Application and Database Server Consolidation on the Sun Fire™ X4600 Server using Solaris™ Containers

Kevin Kelly, Strategic Applications Engineering

Sun BluePrints™ OnLine — October 2006

Part No 820-0040-10
Revision 1.0, 10/12/06
Edition: October 2006

Please
Recycle

Table of Contents

Application and Database Consolidation on the Sun Fire™ X4600 Server using Solaris™ Containers

Reducing the costs of IT infrastructure and improving the manageability and efficiency of application and database services pose significant challenges for many organizations. As performance and functionality requirements increase, it’s not uncommon for the number of systems in a data center to also increase. Although adding servers can help meet increasing demands, using large numbers of servers increases management complexity, drives up hardware and software license costs, and requires increased space, cooling, and power resources. In addition, many companies are discovering that their servers are running at low capacity utilization rates. Recent studies from IDC[5] report that server consolidation is increasing in many data centers. Consolidating database and application services from multiple systems to a single high-performance server can help simplify management, improve system utilization, and increase the efficiency of delivering application services.

The combined capabilities of the Sun Fire™ X4600 server and Solaris™ Containers technology afford considerable promise as a consolidation platform. The Sun Fire X4600 server provides high performance, optimized energy efficiencies, and unparalleled scalability and virtualization options. Solaris Containers provide an isolated and secure runtime environment for applications, enabling multiple services to run efficiently and without conflict on the same platform.

This paper explores the use of a Sun Fire X4600 server as a consolidation platform for multiple database and Java™ 2 Platform, Enterprise Edition (J2EE™ platform) application servers. It describes the processes and methodologies used in the consolidation, and details the steps used to configure the Solaris Containers. In addition, this paper describes the J2EE application server workload testing used to determine the effectiveness of this approach and validate the benefits of consolidating these services on a single system.

Technology Under Evaluation

Successful application and database consolidation requires a range of technologies working together effectively. This article evaluates the Sun Fire X4600 server, running the Solaris 10 Operating System (OS) and using Solaris Containers to partition the application and database instances, as a consolidation platform. Software components used in this evaluation scenario include the Sun Java Platform, Standard Edition (Java SE platform), the WebLogic Server 9.1 from BEA Systems, Inc., and the DB2 8.2.4 Universal Database (UDB) from IBM.

Sun Fire X4600 Server

The Sun Fire X4600 server, a high performance system with up to eight AMD Opteron™ processors, is the industry's first 4 U modular x64 server that is expandable to 16 cores. In addition to the eight AMD processors, the Sun Fire X4600 platform supports up to 128 GB of main memory, four Gigabit Ethernet onboard network interfaces, and four 73 or 146 GB SAS disk drives. The Sun Fire X4600 server includes six PCI-Express and two PCI-X slots for additional I/O throughput. Its four redundant 850W power

supplies, hot-swappable disk drives, and robust remote management capability using Integrated Lights-Out Management (ILOM) help provide excellent support for enterprise-class applications.

The Sun Fire X4600 server used for this workload consolidation contains eight dual-core AMD Opteron Model 885 (2.6 GHz) processors and is configured with 64 GB of memory using thirty-two 2 GB DIMMs. Two of the four internal SAS disks are configured as a mirrored pair using the onboard LSI1064 SAS RAID controller to provide durable media for the application server transaction logs. While the Sun Fire X4600 server supports multiple operating systems, including Linux and Windows, the system in this scenario employs the Solaris 10 OS.

Solaris Containers

Multiple technologies such as *Solaris Projects*, *Zones*, and *Resource Pools* in the Solaris OS provide enhanced capabilities for managing system resources. These technologies allow users to create Solaris Containers, applications or services that have one or more resource boundaries associated with it. These virtual boundaries can specify processor sets or limit CPU and network bandwidth, and thus Solaris Containers are a prime enabler of server consolidation.

The *Project* facility in the Solaris OS provides a network-wide administrative identifier for related work. Users or processes can be grouped into projects, each with a unique identifier and set of resource controls. Thus a project can be used to represent a particular workload associated with a set of users. Resources such as resource pools, CPU shares, and shared memory for each project can be managed using the project database and include controls for scheduling and consumption.

Solaris Zones, a unique partitioning technology in the Solaris OS, can be used to provide an isolated and secure environment for running applications. A zone creates a virtualized operating system environment within a single instance of the Solaris OS and establishes boundaries for resource consumption. Zones provide a means to isolate applications, preventing processes in one zone from interfering with those running in another zone.

Resource Pools provide the capability to partition resources so that consumption by separate workloads does not overlap. Administrators can use resource pools to create a persistent configuration of processor sets and assign scheduling classes. CPUs in a multiprocessor system can be logically partitioned into processor sets and bound to a resource pool, which in turn can be assigned to a Solaris Zone. The dynamic features of resource pools provide flexibility by enabling administrators to adjust system resources in response to changing workload demands.

Application and Database Server Software

The following major software components are used on the Sun Fire X4600 server to execute the J2EE application server workload.

- Application Server

  The WebLogic Server 9.1 from BEA Systems, Inc., is used to deploy and execute the J2EE application component of the benchmark workload used for evaluation. WebLogic Server 9.1 is a fully compliant J2EE 1.4 application server and implements the latest J2EE standards to support enterprise-class web

services, including Enterprise JavaBeans™ (EJB™) 2.1, Java Message Service (JMS) 1.1 and the Java Database Connectivity (JDBC™) API 3.0.

Six unique instances of the WebLogic Server are deployed in this evaluation, each running in a separate Solaris Container.

- Database Server

  The DB2 8.2.4 Universal Database (UDB) from IBM is used to process the database transactions required to support the workload. The IBM DB2 Universal Java Database Connectivity (JDBC) drivers are configured to enable the WebLogic 9.1 application server to access the database. The DB2 database software is also deployed in its own Solaris Container with separate resource controls to manage Solaris IPC facilities.

Consolidation using Solaris Containers

Solaris Containers provide a logical way to consolidate multiple application and database servers onto a single platform. By hosting separate application and database servers in zones, they can be isolated to help ensure transactions in one zone are secure from transactions executed by other server software in another zone. In addition, CPU resources can be managed using dynamic resource pools to prevent CPU-intensive operations running on one server from impacting others.

This paper describes a methodology using Solaris zones and resource pools to consolidate application and database services onto a single Sun Fire X4600 server. For this project, six unique local zones are created and each is configured to execute a separate instance of the WebLogic application server (see Figure 1). Additionally, a separate zone is configured to execute the database software.

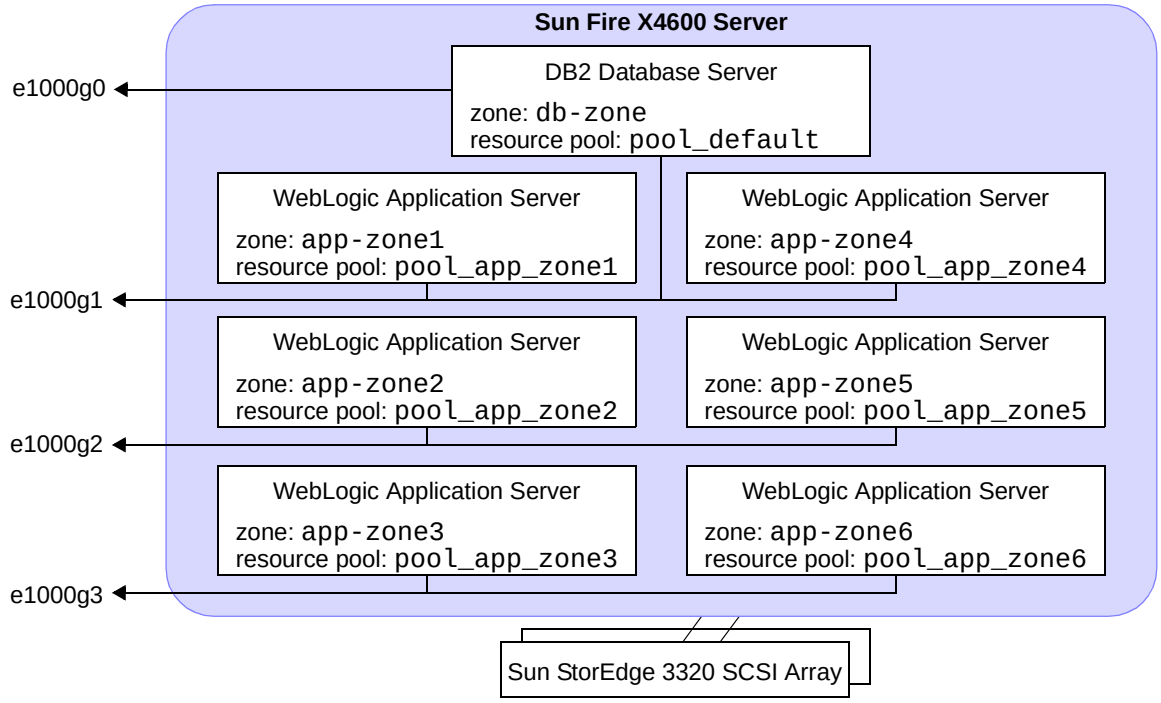

Figure 1. Application and database server consolidation topology on the Sun Fire X4600 server.

The Sun Fire X4600 server features 16 logical CPUs[1]. While Solaris Zones provide a virtual environment to shield each server instance, by default all zones have access to the full set of CPUs enabled in the system. In the scenario described in this article, four logical CPUs are configured in the default resource pool, which is bound to the local zone used to execute the database application. To help ensure that each application server instance does not utilize excessive CPU cycles, the system is configured with six additional resource pools, each configured with two logical CPUs. These resource pools are then bound to the six zones created for the application server instances.

The following sections show the commands used to configure the application and database server software on the Sun Fire X4600 server using Solaris zones and resource pools.

Note – Local zone setup and configuration information can be found in generally available system administration documents [2][3]. Please refer to these documents for detailed information on the steps needed to install and configure local zones and to setup resource pools.

Configure the Resource Pools

The steps in this section create and configure the resource pools that will be associated with the zones.

1. Enable the resource pools facility and create a default configuration using the following two `pooladm` commands. The first command enables the resource pools facility, and the second saves the active configuration to the default file, `/etc/pooladm.conf`:

```
global# pooladm -e
global# pooladm -s
```

2. Create and bind a separate resource pool and processor set for each zone using a minimum and a maximum of two logical CPUs.
   a. The following three `poolcfg` commands create and bind the resource pool and processor set for the first zone. The first command creates a processor set named `pset_app_zone1`; the second command creates a resource pool named `pool_app_zone1`; and the third command binds the newly created processor set and resource pool:

```
global# poolcfg -c 'create pset pset_app_zone1 (uint pset.min = 2; uint pset.max
= 2)'
global# poolcfg -c 'create pool pool_app_zone1'
global# poolcfg -c 'associate pool pool_app_zone1 (pset pset_app_zone1)'
```

   b. Similarly, use the `poolcfg` utility to create the processor sets and resource pools for zones two through six using the same minimum and maximum CPU resource parameters.
3. Use the `pooladm -c` command to save the resource pool configuration created in Step 2 above:

```
global# pooladm -c
```

1. Each logical CPU corresponds to a physical CPU core on the Sun Fire X4600 server.

4. Verify the configuration using the `pooladm` command (with no arguments) to report the processor sets and pool bindings. See “Resource Pool Configuration” on page 17 for sample output.

```
global# pooladm
```

Configure the Zones for the Application Server Instances

The following steps illustrate how to create and configure the zones for the six application server instances used in this scenario.

1. Use the `zonecfg` command to create the first application server zone, `app-zone1`:

```
global# zonecfg -z app-zone1
app_zone1: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:app-zone1> create
```

   a. Specify the zone path, set autoboot value to be true, and associate the resource pool named `pool_app_zone1` with this zone:

```
zonecfg:app-zone1> set zonepath=/export/app-zone1
zonecfg:app-zone1> set autoboot=true
zonecfg:app-zone1> set pool=pool_app_zone1
```

      The `zonepath` parameter specifies the location where the zone root file system is created. This file system consumes approximately 100 Mbytes of disk space.

   b. Add a virtual network interface, specifying the IP address and the physical device for this network interface:

```
zonecfg:app-zone1> add net
zonecfg:app-zone1:net> set physical=e1000g1
zonecfg:app-zone1:net> set address=192.168.3.11
zonecfg:app-zone1:net> end
```

      In this scenario, 192.168.3 is configured as a private network that is used by the benchmark driver systems.

      The physical device (`e1000g1`) specified for this `net` resource in the new local zone is already configured in the global zone with a separate IP address. When the zone is booted, the `ifconfig` utility reports this new interface as `e1000g1:1`. In the global zone, both interfaces `e1000g1` and `e1000g1:1` are reported. However, only `e1000g1:1` is visible in the local zone.

Note – For completeness, the IP addresses are specified in the commands used in this scenario. Typically, the IP addresses would be assigned logical names and entered into the `/etc/hosts` file in the global zone.

---

c. Add a file system for the JVM software, specifying the mount point and the file system type:

```
zonecfg:app-zone1> add fs
zonecfg:app-zone1:fs> set dir=/export/VMs
zonecfg:app-zone1:fs> set special=/export/VMs
zonecfg:app-zone1:fs> set type=lofs
zonecfg:app-zone1:fs> end
```

The path `/export/VMs` specifies the location of the JDK1.5.0_06 JVM software. All six application server instances use this same location.

d. Add a file system for the WebLogic software, specifying the mount point and the file system type:

```
zonecfg:app-zone1> add fs
zonecfg:app-zone1:fs> set dir=/export/appservers
zonecfg:app-zone1:fs> set special=/export/appservers
zonecfg:app-zone1:fs> set type=lofs
zonecfg:app-zone1:fs> end
```

The path `/export/appservers` contains the WebLogic 9.1 software used by all six instances as well as the instance-specific domain data. For ease of use during this evaluation scenario, the WebLogic 9.1 software and user domain directories were configured in the global zone and imported to all six application server zones. In a production environment, the instance-specific domain data (e.g., startup script, logs, and transaction logs) can be configured within separate zones for additional security.

e. Add a comment by using the `attr` resource type:

```
zonecfg:app-zone1> add attr
zonecfg:app-zone1:attr> set name=comment
zonecfg:app-zone1:attr> set type=string
zonecfg:app-zone1:attr> set value="J2EE App Server Zone 1"
zonecfg:app-zone1:attr> end
```

f. Verify and commit the zone configuration, and exit the `zonecfg` utility:

```
zonecfg:app-zone1> verify
zonecfg:app-zone1> commit
zonecfg:app-zone1> exit
```

Note – You can use the `zonecfg -z` command to display configuration information about a Solaris Zone. See “Application Server Zone Configuration” on page 21 for example output from this scenario.

---

2. Install and boot the zone using the `zoneadm` utility:

```
global# zoneadm -z app-zone1 install
Preparing to install zone <app-zone1>.
Creating list of files to copy from the global zone.
...
global# zoneadm -z app-zone1 ready
global# zoneadm -z app-zone1 boot
```

3. Log into the new zone using the console option and configure the name and password information for this zone:

```
global# zlogin -C app-zone1
...
app-zone1 console login:
```

4. Repeat Steps 1 through 3 above to create the zones `app-zone2` through `app-zone6`, using the values in Table 1 for the network interface and the resource pool parameters. Each local zone is bound to a separate resource pool by specifying the `pool` resource parameter during zone configuration. Thus, for example, `app-zone1` is bound to pool `pool_app_zone1,` and `app_zone6` is bound to `pool_app_zone6`.

   Table 1. Web Server zone parameters.

Zone Name	Resource Pool	Physical Network Interface	Virtual Network Interface	IP Address
app-zone1	pool_app_zone1	e1000g1	e1000g1:1	192.168.3.11
app-zone2	pool_app_zone2	e1000g2	e1000g2:1	192.168.3.12
app_zone3	pool_app_zone3	e1000g3	e1000g3:1	192.168.3.13
app-zone4	pool_app_zone4	e1000g1	e1000g1:2	192.168.3.14
app-zone5	pool_app_zone5	e1000g2	e1000g2:2	192.168.3.15
app_zone6	pool_app_zone6	e1000g3	e1000g3:2	192.168.3.16

   As shown in Table 1, both `app-zone1` and `app-zone4` share the same physical network device, `e1000g1`. When booted, `app-zone1` uses the interface `e1000g1:1` for network connections. Although `app-zone4` shares this same physical network device, its network connection is available as `e1000g1:2`. Similarly, `app-zone2` and `app-zone5` share the physical network interface `e1000g2`, with `app-zone2` using the interface `e1000g2:1` and `app-zone5` using the interface `e1000g2:2`. And `app-zone3` and `app-zone6` share the physical network interface `e1000g3`, with `app-zone3` using `e1000g3:1` and `app-zone6` using `e1000g3:2`. Thus, each zone has a unique virtual network address available, and each is mapped to a unique IP address.

   By default, the WebLogic application server listens on port 7001 on all local IP addresses for non-secure requests via the HTTP and T3 (RMI) protocols. When running in a Solaris Container, the application server listens on only the IP address configured for that container. With all six zones configured, there are six WebLogic instances all listening on the same port. However, each of the six zones use a unique IP address.

Configure the Zone for the Database Server Instance

The following steps illustrate how to create and configure the zone for the database server instance used in this scenario. This is similar to the procedure for creating the application server zones (see “Configure the Zones for the Application Server Instances” on page 5), with the following differences:

- The database zone created in this example is bound to the default resource pool, `pool_default`.
- Additional file systems and disk devices are added to support the database.
- An additional network interface is used for network connectivity.

The following values are used for the network interface and the resource pool parameters for the database server instance:

Table 2. Database server zone parameters.

Zone Name	Resource Pool	Physical Network Interface	Virtual Network Interface	IP Address
db-zone	pool_default	e1000g0	e1000g0:1	192.168.200.188
		e1000g1	e1000g1:3	192.168.3.20

1. Use the `zonecfg` command to create the database server zone, `db-zone`:

```
# zonecfg -z db-zone
db-zone: No such zone configured
Use 'create' to begin configuring a new zone.
zonecfg:db-zone> create
```

   a. Specify the zone path, set autoboot value to be true, and associate the default resource pool with this zone:

```
zonecfg:db-zone> set zonepath=/export/db-zone
zonecfg:db-zone> set autoboot=true
zonecfg:db-zone> set pool=pool_default
```

   The default resource pool, `pool_default`, is configured with four logical CPUs in this scenario.

   b. Add two virtual network interfaces, specifying the IP address and the physical device for each network interface:

```
zonecfg:db-zone> add net
zonecfg:db-zone:net> set physical=e1000g0
zonecfg:db-zone:net> set address=192.168.200.188
zonecfg:db-zone:net> end
zonecfg:db-zone> add net
zonecfg:db-zone:net> set physical=e1000g1
zonecfg:db-zone:net> set address=192.168.3.20
zonecfg:db-zone:net> end
```

The additional virtual IP address is added for network connectivity using the `e1000g1` network interface. This additional interface is used for access to the private network, which is shared by the client drivers.

Note – Although the DB2 database executes within its own Solaris Container, the communication between the six WebLogic application servers and the database using the DB2 JDBC drivers occurs using internal TCP mechanisms. Thus, there are no physical network cable connections between the application servers and the database server.

---

c. Add the file systems, specifying the mount point and the file system type:

```
zonecfg:db-zone> add fs
zonecfg:db-zone:fs> set dir=/specdb
zonecfg:db-zone:fs> set special=/specdb
zonecfg:db-zone:fs> set type=lofs
zonecfg:db-zone:fs> end
zonecfg:db-zone> add fs
zonecfg:db-zone:fs> set dir=/export/VMs
zonecfg:db-zone:fs> set special=/export/VMs
zonecfg:db-zone:fs> set type=lofs
zonecfg:db-zone:fs> end
zonecfg:db-zone> add fs
zonecfg:db-zone:fs> set dir=/export/appservers
zonecfg:db-zone:fs> set special=/export/appservers
zonecfg:db-zone:fs> set type=lofs
zonecfg:db-zone:fs> end
```

The additional file system, `/specdb`, is used to store the database files. This file system is located on one of the two Sun StorEdge™ 3320 arrays.

d. Add a raw disk device to store the database transaction logs:

```
zonecfg:db-zone> add device
zonecfg:db-zone:device> set match=/dev/rdsk/c4t0d0s0
zonecfg:db-zone:device> end
```

The raw disk pathname `/dev/rdsk/c4t0d0s0`, located on the second StorEdge 3320 array, is imported from the global zone to the database zone using the `add device` option and configured to store the database transaction logs.

e. Add a comment by using the `attr` resource type:

```
zonecfg:db-zone> add attr
zonecfg:db-zone:attr> set name=comment
zonecfg:db-zone:attr> set type=string
zonecfg:db-zone:attr> set value="Database zone"
zonecfg:db-zone:attr> end
```

f. Verify and commit the zone configuration, and exit the `zonecfg` utility:

```
zonecfg:db-zone> verify
zonecfg:db-zone> commit
zonecfg:db-zone> exit
```

Note – You can use the `zonecfg -z` command to display configuration information about a Solaris Zone. See “Database Server Zone Configuration” on page 22 for example output from this scenario.

---

2. Install the zone for the database using the `zoneadm` utility:

```
global# zoneadm -z db-zone install
Preparing to install zone <db-zone>.
Creating list of files to copy from the global zone.
...
global# zoneadm -z db-zone ready
global# zoneadm -z db-zone boot
```

3. Log into the new zone using the console option and configure the name and password information for this zone:

```
global# zlogin -C db-zone
...
db-zone console login:
```

Install and Deploy the DB2 Database Software

1. Install the DB2 UDB software in the zone `db-zone`.

```
global# zlogin db-zone
db-zone # cd ${DB2_DIST}
db-zone # ./db2setup
```

Follow the DB2 installation wizard, `db2setup`, to install the DB2 software and create a DB2 instance named `db2inst1`.

2. Configure IPC Resource Controls

The System V IPC resource limits in the Solaris 10 OS, such as the maximum shared memory size, are no longer set in the `/etc/system` file. All System V IPC facilities are either automatically configured or are project resource controls. Resource controls allow IPC settings to be made on a per-project or per-user basis on the local system or in a name service environment. Facilities that can be shared include memory, message queues, and semaphores.

a. To set the resource controls for the DB2 instance `db2inst1`, which was created above using the `db2setup` command, first create a custom project using the `projadd` command:.

```
db-zone # projadd -c “DB2 instance 1” user.db2inst1
```

This creates a project named `user.db2inst1` in the local project database stored at `/etc/project` to serve as the default project for the user `db2inst1`.

b. Use the `id(1M)` command to verify the default project for the `db2inst1` user.

```
db-zone # su – db2inst1
$ id -p
uid=109(db2inst1) gid=101(db2iadm1) projid=100(user.db2inst1)
$ exit
```

c. Set the resource controls for the IPC tunable settings:

```
db-zone # projmod -a -K \
 “project.max-shm-memory=(privileged,9578697523,deny)” user.db2inst1
db-zone # projmod -a -K \
 “project.max-shm-ids=(privileged,7168,deny)” user.db2inst1
db-zone # projmod -a -K \
 “project.max-msg-ids=(privileged,7168,deny)” user.db2inst1
db-zone # projmod -a -K \
 “project.max-sem-ids=(privileged,6144,deny)” user.db2inst1
```

These commands add the modified resource control tunings to the project database for the newly created DB2 instance.

d. Use the `projects(1M)` command to verify the project resource settings:

```
db-zone # projects -l
```

See “Database Server Project Configuration” on page 23 for example output from this scenario.

3. Configure the database using the required schema to support the planned workload. In this evaluation scenario, the schema scripts provided with the benchmark publication are used.

   For this server consolidation project, the database files are located in the `/specdb` directory. This directory is mounted using the Unix File System (UFS) option `noatime` to turn off file access time updates on file reads and the UFS option `forcedirectio` to reduce I/O transfer times. The database logs are located on one of the storage arrays and accessed via the raw disk device.

Server Consolidation Evaluation

The purpose of this evaluation is to validate the use of a Sun Fire X4600 server as a consolidation platform and demonstrate the benefits of consolidating multiple application servers onto a single server using Solaris Containers. Two Sun Fire V440 servers and one Sun Fire V890 server are used to generate the application workload. This configuration is then utilized to execute the SPECjAppServer2004 benchmark[1], an industry-standard multi-tier benchmark used to measure the performance of J2EE technology-based application servers.

SPECjAppServer2004 is an end-to-end application that exercises all major J2EE technologies implemented by compliant application servers. SPECjAppServer2004 is an appropriate mechanism to generate and measure relevant workloads as it heavily exercises all parts of the underlying infrastructure that make up the application environment, including hardware, JVM software, database software, JDBC drivers, and the system network.

Consolidation Environment

A Sun Fire X4600 server running the Solaris 10 OS is configured with six WebLogic 9.1 Application Server instances and one DB2 8.2 Database Server instance, each running in a separate Solaris Container (see Figure 2 on page 24). The Sun Java Platform, Standard Edition 5.0 update 6 JVM is used to deploy and execute the J2EE application code. The Domain Name Service (DNS) `named` server process on the Sun Fire X4600 server is configured to round-robin the IP addresses of the six WebLogic application server instances, to provide load balancing for all network requests from the driver machines.

The Sun Fire X4600 server uses two Sun StorEdge 3320 SCSI arrays to store the database files and transaction logs. To meet the database durability requirements of the benchmark, the disks in the Sun StorEdge 3320 SCSI arrays are configured as RAID 1 (mirrored disks) devices using the array controller menu interface. The database files are located on one of the arrays and mounted using UFS. The DB2 transaction logs are stored on the second storage array and accessed using the raw device path. Two internal disks on the Sun Fire X4600 server are configured as a mirrored pair using the onboard RAID controller to meet the benchmark durability requirements for the WebLogic transaction logs.

Two Sun Fire V440 servers and one Sun Fire V890 server are configured as the primary and satellite drivers for the application workload. All three driver systems and three network interfaces on the Sun Fire X4600 server are connected to a single Gigabit Ethernet switch to support the transaction load generated by the client agents. The Sun Fire V890 server also serves as the Supplier Emulator, and uses a separate WebLogic 9.1 instance to provide this function.

J2EE Application Server Workload

This evaluation scenario uses the SPECjAppServer2004 benchmark from the Standard Performance Evaluation Corporation (SPEC) to generate the J2EE workload [1]. This industry-standard benchmark is widely used to measure the performance of application servers using J2EE technology. Developed by SPEC and released in April 2004, this benchmark is based on previous versions (SPECjAppServer2001

1. SPECjAppServer®2004 is a registered trademark of the Standard Performance Evaluation Corporation (SPEC).

and SPECjAppServer2002), but is more complex and implements many enhancements to exercise major services of the J2EE platform.

The J2EE application server workload models a manufacturing, supply-chain management, and order/inventory system. Specifically, the business logic is modeled after an automobile manufacturer whose main customers are automobile dealerships. Dealers use a web-based user interface to browse an automobile catalog, purchase and sell automobiles, and track dealership inventory. The business model includes five domains, where each domain is a logical entity that describes a distinct business sphere of operations:

- Customer domain – deals with customer orders and interactions
- Dealer domain – offers a web-based interface to the services in the customer domain
- Manufacturing domain – performs “just-in-time” manufacturing operations
- Supplier domains – handles interactions with external suppliers
- Corporate domain – manages all customer, product and supplier information

All the activities and processes in the five domains are implemented using J2EE components (Enterprise JavaBeans, Servlets, JavaServer Pages™, Java Messaging System) assembled into a single J2EE application that is deployed to the application servers. The benchmark can be run in either standard or distributed mode. In standard mode, all domains are combined in a single database instance. In distributed mode, the benchmark performs business transactions across domains using multiple database instances. This consolidation environment uses the standard mode of operation using a single database instance.

Performance of the benchmark is reported by the metric SPECjAppServer2004 JOPS (jAppServer Operations Per Second) and is the summation of customer order transactions and manufacturing work order transactions per second.

Benchmark Results

The Sun Fire X4600 server used in this evaluation achieved 1000.86 SPECjAppServer2004 JOPS@Standard. Detailed information on the benchmark results, configuration, and system tuning can be found at the SPEC web site: *http://www.spec.org/jAppServer2004/results/res2006q3/jAppServer2004-20060802-00033.html*.

During benchmark execution, the power consumption of the Sun Fire X4600 server was measured using a commercially available power meter. Measurements show that the Sun Fire X4600 server consumed on average 1200 Watts of power during the steady state period of the benchmark. Power measurements are not a requirement of the benchmark, but were collected to demonstrate power consumption efficiency of the Sun Fire X4600 server.

While traditional metrics are good for calculating the throughput of a server, they do not consider the impact of power, cooling, and space demands in the data center. Thus Sun has introduced SWaP (Space, Watts, and Performance)[1], a new metric that provides a total view of a server’s overall efficiency and helps

1. See *http://www.sun.com/servers/coolthreads/swap/* for more detailed information on Sun’s SWaP metric.

determine the impact of a server in the data center. This innovative metric is calculated as:

$$SWaP = \frac{Performance}{Space \times PowerConsumption}$$

where:

- Performance — measured using industry-standard benchmarks
- Space — the height of the server in rack units (RUs)
- PowerConsumption — number of Watts consumed by the server, using data from either actual benchmark runs or vendor site planning guides

In general, higher SWaP values denote a server that has a higher total efficiency within the data center. Greater efficiency is desirable in a comparison, as it predicts decreased operational costs for the server.

Comparative System Results

Table 3 details a comparison of the Sun Fire X4600 server and other configurations using HP and IBM servers. The comparison includes performance, space and power metrics, along with the calculated SWaP value for each configuration. The systems chosen for comparison have competitive SPEC benchmark results posted on the SPEC web site and use configurations with similar numbers of small servers for the J2EE application tier. Specifically, a HP DL380 configuration, containing a six-node HP DL380 cluster and one HP rx8620 server, and an IBM x365 configuration, containing a five-node IBM x365 cluster and two additional IBM x365 servers, are considered.

Table 3. Comparative SWaP performance results.

Platform	Performance[a] (SPECjAppServer2004 JOPS@Standard)	Space	Power Consumption	SWaP
Sun Fire X4600 server	1000.86	4 RU	1200 Watts	0.21
6-node HP DL380 Cluster, and 1 HP rx8620 server	1664.36	23 RU	5482 Watts	0.01
5-node IBM x365 Cluster, and 2 IBM x365 servers	1343.47	21 RU	4984 Watts	0.01

a.Performance results taken from *http://www.spec.org* as of Aug 16, 2006. The comparison presented here is based on best performance for configurations using similar numbers of small x86 servers.

- Sun Fire X4600 server

  With a performance of 1000.86 JOPS on the SPECjAppServer2004 benchmark, a 4 RU space requirement, and 1200 Watts measured power consumption, the Sun Fire X4600 server has a calculated SWaP value of 0.21 in this evaluation. Detailed benchmark results are posted on the SPEC site at *http://www.spec.org/jAppServer2004/results/res2006q3/jAppServer2004-20060802-00033.html*.

- HP DL380 Cluster

  The HP DL380 cluster has a calculated SWaP value of 0.01. Detailed benchmark results are posted on the SPEC site at *http://www.spec.org/jAppServer2004/results/res2005q3/jAppServer2004-20050802-00015.html*. Each DL380 server occupies 1 RU of space and is estimated at using 438 Watts of power. The HP rx8620 server occupies 17 RU of space and is estimated as using 2850 Watts. The HP DL380 and HP rx8620 power usage was estimated by calculating 75% of the power supply data reported in the product specification[1].

- IBM x365 Cluster

  The IBM x365 cluster has a calculated SWaP value of 0.01. Detailed benchmark results are posted on the SPEC web site, *http://www.spec.org/jAppServer2004/results/res2004q4/jAppServer2004-20041123-00006.html*. Each IBM x365 occupies 3 RU of space and is estimated as using 712 Watts. The IBM x365 power usage was estimated by calculating 75% of the power supply data reported in the product specification[2].

While lower in actual throughput performance, the Sun Fire X4600 server uses only approximately one-fifth of the space compared to the competitive platforms and consumes approximately one-quarter of the power. The SWaP metric, as defined above, shows that using the Sun Fire X4600 server to consolidate multiple server instances on a single system is more than 20 times more efficient than the comparative configurations using multiple server platforms.

Conclusions

Using Solaris Containers to consolidate multiple J2EE application and database servers on a single Sun Fire X4600 server provides numerous benefits over configurations employing multiple servers. The Solaris Containers technology enables multiple applications to each run in their own isolated execution environment. Resource management tools included with the Solaris OS enable resources such as CPUs and memory to be easily allocated to each zone, monitored, and dynamically adjusted to meet changing performance requirements.

Reduced complexity and improved efficiency are well-understood benefits of consolidation. Managing a single system is simpler than installing, administering, and ensuring adequate resources for multiple platforms and possibly multiple operating systems. Deploying multiple applications onto a single server can also increase server utilization and reduce the number of wasted CPU cycles, for greater operating efficiency. Furthermore, using a single energy-efficient platform like the Sun Fire X4600 server can provide significant energy usage and space saving advantages over configuration that utilize multiple individual servers. Application consolidation using Solaris Containers on the Sun Fire X4600 server is a promising way to lower costs and complexity, increase flexibility and scalability, and reduce data center space and energy requirements.

1. The specifiations for the HP DL380 server were taken on August 4, 2006 from *http://h18004.www1.hp.com/products/quickspecs/12166_na/12166_na.html*. The HP rx8620 specifications were taken on the same date from *http://h18000.www1.hp.com/products/quickspecs/11849_div/11849_div.HTML*.
2. IBM x365 specifications were taken from *http://www.redbooks.ibm.com/abstracts/REDP3826.html?Open* on August 4, 2006.

About the Author

Kevin Kelly is Senior Staff Engineer in the Strategic Applications Engineering (SAE) group at Sun Microsystems, Inc. SAE is responsible for producing strategic benchmarks on all Sun Computer Systems platforms. This group explores performance and function of the systems using various workloads resulting in tuning recommendations and tools for optimizing performance.

Acknowledgements

The author would like to thank David Miller, Brian Whitney, Jim Britton, Guido Ficco, and Yan Fisher for their assistance in reviewing the technical content of this article.

References

[1] Standard Performance Evaluation Corporation web site. `http://www.spec.org`.

[2] *System Administration Guide: Solaris Containers—Resource Management and Solaris Zones*, Part Number 817-1592-11.
To access this document, go to `http://docs.sun.com/app/docs/doc/817-1592`

[3] Lageman, Menno. “Solaris Containers – What They Are and How to Use Them,” *Sun BluePrints OnLine*, May 2005.
To access this document, go to `http://www.sun.com/blueprints/0505/819-2679.pdf`

[4] BEA Systems, Inc. web site. `http://www.bea.com`.

[5] *Server and Storage Consolidation 2004: Executive Interview Summary Report.*
`http://www.idc.com`

[6] *Server and Storage Consolidation 2004: End-User Summary Report.*
`http://www.idc.com`

[7] IBM Corporation web site. `http://www.ibm.com.`

Ordering Sun Documents

The SunDocs℠ 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/`

Appendix A: Example Configuration Information

Resource Pool Configuration

This section contains example output from the `pooladm` command, showing the resource pool configuration information for this example scenario.

```
# pooladm

system global
        string  system.comment
        int     system.version 1
        boolean system.bind-default true
        int     system.poold.pid 482

pool pool_app_zone4
                int     pool.sys_id 4
                boolean pool.active true
                boolean pool.default false
                int     pool.importance 1
                string  pool.comment
                pset    pset_app_zone4

pool pool_app_zone1
                int     pool.sys_id 1
                boolean pool.active true
                boolean pool.default false
                int     pool.importance 1
                string  pool.comment
                pset    pset_app_zone1

pool pool_app_zone6
                int     pool.sys_id 6
                boolean pool.active true
                boolean pool.default false
                int     pool.importance 1
                string  pool.comment
                pset    pset_app_zone6

pool pool_app_zone3
                int     pool.sys_id 3
                boolean pool.active true
                boolean pool.default false
                int     pool.importance 1
                string  pool.comment
                pset    pset_app_zone3


pool pool_default
                int     pool.sys_id 0
                boolean pool.active true
                boolean pool.default true
                int     pool.importance 1
                string  pool.comment
                pset    pset_default
```

```
 pool pool_app_zone5
                int     pool.sys_id 5
                boolean pool.active true
                boolean pool.default false
                int     pool.importance 1
                string  pool.comment
                pset    pset_app_zone5
 pool pool_app_zone2
                int     pool.sys_id 2
                boolean pool.active true
                boolean pool.default false
                int     pool.importance 1
                string  pool.comment
                pset    pset_app_zone2
 pset pset_app_zone4
                int     pset.sys_id 4
                boolean pset.default false
                uint    pset.min 2
                uint    pset.max 2
                string  pset.units population
                uint    pset.load 0
                uint    pset.size 2
                string  pset.comment
 CPU
                        int     cpu.sys_id 7
                        string  cpu.comment
                        string  cpu.status on-line
CPU
                        int     cpu.sys_id 6
                        string  cpu.comment
                        string  cpu.status on-line
pset pset_app_zone1
                int     pset.sys_id 1
                boolean pset.default false
                uint    pset.min 2
                uint    pset.max 2
                string  pset.units population
                uint    pset.load 1
                uint    pset.size 2
                string  pset.comment
CPU
                        int     cpu.sys_id 4
                        string  cpu.comment
                        string  cpu.status on-line
 CPU
                        int     cpu.sys_id 5
                        string  cpu.comment
                        string  cpu.status on-line
```

```
pset pset_app_zone6
                int     pset.sys_id 6
                boolean pset.default false
                uint    pset.min 2
                uint    pset.max 2
                string  pset.units population
                uint    pset.load 0
                uint    pset.size 2
                string  pset.comment
 CPU
                        int     cpu.sys_id 3
                        string  cpu.comment
                        string  cpu.status on-line
 CPU
                        int     cpu.sys_id 2
                        string  cpu.comment
                        string  cpu.status on-line
 pset pset_app_zone3
                int     pset.sys_id 3
                boolean pset.default false
                uint    pset.min 2
                uint    pset.max 2
                string  pset.units population
                uint    pset.load 0
                uint    pset.size 2
                string  pset.comment
 CPU
                        int     cpu.sys_id 11
                        string  cpu.comment
                        string  cpu.status on-line
CPU
                        int     cpu.sys_id 10
                        string  cpu.comment
                        string  cpu.status on-line
pset pset_default
                int     pset.sys_id -1
                boolean pset.default true
                uint    pset.min 1
                uint    pset.max 65536
                string  pset.units population
                uint    pset.load 20
                uint    pset.size 4
                string  pset.comment
 CPU
                        int     cpu.sys_id 12
                        string  cpu.comment
                        string  cpu.status on-line
CPU
                        int     cpu.sys_id 14
                        string  cpu.comment
                        string  cpu.status on-line
```

```
CPU
                        int     cpu.sys_id 13
                        string  cpu.comment
                        string  cpu.status on-line
 CPU
                        int     cpu.sys_id 15
                        string  cpu.comment
                        string  cpu.status on-line
pset pset_app_zone5
                int     pset.sys_id 5
                boolean pset.default false
                uint    pset.min 2
                uint    pset.max 2
                string  pset.units population
                uint    pset.load 0
                uint    pset.size 2
                string  pset.comment
 CPU
                        int     cpu.sys_id 1
                        string  cpu.comment
                        string  cpu.status on-line
CPU
                        int     cpu.sys_id 0
                        string  cpu.comment
                        string  cpu.status on-line
pset pset_app_zone2
                int     pset.sys_id 2
                boolean pset.default false
                uint    pset.min 2
                uint    pset.max 2
                string  pset.units population
                uint    pset.load 0
                uint    pset.size 2
                string  pset.comment
 CPU
                        int     cpu.sys_id 9
                        string  cpu.comment
                        string  cpu.status on-line
CPU
                        int     cpu.sys_id 8
                        string  cpu.comment
                        string  cpu.status on-line
```

Application Server Zone Configuration

This section contains example output from the `zonecfg` command, showing the zone configuration information for one of the application server instances in this example scenario.

```
# zonecfg -z app-zone1
zonecfg:app-zone1> info
zonepath: /export/app-zone1
autoboot: true
pool: pool_app_zone1
inherit-pkg-dir:
        dir: /lib
inherit-pkg-dir:
        dir: /platform
inherit-pkg-dir:
        dir: /sbin
inherit-pkg-dir:
        dir: /usr
fs:
    dir: /export/VMs
    special: /export/VMs
    raw not specified
    type: lofs
    options: []
fs:
    dir: /export/appservers
    special: /export/appservers
    raw not specified
    type: lofs
    options: []
net:
    address: ecapps-1
    physical: e1000g1
attr:
    name: comment
    type: string
    value: "J2EE App Server Zone 1"
zonecfg:app-zone1> exit
```

Database Server Zone Configuration

This section contains example output from the `zonecfg` command, showing the zone configuration information for the database server instance in this example scenario.

```
# zonecfg -z db-zone
zonecfg:db-zone> info
zonepath: /export/db-zone
autoboot: true
pool: pool_default
inherit-pkg-dir:
        dir: /lib
inherit-pkg-dir:
        dir: /platform
inherit-pkg-dir:
        dir: /sbin
inherit-pkg-dir:
        dir: /usr
fs:
        dir: /specdb
        special: /specdb
        raw not specified
        type: lofs
        options: []
fs:
        dir: /export/appservers
        special: /export/appservers
        raw not specified
        type: lofs
        options: []
fs:
        dir: /export/VMs
        special: /export/VMs
        raw not specified
        type: lofs
        options: []
...
net:
        address: db-zone
        physical: e1000g0
net:
        address: ecdata
        physical: e1000g1
device
        match: /dev/rdsk/c4t0d0s0
attr:
        name: comment
        type: string
        value: "Database zone"
zonecfg:db-zone> exit
```

Database Server Project Configuration

This section contains example output from the `projects` command, showing the project configuration information for the database server instance in this example scenario.

```
# projects -l
system
        projid : 0
        comment: ""
        users  : (none)
        groups : (none)
        attribs:
user.root
        projid : 1
        comment: ""
        users  : (none)
        groups : (none)
        attribs:
...
group.staff
        projid : 10
        comment: ""
        users  : (none)
        groups : (none)
        attribs:
user.db2inst1
        projid : 100
        comment: "DB2 instance 1"
        users  : (none)
        groups : (none)
attribs: project.max-msg-ids=(privileged,7168,deny)
                project.max-sem-ids=(privileged,7168,deny)
                project.max-shm-ids=(privileged,7168,deny)
                project.max-shm-memory=(privileged,9578697523,deny)
```

Appendix B: Evaluation System Configuration

Figure 2 depicts the system environment used to evaluate the application server consolidation. A Sun Fire X4600 server is utilized as the consolidation platform and runs six application server instances and a database instance, each in a separate Solaris Container. Two Sun StorEdge 3320 SCSI disk arrays are used to store transaction logs and database files. The Sun Fire X4600 server is connected via an Ethernet switch to a Sun Fire V890 server and two Sun Fire V440 servers used to execute the J2EE workload.

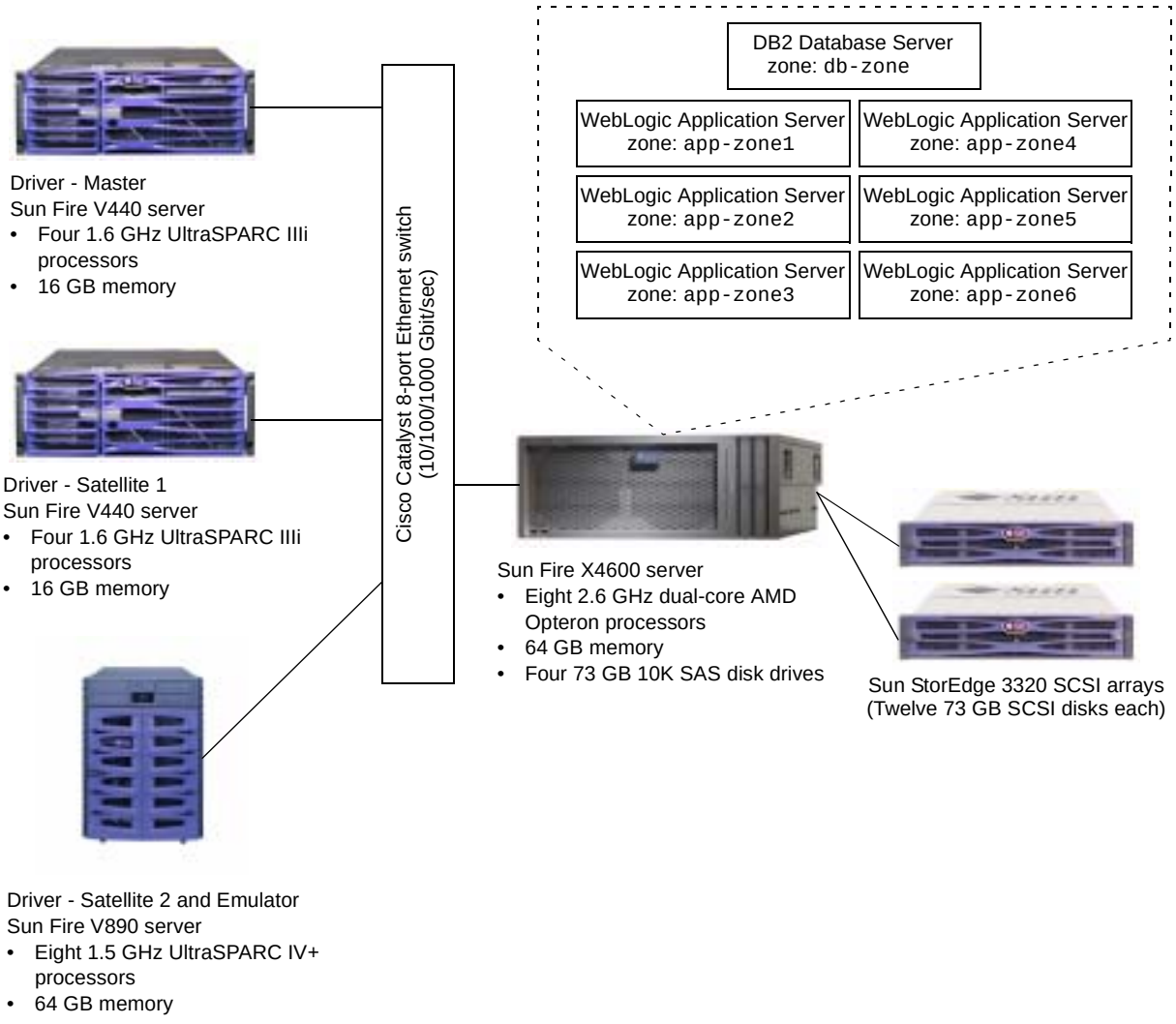

Figure 2. System environment.

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