

WHITEPAPER

RED HAT ENTERPRISE LINUX 6 SCALABILITY

EXECUTIVE SUMMARY

Scalability is one of the major areas of focus in Red Hat® Enterprise Linux® 6. The importance of scalability is driven by the convergence of several factors, which have combined to create a strong customer demand for a high-performance, production-ready Linux infrastructure.

Red Hat Enterprise Linux is now enterprise-proven as the environment for the most demanding enterprise applications. Red Hat Enterprise Linux has demonstrated the power, reliability, and robustness demanded by production environments. A good example of this is the increasing use of Red Hat Enterprise Linux to run SAP, one of the most demanding enterprise application suites.

Customer demand for large Linux systems is driven by the migration from existing large UNIX systems running on proprietary RISC architectures to Linux running on standard hardware. This is driving the demand for large, highly capable Linux systems to replace aging large, highly capable UNIX systems. Another factor is growing demand for IT resources at virtually all companies. Most of this demand is for large numbers of small to mid-range systems, either blade servers or rack-mounted servers.

Primarily based on the x86-64 architecture, these two-socket or four-socket systems now include 8-64 processors or cores, support a terabyte of memory, and include high-performance I/O. Today, a single processor contains 4-16 cores, and this will continue to grow over the next several years. What was an extremely large system a few years ago is a mid-range system today - and will be an entry-level system in a few more years. Further, the large numbers of these systems create challenges for networking, storage, and management.

Thus, Red Hat Enterprise Linux 6 must support the hardware capabilities and customer demands of today while being positioned to support the greatly enhanced hardware capabilities - and customer demands - of the coming decade.

TABLE OF CONTENTS**2 Scalability: Theory and Practice****3 Scalability Drivers**

- 3 64-bit Support
- 4 Processor Count
- 5 Ticket Spinlocks
- 5 Tickless Kernel

6 Split LRU VM

7 Control Groups

8 File Systems

10 Shared Storage File Systems

11 RAS

SCALABILITY: THEORY AND PRACTICE

There are two components to scalability: What are the architectural limits of the operating system, and how well does the system run production workloads. While architectural limits get the most publicity, the real question is how well the system runs production workloads.

To see the difference between theory and practice, look at the Red Hat Enterprise Linux supported system limits page at www.redhat.com/rhel/compare/. Here you can see both the theoretical limits and the certified system limits. The theoretical limits are the architectural design limits of each version of Red Hat Enterprise Linux—the values that can't be exceeded. The certified system limits are the values that have been proven on actual production hardware running real applications.

THERE IS A FAMOUS SAYING

“In theory, there is no difference between theory and practice. In practice there is.” Red Hat applies both theory and practice to ensure that certified systems actually meet customer expectations.

There is a famous saying: “In theory, there is no difference between theory and practice. In practice there is.” Red Hat applies both theory and practice to ensure that certified systems actually meet customer expectations.

As an example, consider what happened when systems supporting 128GB of memory became available and Red Hat increased the certified maximum memory in Red Hat Enterprise Linux 5 from 64GB to 128GB. Since both are well within the Red Hat Enterprise Linux 5 architectural limit of 1TB, this should not require more than plugging in the new, larger memory.

However, there are many subtle aspects of memory management. Going from 64GB to 128GB doubles the number of memory pages that must be managed. Unexpected performance issues can arise in different application workloads, requiring changes to the virtual memory subsystems. Memory management, buffer management, system utilities, and applications may all need to be tuned or modified to provide the expected performance with the larger memory. Without this tuning, applications may actually **slow** down with the larger memory.

With many years of experience supporting production workloads, Red Hat understands that raising the architectural limits is the starting point, not the end point, in supporting large systems. This causes Red Hat to be somewhat conservative when new hardware comes out—it isn't enough for it to just work; it has to work right and meet customer expectations. The only way to achieve this is with real hardware, real applications, real workloads, and solid engineering.

This means that Red Hat Enterprise Linux 6 will be introduced with some certified system limits set below the architectural limits, and that the certified system limits will grow as larger systems become available and Red Hat has the opportunity, together with its OEM partners, to do the engineering and tuning work required to make them work well.

Let's look at some of the changes in Red Hat Enterprise Linux 6 to support the new hardware capabilities and customer demands, focusing on the x86-64 architecture.

SCALABILITY DRIVERS

The big drivers of scalability are 64-bit support, processor count, memory size, I/O, and resource management. Red Hat Enterprise Linux 6 addresses each of these areas.

64-bit Support

Today's processors are "64-bit", which should give them the ability to use 18 Exabytes of memory (1.845 x 10¹⁹ bytes). However, for a variety of reasons, neither hardware nor software actually supports this much memory.

Red Hat Enterprise Linux 5 supports up to 1TB (theoretical) and 256GB (certified). Some systems have been certified with more memory than the certification limit on an exception basis—this requires additional testing and validation and is handled on a case-by-case basis.

Red Hat Enterprise Linux 6 supports up to 64TB of memory (theoretical). The initial certified limits will be smaller than this; Like Red Hat Enterprise Linux 3, 4, and 5 before it, the Red Hat Enterprise Linux 6 certified memory limits will grow over time. Ongoing development and tuning will be performed to ensure effective use of large (and expensive) memory.

Examples of this work are huge pages and transparent huge pages (both implemented in Red Hat Enterprise Linux 6). To explain these, consider how memory is managed in terms of blocks of memory, known as pages. These memory pages are traditionally 4096 bytes. This means that 1MB of memory is made up of 256 pages. Likewise, 1GB of memory is made of 256,000 pages, and 1TB of memory is 256 million pages. There is a memory management unit built into the hardware that contains a list of these pages, with each page referenced through a page table entry.

Hardware and memory management algorithms that work well with thousands of pages (megabytes of memory) have difficulty performing well with millions or billions of pages. This is especially critical since the hardware memory management unit in modern processors only supports hundreds or thousands of page table entries—when more memory pages are used, the system falls back to slower software-based memory management.

Effectively managing large amounts of memory requires either increasing the number of page table entries in the hardware memory management unit, which is expensive and causes other performance issues, or increasing the page size. The most popular sizes are 2MB and 1GB, which are commonly referred to as huge pages. The 2MB page tables scale well to multiple GB of memory, and the 1GB page tables scale well to terabytes of memory.

While these larger page tables help with large amounts of memory, they require changes to system management and some applications. Huge pages must be assigned when the system is booted, are difficult to manage, and often require significant changes to applications to be used effectively.

Transparent huge pages (THP) implement the huge pages but automate most aspects of creating, managing and using the huge pages. Thus, THP hide a lot of the complexity of using huge pages from the system administrator and application developers. Since THP is aimed at performance, the THP developers have done considerable work to test and tune across a wide range of systems, system configurations, applications, and workloads. While special system tuning still produces the best results, transparent huge pages provide excellent performance improvements even with stock settings. Again, note the difference between theory and practice.

An additional complexity is that many new systems are built with NUMA (Non-Uniform Memory Access). While NUMA greatly simplifies designing and building the hardware for large systems, it makes life more challenging for operating system engineers and application programmers.

The big change is that memory on a NUMA system may now be local or remote, and it can take several times longer to access remote memory than local memory. This has many performance implications that impact operating system design, applications, and system management. System designers have concluded that NUMA is the only feasible approach to building large systems—or even medium-size systems—so we can expect to see many more NUMA systems during the Red Hat Enterprise Linux 6 life.

RHEL 6 AND NUMA

RHEL 6 is optimized for NUMA and provides the tools needed to manage users and applications on NUMA systems.

Much work has gone into Linux—especially in Red Hat Enterprise Linux 6—to optimize for NUMA and to provide tools to manage users and applications on NUMA systems. This includes system changes such as CPU affinity, which tries to prevent an application from unnecessarily moving between NUMA nodes. This significantly improves performance. Another tool is CPU pinning, which allows a program or a system administrator to bind a running application to a specific CPU or set of CPUs. These tools, and others, make the difference between a large system that runs well and one that runs poorly.

Processor Count

Red Hat Enterprise Linux 5 supports up to 255 processors (theoretical) and 64 processors (certified). Red Hat Enterprise Linux 6 supports up to 4,096 processors (theoretical). Note that, from an operating system perspective, a core or a hyperthread counts as a processor.

The operating system requires a set of information on each processor in the system. Through Red Hat Enterprise Linux 5 this is done in a simple way, by allocating a fixed size array in memory containing the information for all processors. Information on an individual processor is obtained by indexing into that array. This is fast, easy, and straightforward. For relatively small numbers of processors it works very well. For larger numbers of processors it has significant overhead.

The fixed size array in Red Hat Enterprise Linux 5 is 255 processors and is a single shared resource. This can become a bottleneck if large numbers of processes on large numbers of processors need to access it at the same time. Third, it is inflexible. Adding a single new item to the processor information becomes very difficult, and may be impossible without breaking the existing interfaces.

Red Hat Enterprise Linux 6 addresses this by moving to a dynamic list structure for processor information. The list is allocated dynamically—if there are only eight processors in the system, only eight entries are created in the list. If there are 2,048 processors in the list, 2,048 entries are created.

The list structure allows a finer granularity of locking—if, for example, information needs to be updated at the same time for processors 6, 72, 183, 657, 931, and 1546, this can be done with greater parallelism. Situations like this obviously occur much more frequently on large systems than small systems.

Further, a number of changes and extensions can now be made to the processor information without breaking application compatibility.

This is a major change that has required a lot of work to implement. In addition to implementing the new list structure, every component that touches the processor information had to be updated. Extensive testing was required. And, of course, performance tuning was needed.

The net result of all these changes is that Red Hat Enterprise Linux 6 can support the new systems planned over the next several years. And we have the foundation for supporting truly large systems, if processor manufacturers decide to start building 64,000 processor systems.

Red Hat Enterprise Linux 6 can support new systems planned over the next several years. And we have the foundation for supporting truly large systems, if processor manufacturers decide to start building 64,000 processor systems.

Ticket Spinlocks

As mentioned, NUMA system architecture greatly simplifies hardware design, but places new demands on software. A key part of any system design is ensuring that one process doesn't change memory being used by another process. Data corruption and system crashes are the inevitable result of uncontrolled changing of data. This is done by allowing a process to lock a piece of memory, perform an operation, and then unlock the memory (or free the lock). A common method for doing this is a spin lock, where a process will keep checking to see if a lock is available and take the lock as soon as it becomes available. If there are multiple processes competing for the same lock, the first one to request the lock after it has been freed gets it. When all processes have the same access to memory, this approach is fair and works quite well.

Unfortunately, on a NUMA system, not all processes have equal access to the locks. This results in processes on the same NUMA node as the lock having an unfair advantage in obtaining the lock. Processes on remote NUMA nodes experience lock starvation and degraded performance.

Red Hat Enterprise Linux 6 addresses this issue through a mechanism called ticket spinlocks, which adds a reservation queue mechanism to the lock. This means that processes that need to take the lock will essentially "get in line" and be allowed to take the lock in the order that they requested it. Timing problems and unfair advantages in requesting the lock are eliminated.

While a ticket spinlock has slightly more overhead than an ordinary spinlock, it is much more scalable and provides better performance on NUMA systems.

Tickless Kernel

Another major advance in Red Hat Enterprise Linux 6 is the tickless kernel. Previous versions used a timer-based kernel, which had a clock running that produces a system interrupt or timer tick several hundred or several thousand times a second (depending on what the timer is set to), even when the system has nothing to do. Each time the timer produces an interrupt, the system polls—it looks around to see if there is any work to do. If so, it does it. If not, it goes back to sleep.

On a lightly loaded system, this impacts power consumption by preventing the processor from effectively using sleep states. The system uses the least power when it is in a sleep state. There are several sleep states, with the deeper sleep states requiring even less power. However, the sleep states require time and power for the system to enter them and leave them.

The most efficient way for a system to operate is to do work as quickly as possible and then go into as deep a sleep state as possible and sleep for as long as possible. But it is very difficult to get a good night's sleep when the system timer is constantly waking you up.

The answer is to remove the interrupt timer from the idle loop and go to a completely interrupt-driven environment. This allows the system to go into deep sleep states when it has nothing to do, and respond quickly when there is something to do. This removal of timer ticks from the idle loop produces what is called the tickless kernel.

The most efficient way for a system to operate is to do work as quickly as possible and then go into as deep a sleep state as possible and sleep for as long as possible. But it is very difficult to get a good night's sleep when the system timer is constantly waking you up.

Split LRU VM

One of the secrets to performance is to keep things that may be used close to the processor. All actual work is done in the CPU registers, so the goal is to make sure that the next instruction or piece of data needed by the CPU can be loaded into the registers as quickly as possible. Since there is a trade-off between speed and size, there is a hierarchy of progressively larger-but slower-storage. This hierarchy typically goes first level cache (L1 cache), second level cache (L2 cache), third level cache (L3 cache, included on many new processors), main memory on local numa node, main memory on remote numa node, local storage (disk), and then remote storage.

Data available in the first level cache can be accessed millions of times faster than data on remote storage-but remote storage may be millions of times larger than the first level cache.

Much of the memory on a running system is used for system managed caches-copies of information kept where they can be accessed in microseconds rather than milliseconds. Since the system usually doesn't know what piece of information will be needed next, a variety of algorithms are used to predict what will be needed. This includes keeping in memory information that has previously been used-statistically, if a piece of information has been used once, it is likely to be used again. Data that is written to disk is held in memory until it has physically been written to disk, and the copy is left in memory as long as the memory isn't needed for another purpose-many applications will write data to disk and then promptly read it back to make further changes. Other algorithms will read ahead from storage; if some information has been read from a file, there is a high probability that that the next data needed will be the next data in in the file.

Since there is only a finite amount of memory available, the system quickly gets to the point where existing data must be discarded before new data can be placed in memory. This presents no problems, since this data is a cache (copy) of the original data. The only impact to discarding it is that it will take additional time to access it if it is needed again.

Thus, a key part of system performance is how well the system takes care of predicting what data is likely to be used next, bringing in new data (sometimes before it is needed), and getting rid of old cached data to make room for new data.

One of the most powerful algorithms for determining what data to discard is to get rid of the data that hasn't been used in the longest time-the least recently used (LRU). This approach keeps track of when each piece of data in the various caches is used. This is done by tagging pages of memory, and updating the tag each time data in that page is used (read or written).

The system can then scan through the the cached pages of memory, discard the pages that haven't been used in a long time (evict the pages), and replace them with newer data-a process called page replacement. Or, if an application requests more memory, the LRU algorithm is an excellent way to decide which pages can be discarded to free up memory for the application. LRU is one of the core algorithms in Linux virtual memory management, and is a vital element of system performance.

As powerful as the LRU algorithm is, there is room for performance improvement. The old page replacement mechanism had two major issues: First, it would sometimes evict the wrong pages, causing these pages to have to be read in again. This would often occur when the pages that should be evicted were hidden behind other pages in the LRU list, causing the system to evict the pages it could find, rather than the pages it should evict.

The second issue is that the system would repeatedly scan over pages that should not be evicted. For example, on a system with 80GB anonymous pages, 10 GB of page cache, and no swap, the old algorithms would scan the 80GB of anonymous pages over and over again to get at the page cache. This results in catastrophic CPU utilization and lock contention on systems with more than 128GB of memory.

Starting from the premise that not all data is equal, a Red Hat engineer implemented a set of patches that handle different types of pages differently and finds pages that can be evicted with minimal scanning. These patches were, of course, pushed upstream and accepted into the Linux kernel before being included in Red Hat Enterprise Linux 6. The result is the Split LRU VM (Split least recently used virtual memory manager).

Split LRU VM in Red Hat Enterprise Linux 6 delivers a significant improvement in system performance, especially for large systems.

The Split LRU VM uses several lists of memory pages instead of a single, monolithic memory manager. These include separate page lists for filesystem backed data (the master data exists in a file in the storage subsystem and can be read again whenever needed), swap backed data (the VM can page out memory to disk and read it back in when needed), and non-reclaimable pages (pages that can not be discarded by the VM).

There are also significant improvements to locking, making the system more scalable for large numbers of processors and large amounts of memory.

The end result is that the Split LRU VM in Red Hat Enterprise Linux 6 delivers a significant improvement in system performance, especially for large systems.

Control Groups

We have already noted how small systems are becoming large systems, with a two-socket server or blade now including 32 CPUs-and growing. Many simple approaches to managing system resources that worked fine with one processor-or even four processors-do not work well with 32 or more processors.

Red Hat Enterprise Linux provides many options for system tuning that work quite well. Large systems, scaling to hundreds of processors, can be tuned to deliver superb performance. But tuning these systems requires considerable expertise and a well-defined workload. When large systems were expensive and few in number, it was acceptable to give them special treatment. Now that these systems are mainstream, more effective tools are needed.

Further complicating the situation is the trend to use these more powerful systems for consolidation and placing the workloads that may have been running on four to eight older servers onto a single new server.

Let's explore the situation for a moment. A system with a single CPU can be effectively utilized with a single process. A system with four CPUs requires at least four processes to take advantage of it and avoid wasting system resources. A system with 32 CPUs requires a minimum of 32 processes (one per CPU), and is likely to need several hundred processes to keep the overall system reasonably busy.

Many modern applications are designed for parallel processing, and use multiple threads or processes to improve performance. However, few applications can make effective use of more than eight to ten threads or processes. Thus, multiple applications typically need to be installed on a 32-CPU system to keep it busy.

Altogether, we are at a place where small, inexpensive mainstream systems have all the capabilities of the large systems of a few years ago, multiple applications are being consolidated onto a single server, it isn't cost-effective to spend the same amount of expertise and tuning that was previously dedicated to supporting the large systems, and application workloads have become much more variable.

Further, some resources—such as disk I/O and network communications—are shared resources that are not growing as fast as CPU count. The result is that an application, or a process within an application, can consume excessive resources and degrade the performance of the whole system.

Add virtualization into this mix, and you have an urgent need for better ways to control your systems.

Control groups, or cgroups, are a method for combining sets of tasks and allocating and managing the amount of resources that they are able to consume. For example, you can take a database application and give it 80 percent of four CPUs, 60 GB of memory, and 40 percent of disk I/O into the SAN. A web application running on the same system could be given two CPUs, 2 GB of memory, and 50 percent of available network bandwidth.

The result is that both applications deliver good performance and do not excessively consume system resources. Further, the system is significantly self-tuning, in that changes in workload are not likely to significantly degrade performance.

Cgroups do this in three phases: First, a cgroup is created and a task or set of tasks is assigned to it. These tasks run within the cgroup. Further, any tasks that are spawned are also in the cgroup. This means that the entire application can be managed as a unit.

Second, a set of resources are allocated to the cgroup. These resources include cpusets, memory, I/O resources, and network resources.

Cpusets allows assigning a number of CPUs, setting affinity for specific CPUs or nodes (a node is generally defined as a set of CPUs or cores in a socket), and the amount of CPU time that can be consumed. Cpusets are vital for making sure that a cgroup provides good performance, that it does not consume excessive resources at the cost of other tasks, and that it is not starved for CPU resources it needs.

I/O bandwidth and network bandwidth are managed by other resource controllers. Again, the resource controllers allow you to determine how much bandwidth the tasks in a cgroup can consume, and ensure that the tasks in a cgroup neither consume excessive resources nor are starved for resources.

The result is that an application developer or system administrator can define and allocate, at a high level, the system resources that various applications need and will consume. The system then automatically manages and balances the various applications, delivering good predictable performance and optimizing the performance of the overall system.

File Systems

When Red Hat Enterprise Linux 5 first shipped, 500GB drives had just been introduced and most drives were in the 100-200GB range. Most servers at that time would have storage in the 1-2 TB range. Today 2TB drives are common, 3TB drives are becoming available, and 4TB drives are expected to be widely available in 2011.

We can expect systems to commonly have 10's of TB of storage—or more. Further, SAN base storage can be expected to approach 100 TB.

Control groups, or cgroups, are a method for combining sets of tasks and allocating and managing the amount of resources that they are able to consume. For example, you can take a database application and give it 80 percent of four CPUs, 60 GB of memory, and 40 percent of disk I/O into the SAN. A web application running on the same system could be given two CPUs, 2 GB of memory, and 50 percent of available network bandwidth.

As with other areas, filesystems have both theoretical and practical aspects of scaling, which we will explore in some detail.

The EXT File System Family

Ext3 or the third extended filesystem is the long-standing default filesystem in Red Hat Enterprise Linux. It was introduced in Red Hat Enterprise Linux 2.1 and has been the default filesystem for all subsequent releases through Red Hat Enterprise Linux 5. It is well-tuned for general purpose workloads. Ext3 has long been the most common filesystem for enterprise distributions and many applications have been developed on Ext3.

Ext3 supports a maximum filesystem size of 16TB, but practical limits may be lower. Even on a 1TB S-ATA drive, the performance of the Ext3 filesystem repair utility (fsck), which is used to verify and repair the filesystem after a crash, is extremely long. For many users that require high availability, this can further reduce the maximum feasible size of an Ext3 filesystem to 2-4TB of storage.

Ext4 is the fourth generation of the EXT filesystem family, and is the default in Red Hat Enterprise Linux 6. It supports a maximum filesystem size of one exabyte, and a single file maximum size of 16 TB. Ext4 adds several new features:

- Extent-based metadata
- Delayed allocation
- Journal check-summing
- Support for large storage

Extent-based allocation is a more compact and efficient way to track utilized space in a filesystem. This improves filesystem performance and reduces the space consumed by metadata. Delayed allocation allows the filesystem to put off selecting the permanent location for newly written user data until the data is flushed to disk. This enables higher performance since it allows the filesystem to make this decision with much better information.

Additionally, the filesystem repair time (fsck) in Ext4 is much faster than in Ext2 and Ext3. Some filesystem repairs have been demonstrated a six-fold speedup.

While the Ext4 filesystem itself is theoretically capable of supporting huge amounts of storage, the maximum supported limit in Red Hat Enterprise Linux 6 is 16 TB. Work is needed across a range of system tools such as fsck and performance tuning tools to take advantage of the additional capacity of Ext4.

For filesystems larger than 16TB, we recommend using a scalable, high-capacity filesystem such as XFS.

XFS is a robust and mature 64-bit journaling file system that supports very large files and filesystems on a single host.

The XFS File System Family

XFS is a robust and mature 64-bit journaling file system that supports very large files and file-systems on a single host. As we mentioned above, journaling ensures filesystem integrity after system crashes—for example due to power outages--by keeping a record of filesystem operations that can be replayed when the system is restarted and the filesystem remounted. XFS was originally developed in the early 1990s by SGI and has a long history of running on extremely large servers and storage arrays. XFS supports a wealth of features, including, but not limited to:

- Delayed allocation
- Dynamically allocated inodes
- B-tree indexing for scalability of free space management
- Ability to support large numbers of concurrent operations
- Extensive run-time metadata consistency checking
- Sophisticated metadata read-ahead algorithms
- Tightly integrated backup and restore utilities
- Online defragmentation
- Online filesystem growing
- Comprehensive diagnostics capabilities
- Scalable and fast repair utilities
- Optimizations for streaming video workloads

While XFS scales to exabytes, Red Hat's maximum supported XFS file system image is 100TB.

Given its long history in environments that require high performance and scalability, it is not surprising that XFS routinely is measured as one of the highest performing filesystems on large systems with enterprise workloads. For instance, a large system would be one with a relatively high number of CPUs, multiple HBAs, and connections to external disk arrays. XFS also performs well on smaller systems that have a multi-threaded, parallel IO workload. XFS has relatively poor performance for single-threaded, meta-data intensive workloads—for example, a workload that creates or deletes large numbers of small files in a single thread.

XFS is available with the Scalable File System Add-On for Red Hat Enterprise Linux 6.

Shared Storage File Systems

Shared storage filesystems, sometimes referred to as cluster filesystems, give each server in the cluster direct access to a shared block storage device over a local Storage Area Network (SAN). Shared storage filesystems work on a set of servers that are all members of a cluster. Unlike network file systems such as NFS, no single server provides access to data or meta-data to other members: each member of the cluster has direct access to the same storage device (the "shared storage") and all cluster member nodes access the same set of files.

Cache coherency is paramount in a clustered filesystem to ensure data consistency and integrity. There must be a single version of all files in a cluster that is visible to all nodes within a cluster. In order to prevent members of the cluster from updating the same storage block at the same time, which causes data corruption, shared storage file systems use a cluster-wide locking

mechanism to arbitrate access to the storage. For example, before creating a new file or writing to a file that is opened on multiple servers, the filesystem component on the server must obtain the correct lock.

The most common use of cluster filesystems is to provide a highly available distributed service—for example, an Apache web server. Any member of the cluster will see a fully coherent view of the data stored in the global filesystem, and all data updates will be managed correctly by the distributed locking mechanisms.

Cluster filesystems perform well with workloads where each node writes primarily to non-shared files or where shared files are almost entirely read-only. An example of the first case would be a scientific data capture application, where each node is reading a separate stream of data and writing this to a file that everyone can read. An example of the second case would be a web service where multiple nodes are reading a shared database.

Red Hat Enterprise Linux 6 provides the GFS2 clustered filesystem, available with the Resilient Storage Add-On, which is tightly integrated with Red Hat Enterprise Linux High Availability clustering, available with the High Availability Add-On. This provides excellent support for high-performance, high-availability, scalable, mission-critical applications.

Large Boot Drives

As previously mentioned, 2TB drives are widely available now, 3TB drives are becoming available, 4TB drives will be here soon, and drive vendors continue to invest in new technology. Further, widespread use of RAID means that it is common to combine 8-12 physical drives into a single large and reliable logical drive.

Through Red Hat Enterprise Linux 5, the largest boot drive that can be supported is slightly over 2.2TB. This is due to the limitations of the BIOS and its master boot record (MBR) based disk partitioning. Larger partitions can be used with GPT (GUID Partition Table), a modern replacement for MBR, which allows partitions up to 9.4 ZetaBytes (9.4 x 1021 bytes). Red Hat Enterprise Linux 5 supports the use of GPT on data disks, but BIOS-based booting requires the use of the MBR, thus limiting Red Hat Enterprise Linux 5 to 2.2TB boot partitions.

Red Hat Enterprise Linux 6 supports both BIOS and UEFI (Unified Extensible Firmware Interface). UEFI is a replacement for BIOS, which is designed to support new and emerging hardware. The BIOS was created for the original IBM PC. While it has evolved considerably, the BIOS is becoming problematic for modern hardware. Red Hat Enterprise Linux 6 on systems with UEFI allows use of GPT and larger than 2.2TB partitions for both the boot partition and data partitions.

RAS

In addition to the performance aspects of scalability, it is also important to look at RAS—reliability, availability, and serviceability. RAS ensures that a system functions correctly, provides services when needed, and can be maintained or repaired easily with little impact on operation.

The goal of reliability is to ensure that the system is functioning correctly and does not deliver erroneous results. Several approaches are used to enhance reliability. The first of these is to avoid errors, by building reliable hardware and software. A good starting point is specifying high-quality components, especially for power supplies, fans, capacitors, memory, and connectors (CPUs generally do not have different quality grades). Operational factors are also important, such as running a processor below its maximum speed and ensuring good cooling and well-conditioned power. A highly reliable operating system such as Red Hat Enterprise Linux is vital.

Red Hat Enterprise Linux 6 provides the GFS2 clustered filesystem, available with the Resilient Storage Add-On, which is tightly integrated with Red Hat Enterprise Linux High Availability clustering, available with the High Availability Add-On. This provides excellent support for high-performance, high-availability, scalable, mission-critical applications.

There are limitations on how reliable any single piece of hardware can be made, so the next step is to anticipate failures when designing hardware--to detect errors, recover from errors, and continue. An example of this is the ECC memory used in servers. ECC memory has the ability to detect multiple bit errors and to correct single bit errors and continue, allowing the system to continue to run with many types of memory errors. Today, ECC-style approaches are used inside the CPU, across I/O channels, and even across networks.

Another example is RAID storage. Disk drives are one of the most failure prone parts of computers. Engineers proposed that the best way to deal with disk failures was to use redundant disks, so that the failure of a disk would not cause any loss of data. Furthermore, with proper design, the system could continue running while the bad disk was replaced.

Perhaps the ultimate example is fault-tolerant computing, where every element of the computer from CPU and memory to I/O and storage is duplicated. Not only is every component duplicated, but all operations are synchronized and performed in lock step. Any hardware that fails is automatically turned off, and the duplicate hardware continues operation.

While these examples can be implemented in hardware, the true power of RAS is delivered when hardware, operating systems, and applications cooperate to deliver a robust system. Considerable work has gone into Red Hat Enterprise Linux 6 by both Red Hat and hardware vendors to enhance RAS support.

A simple example of this is stateless operations, such as web servers. It is common to use multiple web servers, for both performance and availability. If a web server fails, requests are sent to other web servers. If a web server fails while servicing a request, a simple page refresh in the web browser will re-run the transaction.

If it isn't possible to correct the error and continue, it is critical to detect the error and prevent it from propagating. If this is not done, faulty results and corrupted data can occur. Detecting a hardware error is called a machine check, and dealing with hardware errors is the responsibility of the machine check architecture (MCA). MCA provides the framework for reporting, logging, and handling hardware errors.

For many types of errors, the system can re-try the operation that failed. If the re-try succeeds, the system can continue operation. Of course, the error should be logged, to give you the opportunity to identify trends and fix problems before they occur.

The simplest way deal with uncorrectable hardware errors is to immediately stop the system when an error occurs--to crash the system. While this is a brute-force approach, it is better than allowing errors to continue. This allows work to fail over to another system, to reboot the system, or to repair the system and then reboot it. The MCA logs and crash dumps provide valuable tools for determining why the system crashed, whether it was a hardware or software problem, and input on how to fix the system.

More advanced versions of MCA, implemented in new hardware and Red Hat Enterprise Linux 6, implement a fine-grained approach. Instead of crashing the entire system, they can disable specific hardware components--CPUs, sections of memory, NICs, storage controllers, or other components. These components will not be used until they are returned to service. This allows the system to continue operating even with hardware failures.

As an example, a failure might occur with a NIC in a PCIe hot-swap slot. Further, assume that there are two NICs in the system which are bonded together so that they share network traffic and appear to the system as a single logical NIC. (Bonded NICs can double the network bandwidth available from a single NIC.) The system (hardware plus Red Hat Enterprise Linux 6) would mark the NIC as having an error and disable it. The error would be reported to the system

administrator. The NIC could be removed and replaced with a new one, while the system was running. Finally, the new NIC could be started and added back to the system. During this entire process, the system would continue to function normally, with the only visible sign being a reduction in network performance. This case requires support from system hardware, NIC hardware, NIC driver, and the operating system. It also, of course, requires support by the system administrator and repair team.

A similar situation applies to multi-path storage. It is common to have two or more paths into a SAN, for reliability and performance. The multi-path I/O subsystem is designed to accommodate failures in the fibre channel links, switches, and storage controllers. As described above for NICs, a hot-swap fibre channel adapter experiencing a hardware error can be identified, stopped, replaced, and re-started with no impact on system operation.

Other types of problems have been more difficult to deal with. Failures in CPUs and memory have traditionally brought the entire system down. However, this isn't always necessary. Especially on larger systems, there is a good possibility that a CPU or memory failure will only impact one (or a few) applications, not the operating system itself. In this case it is feasible to stop ("crash") only the applications affected by the failure, mark the failing hardware so that it won't be used, and continue running the system.

In many cases, the affected applications can be immediately restarted with minimal downtime.

The final part of the RAS story is for the system to provide more details on exactly what and where the hardware problem is. Today's systems may have over 100 memory modules, dozens of I/O controllers, and dozens of disk drives. Traditional repair approaches, such as swapping out memory until the problem goes away, simply aren't sufficient.

This is why the new systems don't just report "memory error" or "disk error." Instead they will report "uncorrectable memory errors in DIMM slots 73,74, and 87" or "fatal error in device be2, pci slot 7." This information tells you exactly which devices have failed and need to be replaced and where they are located.

The combination of new technologies in processors, I/O subsystems, I/O devices, drivers, and Red Hat Enterprise Linux 6 delivers systems that are more reliable, have higher availability, continue to operate in the presence of errors which would crash earlier systems, and can be repaired more quickly—all critical components of an enterprise system.

The final part of the RAS story is for the system to provide more details on exactly what and where the hardware problem is. Today's systems may have over 100 memory modules, dozens of I/O controllers, and dozens of disk drives. Traditional repair approaches, such as swapping out memory until the problem goes away, simply aren't sufficient.

ABOUT RED HAT

Red Hat was founded in 1993 and is headquartered in Raleigh, NC. Today, with more than 60 offices around the world, Red Hat is the largest publicly traded technology company fully committed to open source. That commitment has paid off over time, for us and our customers, proving the value of open source software and establishing a viable business model built around the open source way.

SALES AND INQUIRIES

NORTH AMERICA
1-888-REDHAT1
www.redhat.com

**EUROPE, MIDDLE EAST
AND AFRICA**
00800 7334 2835
www.europe.redhat.com
europe@redhat.com

ASIA PACIFIC
+65 6490 4200
www.apac.redhat.com
apac@redhat.com

LATIN AMERICA
+54 11 4329 7300
www.latam.redhat.com
info-latam@redhat.com