Increasing Storage Area Network Productivity

Scott Tracy, Storage Network Engineering

Ken Gibson, Storage Network Engineering

Sun BluePrints™ OnLine—July 2004

Sun
microsystems

`http://www.sun.com/blueprints`

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95045 U.S.A.
650 960-1300

Part No. 817-7544-10
Revision 1.0, 6/28/04
Edition: July 2004

DOCUMENTATION IS PROVIDED “AS IS” AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REPONDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.

Please
Recycle

Adobe PostScript

Increasing Storage Area Network Productivity

This article describes the Sun StorEdge™ SAN Foundation software (SFS) features that allow dynamic and persistent recognition of storage and configuration changes without rebooting servers running the Solaris™ Operating System (Solaris OS).

This article contains the following topics:

Historical Input/Output Frameworks

Most operating systems (OSs) used today were designed before storage area networks (SANs) became prevalent. Servers used direct attached storage (DAS) through static interfaces such as the Small Computer System Interconnect (SCSI) protocol. OS designers created the storage I/O framework based on the characteristics of direct attached storage like SCSI, which included many of the following design points:

- Device sizes were fixed and logical units (LUNs) could not be added after boot time.
- Storage devices supported a relatively small number of LUNs.
- Storage devices were typically single-ported and visible through only one path at a time.
- Host bus adapters (HBAs) were not hot-pluggable.
- There was no concept of zoning or LUN-masking as used in SANs today.
- Storage interconnects were not hot-pluggable.

Typically, the OS could scan the storage at boot time and assume that it would not change until the system was powered-down for maintenance. Servers attached to no more than a few dozen storage devices and the interconnect fabric did not provide an automated way for devices to register or announce their presence. Servers either scanned every attached device at boot time or allowed administrators to customize the device scan through manually edited configuration files.

Prior to Solaris 8 OS, and like other operating systems of its time, Solaris OS had a driver framework optimized for SCSI DAS. At boot time, Solaris OS built a directory tree that reflected the way storage was physically connected to the server through SCSI busses, HBAs, and internal peripheral component interconnect (PCI) or SBus slots. The administrator was required to create and maintain device driver configuration files (`sd.conf` and so on) to tell the framework the properties of each storage device, because the SCSI protocol did not provide a mechanism for devices to report themselves and their properties.

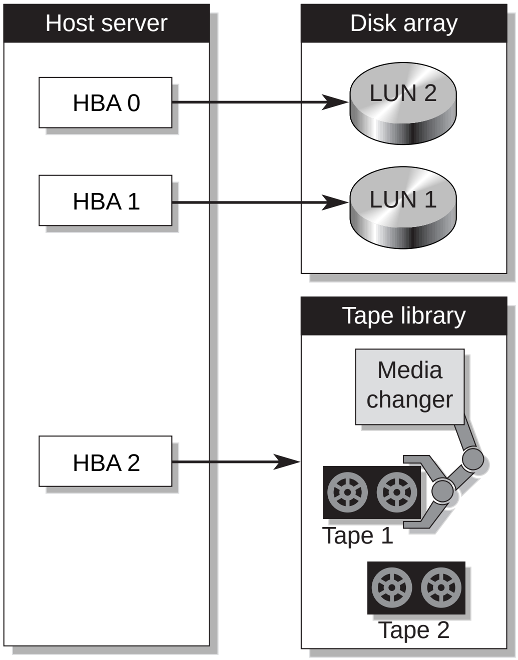

FIGURE 1 Direct Attached Storage (DAS) Configuration

FIGURE 1 shows a typical SCSI DAS configuration with a two-LUN RAID array and a tape library with two tape drives. Using the traditional Solaris OS SCSI driver framework, the administrator would create the disk configuration file `sd.conf` with the following entries:

```
name="sd" class="scsi" target=0 lun=0;

name="sd" class="scsi" target=0 lun=1;
```

Similarly, the administrator would add the following to the tape configuration file `st.conf`:

```
name="st" class="scsi" target=1 lun=0;

name="st" class="scsi" target=1 lun=2;
```

And, the administrator would add the following to sgen.conf (for the media changer at lun 2):

```
name="sgen" class="scsi" target=1 lun=2;
```

During device tree creation at boot time, Solaris OS used the target driver configuration files to create a devinfo node for each line in the file and passed this node to the target driver to probe. Upon successful acknowledgement from the respective target driver, a file in the /devices directory was created. FIGURE 2 depicts the device tree and special files created in the /devices directory and the associated .conf file entries as a result of the configuration shown in FIGURE 1.

Enumerating devices in this manner was not efficient because it required probing of each line of the configuration file, even if a device did not exist. This had an adverse effect on boot time. Also, memory consumption was increased because the devinfo node was not destroyed for devices that were probed, yet were not present (the devinfo node was retained to cover the case of offline devices during the boot process).

Storage applications used the logical links in the /dev directory to configure and use SCSI devices. These links referred to the special files or physical paths created in the /devices directory and remained constant unless either the system was physically reconfigured or the logical links were removed and regenerated.

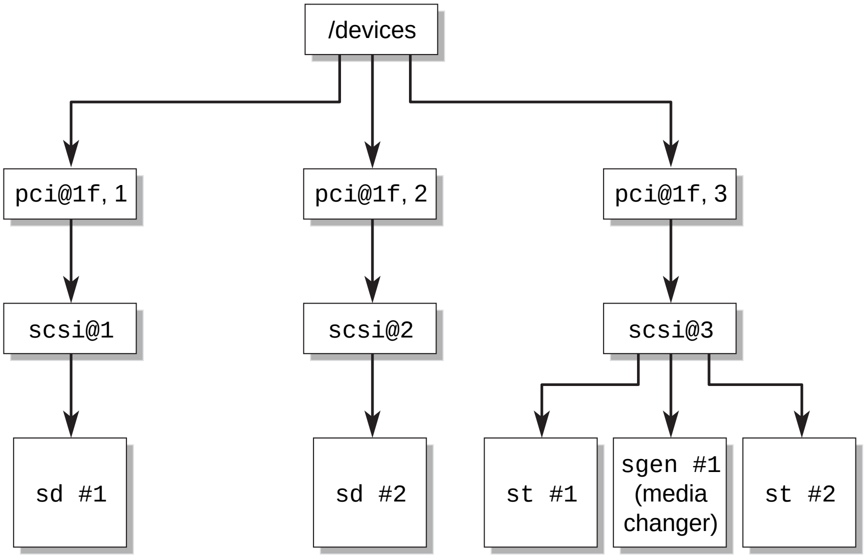

FIGURE 2 SCSI Device Tree

Fibre Channel SANs

The introduction of Fibre Channel (FC) allowed storage vendors to create more dynamic and highly available solutions using large pools of consolidated storage.

Typical characteristics of the new environment are as follows:

- Storage devices that can dynamically connect, disconnect, or move to a new port
- SAN configuration changes made possible by adding or reconfiguring paths to storage
- Storage devices with multiple ports per LUN
- SANs with multiple paths to each LUN
- Storage that can dynamically grow and expose new LUNs
- Thousands of LUNs presented to a server by the SAN
- Large, central storage devices that allow remote booting from many hosts (fabric boot)
- Hot-pluggable SAN HBAs and interconnects

Although the nature of the storage interconnects changed with SANs, early FC HBAs were written to old SCSI driver frameworks. Assume that in the configuration depicted in FIGURE 1, the SCSI interconnects are replaced with Fibre Channel.

Though the interconnect type changed, the way the Solaris OS SCSI driver framework viewed these devices did not. FIGURE 3 shows the associated FC device tree created. Note the similarities to the SCSI device tree in FIGURE 2.

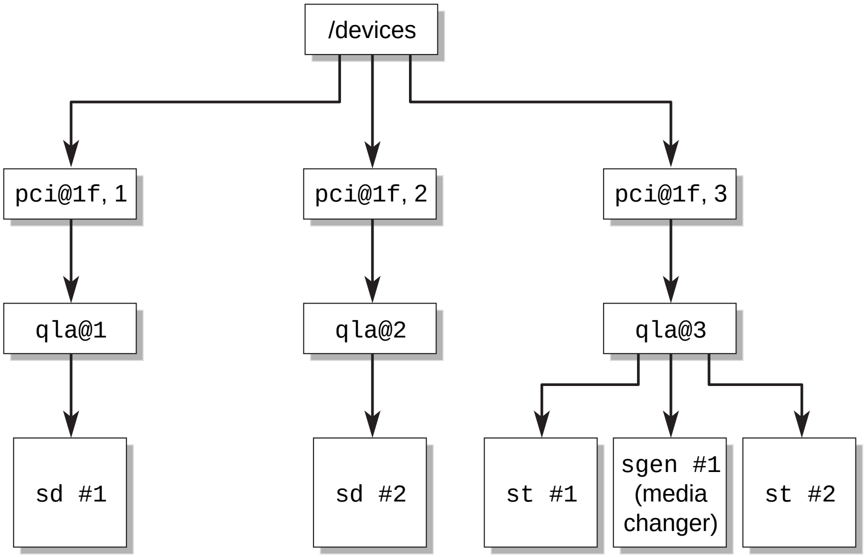

FIGURE 3 Sun Common SCSI Architecture (SCSA) FC HBA Device Tree

In FIGURE 3, the FC driver forces the configuration to appear like a DAS SCSI configuration. Storage applications still follow a path based on the direct physical connection, even though multiple, redundant connections might be available. In fact, the device definition is the fully qualified SCSI path name. Most limiting is that configuration changes still require changes to the device driver configuration files and a reboot to rebuild the device tree.

New Input/Output Framework

To fully and properly support FC SANs, the I/O framework and device driver stack must provide the following capabilities:

- Present persistent device nodes for storage, regardless of the physical path or configurations changes since the node was created
- Automatically identify devices and create device nodes without requiring manual administration
- Suspend I/O to an HBA, and allow it to be removed and replaced without interruption
- Provide high availability through multipathing software
- Allow seamless operation with both legacy and new storage applications
- Present thousands of LUNs to a single host
- Allow fabric boot from large, central storage devices

Beginning with Solaris 8 OS, Sun added these capabilities to the I/O framework and the native SFS stack. Key new features added to Solaris OS included transparent multipathing support, dynamic device node creation, support for 16,000 LUNs, and dynamic reconfiguration. FIGURE 4 shows the new Solaris SAN device driver components as compared to traditional HBAs and driver stacks.

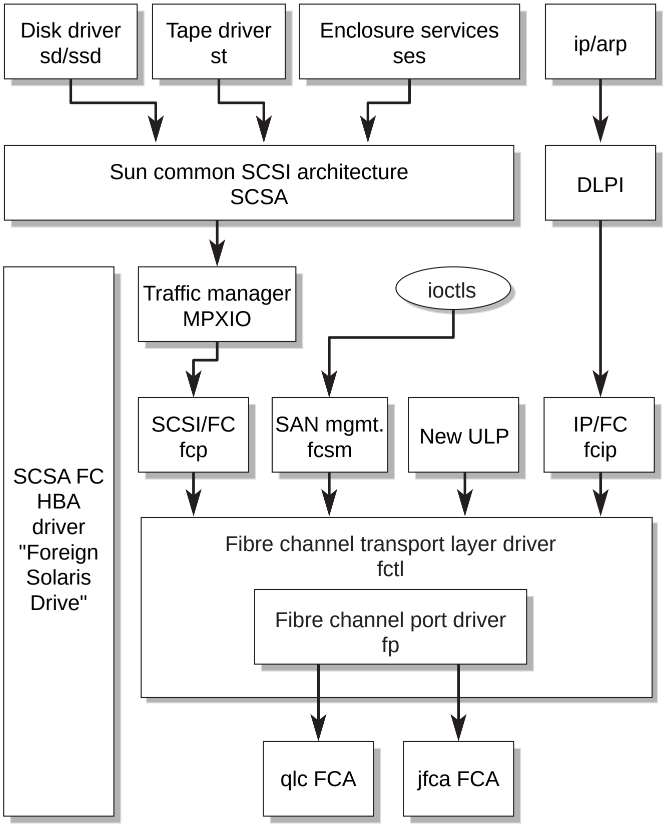

FIGURE 4 Integrated SAN Foundation Software Compared to SCSA FC HBAs

Integrated Multipathing

The integrated multipathing module (Traffic Manager) in the SFS virtualizes and abstracts multiple paths to a disk or LUN, so that instead of representing a device node for each and every path to a single device, a single device node is created that represents the sum of all paths to the disk or LUN. Traffic Manager provides a path-independent name in both the `/dev` and `/devices` directories and uses only one device node of the corresponding device driver (`ssd` or `st`).

FIGURE 5 shows the resulting device tree when Traffic Manager and FC interconnects are enabled in a configuration such as that shown in FIGURE 1. The device nodes remain persistent and data will be accessible even if one or more of the underlying physical paths fail.

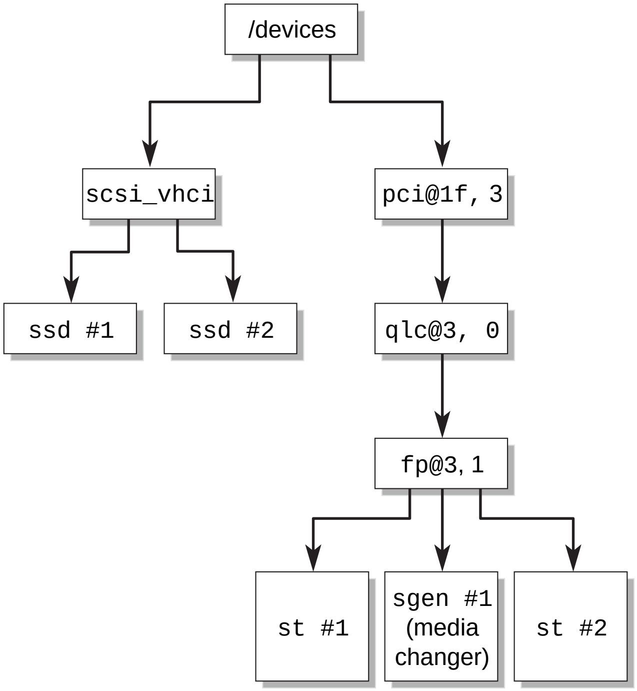

FIGURE 5 Device Tree With Traffic Manager and Multiple Paths

Traffic Manager's integrated multi-path virtualization provides the following advantages over other vendor multipathing techniques:

- Applications, volume manager, file system, and databases do not need to be aware of multiple paths.
- Path management is performed automatically, without manual administration or reboots.
- Device command throttling as designed into the target drivers remains intact, because there is only one device node per device.
- Load balancing provides the ability to increase bandwidth by adding more paths to a device, and it does not require a reboot.

Traffic Manager provides support for all Sun storage devices, as well as all third-party storage solutions that offer fully symmetrical controllers. In a *symmetric controlle*r, all paths are active and all paths can take commands at any time. Examples of well known third-party symmetric controllers include EMC Symmetrix and DMX lines, IBM Shark (Total Enterprise Storage) arrays, and HP XP 512 and XP 1024 series.

Dynamic Reconfiguration

The no-reboot SAN provides the ability to replace HBAs without disrupting access to data. Traffic Manager reroutes I/O around a failed or offlined HBA through one of the “n” way alternate paths.

Sun server platforms that support hot-pluggable HBA devices provide a feature called Dynamic Reconfiguration (DR). This feature allows the I/O framework to logically detach an HBA and prepare it for physical removal.

DR is performed by means of the `cfgadm` (configuration administration) utility. The `cfgadm unconfigure` command removes the associated software resources from the system and Traffic Manager automatically routes I/O away from this path. The `cfgadm connect` and `configure` commands reconnect the OS to the new HBA. Traffic Manager then automatically resumes routing I/O through the new HBA. During the procedure, applications continue to run with full access to data.

Dynamic Device Node Creation

Support for dynamically adding or deleting devices and resulting node creation or destruction use new device driver interfaces created in the Solaris OS framework. These are controlled through the `cfgadm` utility.

The SFS performs device discovery through IEEE-compliant FC methods. Devices are made available to the system immediately upon proper connection to the SAN fabric.

Some subtle differences exist on how new devices are seen by storage applications. If the additional storage is a new LUN with an existing target configured through `cfgadm`, the new LUN shows up automatically. If the additional storage is a completely new target (for example, new array controller), `cfgadm configure` must be run to allow storage applications to properly see these devices. *A system reboot is not required to recognize either a new target or a new LUN*. In both cases, new `devinfo` node and new `/dev` and `/device` entries are created for the new device.

If Traffic Manager is enabled, it determines if this is a completely new device or an additional path to an existing device. If this is a new device, a new virtual device node is created. If this is a new path, no additional /dev and /device files are created. The additional path information is stored and used by Traffic Manager and is transparent to any upstream applications.

For example, assume that LUN 2 is added to the array shown in FIGURE 1. FIGURE 6 depicts the resulting /devices tree. As configuration changes occur, the interfaces to storage applications (the /dev logical links) do not change as any previously configured devices are moved about the SAN. The interfaces /dev and /device entries are uniquely matched for each device/LUN.

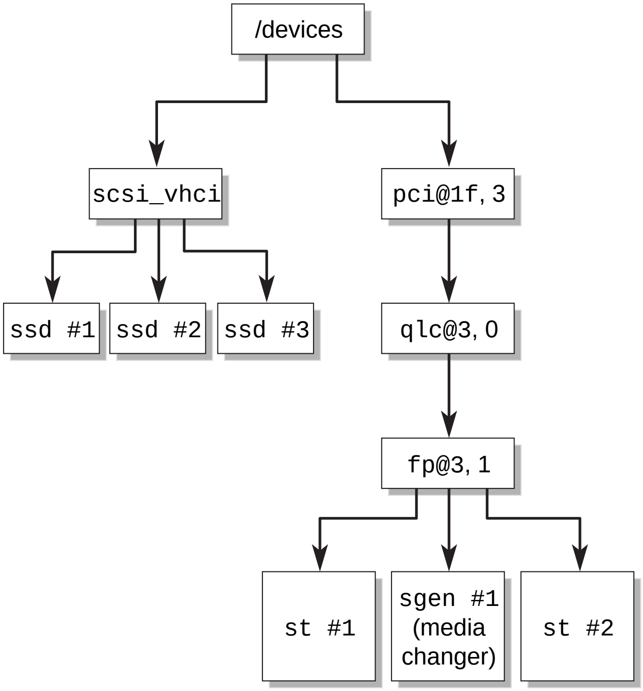

FIGURE 6 /device Directory After Creating a new Disk-Array LUN

Persistent Binding

Although SANs present and implement the dynamic characteristics of storage and paths to storage, the binding between a device and any storage application's logical links in /dev remains persistent throughout the life of the device and link, *so long as the links are not removed and subsequently regenerated*. This statement is true for all devices supported by SFS, including disk, tape, and SCSI enclosure service devices. Therefore, once bound to a physical device in /devices, the logical device names in /dev, such as /dev/rmt<*x*> or /dev/c<*x*>t<*x*>d<*x*>s<*x*>, remain the same across reboots, dynamic reconfiguration events, or any other administrative changes that occur within the SAN.

Initial logical link creation depends upon the order in which the target driver attaches to each device. The order of generation can vary and is not under the control of the system. If logical links are removed and regenerated, there is no guarantee that any new logical links will use the same logical links as had been used for the affected devices. Logical links generated on one system might not map to the same logical links generated on another system.

To maintain persistent binding for any device on the SAN, avoid removing and regenerating the logical links. Logical links are typically removed in one of the following methods:

- Using `devfsadm –C`
- Using the `rm` command on the link itself

Each method presents risks of changing logical links and/or persistent binding.

The manual page for `devfsadm(1M)`describes the following:

```
OPTIONS
The following options are supported:

-C Cleanup mode. Prompts devfsadm to invoke cleanup
routines
that are normally not invoked to remove dangling logical
links.
If –c is also used, devfsadm only cleans up for the
listed devices’
classes.

-c device_class
Restricts operations to devices of the class device_class.
Solaris defines the following values for device_class:
disk, tape, port, audio, and pseudo.  This option may be
specified more than once to specify multiple device
classes.
```

When the `devfsadm –C` command is executed on a system that has a dangling logical link (a logical link through which no communication to a device can be established; for example, the device is in an offline state), the link is removed.

Logical links are automatically regenerated when an offline device comes back online. However, as noted previously, there is no guarantee that any newly created link will be identical to any previously removed link.

The `devfsadm –C` command can be used to clean up dangling links by device class. This approach can alleviate the unwanted removal of all dangling logical links. To remove only dangling logical disk links, issue the following command:

```
# devfsadm –C –c disk
```

This command removes only dangling logical links for disk devices. It leaves all other links untouched (for example, tape in `/dev/rmt<`*`x`*`>`).

The `rm` command can be used to remove logical link entries in `/dev`. Just as in the `devfsadm` case, when an offline device is brought back online, its associate links in /`dev` are automatically regenerated, yet not usually with the previous `/dev` names.

Other operations with `rm`, such as either removing entries in `/etc/path_to_inst` or removing `/devices` entries, are risky and therefore not recommended. Carefully read the appropriate manual pages before attempting such operations.

Persistent binding between devices and their logical links in the device tree is an essential requirement for leveraging FC SAN support. SAN reconfiguration occurs often, and the native SFS used with Solaris OS reduces the overhead of tracking software applications and devices as they are recabled, rezoned, or moved about the SAN. This reconfiguration increases the availability and utilization of storage and associated applications, relieves the administrator of cumbersome target driver configuration files, and most importantly, does not require disruptive system reboots.

Sun SAN Solutions

As SANs have grown to become true networks, Solaris OS and its native SFS have become a true storage network-protocol driver stack. Rather than treating the HBA like a DAS SCSI interconnect, albeit with a longer optical wire, Sun SAN adapters have become true network interface cards (NICs) for the storage network. Redundant paths are virtualized and hidden from applications and storage middleware. Path changes are handled transparently. Thousands of devices can be accessed through the device tree, and new storage is dynamically exposed without rebooting. SAN tapes and media changers are mapped through a consistent device node for backup applications, regardless of SAN configuration changes. All of this is accomplished without manually editing device driver configuration files or performing system reboots.

Sun continues to bring more networking features into the SAN stack. Sun is working with the standards bodies responsible for Dynamic Host Configuration Protocol (DHCP), Internet SCSI Naming Service (iSNS), and Lightweight Directory Access Protocol (LDAP) to bring these standards into the SAN. These services allow hosts and devices to configure parameters automatically and to find storage based on friendly names, using SAN management appliances configured through a central management point. Through its ownership of the whole driver stack and I/O framework, all the way up to the application, and its participation in the networking standards bodies, Sun continues to make Solaris OS the most SAN-enabled OS that is available today and leveraged for tomorrow.

About the Authors

Scott Tracy

Scott Tracy is a Senior Manager of Storage Network Engineering, working for Sun’s Network Storage Division. He has worked for Sun for five years. He manages the Solaris SAN team responsible for software releases related to the SAN Foundation software, specifically the Fibre Channel stack for Solaris OS. Previously he was a manager of Solaris disk and tape driver components as well as non Solaris Fibre Channel failover drivers for AIX, HPUX, and MS Windows. Prior to this, he was a kernel developer working on the Solaris disk driver. Before Sun, he worked as a driver developer for Adaptec on the Easy CD Creator product and for MCI on database applications.

Scott has a BS in Mining Engineering from the Colorado School of Mines, and he currently holds an inactive Certified Public Accountant license in the state of Colorado.

Ken Gibson

Ken is Director of Storage Network Engineering for Sun's Network Storage Division. He has been developing drivers and firmware for networked storage for over fifteen years, starting with hierarchical storage controllers for VAX clusters. He has a BSEE from Michigan State University and an MSCS from the University of Colorado, where he focused on networking and distributed operating systems.

Ordering Sun Documents

The SunDocs^{SM} 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`