HEWLETT® PACKARD

Patching Mission Critical Systems

White Paper

Table of Contents

1. Introduction

This document focuses on patching, which is one aspect of mission critical operations. The term 'patch' is usually associated with code repair, although this definition is not comprehensive. Patches may also be used to enable new hardware, supply new functionality, or deliver utilities. This paper, however, will confine the discussion of patches to defect fixes. In this context, patching falls into the area of ongoing operations and maintenance. Because of this, it is often overlooked or given low priority. Major design efforts and capital expenditures are usually concentrated at the beginning of the project life cycle, during systems design and procurement. Once the systems are installed, it maybe difficult to find additional resources for operations and maintenance activities. Unfortunately, it can take a catastrophic event to drive home the necessity and benefits of proactive support.

Deciding when and how to patch is not always straightforward. It requires both experience in managing software and an understanding of the operational requirements of the business involved. This paper begins with discussion of what patches are and the benefits and risks of patching. It then reviews the process of patching and offers guidelines for developing a patching strategy within the larger context of software change management. It also presents some patching "best practices" from companies that have been successful in managing software in their computing environments.

The recommendations that follow cover areas that are important to the maintenance of systems software in mission critical environments. But even in this area, there is no one "right way" to do things. Every operation is different. Because operations availability involves a tradeoff between investment and uptime, it is up to the business to decide what they are willing to invest for increased system and application availability.

Scope

The recommendations given in this document are targeted for the HP-UX operating system. However, many of them apply to other operating systems. The intent of this paper is to present a mindset and an approach rather than step-by-step instructions. Many of the same principles that work for a mainframe in a data center apply to a laptop in someone's briefcase.

Intended Audience

This paper is intended for IT managers and systems administrators. It includes both general background information on how to approach patching and specific recommendations for improving success. The primary area of focus is mission critical computing. As such, this paper should be useful to any IT personnel involved with mission critical operations and planning.

2. What are Patches?

A Brief History of Patching

Originally, patching was purely reactive. When defects were discovered in a product after it had been released, the development team would create a fix or workaround. These were subsequently released as patches. If a customer encountered the defect and contacted the vendor for support, they were given the patch. Existing users were rarely notified when a problem had been identified and a patch had been created to fix it. Understandably, users weren't happy when they experienced a failure due to a known problem. Even so, many operations today continue to approach patching as a primarily reactive activity.

Customer dissatisfaction over limited access to information eventually led to a different approach to patching. Vendors began making information about patches publicly available. Either through notifications or diligence, systems administrators began patching machines before problems were encountered. Subscribing to the idea that this sort of proactive approach would prevent problems, many systems administrators now undertake to continually apply every new patch they can find. But this can lead to a different problem. Because, like the original software, patches can also introduce defects or side-effects, aggressive proactive patching may introduce new problems as it solves existing ones.

Clearly, these two approaches are extremes. There is a middle ground between purely proactive and purely reactive patching. But there is no one proper approach that applies in all situations. The approach to take depends on several factors. This paper explores those factors, and it gives guidelines for determining the best approach to take depending on the situation and the mission of the system involved.

Types of Patches

The issue of when to apply patches would be easier if patches were limited to defect fixes. But the fact is that many things are released as patches. Patches can be used to deliver driver changes for new hardware, product and application enhancements, utilities, diagnostics, and more. In short, some very useful features are delivered as patches. This fact broadens the scope and importance of strategic patch planning.

Patches are available individually, and they may be combined into groups of related patches known as patch bundles. The grouping can be done by subsystem, platform, or any other logical association. Some patches address a single issue, while others combine several fixes. A patch can range in size and complexity from changing a single character in a file to replacing complete application binaries.

Patches may be used in different ways depending on the task involved. Some examples include:

- New system installations
- Proactive patching
- Reactive patching
- Operating system upgrades
- Hardware and application upgrades

Clearly, there needs to be a plan for each type of patching. For example, an organization may choose to apply patch bundles when performing a new system installation but use individual patches in reactive situations. Or systems administrators may develop a standard system template and create a master image which is applied to all new systems. In short, there is no one "right way" to patch that covers all situations. This topic will be explored in more detail in a separate white paper on patch usage models. For the purposes of this paper, it is enough to say that strategies need to be developed for all aspects of mission critical operations.

3. Benefits and Risks of Patching

Why Patch at All?

Complex software invariably contains some defects. The most common method of correcting software defects is through the application of patches. Patches can also be used to enable new hardware, deliver new or improved functionality, or provide useful utilities. Whatever their use, patches are an important means of software delivery. In terms of reactive support, patches can be used to repair problems and get an operation back up and running. Proactively, patches can prevent failures due to known problems or defects. They can be installed using standard software management tools. They can make complex modifications to critical subsystems, yet they are relatively small, allowing them to be distributed easily in a number of different ways.

Clearly, when used appropriately, patches provide many benefits. Because the application of patches results in changes to system software, patching is a key part of overall software change management. As such, processes and strategies need to be developed for the selection, application, and management of patches.

Proactive Patching

The purpose of proactive patching is to prevent systems downtime due to known defects. In some cases, these are latent defects that will surface under certain conditions or at a certain point in time. Other examples of defects that can be corrected through proactively patching include panics, hangs, security holes, and memory leaks.

The specific methodology used for proactive patching will vary depending on the operating systems, subsystems, products, and applications involved, but the basic mindset should remain the same. A consistent approach to proactive maintenance should be used with all mission critical systems, regardless of what they do or where they are located. This is discussed in more detail in the section on patching strategies.

Reactive Patching

Unlike proactive patching, which can be scheduled and carefully controlled, reactive patching is usually done under stressful conditions. But it is particularly important during this kind of exception situation that care be taken to clearly identify the cause of the problem and then to fix only what is broken. The tendency may be to apply several fixes at once, hoping that one of them will solve the problem, but this is exactly the wrong approach. At best, the problem will be fixed, but the solution will be unclear. At worst, the problem may be compounded, making restoration even more difficult. When approaching reactive patching, it is vital to devote sufficient time to diagnosis

of the problem and to ensure that the patch to be installed will solve it. Taking more time to understand the cause of a problem will lead to a quicker overall solution. Like a physician with a patient, the first rule of reactive patching should be "do no harm". It is also vital to develop a contingency plan before applying any patches. The plan should include a way to back out changes, if necessary.

For mission critical operations, the focus of reactive patching should be the restoration of operation. Even if this means operating with degraded performance, this is usually better than rushing to make changes without proper planning. Unplanned downtime should not be used an opportunity to make changes, even if those changes are already scheduled. The focus during unplanned downtime should be to correct only what is wrong. The problem can then be documented and avoided in the future. A thorough understanding of the problem can be gained through the process of diagnosis, isolation, and resolution. As with any changes to a mission critical system, all reactive changes should first be applied to a test environment.

Patching Risks

Whatever the intended use, it is clear that patches introduce change into the operating environment. When the change is necessary to prevent or correct a problem or to enable needed functionality, patches are extremely valuable tools. However, patches also have the potential to cause as well as solve problems. The key, then, is to understand the risk involved in making changes and to manage that risk.

What Causes Risk?

The core concept to software change management is that change introduces risk. The type and magnitude of the risk depend on the nature of the change. Clearly, care needs to be taken whenever changes are made, especially when dealing with mission critical systems.

This might seem to be an argument against any sort of proactive patching. Why make changes to a working system? The answer is that the risk of making a change needs to be balanced against the risk of not making a change. Take as an example a security hole that leaves key systems exposed. By the time the vulnerability has been discovered, critical information may have been compromised. Proactive maintenance can help to avoid this sort of problem. It can also prevent system hangs, panics, memory leaks, and data corruption.

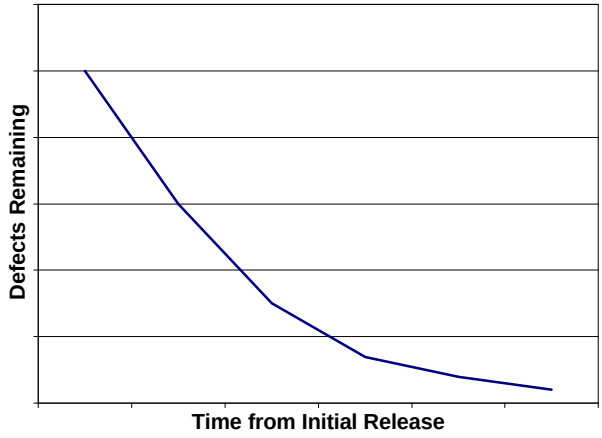
On the other hand, not all updates are beneficial. All too often, what starts as a minor upgrade turns into a major problem. The key question that must be asked before any change is made is whether its benefits outweigh its risks.

Software Quality

Ideally, original software would be perfect, and there would be little or no need for patches. Unfortunately, as software grows in size and complexity, defects are inevitable. While vendors perform rigorous testing of new software prior to release, testing can only catch a percentage of defects. It is nearly impossible for test environments to simulate every possible end-user configuration of hardware, software, and loading. This is true of new versions of operating systems and applications. It is also true of patches.

Following initial development, testing, and release, additional software defects are discovered and corrected over time. This trend is depicted qualitatively in Figure 1. Stated simply, software that has been in use for a long time and in many different environments is less likely to contain critical defects than brand new software. The practical implication of this fact is that brand new patches introduce more risk than patches with more cumulative experience. If experience is defined as total hours of operation, then a rough measure of experience is the age of the patch. Since defects are more likely to be found when a patch is new, older patches are safer and introduce less risk than newer patches.

Figure 1: Discovered Software Defects Over Time



Looking at HP-UX patches specifically, there is a common misconception that once a patch has been superceded, it should no longer be used. However, this is not the case. Patch releases follow a "release stream" similar to the products or subsystems they address. The changes are cumulative. So, for example, a new mass storage patch that supercedes a previous patch has all of the fixes of the older patch along with some new changes. If a mass storage problem is encountered that is solved by two patches, one older and one newer, the older patch is often the better choice.

The only time an HP-UX patch should be avoided is when it has been recalled. Hewlett-Packard recalls patches that introduce either explicit defects or undesirable side-effects. Even with recalled patches, though, it is

important to carefully review the purpose of the patch and the availability of alternatives or workarounds. There are times when it may make sense to keep or even add a recalled patch to a system, if the benefits outweigh the risks.

Types of Risk

Risk cannot be completely eliminated in an IT environment. Just because systems are functioning well today does not mean that they will continue to do so indefinitely. Conditions and requirements may change over time. There are several types of risk involved with software. Some are risks inherent in the product. Others are risks related to the way in which the software is used. Some of the risks involved with patches and the patching process are listed below. In many cases, there are two risks -- one of doing nothing, and another of doing something -- that must be balanced.

Unplanned Downtime

The biggest risk to systems from software quality is downtime. If a critical system or application is unavailable, the negative consequences can include substantial loss of revenue or even loss of life. For mission critical systems and applications, all efforts should be focused on maximization of availability.

Poor Performance

Even if a system is running, it may run the risk of delivering unacceptable performance. If system response is slow, end-users will only wait a limited amount of time before giving up. Some may try again later. Others will find an alternative. In the case of e-commerce, this can mean the difference between a company's survival and its extinction.

Impaired Function

Even though a system may be available and performing its primary mission well, it may still have problems. These can range from minor annoyances to severe usability issues. Take, for example, a problem with a system utility. If the utility is used by only a few users, and if an adequate workaround exists, then making modifications to the system may not be a good idea. If, however, the problem affects a large number of users and hampers their ability to do their jobs, then a fix will provide substantial benefit. In evaluating when to make changes, an assessment must be made as to whether the problem is severe enough to warrant an attempted solution.

The Role of Change Management

Patching falls within the larger context of software change management. It is not enough to just identify and apply patches. There needs to be a consistent framework within which the patch process takes place. Software change management plays a vital role in increasing systems availability. It includes software selection, installations, upgrades, daily operation, and eventual software removal. Formal change management establishes who is authorized to perform which actions on what systems or subsystems at what times. It covers areas such as security and auditing, and it provides oversight so that possible mistakes can be detected and avoided before they are placed into production. The greater the consequences of downtime, the greater the need for formal software change management. For mission critical environments, change management must be fully integrated into all aspects of production operations.

Some of the key elements of software change management include:

- Defined roles and responsibilities
- Documented operating processes and procedures
- Coordinated, cross-functional planning
- System and application templates along with a configuration database
- A formal release-to-production process for new hardware, software, and applications
- Established support and escalation processes
- A detailed change log for each system
- A test environment that closely resembles the production environment
- Procedures for testing and validation of proposed changes
- Contingency and back-out planning
- A comprehensive, validated disaster recovery plan

All of these elements combine to foster a mission critical mindset within the data center.

Business/IT Alignment

Increased system uptime is not free; it requires significant investment. Failure to make the necessary investment puts the business at risk. Investments that increase the availability of mission critical systems should be seriously considered. For critical operations that cannot get sufficient funding to support proper software change management, there may be an alignment problem between business and IT.

At the same time, many operations are classified as 24x7x365 unnecessarily. Since IT spending is limited, the needs of different systems and operations must be balanced against one another. One way to do this is by ranking different operations in the enterprise in terms of priority. This will bring focus to the relative importance of each system. By setting priorities, it becomes clearer where to invest IT resources.

Systems versus Operations

At this point, a further note is warranted on the difference between systems and operations. A system is a machine, while an operation is a function. It is the business function that is important. If it happens to run or depend on a single system or application, then the system or application is mission critical and it represents a single point of failure. If, however, the system has built-in redundancy -- perhaps through the use of system clusters -- then the failure of one component does not cause a loss of function.

4. The Patching Process

The typical software lifecycle involves investigation, planning, implementation, continuous delivery, and obsolescence. While the process of applying patches clearly fits into the delivery phase, there are elements of the patching process that fit into other phases, as well. The success of the overall patching process depends on the successful execution of each step.

The basic stages of patching include:

- **Strategic planning** to determine an appropriate approach to patching for the operations involved
- **Diagnosis** to determine whether patches are needed in a given situation, and why
- **Identification** of an appropriate patch or patches
- **Dependency analysis** to find if the selected patch(s) depends on additional patches, and if those patches, in turn have dependencies
- **Conflict resolution** to resolve any possible structural conflicts among patches resulting from multiple patches affecting the same subsystem
- **Retrieval** of the selected patches, either electronically or via physical media
- **Installation planning**, including allowances for scheduled downtime and procedures for making changes and contingency planning to back out changes if they appear to have undesirable side-effects
- **Installation and testing** of the patches on a target system. This should be a multi-step process, with changes being made first on a pre-production system to validate the changes, and later in the production environment. Pre-production testing may last for an extended period to verify proper function.
- **Verification** that the patch(s) installed correctly and that it has the desired effect. This includes checking the installation logs and verifying that the patch was correctly configured.
- **Distribution** to other systems in the environment, if applicable. Possible steps include setting up a patch depot server and scheduling installation on other systems.

Following established procedures will help to ensure that patching goes smoothly.

5. Software Management Strategies

This section outlines a set of software management focus areas based on usage and tolerance to downtime. As mentioned earlier, there is always a risk that software that has been successfully tested in a lab environment may cause problems when applied to a "new" configuration. For this reason, it is important to limit the number of changes made to a target system. This is a departure from the "install all the latest patches and upgrades" philosophy that HP has used in the past. The new message is "install only what you really need, and understand the risk you're introducing".

Definition of Strategies

Hewlett-Packard has developed three strategies for dealing with software change management in mission critical environments. The strategies are based on operational requirements. The process of selecting an appropriate strategy seeks to align behavior with the key business objectives of the systems involved. The goals of evaluating an operation and choosing an appropriate strategy include:

- Reduced risk
- Increased system and application availability
- Reduced maintenance time

The three strategies for software change management are: Restrictive, Conservative, and Innovative. Four operational factors are used to determine the appropriate strategy:

- **New feature requirement**: the need to introduce new operating system and/or application features into the operating environment
- **Unplanned downtime**: tolerance for the operation not being available outside the scheduled maintenance windows
- **Impact on core business**: the extent to which the business ceases to function as a result of downtime
- **Self-maintenance**: an indication of whether or not all systems planning and maintenance activities are performed in-house without vendor or 3rd-party involvement

Using these factors, Table 1 lists the resulting change management strategies.

Table 1: Criteria for Software Change Management Strategies

Strategy	New Features	Unplanned Downtime	Impact on Core Business	Self-Maintenance
Restrictive	No	Unacceptable	High	No
Conservative	No	Unacceptable	Medium	No
Innovative	Yes	Acceptable	Low	Yes

A more detailed description of each strategy now follows.

Restrictive

This strategy focuses on business operations that require maximum stability and availability. All processes should be designed to eliminate risk and ensure that nothing interferes with standard function. Air traffic control, patient monitoring, telecommunications, and online financial transactions are examples of restrictive environments.

New Features

Since the primary focus of this type of operation is system availability with zero downtime, there should be virtually no requirement or desire to implement new features. Any new functionality is first tested within a stringent test environment and will only be put in place when there is a direct business need to do so.

Unplanned Downtime

There is zero tolerance for downtime in a restrictive environment. Activities that could impact standard operations are seen as possibilities for failure. Restrictive environments often have little or no time available for scheduled downtime, either.

Impact on Core Business

If the operation comes to a halt the business itself comes to a halt. There is an absolute one to one relationship between the operation being available and the business being able to run. There is usually no workable operational alternative.

Self-Maintenance

This operation usually has a staff of very highly qualified personnel who are capable of performing many of the routine tasks. However, when it comes to systems planning and exceptional tasks, the vendor is usually called upon to provide for assistance, support, and backup.

Conservative

This strategy's goal is to ensure the high availability of its operations. Limited changes can be made once all risks are investigated and addressed. To maximize availability, modifications are first evaluated in a test environment. A Conservative strategy for software change is appropriate for many mission critical systems. Examples include payroll processing, financial reporting, and most batch-oriented functions.

New Features

Since the primary focus of this type of operation is high availability, the need to implement new features or functionality is limited. Only features that will increase availability or are business-critical are considered.

Unplanned Downtime

Systems that employ a Conservative strategy have a low tolerance for downtime. Only the most critical changes are considered for implementation. Available windows for scheduled downtime are usually limited.

Impact on Core-Business

The main difference between Conservative and Restrictive strategies with regard to impact on core business is in timing. For Restrictive environments, loss is immediate and catastrophic. For Conservative systems, the impact is not as sudden. If the operation comes to a halt the business itself will continue to run, although an extended outage may have considerable impact.

Self-Maintenance

This operation usually has a staff of very highly qualified personnel who are capable of performing many of the routine tasks. However, when it comes to systems planning and exceptional tasks, the vendor is usually called upon to provide for assistance, support, and backup.

Innovative

This strategy is primarily driven by the need to provide new functionality, cutting-edge technologies, or research and development. This more aggressive position requires a higher tolerance of downtime than the other two strategies. Simulations, prototyping, and new hardware or software qualification are common activities in Innovative environments.

New Features

Since the primary focus of this type of operation is to test and implement new technologies, the need for new features is high. New features are required to meet specific needs, and without them, the business is at risk. They operate at the leading edge.

Unplanned Downtime

Because of the frequent changes required in Innovative environments, unplanned downtime is not uncommon. This is an operational expense and should be planned for. Often planned downtime occurs on an ad hoc basis as well, rather than at pre-planned times.

Impact on Core Business

If the operation stops, there will be little or no short-term impact on the core business. Downtime must be allowed for and recognized as one of the costs of maintaining a leading-edge operation.

Self-Maintenance

Environments employing an Innovative strategy will frequently perform all planning and maintenance activities without vendor involvement. Often technical staff performing maintenance tasks will use the opportunity to further their knowledge of the technology involved. Operations may be performed by developers or software engineers. Only when operations are stopped for an extended period of time will the vendor be called in to help.

Strategic Recommendations

Recommendations for software change management have been developed that correspond to each software change strategy. They cover five different areas:

- **Operating System and Applications**: including the core O/S and major applications used by the operation
- **Proactive Patching**: including all patching activities for which no symptoms or problems are currently evident
- **Reactive Patching**: performed in response to a visible system problem
- **Change Management**: covering all processes and standards used to manage data center operations
- **Test Environment**: including systems, software, and equipment used to support the production operations. The test environment is used to evaluate changes before they are put into production.

Table 2 lists the recommendations for each software strategy.

Table 2: Software Management Recommendations

Strategy	O/S & Apps	Proactive Patching	Reactive Patching	Change Management	Test Environment
Restrictive	Stable releases, available for 1+ years	Use only thoroughly-tested patches with the highest level of field experience	Make fewest changes possible to restore function; perform full diagnostic analysis before attempting a solution	Formal plan with explicit roles & responsibilities; prepared plan to back out changes if necessary; documented DRP that is updated & tested at least yearly	Dedicated equipment that matches production environment including simulated loads
Conservative	Stable release, available for 6+ months	Use only thoroughly-tested patches with substantial field experience	Make fewest changes possible to restore function; perform full diagnostic analysis before attempting a solution	Formal plan with explicit roles & responsibilities; prepared plan to back out changes if necessary	Dedicated equipment that matches production environment
Innovative	Stable release, available for 2+ months	Patches should be carefully reviewed for risks and benefits	Focus on restoration of function; limit number of concurrent changes	Established roles &responsibilities	Test or development equipment or off-hours on production environment

6. Patching Best Practices

As stated previously, there is no single "right way" to patch mission critical systems. What follows is a collection of best practices from IT organizations that have successfully increased their systems availability. Some are directly related to patching. Others fall more within the area of software change management. What they have in common is the element of planning. Because all changes to a mission critical environment introduce risk, they must be planned for as carefully as possible.

Testing

The single most important thing that can be done to ensure the success of software changes is to test them first in a non-production environment. The purpose of this testing is to uncover potential problems unique to the systems environment in which the patches will be used. The success of this approach, however, depends on how closely the test environment matches the production systems. Depending on the magnitude of the change and the required level of confidence, changes may need to be tested for extended periods of time under simulated loads.

Once a change has been tested in a pre-production environment, it should be released to production in phases. That way, it will be possible to check for systems interactions that could not be duplicated in the test environment. If problems are encountered, it is important to factor in recovery time in the move-to-production process.

Use of Templates

In terms of patching, the approach taken needs to be tailored to the mission of the system involved. However, it is not practical to develop individual patching strategies for each system. By grouping systems according to function, a limited number of patching templates can be created. These templates can be incorporated into a comprehensive change management plan and configuration database.

For example, all systems that support enterprise resource planning (ERP) might be assigned to one group. A template can then be created for the ERP group that defines patching standards and rules for change management. The rules should limit the kinds of patches that can be applied and define specific procedures to follow for backing out changes. They might also point administrators to a single location where a "master copy" of software is maintained. This use of a centrally-managed software depot improves manageability and usability. It also allows administrators to monitor usage information that would be difficult to gather by other means.

By combining the depot with a knowledge base, it is possible to record information such as:

- Which patches have been installed on what systems, and for how long?
- Which patches have fixed specific problems?
- Which patches have caused problems?

Preparation and Contingency Planning

Whenever a change is going to be made to a mission critical system, it is important to develop a contingency strategy in case something goes wrong. These precautions will speed the recovery time as well as help in estimating the duration of performing the change. Some steps that can be taken include:

- Performing a full backup before applying changes
- Splitting mirrored volumes to maintain an original copy
- Making a recovery image
- Verifying patch install and de-install on a test system
- Developing a plan to back out changes if necessary

Another important part of planning is factoring in the time and number of personnel necessarily to do each of the steps. These steps should be calculated in serial since most are dependent upon successful completion of the prior step, and many functions cannot be easily split among personnel or done in parallel.

Patch Processes

Patch Selection

A patch selection process provides an added level of safety for production systems. Stated simply, newer patches are not always better. There may be multiple patches that fix a particular problem. In deciding which one to use, consider the fact that older patches have more cumulative time in production than brand new patches. As a rule of thumb, the best patch to use is the oldest one that solves the problem. It is a good idea to develop a process for selecting patches that recognizes this relationship.

Recalled Patches

Likewise, it is a good idea to develop a process for dealing with recalled patches. It might seem that all recalled patches should be removed immediately from every system. But even this is not a firm rule. When HP recalls a patch, it does so because the patch introduces a new defect or an undesirable side-effect. But that same patch may fix a critical defect. In analyzing recalls, it is important to determine the following information prior to taking action:

- Why was the patch recalled?
- Does the reason for the recall affect local systems?
- What does the patch fix?
- Do alternate patches exist that fix the same problem?

In the end, judgment needs to be applied.

Clustering

One way to minimize the impact of both planned and unplanned downtime is through the use of high availability clusters. For HP-UX systems, this involves the use of products such as MC/ServiceGuard or MC/LockManager[1]. For clustered systems with automatic fail-over, the unplanned downtime benefits are clear. When a system or subsystem fails, the application automatically moves to another node in the cluster.

System clustering offers advantages from a software management standpoint, as well. By shifting applications around within a cluster, it is possible to perform "rolling updates" of software, including patches. That is, it is possible to move all applications off of one machine in the cluster and update it without experiencing any downtime. Once the update is complete, the system is returned to the cluster and another machine is updated until all systems have been updated. For operations where there is truly no available downtime, clustering provides the ability to perform maintenance.

Use of Patch Bundles

In many situations, using standard patch bundles is a more efficient and less risk-prone way to patch than applying individual patches. Especially when patching proactively, bundles offer the following advantages:

- all dependency analysis has been performed
- patches in the bundle have been tested together as a group
- unlike individual patches, bundles require a single system reboot
- bundles can be used to create standard patch depots for easy deployment
- bundle labels provide easy tracking of patch levels

For HP-UX, there are a number of standard bundles which available on support media such as Support Plus. Additionally, customized collections of patches can created using tools such as Custom Patch Manager.

The use of bundles is rarely if ever appropriate for reactive patching. When trying to repair a particular problem, it is important to apply changes carefully. A good rule for reactive patching is to apply the minimum change necessary to restore function. This means that more time needs to be spent in diagnosing the problem than in rushing to fix it.

[1] More information on this topic is available in the book "Clusters for High Availability" by Peter Weygant, ©1996 by Prentice Hall [ISBN 0-13-494758-4].

Vendor Partnerships

Given the rapid pace of change in computing, it is not possible to keep abreast of everything. Although open systems promise a wide selection hardware and software components, there clearly are variations among vendors. One of the key things a business can do to ensure the availability of their systems and applications is to form support partnerships with their key hardware and software vendors.

The word 'partnership' is important. It goes much farther than the purchase a support contract that allows the owner to make calls to a support line. Often vendors can provide specific expertise and economies of scale. Internal IT staff should be focused on managing the business. Time spent on operating system and application software maintenance is time that is not available for other activities. By forming support partnerships, an IT operation gains expertise they could not develop and maintain in-house.

In terms of patching, vendors like Hewlett-Packard offer support plans that include patching analysis and consulting. With the cumulative experience of HP's field personnel, they will be able to do a safer, more thorough, and more cost-effective job of patching than an individual company's IT staff.

Proactive Patching Frequency

The final best-practice relates to patching frequency. How often to patch depends on several factors. The important point is that changes should be driven by business needs. If a configuration is stable, with no required changes to hardware or software, semi-annual proactive patching may be all that is needed. For more dynamic environments, monthly updates may be required. It should be pointed out, though, that the burden on systems administrators of proactive patching is proportional to frequency. Organizations that choose to make frequent changes must be willing to accept not only the higher risk, but also the higher IT costs.

As a rule of thumb, operations that adopt a Conservative strategy should proactively patch no more than once every three months, while semi-annual patching is more appropriate for Restrictive environments. This frequency balances the need for system stability with the need to stay abreast of important updates and fixes. Because Hewlett-Packard releases standard media updates quarterly, it may be worthwhile to synchronize proactive patching with these releases.

7. Summary

Patching plays an important role in systems availability. Making changes to an environment introduces risk. For this reason, changes need to be carefully planned and managed. For all systems, and especially for mission critical ones, the process of patching needs to be part of a larger change management plan. By adopting strategies and best practices for software change management and patching, it is possible to improve uptime while reducing administrative workload.

While there is no one "right way" to patch, steps can be taken to improve success. Careless or ad hoc processes are an invitation to disaster. Strategies for both proactive and reactive patching should be developed with the collective involvement of all those who have a stake in the outcome. By working in close partnership with systems and software vendors, it is possible to develop patching strategies that minimize risk and maximize uptime of production operations.