation is typically larger, it eventually gets filled up and requires collection. This leads to less frequent but larger pauses in the old generation.
  For additional tuning tips, see *Tuning Garbage Collection with the 1.4.2 Java Virtual Machine* at: java.sun.com/docs/hotspot/gc1.4.2/index.html
  - Throughput Collector: A parallel young generation collection offers superior scalability on multi-CPU systems. Also known as the parallel collector, it is implemented in the young generation. This enables garbage collection to occur on multiple threads for better performance on multiprocessor machines. Even though it suspends all “mutators” (application threads), it is able to complete the

given amount of garbage collection work much more quickly, by leveraging all available CPUs on the system. This significantly reduces the GC pauses in the young generation. The parallel collector enables applications to scale to larger number of CPUs, as well as larger memory.

  - Short-Pause-Time Collector: A concurrent collection of the tenured generation dramatically reduces pause times on multi-CPU systems. Also known as the concurrent low-pause collector, it is implemented in the old generation. This collector mostly executes concurrently with the application. It trades the utilization of processing power that would otherwise be available to the application for shorter garbage collection pause times.

- Thread Management: The thread management used by the Solaris™ Operating System (OS) implementation of the Java VM offers dramatically improved scalability and eliminates thread starvation. In a benchmark launching worker threads equal to the number of CPUs, the threads are now properly load balanced, whereas with J2SE 1.3, certain threads could be starved for minutes at a time. This provides improved CPU utilization on multi-CPU systems using J2SE version 1.4 with the Solaris 8 OS or greater.

  For more information on threads, see *Threading* at:
  java.sun.com/docs/hotspot/threads/threads.html
- Business Transactions: The Standard Performance Evaluation Corporation offers a benchmark for evaluating server-side Java performance (SPECjbb2000, or the Java Business Benchmark). This benchmark models a three-tier system, the most common type of server-side Java application. SPECjbb2000 focuses on business logic, object manipulation, and the work of a middle-tier Java server workload. The benchmark represents a typical business-critical workload, including order entry, inquiry, and payment processing. The results of this benchmark show a performance improvement in J2SE version 1.4 of 58 percent over J2SE version 1.3.1. (source: java.sun.com/j2se/1.4/performance.guide.html)
- Dynamic Native Code Compilation: Numerous improvements were made to the Java HotSpot Server VM dynamic compiler. Some of the new optimizations include array bounds check elimination, various loop optimizations, and improved inlining.
- EJB Components: These components are a key element of the J2EE platform, and J2SE 1.4 shows a 34 percent improvement over J2SE 1.3.1 on a workload that uses EJB components to simulate a manufacturing and supply chain management application. Improvements in J2SE 1.4 that impact this workload include object serialization, thread management, and reflection.
- Java Servlets™: Another key piece of the J2EE platform. Again, J2SE 1.4 substantially improves performance. In a benchmark using Java Servlets to run an online bookstore, performance increased by 35 percent. Of particular importance are improvements made in the Java HotSpot Server VM dynamic compiler.

Performance Testing and Results

This section describes four performance tests. The tests highlight different areas of the Sun ONE Application Server 7 performance. Two were external, and two were performed by an internal group.

PushToTest

PushToTest is an independent testing organization offering a Performance Kit, which helps organizations to determine how well an application server performs. Recently, PushToTest conducted a test using the kit and found that the Sun ONE Application Server 7 was the fastest at running Web services when compared to application servers from leading competitors. Table 1 displays the results.

Table 1: PushToTest Testing Results

	Sun ONE Application Server 7	Competitor A	Competitor B
Agents	50	50	50
Number of completed transactions	7851	3791	3693
Transactions Per Second	43.62	21.18	20.52

The test was centered on Apache SOAP 2.1 making SOAP RPC encoded requests. The test ran for three minutes, with 50 concurrent test agents making requests.

- Client side: 1.7 GHz AMD XP processor running Windows 2000 SP3.
- Server side: 2x2 GHz AMD MP system running Windows 2000 SP3.

The results make it clear that the Sun ONE Application Server 7 is more than twice as fast, as measured by transactions per second, than its next-closest competitor.

The PushToTest Suite is available for download at: www.pushtotest.com/ptt/saskit.html

Business Benchmark Performance Testing

This performance test was performed by Sun engineers in ongoing performance improvement efforts. Using an industry-standard business benchmark-- an example of a multitiered business logic benchmark simulating ordering, inventory, and manufacturing applications--the test shows improvements in the Sun ONE Application Server PE Edition Update 1 over the release 7.0 version.

Table 2: Business Benchmark Testing Results

	Business Benchmark 2001	
	One Machine	Two Machines
Sun ONE Application Server 7.0	100%	121%
Sun ONE Application Server PE Edition Update 1	123%	231%

In an environment where the application server runs on a single machine with a single instance of the Java VM, the updated release is 23 percent faster.

In an environment where the application server is running on two machines with a single instance of the Java VM on each machine, the updated release is 91 percent faster.

The benchmark testing illustrates that the update release is significantly faster — from 23 to 91 percent faster than the 7.0 release.

Test Bed

A three-tier EJB application was internally tested to run the latest business benchmark, — the following configuration was used:

The application server tier was a Sun Fire™ V880 system configured with 6 x 750-MHz UltraSPARC® III processors and 12-GB of memory for the single-machine test; a second identical machine was added for the two-machine test. Tests were run with a single Java VM per machine.

The database tier was a Sun Fire 4800 server with 8 x 750-MHz UltraSPARC III processors and four GB of memory. A Sun StorEdge™ T3 disk array was used for storage.

The client tier was represented by a Sun UltraSPARC 60 desktop.

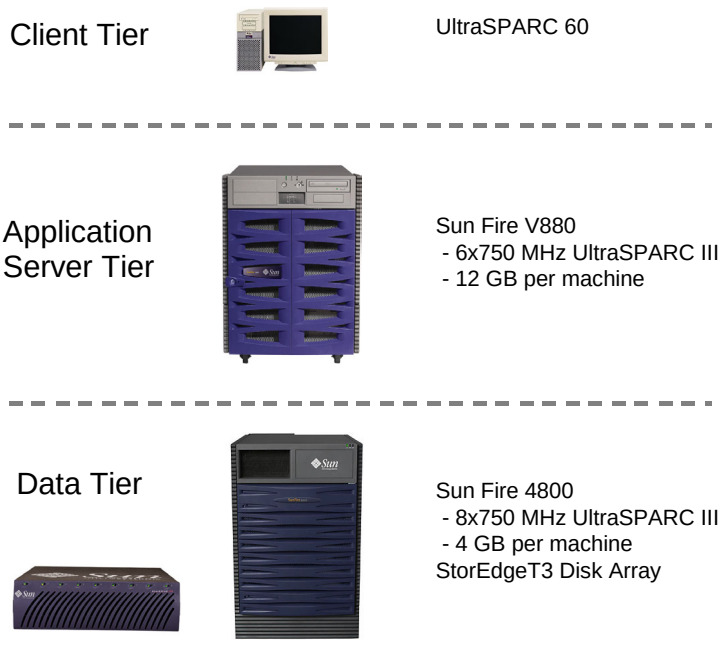

Figure 5: Business Benchmark Test Bed

Network Computing Tests

In the April 3, 2003 edition of *Network Computing Magazine*, six application servers were tested. According to the report, “The Sun ONE Application Server 7 (AS7) screamed its way through our performance tests, offering the most scalable Web services environment among the products we tested. Its ability to process requests as load increased made it stand out from the pack.” (www.nwc.com/shared/printArticle.jhtml?article=/1406/1406f2full.html&pub=nwc#4)

Test Bed

The test platform was an eight-way, Xeon-based server, running Windows 2000 Advanced Server, with two GB of memory. Two Web services were created and deployed. Using two different encoding models, DOC/Literal and RPC/Encoding, a Web service was created that looked up a username and returned first and last names. Performance was tested by sending 50,000 SOAP requests against each of these services. Concurrency levels were varied from 10 to 30 under the same time- and request-limit constraints.

Sun Performance Testing

As part of a regular internal exercise, Sun executes a series of internal application benchmarks to assist in determining the proper size and configuration for enterprise applications. The testing cited here was performed on a reference architecture with the following design goals:

- Stability — Highest priority
- Performance — Important, but secondary to stability
- Scalability — To very high levels
- High Availability — No single point of failure, with 24x7 availability

The internal benchmark exercises a mixture of Web server and EJB logic. To produce sizing information, representative loads and applications that exercise popular application server functionality are identified, and an infrastructure is created to test them, including user load generation.

Applications and User Types

The first application function implemented operations that are typical of an online retail site, such as catalog browsing, shopping cart, ordering, and so on. This used Java servlets to implement the business logic, and employed common Web protocols to communicate with the end user. The first type of user performs typical transactions to the online Web portion of the application. The browser emulation software running on the client's server attempts to implement this user's load by performing a predefined mix of transactions over a predetermined period of time. Appropriate delays, think times, and random start times help distribute the load over the measurement period.

The second application function that was implemented employs different functionality within the Sun ONE Application Server. This application functional block was implemented with the J2EE platform and EJB components to implement an inventory control system and support supplier interactions. A second type of user performs more complicated transactions to the supplier and inventory portion of the application. This user utilizes the functionality of the supplier and inventory part of the application, employing EJB components to stimulate business logic that requires considerably more server resources. It was determined that fewer users would need this application's more advanced functionality. In addition, it was expected that a fixed number of users would employ this functionality during the peak loading period. The latency of transactions for the application was also monitored, to determine if it exceeded acceptable limits.

The two application functional blocks were stressed simultaneously, using the two types of load generation clients. Both environments shared the same resources and worked in the same instance (except where indicated) of the Sun ONE Application Server.

Server Functionality

Four types of software servers were used to support the application's functionality:

- Sun ONE Application Server 7.0. The study was focused on this server, which implements the business logic of the application. It hosts the home Universal Resource Locator (URL) where users make initial contact. No stateful information is kept on the server, and it can be scaled horizontally with minimal effort. In addition to implementing the business logic of the application, the server offers a full text search capability of the product database, and maintains and displays a list of items purchased by the customer.
- Database Server #1. Two database servers are used in this test bed. They supply all of the persistent storage for the online Web portion of the application, including a database of 100,000 products and 10,000 customers.
- Database Server #2. This server supplies all the persistent storage for the supplier and inventory portion of the application. Note that it is separate from the database server that supports the online Web portion of the application.
- Sun ONE Web Server 6 Image Servers. These systems support thousands of images. With a typical online Web application, images and icons of various products, logos, and the like are part of almost every page. In an ideal world, image servers would be geographically dispersed throughout the network, so users could access them locally through DNS techniques, avoiding the consumption of long distance network bandwidth.
- Sun ONE Web Server 6 Secure Server. This offers a Secure Sockets Layer (SSL) transaction for the checkout of the online Web application. It directly accesses the database server to obtain the state of the user's session based on a Web cookie. This server also performs the final portion of the transaction through an SSL connection.

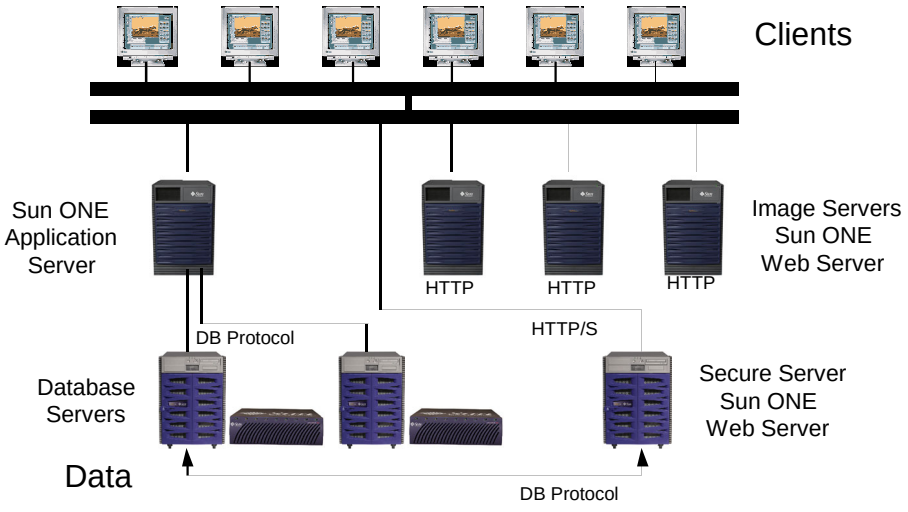

Figure 6 Server Architecture for Load and Performance Testing

It was determined that a mixture of loads in each portion of the application would be utilized on the Sun ONE Application Server to simulate a real-world application environment. This load mixture was determined by the type of users expected to exist at peak periods.

To allow a study of both large and small-scale environments, the test bed employed a 20-processor Sun Fire 6800 server. The CPUs are 1.05-gigahertz, UltraSPARC III modules. By removing CPU cards — which consist of four CPUs and 32-GB of memory from the hardware domain — one card at a time, a server with ever-decreasing capability was produced. As CPUs were reduced, the memory associated with the cards was removed as well.

Results

In the tests, the transaction load is applied from 100 to many thousands of users in order to determine the sizing limitations of a particular server configuration. Multiple test runs of this transaction load are placed against the server under test. For each test run, the number of users increases. Eventually, the transaction latency exceeds acceptable limits, determining the upper limit that can be supported by the server configuration.

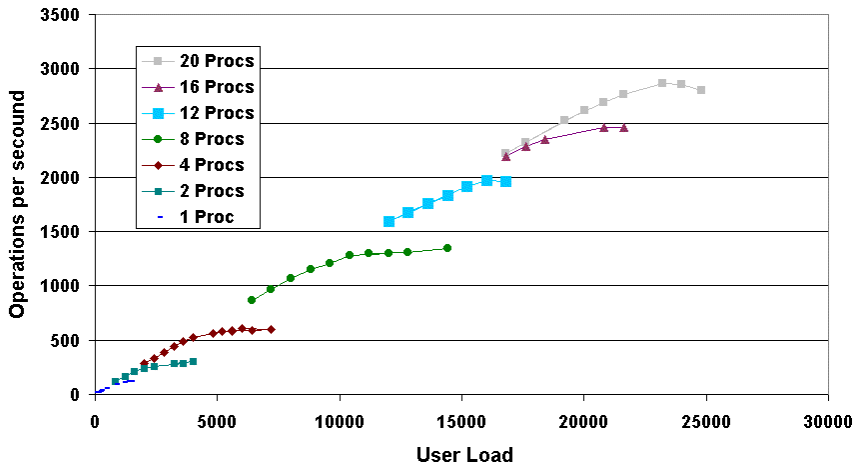

Figure 7: Performance of the Sun ONE Application Server for 1, 4, 8, 12, 16, and 20 CPUs Running the Sizing Load

In Figure 7, the knee of the performance curve for 1, 4, 8, 12, 16, and 20-processor domains of a Sun Fire 6800 server are plotted on the same graph. As illustrated, the performance of all of the configurations follows an almost linear slope of approximately 0.137 operations per second, per user. As soon as the CPUs on a Sun ONE Application Server 7 instance approaches 100-percent utilization, the performance no longer increases as additional user load is applied.

Figure 8 demonstrates the maximum number of users generating the sizing load that can be supported at any one moment in time with standard CPU-configuration servers.

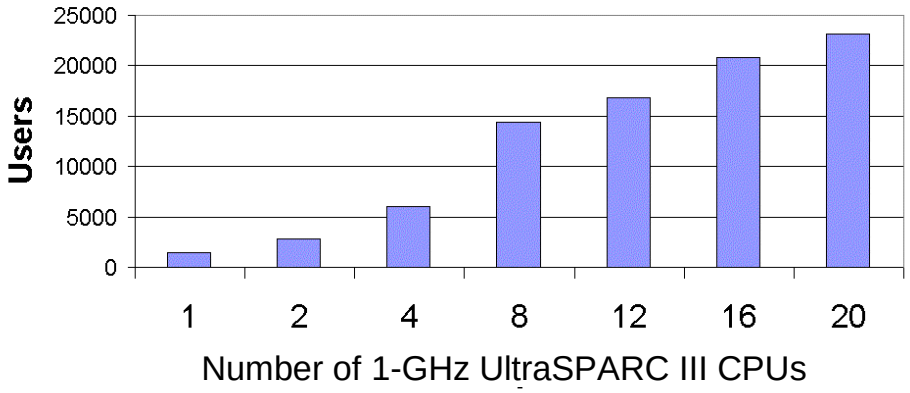

Figure 8: Maximum Number of Users Supported by Number of Processors

Figure 9 shows how the number of operations per second scales with near linearity as the number of processors increases.

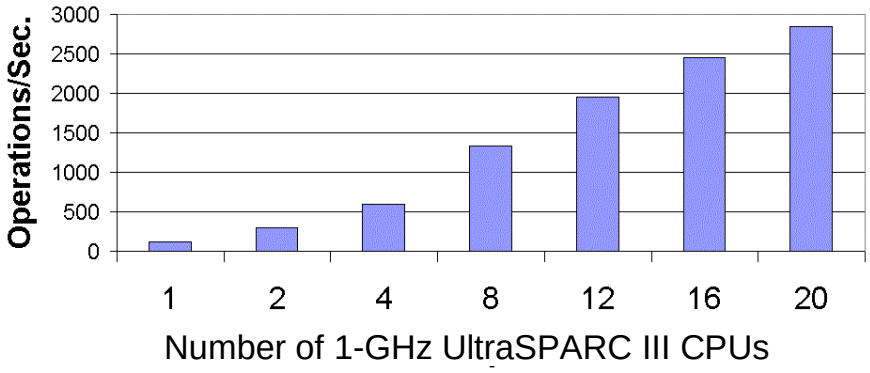

Figure 9: Operations per Second by Number of Processors

EJB Component Performance Benchmark

The University of Mining and Metallurgy conducted a benchmark against the Sun ONE Application Server 7 and two of its competitors — Competitor 1 and JBOSS 3.02. The goal of the study was to perform stress testing of a typical application in a J2EE environment. In this case, a training activity manager application was employed, which supports educational activities such as creating a new students' laboratory, assigning teachers to labs, creating new lessons and tests, adding students, and so on.

The test used two different approaches to manage data: CMP 2.0 entity beans, and the Data Access Object (DAO) design pattern. The tests showed that the Sun ONE Application Server 7 CMP 2.0 entity bean implementation outperformed those from Competitor 1 and JBoss.

The tests also showed that the DAO design pattern offers much better performance than CMP 2.0 containers. Testing DAO implementation as replacement for entity beans, the Sun ONE Application Server outperformed Competitor 1 and JBoss DAO implementations.

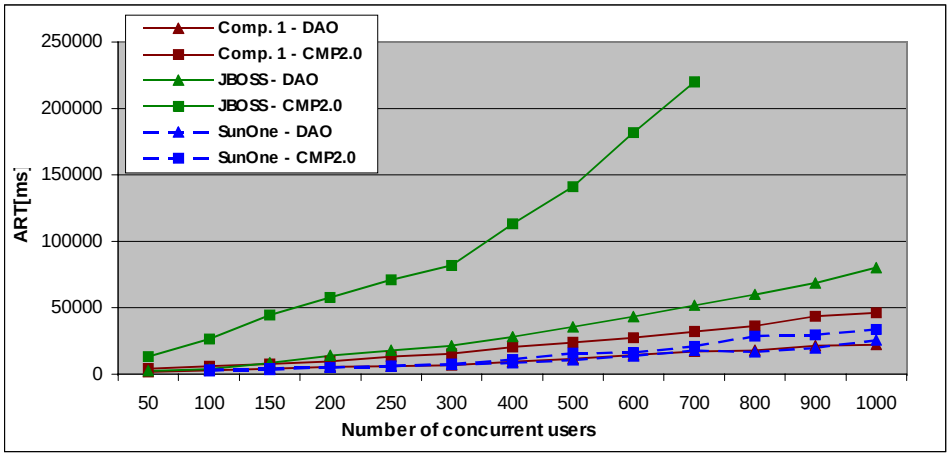

Figure 10: Create Use Case

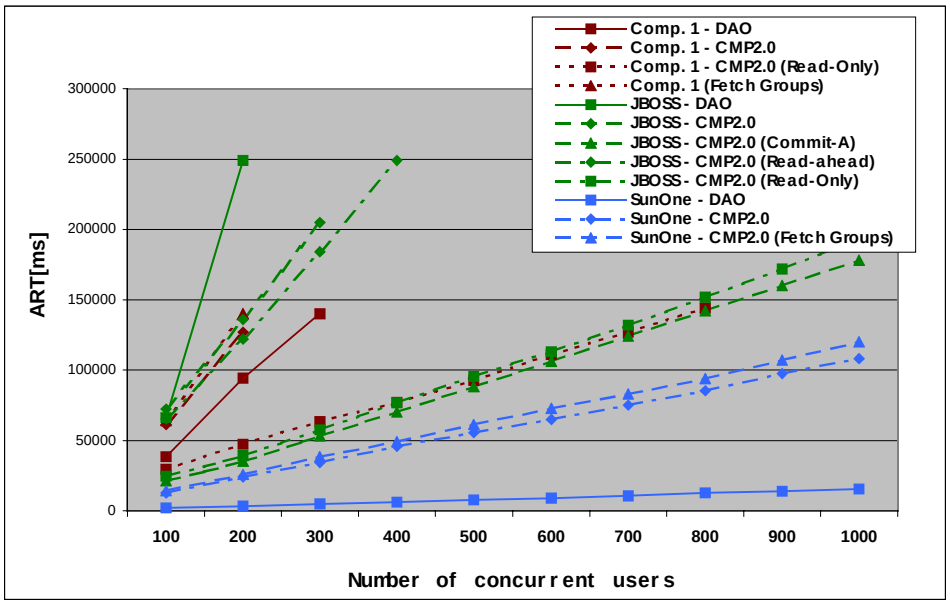

Figure 11: Select Use Case

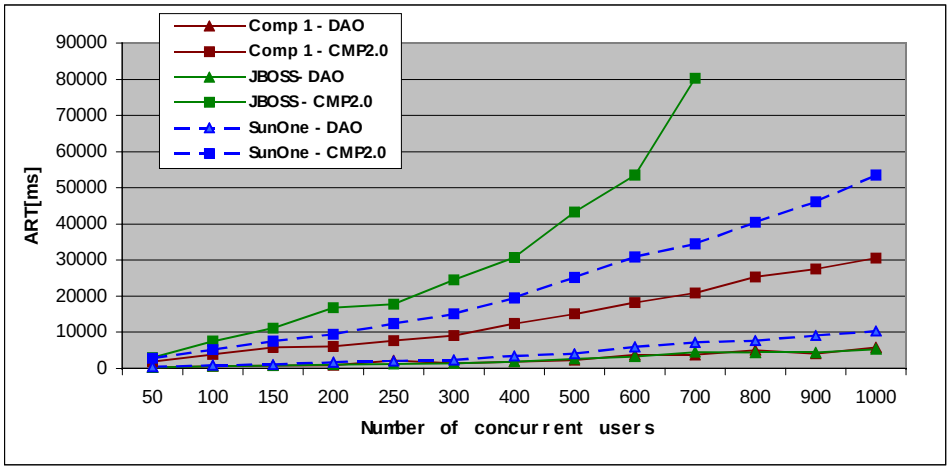

Figure 12: Delete Use Case

Test Bed

Three use cases were employed for testing purposes:

- Create Data: New lessons for a given activity group is created.
- Select Data: Lessons information for a given activity group is fetched.
- Delete Data: Delete information about lessons and test for a given activity group.
- Because the stress tests were focused on database access performance study, implementation of the data persistence mechanisms was the most important goal. Two different approaches were studied:
- Session Facade with DAO: DAO is responsible for implementing appropriate factory classes, which implement the JDBC API for database access. The DAO design pattern

is used to abstract and encapsulate all access to the data source, as well as manage all connections to store and retrieve information.

- Session Facade with entity beans based on CMP 2.0 specification: All business logic responsible for persistence operations provided by the EJB container is implemented using CMP 2.0 services. CMP 2.0 supports container-managed relationships both in selecting and removing data.

Load was generated at the presentation layer using Grinder, a load generation client application that was responsible for direct calls of session beans over RMI-IIOP, measuring performance, and collecting all data. Grinder is a 100% Pure Java™ load-testing framework that is freely available under a BSD-style, open source license. Both the application server and database server were run on the machine, which was a Sun Fire 6800 server with 20 CPUs, 20-GB of RAM, a 120-GB disk drive, a Sun Gigabit Ethernet interface, and the Solaris 8 OS. Grinder was run on a separate machine: a Sun Fire 3800 server with four CPUs, four GB of RAM, a Sun Gigabit Ethernet interface, and the Solaris 8 OS.

Full details of this benchmark is available at www.ics.agh.edu.pl/people/mj/papers/EjbPerf.doc

Tuning Tips and Techniques

To achieve optimal performance for any application deployment, the Sun ONE Application Server requires tuning. Performance may be significantly enhanced by adjusting deployment descriptor settings, modifying server configuration files and other settings, as well as adjusting the underlying hardware and operating environment. Before tuning can begin, it is important to understand the environment and performance goals. An optimal configuration for a production environment may not be ideal for a development environment. This section provides tips and techniques to help architects and administrators understand the tuning and sizing options available, and help them optimize performance from a Sun ONE Application Server 7 deployment.

Before tuning and deploying an application on the Sun ONE Application Server, it is important to become familiar with the application architecture and underlying operational environment. It is not uncommon to see moderately complex enterprise applications developed entirely using servlets and JSP components. More complex business applications are often implemented using EJB components. The Sun ONE Application Server integrates the Web and EJB containers into a single process. Local access to EJB components from servlets is very efficient. However, some application deployment may require EJB components to execute in separate processes and remain accessible from standalone client applications, as well as servlets. Based on the application architecture, the server administrator can deploy the Sun ONE Application Server in multiple tiers, or simply host the presentation and business logic on a single tier. The operating environment — the hardware and operating system — should also be considered in any performance-tuning efforts.

Additional information on tuning the Sun ONE Application Server 7 can be found at:

Sun ONE Application Server 7: docs.sun.com/db/doc/816-7159-10

Sun ONE Application Server PE Edition Update 1:
docs.sun.com/db/doc/817-2180-10

- Many parameters can be configured when tuning the Sun ONE Application Server for peak performance. These fall into three major areas:
- Java VM tuning
- Web Container
- EJB Container and Resources

Java™ VM Tuning

The Java VM is the heart of the application server, so correct configuration can result in significant performance gains. (Note: Using J2SE 1.4.1 or later is highly recommended for best performance.) When tuning the Java VM, consider:

- Correctly sizing the heap: If the heap is sized too small, there will be more frequent garbage collector (GC) activity, memory shortages, and overall reduction in

throughput. On the other hand, a heap that is sized too large will cause long pauses in the GC, and is an unnecessary use of memory. The default heap configuration is designed for a developer environment. Internal testing shows that proper heap sizing can improve the transaction rate by 20 times.

- Choose the appropriate garbage collector: In addition to the existing GCs, two new GCs are available in the Sun ONE Application Server 7.
  - Throughput is parallel, young generation collection, which offers superior scalability on multi-CPU systems. It should be employed when transaction rate and time-to-completion are most important, and is most useful in high-throughput server applications. Internal testing shows that utilizing the throughput GC can improve the transaction rate by 18 percent, over and above a properly configured heap.
  - Short-Pause-Time Collector is a concurrent collection of the tenured generation that dramatically reduces pause times on multi-CPU systems. It should be employed when low pause times are important to focus system resources and ensure consistent response times; for example, near-real-time telecommunications applications. This GC provides the characteristics of an incremental collector. In addition, it has an option enabling a parallel young generation collector.

Note: The concurrent collector essentially takes a CPU away from the system, and dedicates it to garbage collection. This can reduce potential throughput, since a system's CPU is effectively removed from doing the application's work.

Web Container

For application environments that rely on JSP components and servlets, tuning the Web container can provide dramatic improvements. System administrators should pay particular attention to HTTP traffic, while optimizing output streams and caching common queries:

- Optimizing connection handling: The HTTP subsystem is very scalable. The application server defaults are not optimized for environments where there are many HTTP 1.0 clients (clients sending HTTP/1.0 requests without a `KeepAlive` header), single-request HTTP 1.1 clients, or HTTP 1.1 timeouts. The default tunings are also not appropriate for a lightly loaded system that is primarily servicing `KeepAlive` connections. Connection handling can be improved by:
  - Increasing the number of acceptor threads if there are many short-lived connections
  - Decreasing the `RqThrottle` in systems with fewer CPUs
  - Decreasing the `KeepAliveTimout` if clients typically disconnect
- Optimizing JSP components and servlet performance: There are many ways to achieve this. For instance, in environments with HTTP 1.1 clients, sending large amounts of data requires a change to use the `OutputStreamSize` parameter in the obj.conf file. By default, this is set to eight KB; larger amounts will be sent in separate chunks. If there are many large transfers, increase this number to lower the amount of chunking the server must perform.

  In addition, if a great deal of time is spent rerunning the same servlet or JSP component, its results can be cached, then returned out of the cache the next time it is

run. The dynamic content caching of servlet responses is implemented using servlet filters, and the JSP page caching is achieved using a custom tag library. The caching can be configured and enabled using sun-web.xml. For exact configuration details refer to: docs.sun.com/source/816-7150-10/dwservlt.html#27888.
This method is helpful for queries that are run by all visitors to a site. Note that the results of the query should be dynamic, because it might change periodically — but the logic need not be run for every user.

- Optimizing JDBC and SSL performance: JDBC Type 2 drivers and SSL libraries both depend on native libraries. Using the multithreaded memory allocation library `libmtmalloc.so` on a multiprocessor system provides significant scaling improvements for these libraries. Based on internal testing of `libmtmalloc.so` against `libc`'s standard `malloc()` on eight CPU systems, there is upwards of a 200 percent improvement in SSL performance and a 56 percent improvement in business applications that use JDBC Type 2 drivers in Solaris 8 OS and Solaris 9 OS.
- HTTP server connection handling, worker threads, and queue parameters can be tuned using the information obtained from the `perfdump` utility. This is a built-in NSAPI utility that can collect performance data by employing various application server internal statistics. For more details refer to: docs.eng.sun.com/source/816-7159-10/pt_chap4.html#58338.

EJB Resources and Container

The Sun ONE Application Server 7 provides a highly configurable bean pooling mechanism that allows the configuration of bean pools according to the needs of the enterprise. In addition, the Sun ONE Application Server supports a number of tunable parameters that can control the number of beans cached, as well as how long they are cached. These Sun ONE Application Server parameters are easily tuned and configured.

EJB Pool

For stateless session, message-driven, and entity beans, creating an EJB pool reduces the overhead associated with creating and initializing bean instances. (Note: Instances in the pool do not have an identity.) By monitoring the total beans created or destroyed and excessive creation or deletion of instances, equilibrium can be maintained. Accumulated unused instances should be avoided, as they will cause more frequent and longer full GC cycles. A well-tuned cache will maintain an equilibrium of entity beans. Prepopulating the pool with stateless session or MDBs is also recommended.

The pool and cache settings can be set both globally and on a per-bean basis. The global settings are handled through the administrative console. On the page that administers the EJB container, there are tabs for the default pool and default cache settings. These settings apply to all beans that have not explicitly configured pool and cache.

The pool and cache for specific beans are configured in the sun-ejb-jar.xml file contained in the EJB component's Java™ Archive (JAR) file. A sample stanza looks like this:

```
<bean-pool>
      <steady-pool-size>32</steady-pool-size>
      <resize-quantity>16</resize-quantity>
```

```
        <max-pool-size>640</max-pool-size>
        <pool-idle-timeout-in-seconds>600</pool-idle-timeout-in-seconds>
        <max-wait-time-in-millis>0</max-wait-time-in-millis>
    </bean-pool>
    <bean-cache>
        <max-cache-size>512</max-cache-size>
        <resize-quantity>16</resize-quantity>
        <is-cache-overflow-allowed>true</is-cache-overflow-allowed>
        <cache-idle-timeout-in-seconds>600</cache-idle-timeout-in-seconds>
        <removal-timeout-in-seconds>5400</removal-timeout-in-seconds>
        <victim-selection-policy>nru</victim-selection-policy>
    </bean-cache>
```

EJB Cache

Stateful session and entity beans can be cached to reduce the cost of passivation and activation of bean instances. (Note: Bean instances in the cache must have an identity.) The cache can be tuned by monitoring cache hits and misses, the number of stateful bean passivations, and the total number of beans in the cache. The goal is to minimize the number of activations and passivations, while avoiding the accumulation of unused instances, which results in frequent and longer full GC cycles. For entity beans, the cache is tuned by increasing the cache size for beans with concurrent or iterative access patterns. For stateful session beans, the maximum size of the cache should be set to the expected number of concurrent clients. The removal-timeout-in-seconds parameter should be used to prevent large numbers of session states on disk.

EJB Container

Consistency levels are tunable in the EJB container, enabling a trade off between performance and CMP consistency in transactions. There are two general approaches to ensure consistency in database access layers. The first is locking, which is pessimistic concurrency and may lead to decreased concurrency. The second approach is checking, or optimistic concurrency, and may result in a larger number of database round trips.

The consistency level for a bean is set in the sun-cmp-mappings.xml file included in the EJB component's JAR file. To decrease concurrency while increasing data consistency, set the consistency level so that the bean is locked when it is first read, as shown here:

```
<consistency>
    <lock-when-loaded/>
</consistency>
```

Infrastructure Tuning

The hardware and operating system are potential bottlenecks to application server performance. For example:

- CPUs reach 100-percent utilization.
- The system runs out of physical memory.
- The performance of the disk subsystem is too low.
- The network connection between various servers and clients becomes the bottleneck.

CPU Utilization

CPU saturation occurs when the `usr` and `sys` columns of a `mpstat` command add up to a value close to 100. Vertical and horizontal scaling are the two best ways to solve this type of bottleneck. Use either of the methods as follows:

- Vertical scaling can be accomplished by adding CPUs to the server. On Sun Fire servers, hardware domains allow CPUs in one domain to be allocated to another domain. In smaller systems, adding processors might be possible if the maximum for that server type has not yet been reached. Always consider the systems availability requirements when deciding whether to use a single server to support an application.
- Horizontal scaling is accomplished by designing an application to work on multiple servers at one time, so the application's capacity can be increased by adding servers. This also improves the application's availability if one of the servers fails.

Memory

The Sun ONE Application Server can run out of memory in two ways. The first is related to the physical memory required by the Solaris OS, and the second is specific to requirements of the Java VM device running within the Sun ONE Application Server (as previously noted).

Ensuring that there is enough free memory on the Solaris OS is critical to overall server performance for several reasons. When the Solaris OS no longer has free memory for the application to allocate, it uses the `/swap` partition on the disk. While the use of `/swap` works for processes that do not run often, the performance of the server will suffer dramatically if highly active processes cannot fit into memory.

For applications that access the disk often, having as much free memory as possible allows the Solaris OS read cache to work efficiently, because the Solaris OS uses all available free memory as read cache. For high-load environments, it is critical that frequently used files are accessed only once, and from then on they are accessed through the read cache.

Disk Subsystem Performance

Many three-tier Sun ONE Application Server 7 deployments do not need high-performance disk subsystems. This is because the server that executes the business logic will likely depend on other servers (like database servers) to store the data it needs.

In general, redundancy and high availability are the highest concerns to be addressed on any Sun ONE Application Server instance. Therefore, it is essential to install the Solaris OS and the Sun ONE Application Server in a redundant environment that can handle disk failures.

Network Subsystem

A network bottleneck can frequently cause performance degradation that is perceived by the end user. For many applications that produce Web page references external to the Sun ONE Application Server, these bottlenecks occur on the ancillary servers delivering the referenced content. Monitoring and sizing the network subsystem is necessary to maintain high throughput and performance. For example, a single redundant gigabit interface could be used by all clients and ancillary servers that access the Sun ONE Application Server.

Developer Experience

A number of significant improvements have been made to the Sun ONE Application Server 7 developer environment, resulting in improved performance and efficiency in application development and deployment. These improvements enhance the experience in both tightly coupled and loosely coupled environments.

Tightly Coupled Environments

Plug-ins for leading integrated development environments (IDEs) enable close cooperation between the developer environment and the application server development and runtime environment. The Sun ONE Studio 5, Standard Edition is an IDE that enables the creation, assembly, deployment, and debugging of code in the Sun ONE Application Server from a single, easy-to-use interface. Sun ONE Studio 5 Standard Edition is tightly integrated with Sun ONE Application Server PE Edition Update 1.

Loosely Coupled Environments

Many developers are more comfortable using both GUI and command-line tools to develop and deploy applications. For example, an IDE may be used for debugging, while command-line tools are used for deploying — often using *Ant*. (Ant is a Java technology-based build tool that is extended with Java classes. Instead of using shell commands, the configuration files are XML-based, calling out a target tree where tasks are executed.) The Sun ONE Application Server 7 offers enhanced interaction with an Ant-based development and deployment environment. Ant class-deployment APIs and scripts are part of the application server environment. A comprehensive set of sample applications and code packages leverage the Ant capabilities within the Sun ONE Application Server, helping developers quickly understand how to iteratively assemble and deploy applications.

Assembly and Deployment

The dynamic assembly and deployment of applications and components enhances developer productivity because they can be deployed without restarting the Sun ONE Application Server.

- Dynamic reloading: Enables reloading the classes that constitute an application when they change on disk.
- Hot redeployment : Enables redeploying an existing application without restarting the server. As well, applications and modules can be enabled or disabled without undeploying it.

Dynamic redeployment is the ability to redeploy an existing application without a server restart. This happens when an application's configuration (contents of its XML files) and certain classes change. Dynamic redeployment results in behavior identical to that of

reloading the entire application's classes. In addition, dynamic redeployment involves creating new application contexts (Web and EJB components) while removing the old application contexts. Thus, dynamic redeployment produces a brand new instance of the application (except for existing session data). This feature is supported in development mode only, and can result in exceptions similar to those for dynamic reloading. Also, configuration changes that require server restarts do not take effect until the restart is completed. Dynamic reloading is activated only for applications and unshared standalone modules with central configurations that specify it.

Hot deployment is the ability to deploy an application at server runtime, without requiring a server restart. This feature uses the same infrastructure as dynamic redeployment. However, since there is no state left over from a previous incarnation, this feature is supported at production time.

The Sun ONE Application Server 7 Assembly Tool provides a complete solution for packaging and configuring new and pre-existing J2EE applications. The J2EE Archives created by this tool are deployable to Sun ONE Application Server 7. The tool can be downloaded at: www.sun.com/software/download/products/3ec10b05.html

Simplified Developer Environment

The developer environment is greatly streamlined in the latest Sun ONE Application Server 7. The number of processes and application instances is significantly reduced — other than ensuring that a suitable operating system is installed, no additional technologies are required prior to installation. With a few mouse clicks, a full-featured HTTP server, J2EE 1.3, J2SE 1.4, Java Transaction Service (JTS) compatible transaction manager, and JMS messaging server are installed, preconfigured, and ready to use. Optionally, a Sun ONE Studio IDE can also be installed as part of the Sun ONE Application Server 7 installation. The administrative interface for both GUI and command-line interfaces is capable of managing all configuration information and parameters. Unlike previous versions, the server XML file can be modified directly.

Future Directions

Sun is committed to improving the performance of the Sun ONE Application Server, while supporting and complying with existing and emerging Web standards. By optimizing the interfaces and configuration of internal components, Sun expects to continue improving overall scalability and performance.

These technologies include:

- Java Web Services Developer Pack (Java WSDP) 1.1: An integrated toolkit that enables Java developers to build, test, and deploy XML applications, Web services, and Web applications. It provides Java technology implementations of key Web services standards including WSDL, SOAP, ebXML, and UDDI, as well as important Java technology implementations for Web application development, such as JSP components and the JSP Standard Tag Library. The Java WSDP includes:
  - Java Architecture for XML Binding (JAXB) 1.0
  - Java API for XML Messaging (JAXM) 1.1.1
  - Java API for XML Processing (JAXP) 1.2.2
  - Java API for XML Registries (JAXR) 1.0.3
  - Java API for XML-based RPC (JAX-RPC) 1.0.3
  - SOAP with Attachments API for Java (SAAJ) 1.1.1
  - JavaServer Pages Standard Tag Library (JSTL) 1.0.3
  - Java WSDP Registry Server 1.0_04
  - Ant Build Tool 1.5.1
  - Apache Tomcat 4.1.2 container
- Java VM technologies: J2SE and J2EE are the core of the Sun ONE Application Server. Incorporating the latest improvements in these technologies will result in improved application server performance.
  - New *smart tuning* functionality enables the 1.5 VM to evaluate the machine it's running on, and adapt itself for optimal performance.
  - New floating-point instruction sets on x86 processors are expected to enable developers and administrators to take full advantage of new hardware and software platforms. J2SE 1.4.2 includes recognition of specialized hardware characteristics, and exploits them for enhanced scalability and performance.
  - Java VM runtime optimizations are expected to deliver substantial performance improvements over previous versions.
  - Lightweight performance monitoring tools are expected to overcome the intrusive nature of existing tools. New tools will allow nonintrusive, real-time Java VM performance monitoring in production environments. See www.sun.com/developers/coolstuff/.
- Support for 64-bit platforms, in addition to the Solaris OS.

- Solaris x86 improvements are expected to improve the performance of certain applications by a factor of two in J2SE 1.4.2.
- Linux thread optimizations are expected to improve performance and scalability on Linux platforms.

Performance in Practice — The FETISH Network

In Europe, family-run hotels are commonplace. Yet, finding these hidden travel gems has not always been easy, because global, corporate tourism distributors tend not to service smaller companies. For these small and medium-size businesses — the Federated European Tourism Information Service Harmonization project (FETISH) — is offering a compelling new advantage. Funded by the European Commission (EC), the FETISH project has essentially developed a federated community of tourism enterprises and services. This virtual community is brought together through Web services.

The Sun ONE Application Server is the information manager of the FETISH network. It enables all small to medium-size, tourism-related European businesses to provide value-add services by communicating and transacting peer-to-peer via Web services. The FETISH network, with the Sun ONE Portal Server front end, is deployed at three regional sites: Spain, Italy, and the U.S. It leverages the Sun ONE Proxy Server to direct requests to one of three information managers, enabling a worldwide consumer base to make reservations that span across countries via the federated tourism network.

According to the project coordinator, Andrea Nicolai, CEO of T6, “The Sun ONE Application Server, with its support for industry standards such as Java technology and Web services, was a perfect choice for us. It provides a robust, high-performance business logic engine that will scale as we rapidly grow the FETISH Network. Our Java Web services framework has made it easy to add or drop services from our network,” he explained. “Services are interchangeable, so travelers might use a particular application on the network to book a flight one week, and another to do the same thing a month later. The functionality is all the same, and what happens beneath the surface is transparent to end users.” To join the network, a vendor simply registers its service, data, or application with FETISH. Then, the vendor receives access to any other service on the network — potentially hundreds — through a thin-client browser. But this B2B aspect of the network is only one example of what it can support.”

According to Nicolai, the architecture dramatically cuts down on administration load by acting as a self-contained, decentralized network that continuously adapts to new standards and services, as well as fluctuating network availability. It also employs self-healing capabilities — if a software component failed, it would recover itself and continue running.”

The project saw the following business benefits:

- Projected 100-percent reduction in TCO for members
- 500-percent reduction in development cycles
- 20 to 30-percent increase in revenue opportunity for small to medium businesses

The Sun ONE Application Server 7 played an instrumental role in meeting the challenge of building an efficient, flexible global infrastructure that scales to support a global variety of travel offerings

Summary and Conclusion

Sun has the vision, architecture, products, and expertise to deliver Web services in a performance-leading application server. This fifth-generation product is built with more than eight years of expertise in delivering highly scalable, application server technology. The Sun ONE Application Server 7 provides a solid platform for today's applications and Web services.

The results highlighted in this paper show the significant performance capabilities of the Sun ONE Application Server. Whether applications are more dependent on Web server technologies or the business logic enabled in EJB components (and complex n-tier architectures), the Sun ONE Application Server delivers. The tuning tips and techniques discussed can assist developers and system administrators optimize their environments for maximum performance and availability.

Built with Java technology, the Sun ONE Application Server 7 offers many features and capabilities that appeal to developers. Compared to previous versions, it is easier to install, easier to use, and more tightly integrated with developer IDEs and tools. As part of an overall application and service deployment architecture, it can facilitate the integration of backend application logic and data, user management services, and front end presentation options. A reference implementation of the J2EE 1.3 specification, it is distributed as part of the Solaris 9 OS. The Sun ONE Application Server 7 delivers a high-performance, scalable, and reliable J2EE platform for developing, deploying, and managing e-commerce applications and services.

More Information

What Influences the Performance of Application Servers?

developer.iplanet.com/docs/articles/appserver/influences.jsp

Sun ONE Application Server Tuning Guides

- *S1AS7 Tuning Guide*: docs.sun.com/db/doc/816-7159-10
- *S1AS7 Update 1 Tuning Guide*: docs.sun.com/db/doc/817-2180-10

Java 2 Platform, Standard Edition (J2SE) Performance and Scalability Guide

java.sun.com/j2se/1.4/performance.guide.html

Tuning Garbage Collection with the 1.4.2 Java Virtual Machine

java.sun.com/docs/hotspot/gc1.4.2/index.html

J2SE Performance Documentation

java.sun.com/docs/performance

Threading

java.sun.com/docs/hotspot/threads/threads.html

The Coolstuff Program

www.sun.com/developers/coolstuff/

Sun ONE Application Server Performance FAQ

java.sun.com/docs/performance/appserver/AppServerPerfFaq.html

Ask your Sun representative for copy of *Sun ONE Application Server 7 Sizing Guide,* expected to be available in Fall 2003.

SPEC Benchmarks

SPEC® and the benchmark name SPECjbb®2000 are registered trademarks of the Standard Performance Evaluation Corporation. For the latest SPECjbb2000 benchmark results, visit www.spec.org/osg/jbb2000.