



## Table of Contents

Executive Summary . . . . .	1
<b>Using Logical Domains and CoolThreads Technology: Improving Scalability and System Utilization . . . . .</b>	<b>3</b>
Scalability Issues of Web Applications . . . . .	3
Sun's Virtualization Solution . . . . .	5
Configuration Considerations . . . . .	7
Physical Test Environment . . . . .	8
Test Workload . . . . .	10
Test Procedures . . . . .	11
Server and Network Configuration . . . . .	14
Results and Analysis . . . . .	20
Summary . . . . .	23
About the Authors . . . . .	24
Acknowledgements . . . . .	24
References . . . . .	24
Related Resources . . . . .	24
Ordering Sun Documents . . . . .	25
Accessing Sun Documentation Online . . . . .	25
Appendix: Software Installation . . . . .	26

## Executive Summary

Advances in technology have led to the creation of powerful new servers that offer unprecedented computing power and breakthrough performance. Servers like the Sun SPARC® Enterprise T5220 server, designed around the UltraSPARC® T2 processor that incorporates Chip Multithreading Technology (CMT), support up to 64 threads (virtual CPUs) and 64 GB of memory on a single system. However, not all Web and application services currently scale linearly on multicore processors, and this lack of software scalability can result in under utilized system resources and lower performance. The common data center practice of configuring a single network interface to feed data into and out of the application software stack can also become a bottleneck, additionally limiting application throughput.

Virtualization is one approach to solving these problems and fully utilizing the new technological advances occurring in the chip and server industries. This project explores the use of Sun's CMT platform and the Logical Domains (LDoms) virtualization technology as a means to increase application software scalability and improve system utilization. Using LDoms software, multiple logical systems — each with its own operating system and each running an independent instance of the application software — are created on a single server.

This project evaluated the use of a Sun SPARC Enterprise T5220 server running the OpenSolaris™ operating system<sup>1</sup> and the Logical Domains Manager 1.0.1 software as the virtualization platform. Popular open source software products, including the MySQL database server, the Tomcat application server, and a demo application that simulates an online pet store, were used to build the Web application. Grinder, open source software from SourceForge, was used for load generation and testing.

The Sun SPARC Enterprise T5220 server was configured with multiple logical domains, and the Web application performance was measured. The goal was to push the system to fully utilize system resources and achieve maximum transactions per second (TPS). Configurations with 6 logical domains and with 12 domains were compared, to determine any differences in CPU and network utilization. Deploying the application in a virtualized environment resulted in better utilization of system resources and as much as ten-times improvement in performance, when compared to a single application instance running on the system.

---

1. OpenSolaris is an open source project based on the Solaris™ operating system source code.



## Using Logical Domains and CoolThreads Technology: Improving Scalability and System Utilization

Processors like the UltraSPARC T2 processor with Chip Multithreading Technology (CMT) support up to 64 threads on a single chip. However, some Web applications do not currently scale linearly on servers that use these multicore processors. This paper explores the use of Sun's Logical Domains technology and Sun's CMT hardware platform as a means to increase application software scalability and improve system utilization for these Web applications.

### Scalability Issues of Web Applications

A typical Web application has the following architecture, as shown in Figure 1. The application server hosts the business logic, and the database server hosts the data. One or more clients access the Web application via the local network or Internet.



Figure 1. Generic Web application architecture.

When the workload is light, a single instance of each the application server and database server is generally sufficient to handle the load. As the workload grows with more and more clients accessing the Web application, a single instance of the application server can often become a bottleneck. This bottleneck can be the result of the application server software itself not scaling, or can be caused by underlying hardware that is no longer powerful enough to meet increased demands.

The problem of underpowered hardware can easily be addressed by upgrading to more powerful systems. Upgrading the underlying hardware will provide performance improvements — until the software stops scaling. When the software itself stops scaling, a different approach must be taken. Currently, some commonly used open source software is designed to scale on 1, 2, or 4 core systems, but does not scale up on 8 or 16 core systems.

To overcome the scaling issue imposed by the software stack, it is often necessary to run Web applications on multiple smaller systems, or run multiple instances of the software stack on a larger system. Some deployments may implement a one-to-one mapping between the application and database servers, with each application server

communicating with its own database server, as shown in Figure 2. Other deployments may implement a many-to-one mapping, with multiple application server instances communicating with a single database server.

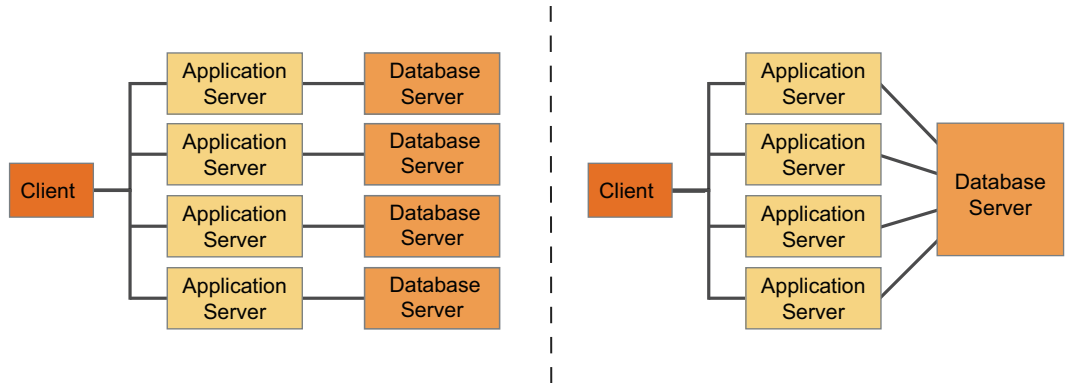


Figure 2. Two example architectures with multiple application server instances.

Running the application on multiple smaller systems can improve performance, but this configuration can present heating/cooling, power, and space challenges. In addition, managing multiple servers increases administration complexity and can increase management costs.

An alternative is to run multiple instances of the application server on a single larger system, as shown in Figure 3. However, running multiple instances on a single larger system can pose configuration and deployment challenges such as memory and CPU resource allocation and load balancing of workloads. In addition, using only a single operating system instance can result in saturating the single network interface used between the clients and application server instances, unless link aggregation techniques are used.

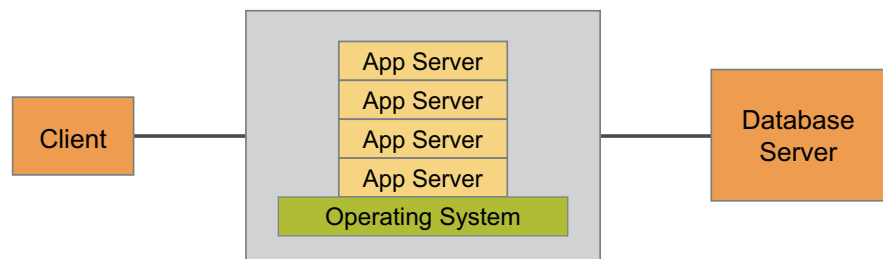


Figure 3. Web architecture with one server running multiple application instances.

Using virtualization technology, such as the Logical Domains (LDoms) technology from Sun Microsystems, on a powerful system such as the Sun SPARC Enterprise T5220 server is an effective solution to this scalability issue. With the Sun SPARC Enterprise T5220 server, multiple instances of the application server software can be deployed on multiple instances of the operating system — all on a single server. Using multiple operating system instances provides the advantage of multiple network interfaces, and

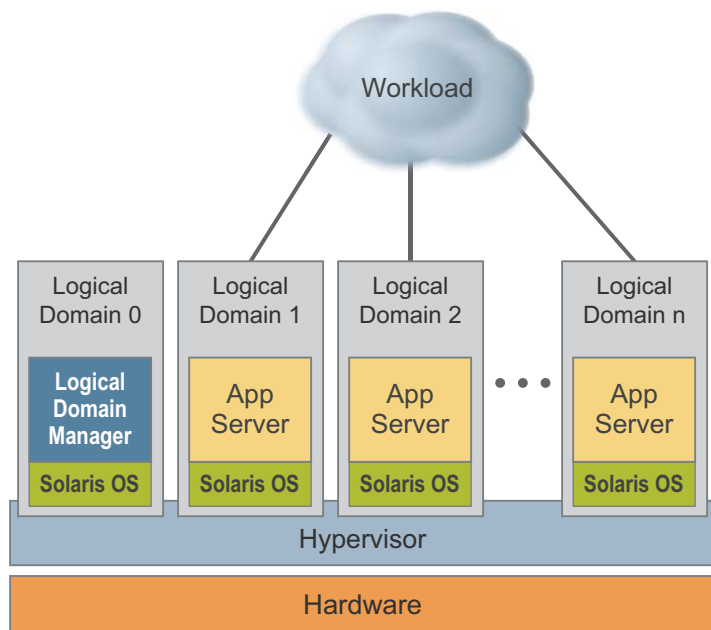
avoids complicated setups using link aggregation. Because a single server is used, there are no configuration and management issues of deploying multiple servers. Consolidating the application on a single energy-efficient server can also potentially save on total power and heating/cooling costs, reduce data center space requirements, and reduce server management overhead.

## Sun's Virtualization Solution

This document studies Sun's virtualization solution featuring Logical Domains (LDoms) technology as a means to address the scalability issues of Web applications.

### Logical Domains (LDoms) Technology

LDoms technology enables a system's hardware resources, such as memory and CPUs, to be subdivided creating partitions called *logical domains*. Each domain has its own set of resources and runs its own independent instance of the operating system, as shown in *Figure 4*.



*Figure 4. Sun's Logical Domains architecture.*

A *control domain* (Logical Domain 0) runs Logical Domain Manager software; this software is used to create and manage additional logical domains called *guest domains*. In this example, each guest domain runs its own instance of the application server software. A thin layer of firmware, called the hypervisor, is interposed between the hardware and the logical domains. The hypervisor, resident on Sun UltraSPARC T1, T2 and T2 Plus processors, enforces resource management at the hardware level and provides an interface between these resources and the operating system software.

LDoms also provide support for virtual devices, such as virtual disks and virtual networks. The actual physical hardware devices are abstracted by the hypervisor and presented to the logical domains on the system. With virtual disk devices, a single physical device can be partitioned and shared by multiple logical domains. Virtual network switches and virtual network devices are used to abstract physical network interfaces and enable communication for the logical domains.

LDoms provide a reliable, robust, and secure virtualization solution. Multiple guest domains provide strong workload isolation, with each guest domain running an independent instance of the Solaris operating system and its own software applications. A software failure in one guest domain does not bring down other guest domains on the system. In addition, the Hypervisor, which has access to all memory and I/O devices, is not accessible from the guest domains or even the control domain. This increases security, and prevents unauthorized access of data and resources. As long as traditional OS security remains uncompromised on the server, no domain can access the content of another domain.

---

**Note** – For more detail about LDoms, see *Beginners Guide to LDoms: Understanding and Deploying Logical Domains for Logical Domains 1.0 Release*, available on the Web at <http://www.sun.com/blueprints/0207/820-0832.html>

---

## Sun CoolThreads™ Technology

LDoms technology is supported on Sun CoolThreads™ technology-based servers that incorporate UltraSPARC T1, T2, or T2 Plus processors and run the Solaris™ operating system (Solaris OS). The UltraSPARC T1 processor contains up to eight processing cores with four threads<sup>1</sup> per core. This hardware support allows the hypervisor to partition the processor into up to 32 domains, with one thread for each domain. The UltraSPARC T2 and T2 Plus processors contain eight processing cores with eight threads per core, for a total of 64 threads per processor.

This project used the Sun SPARC Enterprise T5220 server as the virtualization platform. This server incorporates the UltraSPARC T2 processor and supports up to 64 threads. The SPARC Enterprise T5220 server also supports up to 64 GB memory, up to 8 internal disk drives, and integrated PCI and Ethernet connectivity. Two hot-swappable power supplies and three hot-swappable fan trays provide redundancy and greater reliability.

---

1. The terms thread, strand, hardware thread, logical processor, virtual CPU, and virtual processor are commonly used in the industry to refer to the same concept.

The remainder of this document describes the use of LDoms technology on a Sun SPARC Enterprise T5220 server. Multiple logical domains are used to run multiple instances of application server software. The goal is to explore the suitability of this solution for running applications that do not inherently scale well on large multicore systems such as the Sun SPARC Enterprise T5220 server.

---

**Note** – This document explores the use of the Sun SPARC Enterprise T5220 server as a virtualization and consolidation platform. The recently announced Sun SPARC Enterprise T5240 server, with support for two UltraSPARC T2 Plus processors, a total of 128 compute threads, and up to 128 GB of memory, offers the potential for even greater performance results.

---

## Configuration Considerations

Before attempting to configure the virtualized environment, it is important to estimate how many guest domains should be created. The total number of domains that can be created is platform dependent. At the lowest level of granularity, a domain requires at least one processor thread. Thus, a processor with 32 threads can be configured with a maximum of 32 domains.

In real world applications, however, the number of guest domains that can be configured depends largely on the nature of the application. For example, in some cases it might be possible to share the same physical network interface between two or more guest domains, while in another case, each guest domain would need a dedicated virtual network interface created on top of a dedicated physical network interface. Similarly, the amount of memory that an application server needs to perform at an acceptable level varies and may be a more limiting factor than the available number of CPU threads.

The requirements of a single application server can be used to estimate a suitable number of guest domains. This information can easily be obtained by running a single instance of the application in a non-virtualized environment on a system similar to the target virtualization system.

During the run time, collect the following information:

- Amount of memory required by the application (M)
- Utilization of the network interfaces used (N)
- Amount of CPU utilized (C)
- Disk space requirement (D)

Then, use this information to find the appropriate hardware that meets these performance requirements and budgetary constraints.

As an example, if disks (D) and network (N) resources are available in sufficient quantities, then consider CPU (C) and memory utilization (M) to determine the target number of guest domains. For example, if the application needs 10 percent of the available memory, then the maximum number of domains that can be configured is ten. However, if the percentage of CPU utilization is higher than the percentage of memory utilization, then the CPU utilization is the limiting factor. For example, if the memory utilization stays at 10 percent and 20 percent CPU is needed to run a single instance of the application with satisfactory results, then the maximum number of guest domains that can be configured is limited to five (each with 20 percent of the CPU resources).

In this particular scenario, a single instance of the application was run in a non-virtualized environment, and was able to deliver only 980 transactions per second (TPS) while consuming merely 10 percent of the CPU. Attempts to scale it further caused a significant increase in response time and the creation of too many threads, exhausting the Java heap size. Hence, it was decided to deploy the application in a virtualized environment with 6 domains as the starting point, with each domain allocated 1 CPU core (8 threads). Further measurement found that with this configuration approximately 40 percent of CPU was not utilized on a per domain basis. Hence, it was decided to double the number of domains to 12 for further performance analysis.

## Physical Test Environment

The physical test environment is summarized in Table 1.

Table 1. Physical test environment.

Component	System	Configuration Details
Database server	Sun Fire T2000	<ul style="list-style-type: none"> <li>• 16 GB RAM</li> <li>• 8 core (32 Thread) 1.2 GHz CPU</li> <li>• 2 GB QLogic PCI-Express HBA</li> </ul>
Application server	Sun SPARC Enterprise T5220	<ul style="list-style-type: none"> <li>• 64 GB RAM</li> <li>• 8 core (64 thread) 1.2 GHz CPU</li> <li>• 4 onboard Gigabit Ethernet adapters</li> <li>• 3 dual-port PCI-Express Gigabit Ethernet cards</li> <li>• 2 GB PCI-E HBA (for FC array)</li> </ul>
Front-end Clients	Sun Fire V440	<ul style="list-style-type: none"> <li>• 4 CPU</li> <li>• 16 GB RAM</li> </ul>
	Sun Fire T1000	<ul style="list-style-type: none"> <li>• 16 GB RAM</li> </ul>
	Sun Fire X4100	<ul style="list-style-type: none"> <li>• 2 AMD Dual Core CPUs</li> <li>• 8 GB RAM</li> </ul>
	Sun Fire X4200	<ul style="list-style-type: none"> <li>• 2 AMD Dual Core CPUs</li> <li>• 8 or 16 GB memory</li> </ul>
Storage	Sun StorageTek 3510 FC Array	<ul style="list-style-type: none"> <li>• 12 disks</li> <li>• Connected to Sun SPARC Enterprise T5220 server</li> </ul>

A Sun SPARC Enterprise T5220 server was used as the application server, a Sun Fire™ T2000 server was configured as the database server, and a total of 12 Sun Fire V440, T1000, and X4100/4200 servers were used as front-end clients. In addition, a Sun StorageTek™ 3510 FC Array with 12 disk drives was used for storage. All server and client machines ran the OpenSolaris operating system. The application server also ran the Logical Domains Manager 1.0.1 software.

The network configuration is shown in Figure 5. The Sun SPARC Enterprise T5220 application server contained 7 physical Gigabit Ethernet ports and was configured with 7 virtual switches. Each virtual switch was associated with a physical NIC on the server. In addition, the server was configured with up to 12 guest domains. Each domain was configured with two virtual networks: vNet0 and vNet1.

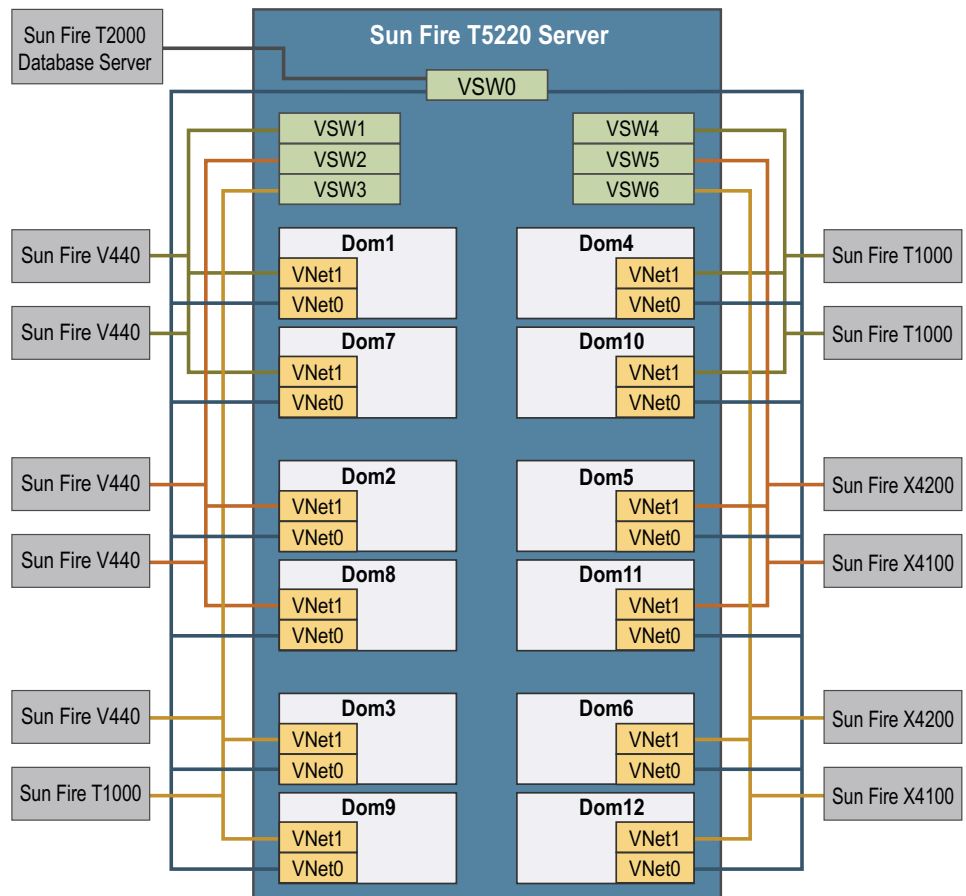


Figure 5. Network configuration used for testing.

One virtual switch, vSW0, was used to connect all domains and the database server. The remaining six virtual switches were used to separate network traffic coming from the load-generating clients. For the configuration with six guest domains (Dom1 - Dom6), each guest domain had access to a dedicated NIC via a virtual switch. For the

configuration with 12 guest domains (Dom1 - Dom12), each virtual switch and associated physical NIC was shared by two domains. For example, vsw1 was used to connect domains Dom1 and Dom7 and two Sun Fire V440 servers used as clients.

## Test Workload

The following software applications were used to generate the test workload for the scalability assessment described in this document:

- JPetStore — A demo application that simulates an online pet store
- Tomcat — Application server software
- MySQL — Database server software
- Grinder — Java load testing framework

Installation details of the JPetStore, Tomcat, MySQL, and Grinder software are described in “Appendix: Software Installation” on page 26.

### JPetStore Application

The Java™ Pet Store demo is a sample application that simulates an online pet store. Users can login, browse for pets, check their account information and order status, check out pets into shopping carts, and purchase pets by placing orders.

The JPetStore application runs in an application server. The application server talks to a back-end database that stores all information about the pets and manages the current inventory and information related to purchases, customer information, order, billing and shipping information. Customers use a Web browser to access the site. All pages are dynamically generated by the JPetStore application running on the application server. Most transactions involve one or more database interactions handled between the application server and back-end database.

Different implementations of the JPetStore demo are available. This study used the open source Apache iBATIS implementation for its simplicity and popularity. This implementation includes a default deployment on a Tomcat application server with the MySQL database. Both Tomcat and MySQL are open source software and both are widely used in the Web 2.0 space.

The JPetStore application is available for download from the following location:

<http://apache.siamwebhosting.com/ibatis/binaries/ibatis.java/>

### Tomcat and MySQL Applications

Apache Tomcat is an application server developed at the Apache Software Foundation (ASF). Tomcat implements the Java™ Servlet and the JavaServer Pages™ (JSP™) specifications from Sun Microsystems, and provides an environment to run Java applications.

MySQL is a popular open source database. This multi-threaded, multi-user SQL database management system runs as a server providing multi-user access to one or more databases.

The Tomcat and MySQL binaries were obtained from the Optimized Open Source Software Stack (Cool Stack). Cool Stack is a collection of some of the most commonly used open source applications optimized for the Sun Solaris OS platform.

For more information, please see <http://www.sunfreeware.com/coolstack.html>.

### Grinder Software

Grinder is a Java load testing framework that makes it easy to run a distributed test using many load injector machines. The Grinder was originally developed for the book *Professional Java 2 Enterprise Edition with BEA WebLogic Server* by Paco Gomez and Peter Zadrozny.

The Grinder 3 Jython-based<sup>1</sup> script engine is used to execute test scripts and inject load into the application tier. The Grinder software also provides a proxy mechanism that can be used to record a browser user session. The recorded session, a Jython-based HTTP script, is suitable for use with the Grinder software.

For more information about the Grinder software, please visit the Web site <http://grinder.sourceforge.net/>.

### Test Procedures

The Grinder proxy was first used to capture an example JPetStore user session. The recorded session was then modified to more closely match real world behavior and better stress the application server software. Next, the Sun SPARC Enterprise T5220 server was configured with varying numbers of Logical Domains, and the Grinder software was used to generate a test workload. Performance statistics were measured using standard Solaris monitoring utilities.

### Workload Generation

In this study, a JPetStore user session was recorded using the Grinder Proxy. The recorded session, a Jython-based HTTP script, was then modified to remove all inter-request sleep time, reduce database intensive requests, increase the static image requests and add a dynamic text-to-image generation request. The resulting access pattern contains more read-only activities, such as query items, than database modification operations such as the purchase activities.

---

1. Jython is a Java implementation of the popular Python programming language.

---

**Note** – For more information about Grinder 3 in general and how to use the Grinder Proxy to record a session, please visit the following Web sites:

- <http://grinder.sourceforge.net/g3/whats-new.html>
  - <http://grinder.sourceforge.net/g3/tcpproxy.html#HTTPPluginTCPProxyFilter>
- 

### Modifying Workload to Stress the Application Server

The business logic of the JPetStore application is very lightweight and consumes very little CPU of the system running the application — in this case a logical domain. However, scaling up the JPetStore application increases the load on the database. Since this exercise focused on the mid-tier hosting the application, it was decided to reduce the CPU usage on the database and increase the CPU usage on the mid-tier to make it more like real world experience. Hence, the load generator script was modified to reduce the frequency of certain database-heavy transactions and intersperse the JPetStore requests with a CPU intensive image generation request, to mimic the on-the-fly thumbnail generation of real life applications.

To perform the thumbnail generation, a small JSP script (see Table 2) was written that takes any arbitrary string as a request and converts it to a PNG format image with size, font, background and foreground color specified in the QueryString of the request. Standard Java graphics APIs from the Swing, AWT and ImageIO packages were used to convert the text into a PNG format image. Because the generated images need to be stored to disk before they can be sent out, this JSP script also added moderate disk I/O load.

*Table 2. JSP script used for thumbnail generation.*

```
f = Font.createFont(Font.TRUETYPE_FONT, new FileInputStream(font));
f = f.deriveFont(size);

BufferedImage buffer = new BufferedImage(1, 1, BufferedImage.TYPE_INT_RGB);
Graphics2D g2 = buffer.createGraphics();

FontRenderContext fc = g2.getFontRenderContext();
Rectangle2D bounds = f.getStringBounds(text, fc);

width = (int)bounds.getWidth();
height = (int)bounds.getHeight();

g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                    RenderingHints.VALUE_ANTIALIAS_ON);
g2.setFont(f);

g2.setColor(background);
g2.fillRect(0,0,width,height);
g2.setColor(color);
g2.drawString(text,0,(int)-bounds.getY());
```

## Workload Summary

Table 3 summarizes the relative access frequency of each request type used in the test workload.

Table 3. Access frequencies of each request type.

Number of Requests	Request Type
10	GET jpetstore
10	GET /
10	GET jpetstore.css
40	GET index.shtml
10	GET signonForm.shtml
10	POST signon.shtml
11	GET editAccountForm.shtml
1	POST editAccount.shtml
30	GET viewCategory.shtml
30	GET viewProduct.shtml
30	GET viewItem.shtml
3	GET addItemToCart.shtml
2	POST updateCartQuantities.shtml
1	GET switchMyListPage.shtml
1	GET checkout.shtml
1	GET newOrderForm.shtml
1	POST newOrder.shtml
1	GET newOrder.shtml
10	GET listOrder.shtml
10	GET signoff.shtml
92	GET imgtext.jsp
92	GET static.html (19 thumbnail images)

## Server and Network Configuration

For this project, the application server (a Sun SPARC Enterprise T5220 server) was first configured with 1 control domain and 6 guest domains. Each guest domain was configured with one Tomcat application sever running the JPetStore application. After measuring the performance from this configuration, the application server was reconfigured with 12 guest domains and the tests were repeated.

Configuration parameters for testing with 6 guest domains are provided in “Configuration with 6 Guest Domains” below. Configuration parameters for testing with 12 guest domains are provided in “Configuration with 12 Guest Domains” on page 17. Results from these test configurations are included in “Results and Analysis” on page 20.

### Configuration with 6 Guest Domains

The following steps describe the process of configuring the application server with six guest domains. The steps include creating a control domain and six guest domains on the Sun SPARC Enterprise T5220 server; configuring Tomcat on each guest domain; and configuring the Grinder software on the client machines.

In this configuration, the control domain is configured with 2 cores (16 strands) and 16 GB RAM. Each guest domain is configured with 1 core (8 strands) and 8GB RAM.

---

**Note** – Refer to *LDOMs 1.0.1 Administration Guide* for detailed instructions on how to create and configure LDomS.

---

1. Set up the control domain on the Sun SPARC Enterprise T5220 server:
  - a. Create six Virtual Disk Servers:

```
ldm add-vds primary-vds1 primary
...
ldm add-vds primary-vds6 primary
```

- b. Create seven Virtual Switches and assign them to the physical NICs:

```
ldm add-vsw net-dev=e1000g1 primary-vsw0 primary
...
ldm add-vsw net-dev=e1000g7 primary-vsw6 primary
```

- c. Create the Virtual Console Concentrator Service:

```
# ldm add-vcc port-range=5000-5100 primary-vcc0 primary
```

- d. Set up the Control Domain with the following configuration:
- 16 VCPUs
  - 16 GB RAM

```
# ldm set-vcpu 16 primary
# ldm set-memory 16G primary
# ldm add-config initial
```

- e. Confirm the configuration is ready to be used at the next reboot:

```
# ldm list config
factory-default [current]
initial [next]
```

- f. Reboot the server to enable the configuration changes to take place. This step is necessary to release the resources needed to create the logical domains.

2. Configure the virtual switches as the primary interfaces.

- a. Print out the address information:

```
# ifconfig -a
```

- b. Plumb the virtual switches:

```
# ifconfig vsw0 plumb
..
# ifconfig vsw6 plumb
```

- c. Unplumb the physical device assigned to the virtual switch (`net-dev`), which is `e1000g1` in this example. Repeat for all seven virtual switches.

```
# ifconfig e1000g1 down unplumb
```

- d. Migrate properties of the physical network device to the vswitch. Repeat for all seven virtual switches.

For static IP addresses, use the following command:

```
# ifconfig vsw0 IP_of_e1000g1 netmask netmask_of_e1000g1 broadcast + up
```

For dynamic IP addresses (DHCP), use the following command:

```
# ifconfig e1000g1 down unplumb
```

- e. Make the configuration file modifications permanent. Repeat for all seven virtual switches:

```
# mv /etc/hostname.e1000g1 /etc/hostname.vsw0
```

3. Configure the guest domains, Dom1 - Dom6, on the Sun SPARC Enterprise T5220 server with the following characteristics:

- 8 VCPU
- 8 GB RAM
- 2 VNets (vnet0 and vnet1)

All domains use `primary-vsw0` for connection to the database. Each domain uses an additional switch for connection to the application server. For example, Dom1 uses `primary-vsw1`, Dom2 uses `primary-vsw2`, and Dom3 uses `primary-vsw3`.

The following commands were used to configure Dom1:

```
ldm add-domain dom1
ldm add-vcpu 8 dom1
ldm add-memory 8g dom1
ldm add-vnet vnet0 primary-vsw0 dom1
ldm add-vnet vnet1 primary-vsw1 dom1
```

4. Set up the Tomcat configuration files for each guest domain, Dom1 – Dom6.

In each domain Dom1 - Dom6, modify the `/opt/catalina.sh` file to have the following values for the JVM:

```
JAVA_OPTS="-server -Djava.awt.headless=true -XX:+AggressiveHeap -Xmx3g
-Xms3g -Xmn2g -XX:+UseParallelOldGC -XX:+UseParallelGC
-XX:ParallelGCThreads=4 -XX:+UseBiasedLocking -Xloggc:/tmp/gc.log"
```

5. Configure the Grinder software on all clients.
  - a. Choose one client to configure as the master client for the Grinder software. The master client is used to start test runs, aggregate results, and report performance data from all other load generating clients.
  - b. Create a `jpetstore.py` file and place it on each Grinder client. This test script defines the flow of the requests that are sent to the server. See “Workload Generation” on page 11 for details on the test script used in this exercise.

- c. Modify the `grinder.properties` file on each client. The following properties were modified for testing:

```
grinder.processes=1
grinder.threads=250
grinder.runs=0
grinder.duration=600000
grinder.script=/usr/grinder/jpetstore.py
grinder.jvm.arguments=-server -XX:+AggressiveHeap
    -Xloggc:/tmp/gc.log -Xms1024m -Xmx1024m -Xmn300m
grinder.reportToConsole.interval=500
grinder.initialSleepTime=0
grinder.sleepTimeFactor=1
grinder.sleepTimeVariation=0.2
grinder.logProcessStreams=false
grinder.reportTimesToConsole=false
grinder.debug.singleprocess=false
grinder.useNanoTime=true
```

### Configuration with 12 Guest Domains

The following steps describe the process of configuring the application server with 12 logical domains. The process includes creating a control domain and 12 guest domains on the Sun SPARC Enterprise T5220 server; configuring Tomcat on each guest domain; and configuring the Grinder software on the client machines.

In this configuration, the control domain is configured with 2 cores (16 strands) and 16 GB RAM. Each guest domain is configured with 1/2 core (4 strands) and 4 GB RAM.

---

**Note** – Refer to *LDOMs 1.0.1 Administration Guide* for detailed instructions on how to create and configure LDOMs.

---

1. Set up the control domain on the Sun SPARC Enterprise T5220 server:
  - a. Create 12 Virtual Disk Servers:

```
ldm add-vds primary-vds1 primary
...
ldm add-vds primary-vds12 primary
```

- b. Create seven Virtual Switches and assign them to the physical NICs:

```
ldm add-vsw net-dev=e1000g1 primary-vsw0 primary
...
ldm add-vsw net-dev=e1000g7 primary-vsw6 primary
```

- c. Create the Virtual Console Concentrator Service:

```
ldm add-vcc port-range=5000-5100 primary-vcc0 primary
```

- d. Set up the Control Domain with the following configuration:
- 16 VCPUs
  - 16 GB RAM

```
# ldm set-vcpu 16 primary
# ldm set-memory 16G primary
# ldm add-config initial
```

- e. Confirm the configuration is ready to be used at the next reboot:

```
# ldm list config
factory-default [current]
initial [next]
```

- f. Reboot the server to enable the configuration changes to take place. This step is necessary to release the resources needed to create the logical domains.

2. Configure the virtual switches as the primary interfaces.

Assign virtual switches VSW1 – VSW6 to VNet 1 in each of logical domains, Dom1 – Dom12.

- a. Print out the address information:

```
# ifconfig -a
```

- b. Plumb the virtual switches:

```
# ifconfig vsw0 plumb
..
# ifconfig vsw6 plumb
```

- c. Unplumb the physical device assigned to the virtual switch (`net-dev`), which is `e1000g1` in this example. Repeat for all seven virtual switches.

```
# ifconfig e1000g1 down unplumb
```

- d. Migrate properties of the physical network device to the vswitch. Repeat for all seven virtual switches.

For static IP addresses, use the following command:

```
# ifconfig vsw0 IP_of_e1000g1 netmask netmask_of_e1000g1 broadcast + up
```

For dynamic IP addresses (DHCP), use the following command:

```
# ifconfig e1000g1 down unplumb
```

- e. Make the configuration file modifications permanent. Repeat for all seven virtual switches:

```
# mv /etc/hostname.e1000g1 /etc/hostname.vsw0
```

3. Configure the remaining LDoms (Dom1 - Dom12) with the following characteristics:
  - 4 VCPUs
  - 4 GB RAM
  - 2 VNets

All domains use `primary-vsw0` for connection to the database. Each domain uses an additional switch for connection to the application server. For example, Dom1 uses `primary-vsw1` and Dom2 uses `primary-vsw2`.

The following commands were used to configure Dom1:

```
ldm add-domain dom1
ldm add-vcpu 4 dom1
ldm add-memory 4g dom1
ldm add-vnet vnet0 primary-vsw0 dom1
ldm add-vnet vnet1 primary-vsw1 dom1
```

4. Tomcat configuration files for Dom1 – Dom12.

In each domain Dom1 - Dom12, modify the `/opt/catalina.sh` file to have the following values for the JVM:

```
JAVA_OPTS="-server -Djava.awt.headless=true -XX:+AggressiveHeap
-Xmx2048m -Xms2048m -Xmn800m -Xss256k -XX:+UseParallelOldGC
-XX:+UseParallelGC -XX:ParallelGCThreads=4 -XX:+UseBiasedLocking
-Xloggc:/tmp/gc.log"
```

5. Configure the Grinder software on all clients.
  - a. Choose one client to configure as the master client for the Grinder software. The master client is used to start test runs, aggregate results, and report performance data from all other load generating clients.
  - b. Create a `jpetstore.py` file and place it on each Grinder client. This test script defines the flow of the requests that are sent to the server. See “Workload Generation” on page 11 for details on the test script used in this exercise.

- c. Modify the `grinder.properties` file on each client. The following properties were modified for testing:

```
grinder.processes=1
grinder.threads=250
grinder.runs=0
grinder.duration=600000
grinder.script=/usr/grinder/jpetstore.py
grinder.jvm.arguments=-server -XX:+AggressiveHeap
    -Xloggc:/tmp/gc.log -Xms1024m -Xmx1024m -Xmn300m
grinder.reportToConsole.interval=500
grinder.initialSleepTime=0
grinder.sleepTimeFactor=1
grinder.sleepTimeVariation=0.2
grinder.logProcessStreams=false
grinder.reportTimesToConsole=false
grinder.debug.singleprocess=false
grinder.useNanoTime=true
```

## Results and Analysis

The following monitoring utilities (see Table 4) were used to monitor the critical resources during testing. In addition, the JVM flag `-Xloggc` was used to monitor garbage collection activity.

Table 4. Monitoring utilities.

Utility	Execution Location	Description
<code>mpstat 1</code>	DB server, control domain, all guest domains	Monitor CPU utilization
<code>vmstat -p 1</code>	Control domain, all guest domains	Monitor memory utilization
<code>iostat -xtczn 1</code>	DB server, control domain	Monitor disk utilization
<code>nicstat 1</code>	All guest domains	Monitor network utilization between guest domains and clients, and guest domains and DB server.

## Results for Six Domains

For this set of tests, the Sun SPARC Enterprise T5220 application server was configured with one control domain and six guest domains. The control domain was configured with 16 strands (2 cores) and 16 GB RAM. Each guest domain was configured with 8 strands (one core) and 8 GB RAM. On average each guest domain was approximately 60 percent utilized during the load test.

The throughput results for testing with six domains are listed in Table 5. As these results indicate, a baseline rate of 952 TPS was achieved with one guest domain. With six guest domains, the number of TPS increased by a factor of 5.84 to 5560 TPS.

Table 5. Throughput results for testing with six domains.

Number of Domains	Throughput (TPS)
1	952
2	1920
3	2810
4	3780
5	4660
6	5560

Figure 6 illustrates these results graphically, showing linear throughput scalability as the number of domains was increased from one to six. The average throughput per domain decreases by only a small percentage as the number of domains increases, with the six domain configuration averaging 927 TPS per domain as compared to the 952 TPS for a single domain configuration.

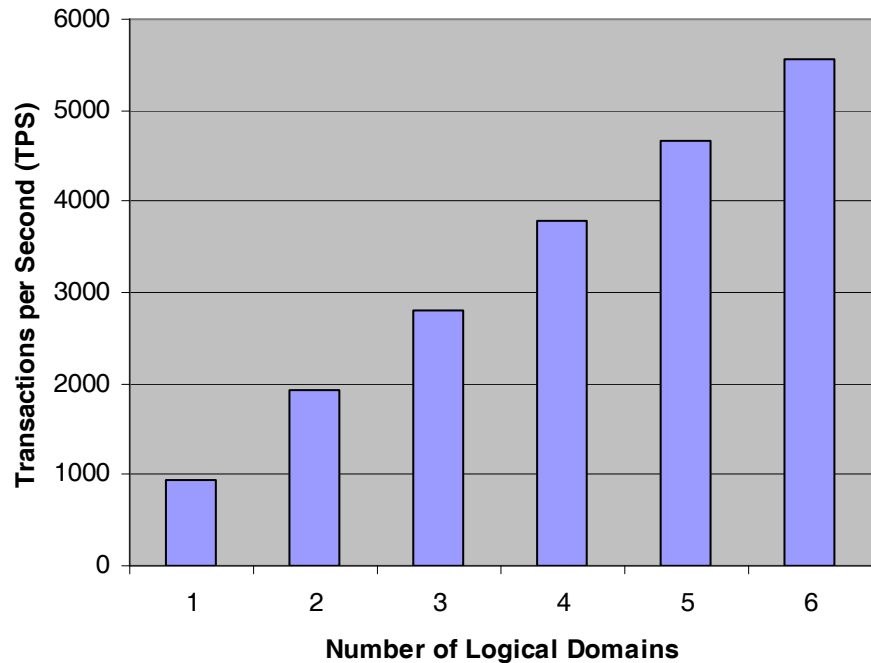


Figure 6. Throughput results for testing with six domains.

### Results for Twelve Domains

For this set of tests, the Sun SPARC Enterprise T5220 application server was configured with 1 control domain and 12 additional guest domains. The control domain was configured with 16 strands (2 cores) and 16 GB RAM. Each guest domain was configured with 4 strands (half of a core) and 4 GB RAM. Each guest domain was running at almost 100 percent CPU utilization during the load test.

Performance results for testing with 12 domains are included in Figure 7. CPU utilization is aggregated against the 48 strands that were assigned to the 12 guest domains. Network throughput is shown in the last 2 columns, one for aggregated throughput with all clients, the other for throughput with the back-end MySQL database.

Table 6. Throughput results for testing with 12 domains.

Number of Domains	Throughput (TPS)	Control CPU% (16-strand)	Guest CPU% (48-strand)	Network from/to clients (Mbit/s)	Network from/to DB (Kbit/s)
1	822	1.6	7.9	4.8 / 33.5	323 / 193
2	1560	3.0	16.2	8.3 / 60	566 / 323
4	3140	5.9	32.4	16.6 / 121	1206 / 643
6	4560	8.5	48.8	22.6 / 173	1439 / 911
8	5860	11.2	65.2	28.6 / 224	1650 / 1160
10	7160	14.0	82.3	33.4 / 265	1838 / 1329
12	8400	16.6	98.4	37.5 / 306	2180 / 1589

Figure 7 illustrates these results graphically, showing performance gains as the number of logical domains was increased from 1 to 12.

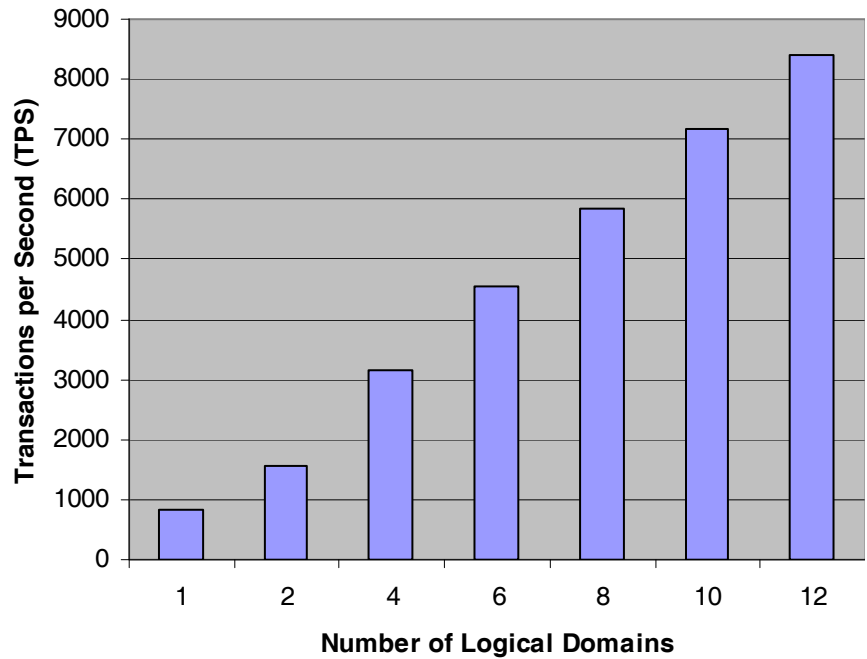


Figure 7. Throughput results for testing with 12 domains.

## Analysis of Results

For the purpose of this exercise, the load on the database server was kept low. Based on CPU and network utilization data collected during the 6-domain test, it was clear that the server was under utilized. Thus, it was determined that the number of domains could be doubled at a slightly reduced per-domain throughput, especially if the per-domain Java heap size could be reduced from 5 GB to 2 GB. With increased memory pressure and lower free CPU availability, it was found that, though the throughput dropped by 15%-21% on a per-domain basis, the total throughput obtained from a single Sun SPARC Enterprise T5220 server increased by 51% after moving from 6 Domains (5560 TPS) to 12 Domains (8400TPS).

Because disk utilization was relatively low, it was not considered in the decision making process to move from 6 to 12 domains. Also, since the network utilization by each guest was below 50% of the maximum capacity of the physical network interface, it was decided to deploy two guest domains on the same physical interface when moving the configuration from 6 to 12 domains.

The configuration with 12 guest domains fully used 6 of the 8 cores of the Sun SPARC Enterprise T5220 server. The control domain still had approximately 80 percent CPU available. It is possible to add one more domain by removing one core from the control domain. While no testing was done to confirm this, it is estimated that adding an additional guest domain would increase the total TPS by approximately 650. Attempting to push the load further than 13 guest domains would likely cause an increase in response time and the throughput would start leveling off.

## Summary

This paper explored the use of Sun's LDom technology and the Sun SPARC Enterprise T5220 server as a means to increase application software scalability and improve system utilization. A SPARC Enterprise T5220 server was configured with multiple logical domains, with each domain running its own independent Solaris OS and instance of the application software. Web application performance was measured for configurations ranging from 1 to 12 guest domains.

From the results produced in this scenario, it is clear that configurations with 6 logical domains exhibited scalable performance improvements and still did not fully utilize system resources of the SPARC Enterprise T5220 server. A configuration with 12 logical domains increased the overall throughput by over 50 percent compared to the 6-domain configuration, while almost fully utilizing the available CPU resources assigned to the logical domains. Using LDom virtualization technology on Sun's CoolThreads technology-based servers, like the SPARC Enterprise T5220 server, was found to be an effective solution for increasing the scalability of Web application software and improving system utilization.

## About the Authors

A member of the Global Systems Engineering group, Lee Anne Simmons is an Engagement Architect, focused on providing Sun's customers technical solutions to solve their business requirements. She has extensive background in Information Technology with a focus on enterprise applications.

A member of Sun's Performance Application Engineering (PAE) group, Ning Sun is currently at work on several Web and application server performance projects. Her technical focus centers on the performance of networks, Java Technology, servlet and EJB™ containers. Ning has extensive background in the development of industry-standard benchmarks including SPECWeb, SPECjAppServer and TPC-W.

## Acknowledgements

The authors would like to recognize Pallab Bhattacharya for his contributions to this article.

## References

*Logical Domains (LDoms) 1.0.1 Administration Guide*. Sun Microsystems, Inc. September 2007, Revision A. Part No. 820-3268-10.

Shoumack, Tony. "Beginner's Guide to LDoms: Understanding and Deploying Logical Domains," *Sun BluePrints OnLine*, July 2007.

<http://www.sun.com/blueprints/0207/820-0832.pdf>

Sun SPARC Enterprise T5220 Server: <http://www.sun.com/servers/coolthreads/t5220/>

## Related Resources

- **Grinder:** <http://grinder.sourceforge.net/>  
Grinder is a Java load testing framework for running a distributed test using many load injector machines. It is freely available under a BSD-style open-source license.
- **JPetStore:**  
<http://apache.siamwebhosting.com/ibatis/binaries/ibatis.java/>  
JPetStore is a sample application that simulates an on-line pet store.
- **MySQL Enterprise: Drivers:** <http://www.mysql.com/products/connector/>  
MySQL provides standards-based drivers.
- **Optimized Open Source Software Stack (Cool Stack) for the Sun Solaris Operating System:** <http://cooltools.sunsource.net/coolstack/>  
Contains commonly used open source applications for Solaris, including MySQL and Tomcat.

## Ordering Sun Documents

The SunDocs<sup>SM</sup> program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

## Accessing Sun Documentation Online

The docs.sun.com web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is

<http://docs.sun.com/>

To reference Sun BluePrints OnLine articles, visit the Sun BluePrints OnLine Web site at:

<http://www.sun.com/blueprints/online.html>

## Appendix: Software Installation

The following sections describe the installation procedures used in this project:

- “Database Server Installation” on page 26 describes the installation procedures for the database server.
- “Application Server Installation” on page 29 describes the installation procedures for the application server.
- “Client Installation” on page 30 describes the installation of the Grinder load generation software on the client machines.

### Database Server Installation

The following steps were used to install and configure the OpenSolaris OS, MySQL database, and JPetStore database on a Sun Fire T2000 server.

1. Install the OpenSolaris software, which includes J2SE. The OpenSolaris software can be downloaded from the following Web site:

<http://www.opensolaris.org>

---

**Note** – While this example used the OpenSolaris software, the Solaris 10 OS (version 08/07 or greater) can also be used. Solaris 10 OS is available for download from the following location: <http://www.sun.com/software/solaris/get.jsp>

---

2. Install the CoolStack MySQL database.
  - a. The MySQL software can be downloaded from the following location:
 

<http://cooltools.sunsource.net/coolstack/>
  - b. Download the Cool Stack 1.1 MySQL software to the database server. From the download page, choose:

MySQL 5.0.33 64bit Database Server, English

The following bzip2 file is downloaded to the local directory:

CSKmysql\_sparc.pkg.bz2

- c. On the database server, use `bunzip2` to uncompress the file:

```
# bunzip2 CSKmysql_sparc.pkg.bz2
```

- d. Use the `pkgadd` file to add the package to the OpenSolaris OS:

```
# pkgadd -d CSKmysql_sparc.pkg
```

The files are installed in the `/opt/coolstack/mysql` directory.

- e. Use the following command to install the database:

```
# /opt/coolstack/mysql/bin/mysql_install_db
```

---

**Note** – Reference the README file for more information on the MySQL installation.

---

3. Configure the MySQL database.
  - a. Copy a MySQL configuration file to configuration directory.

```
# cp /opt/coolstack/mysql/share/mysql/my-large.cnf /etc/my.cnf
```

In this example, the `my-large.cnf` configuration file was used, as the database was installed on a large server. Other configuration files are available, and may be more appropriate for other installations (such as installation on a laptop or smaller desktop system).

- b. Edit the `/etc/my.cnf` file to contain the MySQL parameters used for testing. Remove the comment symbols from the InnoDB lines, to enable the use of the InnoDB engine and modify the values to better handle the number of connections to the database.

The following settings were used for this exercise:

```
[mysqld]
sql-mode = IGNORE_SPACE
#transaction-isolation = READ-COMMITTED
datadir=/data/mysql/var/
port = 3306
skip-locking
max_allowed_packet = 1M
max_connections=700
max_connect_errors=100
sort_buffer_size = 2M
read_buffer_size = 2M
read_rnd_buffer_size = 8M
thread_concurrency = 4
server-id = 1
key_buffer_size=2048M
key_buffer=2048M
query-cache-type = 1
query-cache-size = 1024M
table_cache=256
sort_buffer=2M
myisam_sort_buffer_size = 64M
tmp_table_size = 1024M
thread_cache_size = 8
thread_cache = 8
innodb_data_home_dir = /data/mysql/var/
innodb_data_file_path = ibdata1:10000M:autoextend
innodb_log_group_home_dir = /data/mysql/var/
innodb_log_arch_dir = /data/mysql/var/
innodb_checksums = 0
innodb_buffer_pool_size = 5000m
```

```

innodb_log_file_size = 1900M
innodb_log_buffer_size = 8M
innodb_flush_log_at_trx_commit = 1
innodb_lock_wait_timeout = 300
innodb_thread_concurrency = 1000
innodb_sync_spin_loops = 10
innodb_locks_unsafe_for_binlog = 1
innodb_max_dirty_pages_pct=15
innodb_support_xa=0
mysqldump]
quick
max_allowed_packet = 16M
[mysql]
no-auto-rehash
[[isamchk]
key_buffer = 256M
sort_buffer_size = 256M
read_buffer = 2M
write_buffer = 2M
[myisamchk]
key_buffer = 256M
sort_buffer_size = 256M
read_buffer = 2M
write_buffer = 2M
[mysqlhotcopy]
interactive-timeout

```

4. Install the JPetStore database on the MySQL server.

- a. Download the JPetStore database from the following location:

<http://archive.apache.org/dist/ibatis/binaries/ibatis.java/jpetstore-5.0.zip>

Follow the directions included with this distribution to install the necessary files on both the application server and the database server.

- b. Download the following three files from the default JPetStore location (`/usr/jpetstore/JPetStore-5.0/src/ddl/mysql`) to the MySQL database server:

- `jpetstore-mysql-create-user.sql`
- `jpetstore-mysql-dataload.sql`
- `jpetstore-mysql-schema-innodb.sql`

- c. Execute the following scripts to load the data:

```

# mysql -uroot -p < jpetstore-mysql-schema-innodb.sql
# mysql -uroot -p < jpetstore-mysql-dataload.sql
# mysql -uroot -p < jpetstore-mysql-create-user.sql

```

- d. To verify the installation, use the following MySQL commands:

```

# mysql -uroot -p
PASSWORD: *****
mysql> show databases

```

If the database has been configured correctly, these commands will display the JPetStore database.

## Application Server Installation

The following steps were used to install the OpenSolaris OS and install and configure related application software on a Sun SPARC Enterprise T5220 server.

1. Install the OpenSolaris OS software. Refer to instructions in Step 1 of the “Database Server Installation” on page 26.
2. Install the Logical Domains Manager 1.0.1 software. Refer to the *Logical Domains (LDoms) 1.0.1 Administration Guide* for detailed installation instructions.
3. Install the Tomcat application server software.

- a. Download the Tomcat application server software from the following location:

```
http://cooltools.sunsource.net/coolstack/
```

- b. Unzip the file on the application server:

```
# bunzip2 CSKtomcat_sparc.pkg.bz2
```

- c. Use the `pkgadd` command to add the package to the Solaris OS:

```
# pkgadd -d /etc/patch/ CSKtomcat_sparc.pkg
```

- d. The files are loaded into the `/opt/coolstack` directory. On completion, the following message is displayed:

```
Installation of <CSKtomcat> was successful.
```

4. Configure the Tomcat application server software

- a. Use the `startup.sh` script to start Tomcat:

```
# cd /opt/coolstack/apache-tomcat-5.5.25/bin
# ./startup.sh
```

- b. Start a Web browser on a client machine, and go to the following location, where *appserver* is the name of your application server:

```
http://appserver:8080
```

If Tomcat is installed correctly, the Tomcat application server is displayed in the browser window.

5. Install the MySQL JDBC Driver.
  - a. Download the JDBC driver, `mysql-connector-java-5.0.7-bin.jar`, from the following location:

```
http://www.mysql.com/products/connector/
```

- b. Copy the `.jar` file to the Tomcat `common/lib` directory.
6. Deploy JPetStore on the Tomcat application server.
  - a. Locate the `jpetstore.war` file in the JPetStore directory structure:

```
/usr/jpetstore/JPetStore-5.0/build/wars
```

- b. Copy the `jpetstore.war` file to the local Tomcat `webapps` directory. In this example, the file was copied to `/opt/apache-tomcat-5.5.25/webapps`.
  - c. Configure the JPetStore driver.

First, locate the `database.properties` configuration file in the local Tomcat directory. In this example, the configuration file can be found at:

```
/opt/apache-tomcat-5.5.25/webapps/jpetstore/WEB-INF/classes/properties/database.properties
```

Edit the `database.properties` file, and change the values for the following MySQL parameters, substituting the name of database server:

```
driver=com.mysql.jdbc.Driver
url=jdbc:mysql://jpetdb:3306/jpetstore
username=jpetstore
password=jpetstore
```

## Client Installation

1. Install Grinder on each of the client machines. The Grinder source can be downloaded from the following location:

```
http://grinder.sourceforge.net
```

Follow the instructions provided with the Grinder software distribution to configure Grinder on each client.



**Sun Microsystems, Inc.** 4150 Network Circle, Santa Clara, CA 95054 USA **Phone** 1-650-960-1300 or 1-800-555-9SUN (9786) **Web** [sun.com](http://sun.com)

© 2008 Sun Microsystems, Inc. All rights reserved. Sun, Sun Microsystems, the Sun logo, BluePrints, CoolThreads, Java, JavaServer Pages, JSP, OpenSolaris, Solaris, StorageTek, and Sun Fire are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the US and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc. Information subject to change without notice. Printed in USA 04/08

