OPEN AVAILABILITY ARCHITECTURE: Building highly available systems with Linux and CompactPCI

By Bill Weinberg

With the convergence of voice and data onto next-generation networks, and the growing number of commercial Internet users, service providers must provide quality of service (QoS) levels comparable to today's Public Switched Telephone Network (PSTN) in order to remain competitive. Such levels of service demand so-called "5-nines" (99.999%) reliability and imply that hardware and/or software faults, repairs, and upgrades to the system not interrupt or disrupt ongoing transactions or inhibit establishment of new network connections.

To meet the needs of the rapidly growing Internet, infrastructure equipment manufacturers are now struggling to develop, deliver, and integrate these mission-critical systems. With huge pressures to bring products to market faster, and faced with skyrocketing costs and complexity of system development, these manufacturers are migrating to open architecture hardware and software building blocks sourced from multiple vendors instead of their own tradition of vertical integration.

This building block approach allows them to quickly and efficiently enhance customers' systems and networks, while maintaining the highest levels of service and reliability. Termed the "Open Availability Architecture," this latest technology enables carrier-class OEMs to define and purchase application-ready computing platforms to meet their goal of 5- and 6-nines availability targets without resorting to proprietary hardware or system software components.

High Availability Forum

High availability (HA) is a term for technology that enhances the "uptime" of computer-based communications systems by distributing functionality across multiple, usually off-the-shelf CPUs. In response to hardware and software failures, HA systems facilitate the rapid transfer of control (failover) from a faulty CPU, peripheral, or software component to a functional one, while preserving operations or transactions in-progress at the time of failure. Many embedded designers choose the standard CompactPCI board and system form factor for HA designs since it supports replacement and insertion of system and peripheral cards without the need for cycling power, reducing technician-assisted time-to-repair.

Earlier this year, the HA Forum released its report on open architecture HA solutions. Members of the Forum include, Dialogic (an Intel company), GoAhead Software, Hewlett-Packard, Intel Corporation, MontaVista Software, Motorola Computer Group, RadiSys, and Ziatech (an Intel company). The Forum's report supplies important guidelines that facilitate the development of systems for the Internet and telecom infrastructure with virtually no. The report also promotes the adoption of the open architecture model by providing important best-known methods and capabilities for the development of HA systems. Topics addressed within the report include redundant hardware and software components, storing information, fault management, replacing and upgrading components, and dynamic reconfiguration of the system – all essential factors that lead to the successful implementation and operation of HA systems.

Availability defined

Collections of four, five, or more nines are bandied about in marketing settings without much thought to what availability really means. While "5-nines" or 99.999% uptime literally implies exactly that – 365 days, 23 hours, 54 minutes, and 45 seconds of operational time during a year (or 5 minutes and 15 seconds), in practice that is a not a "useful" number out of context, in that it does not take into account the reliability of constituent components and processes for failure resolution. Availability is best defined as:

$$\frac{\text{MTBF}}{\text{MTTR}}$$

MTBF = Mean Time Between Failures

MTTR = Mean Time to Repair

So, if a system offered a MTBF of 20,000 hours with a MTTR of 2 hours, then its availability would be 99.99%, "4-nines."

Service companies, ISPs, and others have for years promoted availability on a human time base, that is, in terms of days or hours in a week – PST business hours, 7-to-7-by-5, or 24 x 7. In most discussions of telecommunications applications, systems builders want to measure availability on a precise annual basis – how much downtime a system can expect to experience over the course of year (shown in Table 1).

HA for Linux thus either focuses on *increasing* MTBF (clustering, redundancy) or *reducing* MTTR (hot swap, failover).

The communications infrastructure challenge

HA systems for communications infrastructure, unlike standard enterprise computing equipment, cannot merely rely on manufac-

"Nines"	Application	Uptime %	Actual Downtime
2	Office Equipment	99%	3 days 15.6 hours
3	Most IT Infrastructure	99.9%	8.76 hours
4	Internet Infrastructure	99.99%	52.56 minutes
5	PSTN & Other Business Critical Apps	99.999%	5.26 minutes
6	Life Critical Applications	99.9999%	31.54 seconds

Table 1. "Nines," applications, and uptime

turer-specified hardware MTBF figures, since they focus on single components or smaller subsystems. Moreover, high software content equipment must also factor in practically unobtainable software MTBF numbers. On the repair side as well, HA systems for infrastructure usually require theoretically instantaneous failover, that is, the 5.26 minutes of annual cannot occur in one chunk (planned or otherwise), or even several shorter intervals, but rather needs to be spread out in increments lasting no more than parts of second as repair *systems*, not human *technicians*, do their work.

High availability vs. traditional (MxN) fault tolerant systems

In the past, traditional fault tolerance and fault resilience were achieved through the use of redundant systems working in parallel. Two or more hardware resources were dedicated to replicate the computing task at hand, proceeding in lock-step with one another, such that when a fault occurred, at least *one* of *N* systems continued to operate.

Under nominal conditions (pre-fault), an external arbitration mechanism decided which redundant computing node held sway; in some systems, key calculations and programmatic decisions were put up to a vote, with the majority winning out. Example applications include many field central office systems, power grid managers, and some high profile designs, like the space shuttle.

Such systems typically relied on over-engineered, customized hardware (to increase MTBF), built and integrated at great expense. While traditional fault tolerant systems thereby offered many "nines" of availability, the added-value hardware also introduced significant added cost: the systems were expensive to acquire, integrate, maintain, and nearly impossible to upgrade since they did not leverage off-the-shelf components and subsystems.

While such redundancy and arbitration were costly to implement, their impact on application software was usually minimal and often completely transparent. Applications, as long as they presented their output in an acceptable format, didn't have to acknowledge or accommodate the parallelism or monitored nature of their environment. Programs simply ran in parallel, neither taking advantage of additional system resources nor paying a *software* performance price for their enhanced uptime. When faults did occur, failover to working nodes appeared to be instantaneous, since hand-off was managed externally and all nodes maintained parallel, independent state information. Faults effectively consumed, however, entire nodes, since arbitration could not determine the scope of a failure: the fault could lie with a node's CPU (fatal), with an individual peripheral or subsystem (possibly recoverable), or with individual software components, interfaces, or programs (usually recoverable). So, regardless of the actual fault scope, fault tolerant systems classically remove the entire node from action.

Highly available system architectures contrast starkly with fault tolerant designs, as outlined in Table 2. HA leverages less expensive, commercial off-the-shelf (COTS) components (cf. RAID), in a distributed environment.

Because the hardware is primarily COTS, upgrade paths exist at all levels, allowing designers, integrators, and IT teams to reap the benefits of Moore's Law for CPUs, plummeting memory prices, expanding storage capacities, faster networking, etc. And, rather than "burning" CPU and peripheral bandwidth in spares that only replicate active functions, extra CPU horsepower in a nominal, functional (pre-fault) system can be applied to computing tasks at hand instead of being held in reserve.

	Fault Tolerance	High Availability
Architecture	**M x N**	**N + M**
Hardware	Redundant, Highly Custom	Distributed COTS
Upgrade Path	Difficult, Expensive	Cost-effective
Synchronization	Lock-step execution	Asynchronous execution
Application Overhead	None / Transparent	Nominal / Cooperative
CPU Utilization	Low: < 50% – 1/M	High: 50-80%
Failover Latency	Instantaneous	Finite, Characterized
Failover Resolution	Entire Nodes	CPUs/Peripherals/Processes

Table 2. A comparison of traditional fault tolerant and highly available systems

Moreover, when faults do occur, rather than retiring or restarting entire fault tolerant nodes, highly available systems need only failover the individual CPU or hardware interface involved, and have the freedom to choose whether to replace or restart failed applications or software interfaces.

The developer's job, however, is more challenging with highly available design. Programmers must rethink assumptions about hardware constancy, operator proximity, and plan for failure. Applications themselves must be written with integrated health monitoring or at least encased in HA "wrappers" and be capable of accommodating ephemeral resource availability. Applications with real-time requirements must acknowledge that health monitoring and resource management can impact response latencies. The same is true for design, giving rise to the notion of kernel "hardening."

CompactPCI and Linux for HA

From an application software perspective, the choice of form factor is often transparent. Differences among PC-AT, PC/104, VME, PCI, CompactPCI, etc., are encapsulated by the kernel, drivers, and interfaces. Linux is very much at home on these architectures, and solid, commercial ports are available for the embedded architectures represented thereon, specifically IA-32/x86, PowerPC, and to a lesser extent, MIPS (the same cannot be said of Windows and most RTOS platforms).

Why CompactPCI?

Embedded systems designers often wear both software and hardware hats, and cite a mix of reasons for choosing CompactPCI:

- CompactPCI 6U form factor is comfortable for migrating VME designs: same footprint, more capability, more bandwidth
- CompactPCI offers infrastructure for hot swap and other availability enhancers
- Embedded PCI buses and carriers offer higher CPU density per slot
- PCI address space is larger and flatter than VME address windows
- CompactPCI is well supported by Linux

These same designers also complain of:

- CompactPCI's limited slots (1+6) per chassis/domain, vs. VME's 18+ slots per chassis
- High cost of boards and chassis vs. VME (and the unfulfilled promise of leverage desktop PCI hardware economies of scale)
- CompactPCI's limited backplane data rates/bandwidth and cost vs. 100 Mbit and Gigabit Ethernet
- The complexity of building hot-swap aware operating system kernels, drivers, and the challenge of dynamic PCI memory space mapping

The vector sum of the above factors ends up favoring CompactPCI for HA, and my personal experience with embedded Linux certainly confirms this continuing preference. In my two years at MontaVista Software, I have engaged with perhaps 35 development projects for communications-intensive, highly available systems that originally specified VME hardware platforms, both for legacy and new designs. These designs included projects in telecommunications, data communications, aerospace/defense, industrial control, and other areas. All eventually chose CompactPCI and are now in the process of deploying embedded Linux instead on that architecture.

Developers' choices are of course echoed by board vendors introductions and road-maps, and where VME once ruled in terms of board-vendor shelf-space, CompactPCI now dominates. These same vendors until recently promoted their hardware offerings as supported first and foremost by Microsoft Windows NT/2000 and Wind River Systems' VxWorks; now they tout Linux.

As it turns out, MontaVista's Hard Hat Linux version 1.0, introduced in 1999, exclusively supported CompactPCI, and later added support for other form factors, only introducing VME support in our latest Hard Hat Linux 2.0 release. Today, Hard Hat Linux 2.0 Professional Edition targets almost 20 CompactPCI system boards, peripheral slot CPUs, PMC mezzanine cards, and carriers from Motorola Computer Group, Force Computers, Artesyn, MEN, Momentum, SBS, and Ziatech with support for PowerPC, IA-32, and MIPS architectures.

The case for using CompactPCI for HA design remains compelling even when the form factor is leveraged only for its electromechanical characteristics. That is, many HA systems designers will still choose CompactPCI boards and chassis while foregoing connectivity via CompactPCI bus transfers, in many cases completely ignoring the PCI bus and the signals associated with hot swap (ENUM, etc.). CompactPCI in this case "merely" presents a highly reliable, cost-effective hardware platform with functions like health monitoring, inventory management, sparing, etc. all implemented via external network connections.

Such is the case with the above-mentioned HA forum architecture and a demonstration introduced at SuperComm this Spring in Atlanta by Radisys, MontaVista, and GoAhead. The demonstration combined a RadiSys fault tolerant CompactPCI chassis (with built-in redundancy of fans and power supplies); switched Ethernet CompactPCI cards; IPMI platform management; GoAhead's SelfReliant 7000 Series management middleware; and MontaVista's Hard Hat Linux operating system. While the joint architecture leverages CompactPCI for both MTBF enhancement and MTTR reduction, all application communications occur via high-speed Ethernet connections.

Linux, open source, and HA

Now then, why Linux? Open source Linux is a dynamic and growing operating system platform with ample capabilities in many areas, including HA. The Linux community is fostering HA innovation, and the open source development paradigm is providing a fertile proving ground for new ideas.

Developers in all areas of embedded systems development are flocking to Linux for myriad reasons. Key stated motivations are:

- Availability of source code
- Standard APIs and other interfaces
- Stable and robust platform
- Integrated, high performance networking
- Support for a broad range of processors and peripherals
- No runtime royalties

Excellent performance in terms of throughput and real-time response

These criteria translate into two obvious benefits: lower overall cost of deployment and faster time to market.

MontaVista open source HA-related Linux projects

MontaVista Software has contributed a variety of technologies to the open source community as parts of Hard Hat Linux, in shared specification, and as free-standing open source projects, including:

- Authoring the PICMG 2.12 (CompactPCI Software Interoperability) appendix for Embedded Linux
- PICMG 2.12 hot-swap driver as an open source project on SourceForge – http://sourceforge.net/projects/picmg212-hs
- Authoring the low-level interface specification for PICMG 2.13 (system slot hot swap)
- Leveraging the Linux 2.4.x SMP kernel for preemptible, real-time performance, with the MontaVista preemption patch
- The MontaVista real-time characterization project (http://www.mvista.com/realtime)

Developer requirements

In a study involving over two dozen customers, prospects, and partners with high availability requirements, MontaVista confirmed a variety of suspicions about how developers are leveraging COTS software and hardware to build highly available systems. Figure 1 highlights HA technology requirements from these developers and systems suppliers.

The following sections discuss and detail how both open source Linux and MontaVista address these requirements with technologies and products.

Embedded CPU support

CPU preference among HA systems developers and suppliers appears to some degree to track embedded systems CPU choice, biased by the embedded PC product lines offered by key board vendors and by Windows NT/2000 penetration into computer telephony and communications. That is, IA-32/x86 and PowerPC are the two most popular 32-bit architectures for embedded designs in general [1], and also for embedded Linux in particular [2]. While more deeply embedded applications data puts PowerPC architecture ahead of IA-32/x86, economies of scale,

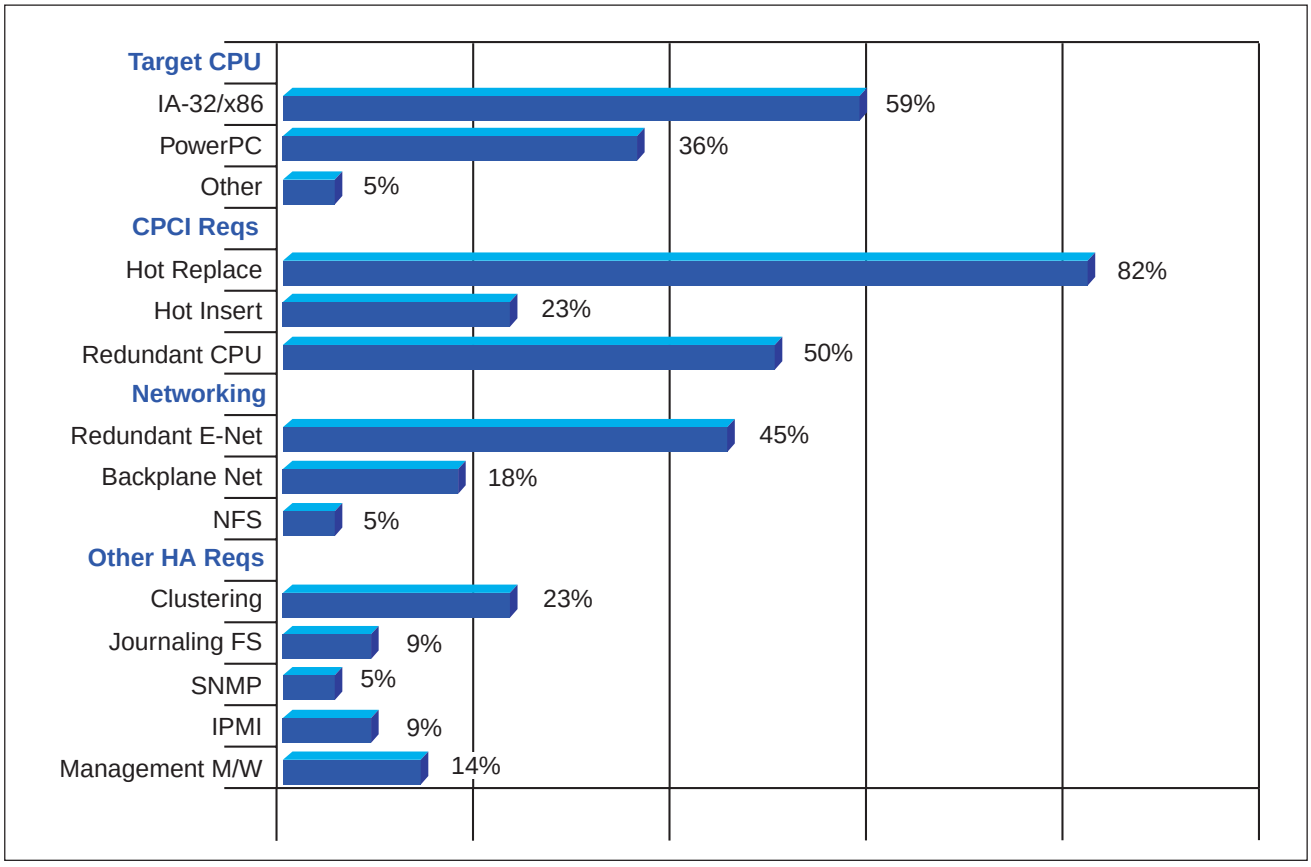

Figure 1. HA technology requirements from developers and systems suppliers

the ability to leverage desktop PCI technologies, and prior assumptions about Windows overtaking embedded operating systems (to say nothing of Linux) in this space motivate CompactPCI board vendors to favor embedded PC architecture in their product lines, and developers in their hardware choices.

Since Linux started out and continues to thrive as a white box PC operating system (see http://www.kernel.org), its support for x86 and for PCI bus derivatives is by default quite good. PowerPC workstation support is also rather good from the open source community, with distributions from linuxppc.org, Yellow Dog, and others readily available for Macintosh and other PowerPC hardware.

Embedded Linux follows suit with MontaVista and other suppliers focusing their energies first and foremost on these CPU platforms in embedded settings, leveraging open source where available, investing as needed, and fostering the open exchange of many derived technologies. Indeed, across all application spaces (not just communications infrastructure), x86 and PowerPC combined, account for approximately three quarters of MontaVista business.

Hot swap: Replacing and augmenting board inventory

Anyone today with exposure to CompactPCI is already familiar with the hot-swap process: go up to a running chassis, pop an extraction handle, wait for the little blue light, and voilà – you can safely remove a board without cycling power. Fewer people are aware of the distinction between hot replace and hot insert, and fewer still have an inkling of the operating system kernel and driver infrastructure required to accomplish this sleight of hand.

Hot replace vs. hot insert

As stated above, the goal of hot swap is to reduce MTTR and so enhance overall availability. It is important to remember, however, that hot swapping boards in and out does not always occur in response to a fault, but as a means to enhance capacity or future fault resilience through hardware upgrades.

Most so-called hot swap solutions to date only implement hot replace. That is, they support only the hot extraction and insertion of instances of board-level hardware already present at boot time. As such, they only maintain the initial inventory of primaries and spares, in terms of number and of type, in a static runtime chassis population. Most such solutions do not even allow the insertion of replacement boards in a slot other than the one vacated by the faulty board being replaced.

Key kernel technologies to support hot replace include:

- Device/driver instance management
- #ENUM handling
- Optional support for nonconformant slot management
- Basic PCI memory space mapping (boot time)
- Some kind of interface abstraction with notions of suspend/resume

Linux supports hot replace very well, as do solutions from hardware and after-market software vendors for Microsoft Windows and Wind River Systems' VxWorks, and indeed the need for hot replace only is the major requirement from the companies polled above.

Developers are not requiring hot insert because they do not believe it exists! This "Holy Grail" of CompactPCI requires all the same capabilities, plus:

- Ability to recognize ID codes for new devices
- PCI memory space expansion, remapping, reuse
- Interaction with a policy system to govern disposition of new hardware resources

While various vendors and platforms attempt to address these requirements in a "band-aid," add-on fashion, MontaVista and the Linux community are building in deep support for hot insert and hot swap underpinnings, and MontaVista is delivering it this year.

Hot swap infrastructure (PICMG 2.12)

Most of the implementations of hot swap to date have been ad hoc at best, and outright kluges at worst. Fortunately, PICMG standard 2.12 for Hot Swap Infrastructure is finally taking hold, allowing developers of system code, devices and device drivers, middleware, and to some extent HA-ready applications to leverage a common technology base.

MontaVista, for its part, already announced an early PICMG-compatible deliverable earlier this year and will be delivering a fully 2.12 compliant solution for multiple CPU architectures and platform vendors early in Q4 2001 for Hard Hat Linux 2.0.

About the HyperTransport Technology Consortium

The HyperTransport Technology Consortium, formed in July 2001, conducts a forum for the future development and adoption of the specifications for HyperTransport technology. The consortium also manages and refines the specifications, and proliferates an infrastructure of test and verification tools to speed up market delivery of devices enabled with this technology. Product samples using HyperTransport are currently available, and other components are planned to be in volume production by the end of the year.

This new high-speed, high-performance link specification is being offered as an open standard through membership in the HyperTransport Technology Consortium. Charter members of the consortium include: Advanced Micro Devices, Sun Microsystems, API NetWorks, Cisco Systems, PMC Sierra, Inc., NVidia Corporation, Transmeta Corporation, and Apple Computers. For membership information, go to www.hypertransport.org.

Hot Swap Aware Driver Architecture

Hot Swap Aware Drivers (HSAD) are device drivers modified and/or originally architected to support the hot swap interface. To be Hot Swap Aware, the driver must be capable of being deactivated, and of being able to restore appropriate state to a board that has been newly inserted. A desirable capability for a HSAD would be to checkpoint its state at appropriate points to facilitate a smooth failover in the case where a hot standby CPU takes over the bus.

The HSAD architecture for Linux draws on earlier work by and for PICMG and hardware vendors like Motorola Computer Group, Force Computers, and others. While detailed descriptions of HSAD architecture have been published elsewhere [3], the key features of the Linux HSAD and differences from conventional device drivers can be summarized as:

- State machine architecture
- Standard multi-point UNIX/Linux device driver extended with IOCTL calls
- Leverages Linux dynamically installable kernel module interface
- Built in, required support for certain types of alarms and logging

Customer expectations and vendor deliverables for HSADs have been at odds. Software platform vendors have delivered HSADs in kit form, as templates, as part of loose frameworks, in binary-only, or for obsolete device sets. In my personal interactions, developers consistently request customizable source code, reasonably standardized APIs, and a good out-of-the-box experience (OOBE) for popular, easily obtainable peripheral hardware.

At present, MontaVista and others are delivering hot swap aware drivers for Ethernet only. Ethernet is the perfect fit for first delivery because:

- NIC devices have simple internal state
- TCP/IP over Ethernet provides delivery guarantees beyond HA system promises
- Numerous interfaces exist off the shelf in multiple form factors
- TCP/IP over Ethernet is used for both applications networking and for health monitoring
- It can be integrated into a variety of topologies and bandwidth sharing schemes
- It demonstrates well
- The Linux kernel and driver family supports Ethernet and TCP/IP as “native” capabilities

Other candidates for HSAD support in the near term include SCSI, IDE, and other peripherals with increasingly complex internal state models. While the ability to hot swap physical media is common in many HA CompactPCI chassis and dedicated RAID systems, it still represents a significant challenge to all embedded and enterprise operating systems platforms including Linux.

Hot swap and intelligent I/O cards

All discussions of hot swap have focused how a system-slot CPU and its chassis handle the appearance and departure of electro-mechanical and logical presence in a system of peripheral devices and their associated mapped memory. The HSADs themselves by definition reside on the system slot CPU built into or bound onto an operating system kernel, in our case, Linux. Hot swap peripherals are presumed to reside on unintelligent I/O cards (no CPU) or on equally vapid PMCs riding on carrier cards with transparent bridges.

Communications equipment manufacturers choosing to leverage CompactPCI several years ago began regarding this division of intelligence (smart system slot, dumb I/O slots) as physically wasteful, especially when compared with VME. In response to their customers’ request for greater distributed computing intelligence, i.e., CPU density, CompactPCI board vendors began offering 6U, 3U, and PMC-based non-system slot CPUs, such that every slot in the local chassis could carry up to three additional microprocessors (a 6U intelligent carrier with 2 PMC CPUs).

These intelligent I/O cards present a “virtual” device footprint to Linux (or any other operating system) running on the system slot. That is, transparent bridges allow the host controller to “see” limited portions of the intelligent I/O card’s memory and memory-mapped resources for sharing (and backplane communications), but not usually as part of a memory-mapped I/O scheme.

Indeed, supporting these intelligent I/O cards is not in the domain of peripheral support, but of standard kernel board support. Each CPU looks like a “little host” and must manage its own, “local” issues, including:

- Boot media – from local flash, via Ethernet, or over the CompactPCI backplane
- Boot image – how much local flash and RAM is available (Linux is smaller than you imagine!)
- Root file system – without local rotating media (disk), Linux/bin and other key directories must mount with NFS or on local RAM or local non-volatile memory
- Boot time – how long after hot insertion is an intelligent node ready for “action”
- Shutdown – a system must be able to accept shutdown commands from the system slot and/or service locally with initial extraction and/or premature extraction attempts
- Halt time – during a hot removal, how long does the system need to shutdown, sync file systems, etc. before enabling the Blue LED
- Slot-to-Slot I/O – how much system slot intervention is needed to set up and maintain communications among intelligent I/O cards; what happens to such communications during a system slot failover

MontaVista and others are meeting these challenges with flexible options for booting compressed and remotely hosted kernel images, for supporting compressed r/w and read-only flash file systems (JFFS and CramFS), for accelerating boot and daemon start times from several minutes to seconds, speeding shutdown

and eliminating costly file system operations with journaling file systems, and by evolving our current Hard Hat Net technology to track the emerging PICMG 2.16.

Redundant system slot

Sparing to eliminate single points of failure does not end with doubling up on peripherals and interfaces, and interconnects – it extends to the system slot CPUs as well, and has spawned a variety of chassis and system architectures to facilitate Redundant System Slot (RSS) functionality.

All hardware platforms targeted by Hard Hat Linux involve dual-domain chassis with dual system slot CPUs. Approaches differ in how the processors manage and connect to multiple PCI bus domains and whether those CPUs are run in a simple N+1 active/standby pair, in a parallel 2N redundant configuration, or in various clustered permutations to squeeze more computing bandwidth out of the system.

Linux itself, while needing to support all such configurations, can accommodate most mixes of CPU and PCI domain connectivity, quite flexibly. Our HA add-on for Hard Hat Linux, for example, today manages the N+1 redundancy in a Ziatech chassis where each CPU must own both PCI domains connected over local PCI-to-PCI bridges, and by Q1 2002 will be able to accommodate other vendors' hardware with either split domains or cross-bar isolated hot standby CPUs.

Even more challenging than managing the internal states of complex I/O slot peripherals is maintaining the potentially heavy and complex states of system controllers themselves on back-up nodes. Since in HA systems, redundant nodes do not run in lock-step and may not process identical data in comparable time frames, applications designers must decide how much state information is transferred among redundant CPUs and how often a checkpoint occurs. Classically, applications designers and integrators must trade off among the scope and frequency of check-pointing against failover time (when and if remaining state data must be copied over or synthesized).

While the Linux kernel will be able to accommodate the hand-off of active PCI bus domains later this year, and Linux device support can handle almost any standard or proprietary checkpointing channel, the check-pointing and health monitoring processes will remain the province of middleware vendors (like GoAhead) or certain key open source projects (see clustering with Linux).

CompactPCI backplane networking

Whether an application's primary use of the CompactPCI backplane is for data transfer, health monitoring, or just for power, it is also desirable to be able to leverage this communication path as a networking medium. Several off-the-shelf solutions exist for passing network packets across the PCI bus at levels ranging from raw transfers across bridges to shared memory, to Ethernet emulation, to varying depths of IP networking, to very high-level messaging abstractions.

Ethernet emulation with Hard Hat Net and PICMG 2.16

MontaVista Hard Hat Net (and other tools like Ziatech's CompactNet) enable both CompactPCI system controllers and peripheral devices to communicate using standard networking protocols across the CompactPCI backplane. By emulating standard Ethernet, such a scheme leverages the CompactPCI bus for intra-chassis networking, and anticipates the emerging PICMG 2.16 specifications for backplane networking, giving embedded Linux developers a tool for today and a path for the future PICMG standard.

Hard Hat Net functions by emulating a standard Linux Ethernet driver to implement the physical layer of the desired protocol (most often IP). Thus, the higher levels of a given Linux network stack operate just as they would for any traditional network interface, allowing support of any type of network packet, not just TCP/IP. Individual CompactPCI cards using Hard Hat Net can communicate with other cards in their local PCI bus or route through network interfaces to other nodes across traditional networking media.

Hard Hat Net abstracts hardware and CPU dependencies (e.g., endianness) to facilitate the transparent mix of vendors and processor architectures present in today's communications systems. Hard Hat Net supports a variety of protocols, including Internet protocol (IP), IPX, and AppleTalk, among others. As a result, Linux developers can take advantage of the high-speed Hard Hat Net interface to implement a variety of loosely coupled and distributed architectures, including Linux clustering, MPI, PVM, and CORBA.

Redundant Ethernet (bonding)

A typical Ethernet-based TCP/IP network connecting two machines, shown in Figure 2, consists of a host connected to a network interface connected to a physical link that is connected to a switch. This switch is then connected to a physical link, which is connected to a network interface, which is connected to another host.

Unfortunately, this connection scheme involves several single points of failure. Putting aside the hosts themselves, these include:

- Single Ethernet cables joining the hosts and the switch
- Individual NIC instances in each host
- The switch itself

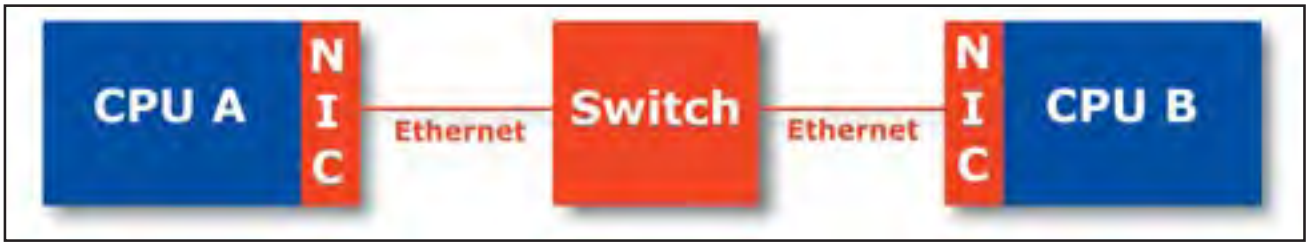

Figure 2. Network with multiple single points of failure

As it turns out, each of these single points of failure represents an easily addressable opportunity to enhance the reliability of the total system through minimal redundancy, while also potentially enhancing system throughput.

MontaVista Hard Hat Linux provides software to support redundant active/backup networking. The solution entails two network interfaces, two physical links, and two Ethernet switches in active/backup configuration, as shown in Figure 3.

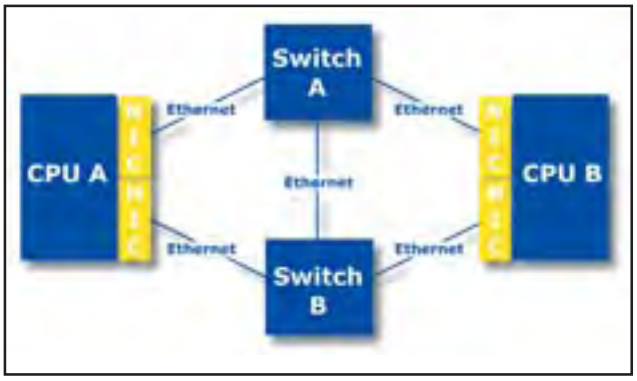

Figure 3. Network with fully redundant components

This redundant configuration can support a failure of any single network interface, physical link, or Ethernet switch, as well as certain multiple failures, and still maintain network connectivity.

The challenge in supporting active/backup network configuration is not in configuring the Ethernet hardware itself, but instead in having software support in Linux for active/backup configuration. To address this challenge, MontaVista contributes software and testing to the development of active/backup software in the Linux 2.4 kernel.

The network bonding driver

The software driver joins two or more standard Ethernet drivers, as exported by the Linux operating system, into one "bonded" network interface. These NICs can be identical Ethernet devices on a CompactPCI system controller, intelligent peripheral card, or PMC devices on carriers, or an asymmetrical admixture of NIC types. The bonding driver software can even control divergent physical interfaces, melding 10 Mbit, 100 Mbit, or Gigabit Ethernet and backplane networking.

The software monitors for link failures using MII link monitoring. If a link failure is detected on the active link, the backup link automatically becomes active. Other solutions include end-to-end heart beating and other diagnostics to detect failures in the network that can not be detected by MII link monitoring.

Clustering with Linux

To date, most Linux-based HA work has focused on clustering, including open source efforts like the Linux-HA project, which provides the heartbeat code that is central to many of the clustering solutions offered by enterprise-focused Linux companies.

More traditional examples of clustering that enhance performance and augment application availability (MontaVista actually demonstrated a cluster running over CompactPCI in a multi-vendor, multi-architecture chassis at LinuxWorld New York in January, 2000). Options include:

Beowulf
By far the best known, most widely implemented standard networking-based clustering software for high performance technical applications. For additional information see http://www.beowulf.org.

FailSafe
SGI's placed its Failsafe product into open source, and Linux developers are now benefiting from and enhancing many of the features provided by Failsafe's years of development as a product under the IRIX operating system. See project home page at http://oss.sgi.com/projects/failsafe/.

Piranha
RedHat's open source clustering resource suite, including the Linux Virtual Server (LVS) and various GUI-based management tools. See project home page at http://sources.redhat.com/piranha/.

Kimberlite
Mission Critical Linux sponsors the Kimberlite project to build clusters that provide support for dual server nodes connected to a shared SCSI or fibre channel storage subsystem, in an active-active failover environment. See download and information page at http://oss.missioncriticallinux.com/projects/kimberlite/download.php.

Management interfaces: SNMP, IPMI, Web

Requirements for HA system management interfaces vary greatly in scope and paradigm preference [3]. Whether your project's preference is for SNMP, a Web interface, or a management scheme based on GoAhead's MOC architecture, you will still need to:

Support hardware management and monitoring in HA systems hardware (MIBs, CGIs, MOCs, etc.)

Leverage available IP networking infrastructure

Unlike traditional RTOS platforms, Linux brings with it rich networking and off-the-shelf options for standard interfaces like SNMP agents and dozens of http servers with customizable CGIs, ranging from the richness of the ubiquitous Apache server down to pico servers implemented in as little as 200 lines of open source C code.

Fault resilient storage

HA systems developers, at least at the system slot level, often require redundant or otherwise fault resilient storage, usually disk-based. Hard Hat Linux 2.0, for example, leverages the various options for journaling/transaction logging file systems in the corpus of open source Linux, primarily JFS, Reiser, and flash-based JFFS.

Hard Hat Linux and software disk mirroring (RAID 1)

A typical host contains a hard disk to store an operating system and applications. Hard disks have moving parts and relatively higher failure rates than passive motherboard components. Since the system disk is a single point of failure, Hard Hat Linux provides disk redundancy by using software disk mirroring as shown in Figure 4.

With disk mirroring, the SCSI cabling and SCSI HBA are still single points of failure. To eliminate these weak points, you can use redundant multi-initiator SCSI interfaces and cabling as shown in Figure 5.

The software driver joins two or more standard partitions on a block storage device. A block storage device could be SCSI or IDE disk drive. Once joined, the software automatically mirrors data written to one disk to both disks. If a disk fails, data will be read from the surviving disk.

To support Open Availability Architecture, two hosts must be able to read from and write to the same set of disks. This environment, multi-initiator, exists in a highly redundant configuration, as shown in Figure 5. MontaVista currently supports development and testing of multi-initiator RAID systems for highly-available configuration.

Middleware

Applications in a HA setting have all the enterprise-type requirements for middleware to support their functionality, as well as needs for middleware and software services to support the cooperative nature of building applications on a highly available but not necessarily fault tolerant base.

Embedded Databases

Telephony and communications infrastructure applications running over CompactPCI-based Linux have database requirements of their own (e.g., SQL servers, OODBC clients, front/back office integration), and will need to provide a reliable foundation for locally hosted data for applications as diverse as usage billing, subscriber info, mobility/roaming support, and routing tables.

Within the domain of monitoring and maintaining the integrity of system hardware and platform software, requirements for a number of databases or database-like software components exist. Specifically, some sort of database entity is needed to manage CompactPCI chassis topology, board-level inventory, driver-hardware associations, and to manage event handling policy.

Other middleware

Beyond the scope of this paper are other types of middleware for Linux-based HA systems, including:

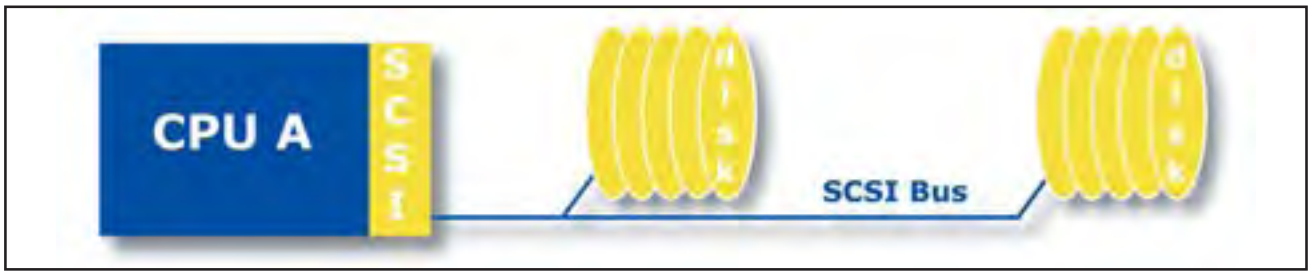

Figure 4. Software disk mirroring removes single point of failure in disks

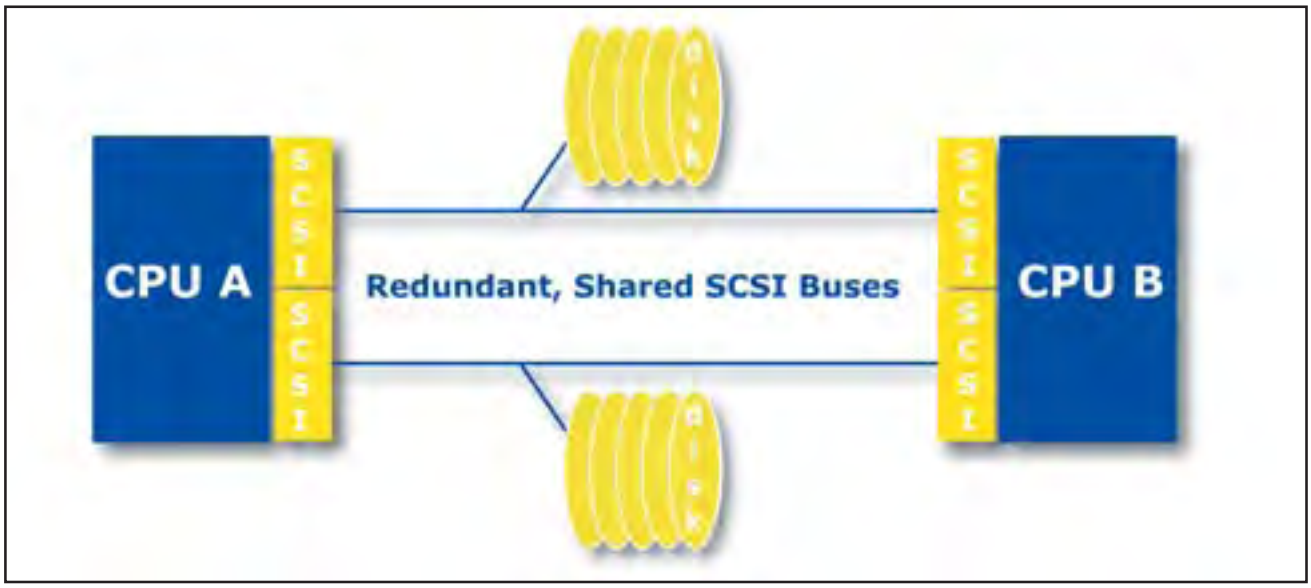

Figure 5. Storage system with fully redundant components

- Management interfaces (see Management Interfaces section)
- Distributed computing layers, e.g., CORBA, COM/DCOM, MPI, others
- Java (MontaVista integrates and offers IBM's Visual Age Micro Edition and J9 Virtual Machine)
- Health monitoring, heart-beating, etc.

Kernel hardening

Traditional off-the-shelf operating system designs, whether architected for embedded, desktop, or enterprise applications, make broad assumptions about hardware reliability and availability. Since MTBF of the most popular operating systems (e.g., Windows) is typically characterized in hours, and the aggregate MTBF of PC, server, or most embedded hardware is characterized in tens of thousands of hours, software designers have felt safe to assume that software will ALWAYS fail before hardware. Such assumptions sadly contribute to the low quality standards for platform software, and render most COTS operating systems inappropriate for use in highly available settings.

Linux benefits from the multi-user, multi-process memory protected architecture that it inherits from UNIX and also from the intense scrutiny that its source code undergoes as an open source entity. Any standard release (even-numbered kernel revisions, e.g., 2.4.x) enjoys the attention of hundreds of developers and kernel architects and millions of deployments, a quality assurance regimen to which few pieces of software are ever subjected. As such, the Web is full of anecdotal evidence that Linux MTBF can often exceed that of the hardware on which it executes.

While this global, pervasive QA effort has gone very far in assuring that Linux itself is highly stable, it has not resulted in all the changes needed for "designing for failure." While the Linux kernel does easily recognize missing components and resources at boot time (e.g., off-line removable media or unplugged notebook CD-ROM drives), it does not readily handle runtime faults like:

- Wholesale device failure (SCSI, IDE, serial, Ethernet, etc.)
- Disk media faults in key file system areas
- Networking interface failures
- Memory failure
- Resource shortages

In general, Linux, like other desktop operating systems, has a tendency to "panic" in the face of some faults and to ignore other fault conditions and blithely proceed with errant execution, and/or to hang. It is important to note that while some such fault response criteria lies with Linux kernel implementers, an even greater responsibility lies with the even larger community of device driver and application programmers.

To address the oversights that result from the disparity between hardware and software MTBF, MontaVista Software and others are embarking on projects to "harden" the Linux kernel and drivers so as to anticipate fault conditions with built-in detection, isolation, reporting, and recovery mechanisms. In particular these efforts entail:

- Rigorous function return value evaluation/provision and appropriate response in kernel and driver code (i.e., evaluate and act on ANSI non-void function return values)
- Log ALL error and even boundary conditions in a standard fashion for easy logging, alarm generation, and error "percolation"
- Elimination of purely polled device interface code and substitution of timeout-generated panics with error logging
- Establishment of options for high and low water marks on all classes of system resource pools (e.g., buffers, network frames, global memory allocation) with alarm generation/event logging when resources pass application-defined low levels and/or return to acceptable norms (or even to abnormal abundance)
- Definition and implementation of monitoring and management interfaces to manage system resources

Conclusion

Despite extravagant claims by vertical solutions vendors and various proprietary software platform suppliers, HA applications benefit little from one-size-fits-all products. As such, developers have had to roll their own technology for HA in addition to building their main value-added offering.

Unfortunately, top-down clustering solutions from vertically integrated systems suppliers could not accommodate developers' particular hardware needs. Conversely, the COTS embedded systems market focused too narrowly on bottom-up solutions offering little more than board support and primitive hot replace technologies. "Filling in the middle" could often cost more than building your own architecture from top to bottom.

Until recently, Linux could only offer an open source tool box, with a steep learning curve and limited out-of-the-box added value beyond platform stability and overall openness. Now, as Linux is maturing and the Linux community gathers momentum for HA technologies, developers can leverage rich frameworks like the "Open Availability Architecture," making better-informed buy vs. build decisions, to bring their systems to market faster, and with lower overall cost.

References:

[1] Venture Development Corp. World Market for Embedded Software Development Tools, in The 2000 Embedded Software Strategic Market Intelligence Program.

[2] Venture Development Corp. Vollume III – Linux's Future in the Embedded Systems Market, in The 2000 Embedded Software Strategic Market Intelligence Program.

[3] Rose and Weinberg. [1999] "Software Concepts for High Availability," in Proceedings of the Embedded Systems Conference, June (Boston).

William Weinberg *is Director of Product Marketing at MontaVista Software where he drives open source embedded tools and technology from conception to market. Bill combines more than 15 years of embedded with real-time experience and expertise in operating system and software tools to leverage Linux for pervasive computing. Previously, he managed Java, embedded Web technologies and alliances programs at Lynx Real-Time Systems. Prior to Lynx, as Acer's Brazil Country Manager, he spearheaded introduction of Pentium technology and Internet-ready PCs. Other experience includes engineering, product marketing, and technical sales roles at companies like DocuGraphix, Animatics, and Microtec Research. Throughout his career Bill has been a featured speaker at industry conferences and a frequent contributor to electronics and telecommunications publications.*

For more information, contact:

Bill Weinberg
Director, Strategy/Evangelism
MontaVista Software, Inc.
1237 East Arques Ave.
Sunnyvale, CA 94085
Tel: 408-328-9213
Fax: 408-328-3875
E-mail: bill_weinberg@mvista.com
Web site: www.mvista.com