SUN BLUEPRINTS APPROVED

[illegible]

Planning to Fail

By John S. Howard - Enterprise Engineering

Sun BluePrints™ OnLine - December 2000

Sun microsystems

ENTERPRISE ENGINEERING

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

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303 USA
650 960-1300 fax 650 969-9131

Part No.: 806-7710-10
Revision 01, 12/08/00
Edition: December 2000

Please
Recycle

Adobe PostScript™

Planning to Fail

Introduction

The architecture of the Sun Enterprise™ 10000 (also known as StarFire™) and Sun's datacenter servers affords several features for increased availability and resilience. However, these features are often overlooked or compromised by the configuration and installation of the hardware and software. By assuming that the hardware and software will fail and being prepared for those failures, you can achieve increased availability and resilience.

Further, there is a wide and often confusing assortment of system software and patches intended for increasing system availability. If improperly combined or mis-configured, these software products may actually contribute to system failures and decreased system reliability.

This paper presents design guidelines and "best practices" for the selection and configuration of system software such as Veritas Volume Manager (VxVM), Dynamic Multi-pathing (DMP), Dynamic Reconfiguration (DR), and Live Update (LU). This paper also focuses on which versions and combinations of these software tools result in viable configurations, and which combinations to avoid.

Design Tenets

The goal is to architect systems that are as simple as possible while being resilient, recoverable, and consistent. To achieve this goal, use the following design tenets.

Plan and Practice

A recovery plan implemented by the technical writing staff is far superior to a brilliant engineering staff working without a plan, attempting a “seat-of-the-pants” recovery. Spontaneity is wonderful in life, but counter-productive in a datacenter.

Even knowing what you don’t know is useful. Consider the dynamic reconfiguration (DR) capability of the StarFire platform. You may be uncertain if you will ever need DR, but suspect it may be required someday. Use this insight to avoid building anything into the system design that would preclude using DR at a later date.

Practicing a plan diminishes the fear of the unknown and knowing what to expect during a failure situation reduces stress during recovery procedures. Never underestimate the human effects of systems failure; stress and panic often lead to mistakes which hinder recovery. Also consider the effective complexity of a design. When under stress, will this design be too unwieldy or complex, and possibly hinder recovery?

Simplicity versus Complexity

In most recovery situations, the actual recovery is conceived and directed by people. System complexity may prevent people from fully understanding a system configuration and its interdependencies. As such, complexity can hinder recovery. Therefore, systems and software should always be configured and implemented as simply as possible.

However, given the realities of software systems and the needs of datacenters, complexity can not be completely avoided. The key is to minimize complexity whenever possible. Complexity should be questioned; ask, what are you trying to achieve, and is there a better (i.e., simpler) tool for the job?

Availability versus Serviceability

The reliability, availability, and serviceability (RAS) needs of the application and services should drive your architecture decision making process. Consider the case of mirroring several disks with a Sun StorEdge™ D1000 enclosure. The Sun StorEdge D1000 enclosure has redundant power supplies, multiple SCSI controllers and the bus can be split into 2 electrically distinct busses. However, the Sun StorEdge D1000 enclosure has a common backplane for both busses. This backplane is not a single point of failure (SPOF) as its failure will not effect both halves. However, as the entire Sun StorEdge D1000 enclosure must be powered down to replace this backplane, it does provide a serviceability issue.

This is not a flaw in the Sun StorEdge D1000 enclosure; it is a design point that must be considered. Only the needs of your application can determine if the Sun StorEdge D1000 enclosure is sufficient. If your application requires a higher RAS level, mirror the disks across two Sun StorEdge D1000 enclosures. All such RAS aspects of all components need to be considered when making your hardware choices.

Don’t Fix It if it isn’t Broken

If you have a solution, that is already configured, installed, active, and you are certain that it is working: don’t change it. When making design changes to an already functional system, be aware of the fine line between being proactive to eliminate problems before they occur, and introducing major changes that can increase opportunity for human error. The deciding factor on whether to change a design should be based on the *actual* stability of a system versus the *perceived* stability. If the system is robust and stable but lacking style or grace, sacrifice style and grace and prioritize stability.

Similarly, do not change things for the sake of change. There is nothing wrong with older systems if the performance is within acceptable parameters. Resist the temptation to tinker with systems and never attempt to fix ones that are functional.

People and Process

When NOT to recover - Commodity Systems

With an n-tier systems architecture, some systems will fall into the category of being “commodity systems.” These commodity systems operate as stateless service providers. For an e-commerce site, the web server front ends are an example of commodity systems. Commodity systems can be considered as disposable systems; when one of these systems fail, it is more efficient to replace the failed system as a whole. For example, having a warm standby for a webserver or using a JumpStart™ server to reinstall the Solaris™ Operating Environment and http server software onto a system from a pool of spare systems may be more effective than attempting to troubleshoot or recover the failed webserver. However, it is crucial that the commodity systems be identified and that spare systems be on hand and “burned-in.” It is also crucial that JumpStart server be configured prior to any failure.

Naming Conventions

Hostname and component naming conventions vary from corporation to corporation and often from department to department. Naming conventions often evolve without any planning or even acknowledgement of their importance. When establishing a naming convention, ensure that it applies to the largest number of components possible, from individual disk drives and system boards to complete systems. Also, consider the type of datacenter the naming convention is for. For example, a naming convention for an e-commerce start-up will have different requirements and design goals than one for a relatively static, established corporate datacenter.

Most naming conventions need to allow for the reality that one name for a component or host is often insufficient. All naming schemes need to accommodate devices that serve multiple purposes. In some situations, it is easier or necessary to refer to a device by a name that is appropriate to that situation. For example, a helpdesk may get a call from a user simply saying “The mailserver is down.” The helpdesk needs to map the abstraction that the user is referring to as the system, to a more useful reference in order to resolve the problem. To accomplish this, each device will need three names: a name that reflects its physical location in the datacenter, a name that embodies the function or abstraction of the service the device provides, and finally a name by which people commonly refer to the device (this paper refers to this as the “truename” of a device).

Storage Naming - Disks, Enclosures, and Cabinets

With the ever increasing storage demands and rapid growth of storage subsystems, a naming convention should provide an accurate, efficient and obvious method to locate components. Further, the method chosen needs to function while people are working under stress or in an emergency.

Make the storage naming convention consistent at the physical storage hardware level (Sun StorEdge T3, A5x00, A3500, etc. arrays) and provide the means for an efficient and obvious mapping to the logical device naming level (VxVM, RM6, SDS).

When given a logical volume name, the naming convention should provide an answer to the following question: “What physical devices make up this logical volume?” Conversely, when given a logical volume or physical name, the naming convention should be able to answer: “Where is the physical component that makes up this device located in the datacenter?” This bi-directional mapping of logical units (volumes) and physical units (disk, enclosure, and cabinet) should be handled by the volume management software such as VxVM, but may need to be done in an external form, i.e. a spreadsheet.

The “Trinomic” Naming System

As an example of an extensible and flexible naming scheme for an established datacenter, the following is an examination of the trinomic naming convention used in the Enterprise Engineering lab. For hostnames, the three names are mapped by using CNAME dns entries and use `nslookup` to map from one name to another.

For storage naming, VxVM utilities such as `vxprint` and the `vmsa` GUI are used to map between physical location names, truenames, and abstract names.

Physical Location

To create a physical location name, an imaginary grid is placed over the datacenter and floor tiles are used as the units. The x-axis is denoted by capital letters, A, B, C, etc. The y-axis is denoted by numbers, starting at 0. The first portion of the physical location name of a system is denoted by its location in this coordinate system. Since systems or cabinets rarely occupy a single floor tile, by convention, a system is “in” the floor tile the cabinet’s front left corner is in.

The second portion of the physical location name is the location within the cabinet as denoted by the number of rack units, starting at 0 and counting rack units up from the floor.

The third portion of the physical location name is either an ‘f’ or ‘r’ for a component in the front or rear of the cabinet, respectively.

Finally, the optional additional fields of the physical name are component specific identifiers, such as system board number, system board slot number, disk slot number, etc.

These parts are then concatenated together with alphabetic field separators to form the physical location name. For example, for a Sun Enterprise 6500 server at grid location D3 and an elevation of 0 rack units (the system is sitting on the floor) its physical name would be: `gD3e0`. Similarly, for the physical name of the front disk in slot 5 of an Sun StorEdge A5200 array at grid location B1 and an elevation of 2 rack-units within the cabinet would be: `gB1e2f05`.

Truename

The “truename” is what operations or system administrations staff use to refer to the device. For a system or StarFire platform domain, this would be the hostname of the systems, such as `blackmesa` or `imp905.` For storage devices this would be an enclosure name, for something like a Sun StorEdge A5200 array, or the network name for a device like the Sun StorEdge T3 array.

For storage, the truename would be the mount point of a filesystem or the sub-disk name or partition name in the case of raw devices.

Abstract Name

The abstract name reflects the abstraction of service that the device provides. For a system or StarFire server domain, the abstract name would be a reference to the service, such as `mailhost06` or `eeweb`. In the case of Sun™ Cluster, this would be the name of the logical service provided by the cluster.

For storage, the abstract name should be the logical volume name. For example, `u01` or `ardblogs`.

Software Frameworks

Patch Management

Patch management is crucial proactive service. Within the recent past, Sun has provided tools such as SunAlert and PatchPro Expert on the SunSolve Online^{SM} website (http://sunsolve.sun.com) to assist with patch management. It is highly recommended to stay current with the kernel update patch for the Solaris Operating Environment and storage subsystem patches, such as device firmware patches and patches for the unbundled volume manager packages.

As with all software and software upgrades, patches should be tested in a production-like environment, such as a staging or test environment, before being installed on production servers. Also provide a roll-back plan, that details how and under what circumstances the patch is to be backed out, with every patch installation request.

Using a JumpStart Server For Recovery

Each subnet with systems running the Solaris Operating Environment should have at least one JumpStart server available on it. The JumpStart server is not only used for controlled, consistent installations of the Solaris Operating Environment, it can also be used as an effective tool for system recovery.

The JumpStart system architecture provides a readily available, known good Solaris Operating Environment boot image that can be used to bring a failed system up to a known state in order to allow recovery. Additionally, the JumpStart mechanism is flexible and extensible, and enables the boot image to be customized with software tools needed to perform recovery, such as VxVM, data restoration tools, troubleshooting applications, etc.

Live Upgrade

Live Upgrade (LU) provides a mechanism to manage and upgrade multiple on-disk Solaris Operating Environments which allows the upgrade of one environment without taking the system down. LU provides a framework to upgrade and work within these multiple on-disk environments, then reboot into the new Operating Environment after the changes to the on-disk software images have been completed.

An Early Access version of LU was provided with the Solaris 8 Operating Environment, however, LU works with and may be installed on all releases of the Solaris 2.6, 7, and 8 Operating Environments.

Live Upgrade Boot Environments

Central to the operation and implementation of LU is the concept of a Boot Environment. A Boot Environment (BE) is a grouping of filesystems and their associated mount points. LU uses the term boot environment instead of boot disk because a BE may be contained on one disk, or may be spread over several disks. LU provides a command line interface and a GUI to create, populate, manipulate, and activate boot environments.

BEs may be created on separate disks or on the same disk, however, a single root "/" filesystem is the recommended layout for the Solaris Operating Environment. For further information on the preferred method of laying out the root filesystem, and the reasoning behind that method, consult the numerous Sun BluePrints™ OnLine articles on boot disk layout.

The active BE is the BE that is currently booted and active, all other defined BEs are considered inactive. Inactive BEs are also referred to as Alternate Boot Environments (ABEs).

The Early Access version of LU does not allow the user to choose which filesystems are included in a BE. However, BEs may be completely self-contained or they may share filesystems. Only filesystems that do not contain any Operating Environment specific data and must be available in any Operating Environment, should be shared among BEs. For example, user's home directories on `/export/home` would be a good candidate to be shared among several BEs.

Further, LU provides a mechanism to synchronize individual files among several BEs. This feature is especially useful for maintaining files such as `/etc/passwd` in one BE and then propagating changes to all BEs.

Sidegrades

The ability to create multiple boot environments and populate those boot environments with live Operating Environment data enables the system administrator greater flexibility to react to changing user needs with minimal downtime. LU enables the system administrator to perform “sidegrades,” the large scale reorganization of the Operating Environment with minimal impact to the user. The following section details using LU to perform such a sidegrade.

Migration to a site-standard boot environment

Over the course of time, the on-disk data of systems and Operating Environments tend towards a state of greater disorder. Work-arounds and special cases are implemented and never re-architected to the site standard. These work-arounds and special cases are usually left in place because the downtime to resolve them is unavailable. LU can be used to reinforce a site standard for boot environments onto systems that have suffered at the hands of entropy and work-arounds.

For example, consider a system that was originally installed with an undersized `/` filesystem. `/` was sized large enough for the initial installation of the Operating Environment, however over the course of time, several patches were installed. The disk space requirements of these patches, and the space needed to save the previous versions of the files incase of backout, caused `/` to become 100% full. As a work-around to alleviate the space constraints on `/` the system administrator moved `/var/sadm` to another filesystem (`/opt2/var/sadm`) and then created a symbolic link from `/var/sadm` to `/opt2/var/sadm`.

In this situation, use LU’s `lucreate` command to create a copy of the current BE into an alternate boot environment with a larger root that has enough space for future patch needs. Then use the LU command `luactivate,` to select the new BE and reboot when convenient.

LU for Patch testing and implementation

As seen in the previous example, LU provides a mechanism with which the system administrator can readily manipulate an inactive Operating Environment. This mechanism can be exploited to provide a software and patch testbed and fallback procedure for systems where a completely duplicated test environment is unavailable or uneconomical. Consider a Sun Enterprise™ 450 workgroup server running the Solaris 8 Operating Environment, this system may provide a necessary, but non-mission critical service. Patches and new software releases should be tested before installation on the Sun Enterprise 450 server, however, completely duplicating the Sun Enterprise 450 server for a test environment is not possible in this instance.

By using LU, an alternate BE can be created with the Solaris 8 Operating Environment installed in it. Patches and software upgrades can then be applied to the alternate BE and the alternate BE then activated. If problems with the newly applied patches or software are encountered, the system administrator can quickly and easily fallback to the stable, known-good original BE.

VxVM

There are a variety of recovery situations in which it is useful to boot from some sort of fail-safe media and still maintain the ability to manipulate VxVM objects. There are a number of ways to achieve this, such as customizing a JumpStart boot image to include the VxVM drivers and command line utilities. However, the straight-forward approach to VxVM recovery preparation is to create a clone disk.

Creating an Operating Environment Clone Disk For Recovery

The clone disk is a filesystem copy of the core Solaris Operating Environment which is written to slices, not volumes. The function of the clone disk is to provide a bootable image without depending on VxVM volumes, yet still maintain the ability to use VxVM utilities and drivers. This allows us to effect a wide range of simple repairs or service procedures without having to unencapsulate the boot device.

Making a clone disk can be complex, be exceptionally careful of the target disk to not overwrite important data. The choice of data copy utility is also important, the common choice for this sort of activity is `dd(1)`. While `dd` is an acceptable utility to perform this function under certain circumstances, it must be used with far more care than filesystem-aware utilities such as `cpio`, `tar`, or `ufsdump`. Because of the lower potential to do inadvertent harm, we prefer a utility which is aware of and works within the structure of the filesystem, such as `ufsdump` and `ufsrestore`.

Once the filesystem copy is made, some modifications must be made to various configuration files on the clone disk in order to successfully boot from slices. The `/etc/system` file and `/etc/vfstab` must both be modified in order to prevent the cloned Solaris Operating Environment from depending on VxVM volumes. However, we still want to load the VxVM device drivers when booting off of the clone disk.

As discussed earlier, the Live Upgrade (LU) product is well suited to manage the clone disk, manage multiple boot environments, provide a safe fall-back boot environment, and provide a synchronization mechanism to keep user specified files up to date across multiple boot environments. LU provides a standard and an easily managed mechanism for duplicating, activating, and managing multiple boot environments. Once LU is released, it should be used as the preferred mechanism for duplicating and managing multiple boot environments.

Slices

As a secondary precaution to help ensure easy and effective recovery in an emergency, allow a way to access the underlying slices on the root disk even if the clone disk does not function. In order to access the underlying filesystems on the root disk, map the subdisks to the corresponding disk partition. This is automatically done by VxVM for the root volume, but no others. Ensure that all file systems required during a recovery, are mapped to partitions so that they may be used in such dire circumstances. Then, in extreme cases where VxVM simply will not function, boot off of CD-ROM or over the network and mount the underlying slices which map to the subdisks for each volume.

Resorting to slices to access the file systems is a common recovery tactic, but it often takes longer to perform this action (and recover from it) than it does to correct the original problem. For this reason, leave it as a recovery option of last resort.

Setting up the Clone Disk

Reserve the clone disk exclusively for recovery operations and keep it under VxVM control to help ensure that it will not be diverted to some other task later on. To setup the clone disk, a similar technique to the VxVM boot disk setup will be used. The following commands will setup and reserve the clone disk:

```
# /etc/vx/bin/vxdisksetup -i c2t9d0
# vxdg -g rootdg adddisk clone=c2t9d0
# size=‘vxassist -g rootdg maxsize nmirror=1 clone | \
  awk ’{print $4}’‘
# vxmake sd DO_NOT_USE \
     dmname=clone \
     dmoffset=0 len=$size \
     comment=”Do not delete or relocate this subdisk.”
```

These commands initialize the target disk and add it to the `rootdg` disk group. A large subdisk mapping the entire usable space of the public region is then created with `vxmake`. This subdisk is named “DO_NOT_USE” to remind system administrators not to allocate volumes out of this space.

The `clone` Script

The process to clone the OS filesystems to the target disk is fairly straightforward, although moderately tedious. The best solution for this is to script the entire operation to help ensure consistent and safe results. The clone script should do the following:

1. Verify the clone disk exists and has no volumes on it
2. Re-partition the clone disk to help ensure slice boundaries are in place
3. Copy filesystem data from the default boot file systems to file systems on the clone disk
4. Install the boot block on the clone disk
5. Rebuild the system file and `vfstab` on the clone disk to allow slices-only booting

After cloning the Solaris Operating Environment, be certain that the clone disk is bootable before placing the system in production. Also, run the clone script out of `cron` on a regular basis to help insure that the recovery disk is kept up to date. And as mentioned earlier, once Live Upgrade is released, use it to clone the boot environment rather than this scripting approach.

Open Boot PROM Configuration

Creating and maintaining Open Boot PROM (OBP) definitions is critical to help ensure easy startup and recovery. Without clear and descriptive device aliases defined and kept up to date, it may be difficult to discern which is the correct boot device. This is especially important for recovery attempts in outage situations. For example, if you need to boot from a mirror or the clone disk, you need to be certain of the full device path to that disk, and if the disk has been replaced or moved.

At the OBP, the host’s file systems are inaccessible, so configuration data may not be verified. Ensure that descriptive device aliases exist ahead of time so that it is clear which device to boot from in any situation. Establish device aliases for “`rootdisk,`” “`rootmirror,`” “`hotspare,`” and “`clone`” and ensure these names are defined on all systems to prevent confusion.

The boot list as defined in the OBP parameter "`boot-device`" should list all valid, possible boot devices that the system attempts to open when it tries to boot. The clone disk should only be used to manually boot in recovery or service situations.

There are several mechanisms to set and control these settings in the OBP environment such as:

- OBP commands `setenv`, `show-disks` and `nvalias`
- Solaris Operating Environment `eeprom` and `luxadm` commands
- VxVM `vxeeprom` command

Dirty Region Logging (DRL)

A Dirty Region Log (DRL) is a bitmap used by VxVM to track which regions of a subdisk are considered "dirty." A dirty region is defined as being a region which has been changed in memory but not yet committed to disk. Additionally a region of a disk that is part of an uncompleted I/O request is also considered to be a dirty region.

DRLs are synchronously written to disk before the data write operation is performed, as such DRL's should be kept off of heavily used disks. Additionally, DRLs and the data regions that they log should never be placed on the same disks. Further, DRL's should be mirrored for maximum availability and recoverability.

The use of DRLs can reduce filesystem `fsck` time after a system crash. The use of DRLs are mandatory with Sun Cluster HA. This requirement is due to the need of the Sun Cluster framework to `fsck` all filesystems of a diskgroup during the diskgroup import step of a cluster reconfiguration.

`vxrelocd`: Threat or Menace?

`vxrelocd`, the VxVM hot relocation feature, evacuates subdisks from components that VxVM identifies as failed or failing. `vxrelocd` performs this sub-disk evacuation without regard for the viability of the resulting configuration. Because `vxrelocd` is unconcerned with system availability constraints, upon a disk failure, hot relocation will evacuate VxVM objects from the failed disk to wherever there is available space. This behavior of `vxrelocd` can potentially compromise the carefully planned reliability and availability aspects of the system with the every disk failure.

Consider the following failure, a disk comprising a mirrored volume (RAID1+0) fails. With hot relocation active, `vxrelocd` evacuates all VxVM objects from the failed disk. The evacuated sub-disks, plexes and volumes may be relocated to the disk being used for the other side of the mirror. This hot relocation has most likely compromised the availability and reliability of the storage subsystem.

Disabling Hot Relocation and Enabling Hot Sparing

VxVM’s hot relocation feature should be disabled in favor of the old hot sparing feature, which uses a pre-designated spare disk rather than evacuating individual sub-disks to anyplace there is available space. To disable hot relocation and enable hot sparing, kill the running relocation daemon, `vxrelocd`, (warning: do not kill `vxrelocd` if it is currently in the process of performing a relocation). After killing the relocation daemon, start the hot sparing mechanism by typing ‘`vxsparecheck root &`’. To make these changes permanent, you must edit `/etc/rc2.d/S95vxvm-recover` to comment out the ‘`vxrelocd root &`’ command and uncomment the ‘`vxsparecheck root &`’ line. After editing, the last 10 lines of `/etc/rc2.d/S95vxvm-recover` should look like this:

```
# start the watch daemon.This sends E-mail to the administrator
when
# any problems are found.To change the address used for sending
problem
# reports, change the argument to vxrelocd.
#vxrelocd root &

# to enable hot sparing instead of hot relocation.
# ( comment out vxrelocd before uncommenting vxspare )

vxsparecheck root &

exit 0
```

The Trade-offs of Complexity: DMP and DR

Complexity may not be completely avoided, but it should be minimized. Dynamic Reconfiguration (DR) and Dynamic Multi-Pathing (DMP) are very powerful tools. However, this power is gained at the expense of simplicity.

Do not implement DR or DMP simply because they are there, only implement DR or DMP if they address a specific need or requirement in your datacenter architecture.

After having analyzed the requirements and verifying that DR or DMP are the correct tools for the need, keep the following features and limitations in mind when planning and implementing your architecture.

Dynamic Multi-Pathing

Use DMP only where appropriate, with the exception of the Sun StorEdge T3 array, DMP is not used for availability; DMP is a tool for performance and load balancing between paths.

The exception to this statement is a Sun StorEdge T3 partner group using VxVM version 3.0.4 for logical volume management. This Sun StorEdge T3 configuration will use DMP for availability by using the multiple paths as a failover mechanism between the two Sun StorEdge T3 arrays of a partner group.

Dynamic Reconfiguration

Dynamic Reconfiguration (DR) configurations are always a moving target, check with Service before making any changes to DR configuration.

At the time of this writing, there are no reported DR issues with versions 7 and 8 of the Solaris Operating Environment. Version 2.6 of the Solaris Operating Environment requires the kernel update 105181-20.

At the time of this writing, Sun does not recommend enabling DR for the Solaris Operating Environment version 2.5.1 due to driver bugs introducing the possibility of system panics. It is not necessary to disable DR if it is actively being successfully used.

DMP is mutually exclusive with many other products and system software. DMP must be disabled in order to use Sun Cluster. Similarly, DMP and DR do not work together.

Sun Cluster

Within your datacenter, identify which data needs to be persistent data. These persistent data stores should become your highly available (HA) data services.

In addition, minimize the number of data services (logical services) per cluster. Multiple services on clusters tend to encourage service interdependencies. These interdependencies can be difficult to recognize until a failure makes them apparent. As these subtle or hidden interdependencies can be especially troublesome within the Sun Cluster framework, try and keep to less than four data services per cluster.

When creating a logical service, do not use auto-failback, always specify the `-m` option to enable manual failback of a service.

Practice logical service failovers and use a regular schedule of failing services over. This will help identify issues such as configuration inconsistencies, configuration errors and help detect marginal (or intermittently failing) hardware components. Regularly failing over the service will also make the system administration and operations staff familiar and comfortable with the Sun Custer framework in general and the failover procedures in specific.

Practice

Always keep the human aspect of datacenters in mind when developing your datacenter architecture. The operations and systems administrations staff will be working under stress, most likely for long periods of time. Plan for staff rotations, food and showers during a recovery event. Part of the pain of recovery is due to staff inexperience with failure and recovery procedures. Frequent recovery drills can make your administration and operations staff familiar and comfortable with your datacenter recovery procedures and these drills can also be used to verify correctness of your runbook.

The Staging Environment

In order to practice recovery procedures, you need a staging environment which is duplicated exactly as possible to the production environment. A staging environment will allow you to test your production environment and your recovery procedures. This staging environment is where you can induce failures and practice recovery from those failures. For example, use the staging environment to test procedures for recovering from failed disks or pulled cables. In addition, test that your applications recover, and that the monitoring methods accurately locate the failure.

Also, use the staging environment to test the procedures for software upgrades, patch installations, new applications, and disaster recovery drills.

In addition to practicing the mechanics of a recovery, test the restoration of files from your backups. This will not only test your restore procedures, it will also validate your backup procedure and backup media.

Conclusion

This paper presented design guidelines and “best practices” for the selection and configuration of system software. Versions and combinations of software tools that result in viable configurations were discussed in addition to which combinations to avoid. As this information is extremely time sensitive by nature, the local Sun Microsystems service team should always be consulted for the latest and most accurate information.

Additionally, it has been stressed that recoverability must be planned for in all aspects of system design and datacenter architecture. Once the systems have been installed, practice drills should be staged to test the recovery procedures as well as the disaster recovery procedures.

Acknowledgements

Many of the ideas in this paper grew out of hallway “meetings” and conversations with the other members of the Enterprise Engineering staff. Especially helpful were the suggestions of Richard Elling, Mark Garner and David Deeths. Also, thanks to Cathleen Plaziak, Terry Williams and David Gee for applying their technical writing skills to this paper. And, as always, thanks to Chuck Alexander and Bill Sprouse (The Management) for giving us the latitude to do what we do.

Author’s Bio: John S. Howard

John S. Howard is currently a Staff Engineer in the Enterprise Engineering group at Sun Microsystems in San Diego, California. He has worked as a software engineer and systems administrator for the past 19 years. Prior to Enterprise Engineering, John worked in Enterprise Services as an Area System Support Engineer for five years. As an ASSE, he was responsible for developing and performing Reliability, Accessibility, and Serviceability (RAS) studies of customer datacenters and the development of proactive Enterprise RAS Services. Prior to Sun, John held engineering positions at: The Chicago Board of Trade Clearing Corporation, Datalogics Inc, and Rand McNally. Throughout his career he has developed: pagination and publishing software, loose-leaf publishing systems, extensive SGML systems development, database publishing systems, text editors and WYSIWIG systems, and device drivers.