

High availability system platforms

I was recently involved in a discussion with a group of industry experts on the topic of "high availability computer systems." As often happens with such a group, there were many opinions, and some areas of serious disagreement. One item that everyone agreed with, though, was that the words "high availability," when applied to a computer system, were fraught with confusion and misunderstandings. On the surface, this is surprising, because the concept of high availability itself is simple and straightforward: It just means that the computer system is almost always running.

ALMOST ALWAYS

"Almost always" is usually quantified as a percentage. A system with 99% availability will be running 99% of the time (doing whatever job it was designed to do). Thus, during any arbitrary 100 hour period, the system would be expected to be able to provide services for about 99 of those hours. The realm of systems which are generally considered "high availability systems" starts at an availability of about 99.99%, and goes up from there. Table 1 shows various levels of availability, and what they mean in terms of how much time during the course of a full year the system can be out of service. Thus, if a system has an estimated availability of 99.999% (often referred to with the shortcut, "5 nines"), it can be expected to be continuously available for use, 24 hours per day, 7 days per week, with less than 5.5 minutes of "down time" per year.

Availability	Average down time per year
99%	876 Hours
99.9%	8.76 Hours
99.99%	52.6 Minutes
99.999%	5.26 Minutes
99.9999%	31.5 Seconds
99.99999%	3.15 Seconds

Table 1.

Availability ratings typically represent averaged system availability. If a system runs for 4 years with one outage that lasts 20 minutes, this would usually be considered as meeting the "5-nines" availability goal. Similarly, if a computer manufacturer has installed 10,000 systems, and during the course of a year only 100 of them experienced any outages - but these 100 were out of service for 8 hours each - that manufacturer could (and, no doubt, would) make the case that their systems had exhibited an availability of 99.999%.

Although this quantitative measurement of availability is useful for describing the desired or actual capabilities of a system, more often than not the term "high availability system" is used in a qualitative, rather than a quantitative way. This is because achieving a system availability of "5-nines" or greater requires the use of techniques which are specifically aimed at increasing

system availability. With today's technology, it simply is not practical to build system components of sufficient reliability to allow "normal" computer system designs to achieve 99.999% or greater availability.

To appreciate why this is the case, consider the hardware and software which makes up a typical computer system. Even with the very high reliability of modern solid state electronics, complex computer circuits, power supplies, cooling fans, etc., generally have a predicted mean time between failures (MTBF) of 50,000 to 100,000 hours, or approximately 10 to 20 years.

Software is potentially an even greater source of system failures, although software does not have a true MTBF which can be estimated like hardware failure rates can. If software is "correct" there is no reason for it to ever fail. There are, in fact, many systems - most of them relatively small - which do not ever experience software failures. However, most complex software systems do contain programming errors. Real time systems, in particular, are likely to contain errors which exhibit themselves only when certain events occur at precisely unlucky times and/or in a precise sequence. If the conditions which the software does not handle correctly occur at some random interval over time, these software errors will result in an external behavior which is very similar to a software MTBF.

In considering the MTBF of a complete computer system, the failure rates of all components, potentially including the software, must be combined. A system with five components which each have an MTBF of 50,000 hours would have a system MTBF of just 10,000 hours. Moreover, even if the complete computer system has an MTBF of 50,000 hours, if nothing is done to prepare for that rare failure, the system will be out of service for some time once it does eventually fail. If the mean time to repair (MTTR) of this quite reliable computer is just eight hours, its system availability will be just 99.98% (on the average, the system would be expected to be in service only 49,992 out of every 50,000 hours, or 99.98% of the time).

Because it is essentially impossible to achieve "High Availability" by building hardware and software that does not fail, high availability systems are designed so that they are able to provide service after a failure has occurred, either without any interruption of service, or, at worst, with only a very brief service interruption. This is accomplished by building a system which contains

redundancy and then managing that redundancy effectively.

Confusion about what might or might not be a "high availability system" ensues because there are many different ways to provide and manage redundancy within a system. Some systems are designed so that there is absolutely no single point of failure. That is, no matter what (single) thing goes wrong, the computer system will be able to continue to provide service without pause. Other systems may provide for redundancy of some components, but not others. Some concentrate on hardware redundancy; some consider that fast recovery from software failures the most important goal. Some provide bare minimum management capabilities; some rely primarily on highly sophisticated fault management software to maintain service.

Beneath this variety, however, there is a core commonality. Through the use of one key abstract concept, the redundancy and management capabilities of any high availability system can be described, analyzed, and evaluated. Similarly, a solid grasp of this concept is key to the design of high availability systems in the first place.

THE KEY CONCEPT: FAULT DOMAINS

This concept, central to designing or analyzing any high availability system, is the Fault Domain. Put simply, a fault domain defines the extent of the impact in a computer system of a single hardware or software fault. If a fault in a particular hardware or software component would result in the entire system failing - an interruption in service to the users of the system - then the system consists of a single fault domain (at least relative to a fault in that component) and that hardware or software component can be described as a single point of failure for the system. High availability systems are designed so that any fault - or at least the vast majority of faults - will impact less than the entire system, so that service can be continued. Thus, the part of the system which will be impacted by a particular fault is a fault domain.

Fault domains are important to high availability systems, because they define the unit of redundancy required in any particular system. Since a single event in a fault domain can cause the outage of an entire set of components, high availability systems provide a strategy for continuing to provide service while any entire fault domain is non-functional. Therefore, any high availability system can be described as a collection of fault domains, with redundancy among the domains, designed so that if any one domain fails, operations can continue using a redundant domain which is still functional.

For this abstract concept to be useful, it must be possible to locate all the fault domains in a high availability system. Since a fault domain can be defined for each and every possible fault which may occur, this has the potential of being an extremely tedious task. However, that is not generally the case. Rather, most high availability systems are designed with a relatively straight-forward set of fault domains, using the simpli-

fying assumption that any fault within certain sets of components will cause the failure of the entire set, thus effectively equating the fault domains for all faults within that set of components. Often, systems will enforce this assumption by taking a management action to force the entire fault domain to shut down if any fault is detected within the set of components which comprise a fault domain.

High availability system architectures are created by defining and combining fault domains in such a way that the system can continue to operate even when any particular fault domain is out of service. A wide variety of fault domain configurations are possible. Roughly speaking, high availability system architectures fall on a spectrum based on the granularity and complexity of the fault domain model. At the two ends of this spectrum are:

- a. Clustering. A fault domain consists of an entire computer, complete with CPU, memory, I/O controllers, I/O devices, power conversion and distribution systems, cooling systems, etc. Multiples of these computers (often called "nodes") are then used as redundant fault domains.
- b. Hardware Fault Tolerance. A single computer is made up of multiple, redundant fault domains. The hardware design is such that the computer continues to provide full service even if any of its constituent fault domains fails.

Today, many high availability computer systems fall between those end points. These systems contain some fault domains which look and operate much like nodes in a clustering system, but have other fault domains which are managed in a fault-tolerant mode. An example of such a system is the RadiSys CP80 system shown in Figure 1, which contains two processing units, complete with CPU, I/O controllers, and backplanes housed in a single enclosure with common power supplies and fans, and connected to a single, shared RAID disk subsystem.



Figure 1. The RadiSys CPP80 High Availability System.

INFRASTRUCTURE

Regardless of the particular system architecture, though, all high availability systems somehow include multiple fault domains, with redundancy among those fault domains, and at least certain minimal capabilities for managing those fault domains. The remainder of this article will explore common characteristics of high availability system platforms by examining fault domain redundancy and management.

REDUNDANCY

The most fundamental common characteristic of high availability systems is the presence of redundant fault domains. As an example, in clustered systems, fault domains are the complete computer nodes. Redundancy is provided by including at least enough nodes to support the minimum required performance of the system even when any one of the nodes out of service.

For systems which contain fault domains which are smaller than a complete computer node, each fault domain must be made redundant - at least in an "N+1" mode, so that whatever services a particular fault domain provides to the system, those services will still be provided when that fault domain is out of service. Typical subsystems which make up fault domains, and thus are provisioned redundantly include -

- Processing Subsystems
- I/O Controllers
- Mass Storage Subsystems
- Peripheral Devices
- Internal communication paths
- Power Supplies
- Cooling Modules

Identifying fault domains and analyzing their redundancy are the primary techniques for assessing the

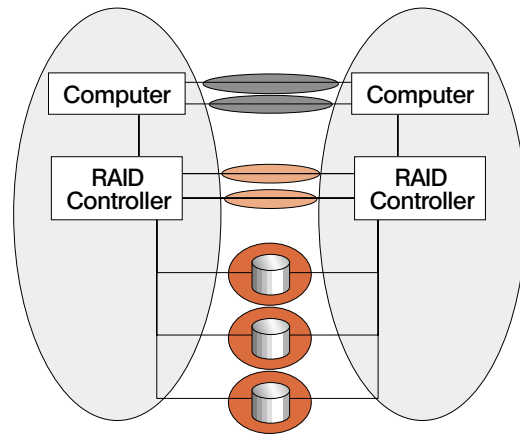


Figure 2. A high availability system with fault domains indicated.

level of availability that a particular high availability system can obtain. Each fault domain will have an associated MTBF and MTTR which can be estimated using empirical or theoretical methods. Based on these estimates, a calculation can predict how likely it is that multiple failures will occur such that the system is no longer able to provide a critical service.

For example, consider a system with three power supplies in an "N+1" redundant configuration, meaning that two power supplies are sufficient to power the entire system, and any one of the three can thus fail without bringing the system down. Each power supply is an individual fault domain, because faults within the supply do not cause failure of anything beyond the power supply itself. If each supply has an MTBF of 20,000 hours, and an MTTR of 24 hours (the time it will

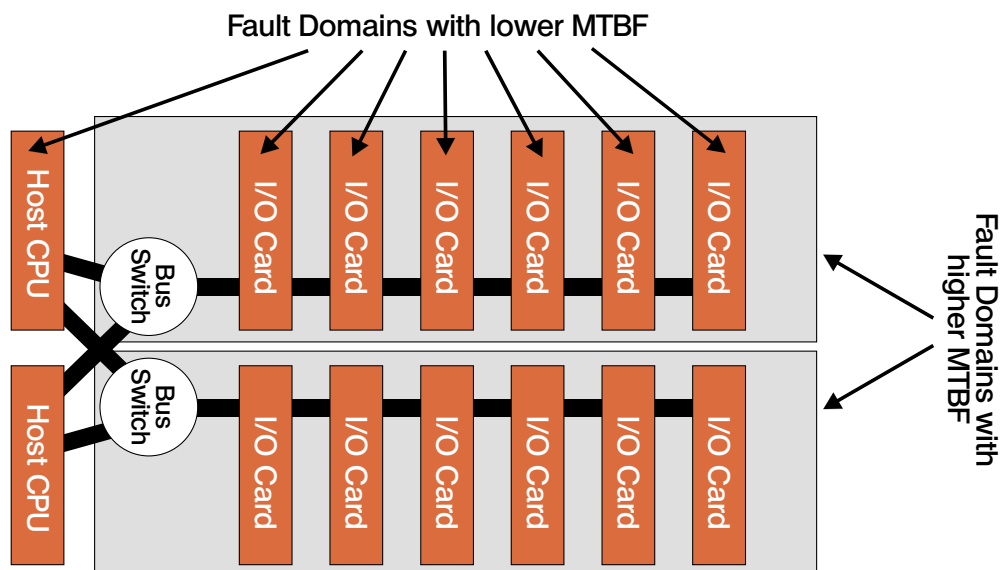


Figure 3. Nested fault domains.

AD VMIC

take to dispatch someone to the site to replace the failed supply), then a single supply will have an availability of 99.88%. Using the same data, though, the probability that a second supply will fail during the time it takes to repair a first failure can be calculated. Then, using this probability, the availability of the "power service" provided by the three supplies in an "N+1" configuration can be estimated. In this example, the expected availability can be shown to be over 99.9995%.

The assumptions which go into these sorts of calculations can be significant. One of the main assumptions often made is that component failures are random events. If, instead, components tend to wear out (e.g., because they contain moving parts, like fans), then the probability of failures will increase over time, which will make it more likely that "double failures" will occur later in the product's life, unless preventative maintenance is done. Another possibility is that parts in surviving fault domains are put under more stress once another fault domain fails. This could mean that a component is more likely to fail just when it is most critically needed. These sorts of special circumstances need to be carefully considered when estimating overall system availability.

With or without these complicating factors, though, high availability systems can always be described as a finite set of redundant fault domains, and once so modeled, the availability of the overall system can be estimated. Figure 2 shows a relatively simple, but common architecture, consisting of two complete computer systems, redundant communication paths between the two systems, and a fault-tolerant RAID subsystem driving three disk drives. In this system, there are nine fault domains which are indicated in the diagram by the colored ovals. The fault domains of the same color form redundant sets. The reason each computer system along with one of the two RAID controllers is in a single fault domain is because there is a single dedicated path between each computer and its associated RAID controller. Thus, if either the computer, the RAID controller, or the communication path between them fail, all three components would be out of service.

More complex configurations may contain "nested" fault domains. For example, if there is a non-redundant communication path between a set of fault domains (and that communication path is critical to those fault domains providing required services), then that set of fault domains plus the communication path becomes a larger fault domain. Figure 3 shows an example of this arrangement. Such a system architecture may make sense if the MTBF of the communication path (an I/O bus in figure 3) is much longer than the MTBF of the nested fault domains (the I/O controllers in figure 1), or if the MTTR is much shorter.

In contrast, Figure 4 shows a similar system where the bussed interconnects are replaced by a redundant switched network interconnect, of the sort promised by Infiniband or RapidIO, as well as PCI/switched-fabric bridges being developed by PLX and Starbridge. In this system, each of the switched networks are a separate fault domain, but, because each I/O controller and host

processor have redundant paths to every point, the fault domains remain separate from each other rather than nesting.

MANAGEMENT

Beyond simply having redundant fault domains, high availability systems must also contain the mechanisms necessary to enable continued (or quickly restored) operations when a fault domain suddenly fails. This implies an ability to monitor and control the hardware components which constitute the fault domains in the system. Because of this requirement, high availability systems often contain sophisticated "platform management" capabilities. Just as a description of a system in terms of its fault domains identifies the key pieces of redundancy, it will also indicate the specific minimum requirements for platform management in the system.

When a fault occurs in a high availability computer system, there are five operations which need to take place as a result:

1. **Detection.** Somehow the fact that a fault exists must be detected by the system.
2. **Diagnosis.** If it is not obvious from the method of detection, the system must determine which fault domain contains the faulted component.
3. **Isolation.** If the fault domains are not already electrically and logically isolated from each other, the domain with the fault needs to be isolated from the rest of the system so that the fault is "contained" within a single fault domain.
4. **Recovery.** The system needs to adjust itself to continue (or quickly restore) operations without the services of the failed fault domain.
5. **Repair.** The failed component - or perhaps the entire failed fault domain - needs to be repaired or replaced while the rest of the system continues operation. Then the system needs to adjust itself again to the new situation of having the failed fault domain once again able to provide service.

The minimum requirements for platform management in a high availability system would be whatever is needed to enable these operations to occur, at least at the granularity of fault domains. Thus, in a high availability system, it should be possible to describe the mechanism for fault detection, diagnosis, isolation, recovery, and repair for each and every fault domain. In many cases, some of these steps may be trivial, but in other cases, they may be significant.

Each of these classes of management requirements are considered individually below.

Fault Domain Failure Detection

However fault domains are constructed, there must be some way of detecting faults anywhere in the system. This may be communicated "out of band" through a platform management system or "in band" via unambiguous observable behavior (or non-behavior). An example of the former would be monitoring the RPMs of a cooling fan, and sending an alarm to management software if it drops below a critical value. An

Each red box, the grey switched network and the black switched network are fault domains in this system.

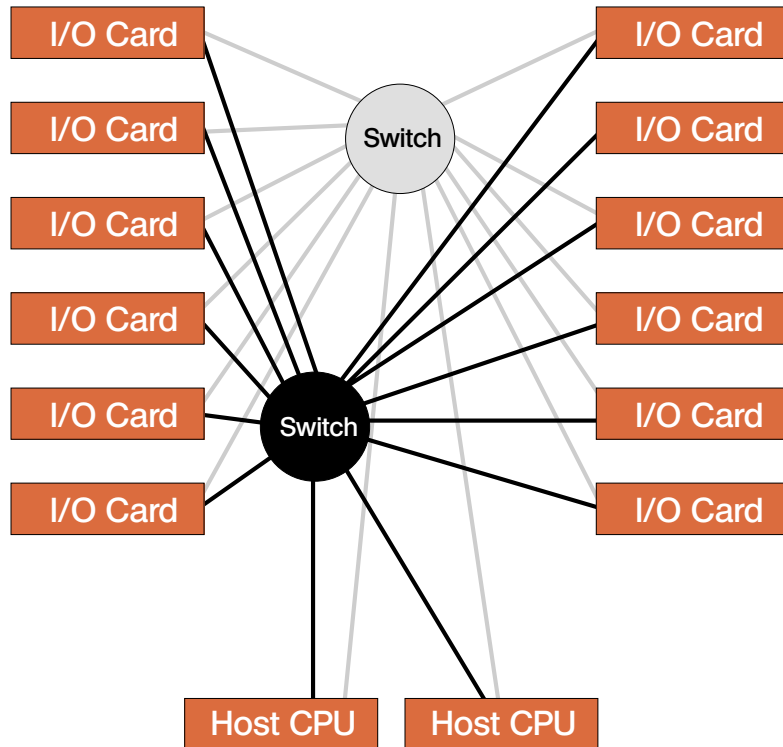


Figure 4. Redundant switched interconnects.

example of the latter is "fail safe" behavior, where a fault domain contains a self-checking capability (e.g., parity checking on a memory read) which causes it to promptly shut down when a fault is detected. The resulting shut down is then observed by other parts of the system.

Beyond the immediate communication required for fault diagnosis and isolation, fault domain failures must also be communicated to appropriate subsystems (or people) in order to trigger recovery and repair actions.

A special case of fault detection exists when a system contains fault domains which are normally in a "stand-by" mode. When subsystems are normally idle, there may be a need for special techniques to detect latent faults in these domains. If the primary failure detection mechanism is observation of normal operating behavior, the system will need to provide a separate mechanism for detection of faults in fault domains which are not operating by design (because they are in a stand-by state). Unless the system can be assured that the standby fault domains are in good shape, all plans on how the system will be able to survive from a failure of an active fault domain are ineffective.

Fault Domain Diagnosis

In cases where a failure of a fault domain is detectable,

but it is not immediately evident which of several redundant fault domains has failed, the system must support a diagnostic function which will determine where the actual failure exists. For example, consider a system with multiple fans, each of which makes up a separate fault domain, because the failure of any one fan can be compensated by increasing the speed of others. If a failure of a fan is detected indirectly (via a temperature or air-flow detector, say), additional diagnosis may be required to determine which fan had failed so that the proper isolation, recovery, and repair actions may be initiated.

Diagnosis of failures may occur through diagnosis capabilities of the hardware components themselves, or through capabilities of management software which can analyze a variety of indications and/or initiate various actions in an attempt to deduce which hardware component has failed. A diagnosis capability in the hardware itself basically means that it is able to answer the question, "Are you ok?". If the hardware cannot directly answer this question, then it must at least have the capabilities which are needed by management software for it to be able to diagnose the failure.

In a system with nested fault domains (e.g., figure 3, above), diagnosis may aid in converting a detected failure of the larger fault domain to a failure of the small-

er, nested fault domain. For example, in figure 3, if one of the I/O controllers failed in a way which caused the bus to "hang", the fault domain associated with the bus segment would fail, which is what would be detected. It would be highly desirable in this case to be able to do a diagnosis of the individual I/O controllers, determine which one is failing, and isolate it from the bus. This would result in removing the fault from the larger fault domain, and restoring the functionality of all the other I/O controllers on the bus segment.

Additional diagnosis capabilities can also be useful to effect a repair to a fault domain more quickly. For example, if an I/O controller fails, after the system isolates and recovers from the failure, it would be desirable to have the platform management system order the controller to run a self-test to determine if the failure was transient or permanent. If transient, then the I/O controller could be immediately reintegrated. If permanent, then a technician would have to be dispatched to replace the board.

Fault Domain Isolation

Critical to the design of a high availability system is the ability to isolate fault domains from the system. Isolation means taking whatever action is needed to prevent a fault from affecting other fault domains.

Often this capability is an integral part of the design of a hardware component. For example, power supplies include circuits which monitor output voltages being produced, and quickly shut the supply off if voltages venture out of specifications. In other cases, another part of the system may need to initiate an action to isolate a fault domain, such as removing power from an I/O controller which is attached to a common I/O bus. Even when isolation is automatic by design, it can be highly desirable, as a further safety feature, for any fault domain to be able to isolate itself from the rest of the system upon request from a platform management system.

Because of the importance of domain isolation, and because of the difficulty in guaranteeing the behavior of a fault domain which is already misbehaving, there may be multiple levels of fault isolation capability in systems. For example, if a host processor is not responding, it may be ordered to execute a system reset operation. If this still does not clear the problem, it may be ordered to power itself off. Similarly, if a particular I/O controller has failed in a system, it may be ordered to isolate itself from the I/O bus. If this does not work, a second level of isolation may be to isolate a slot on the backplane, or even an entire I/O bus segment (at the point of a PCI to PCI bridge, for example).

Fault Domain Failure Recovery

"Fault Domain Failure Recovery" means having the overall system recover from the failure of a fault domain; that is, to continue to provide its required service, while no longer using the services of the failed fault domain. The actions required for recovery are highly dependent on the specific architecture of the system fault domains and the characteristics of the services provided by particular fault domains. In many

cases, when fault domains operate in an "active/active" mode, there may be no specific actions required - the system just continues to operate but with less excess capacity. In other cases, system reconfiguration actions may be required to put the system back into operational shape without the failed fault domain.

Examples of the sorts of hardware capabilities which may be needed are:

1. Reprogramming a spare Ethernet controller MAC address so it can assume the role of a failed controller,
2. Reprogramming a switched fabric routing network to bypass failed nodes,
3. Switching a PCI to PCI bridge from non-transparent to transparent mode so that a slave processor card can become a system controller.
4. Selectively enabling or disabling hardware components
5. Triggering a hardware condition which in turn triggers an automatic failover to a functioning fault domain.
6. Reassigning a boot device and forcing a reboot to load a specific configuration.

Fault Domain Repair

Often one of the most complex features of high availability systems is the need to repair failed fault domains while the system continues to operate. The specific capabilities required in the hardware to support this are dependent on the design of the fault domains. Because of the complexity, it is not unusual for the requirement for on-line repair of fault domains to be a major driver in the system architecture, and the identification of fault domains in the first place.

The capabilities needed to support on-line repair have several dimensions. These include

- a. the basic ability to hot-swap fault domains
- b. notification to other parts of the system (e.g., the operating system) that a system configuration has changed,
- c. guidance to service personnel to help insure correct repair actions
- d. on-line firmware/software upgradability
- e. maintenance of a system inventory

A few comments on each of these follow.

a) Hot-swap

There probably still are systems which are "mission critical," but which can tolerate planned down-time for repair actions. More and more, though, "mission critical" implies a need for service 24 hours per day, 365 days per year. Consider something as simple as a corporate email system. Many businesses today effectively shut down if their email system is down. And whereas it once might have been acceptable to do maintenance on the system at night, globalization of the economy has effectively eliminated "night" for many companies.

To support repair actions while the system remains

AD IMAGE MEDIA

active, the system must allow for the physical removal and replacement of a fault domain while the redundant fault domains which are providing service remain active. Furthermore, the hardware needs to be constructed to make this operation as failsafe as possible. Even if repairs are carried out by trained technicians, something as simple as dropping a screw can cause a system failure if the system is not designed with the thought of on-line repairs from the beginning. Generally, high availability systems are designed to make fault-domain removal and replacement a very simple and safe operation.

b) Notification of configuration change

When a fault domain is repaired, the platform configuration changes. At the least, a new resource is made available which the operating system and application program can begin using. This fact needs to be communicated to the operating system and/or application so that they can begin using the newly repaired fault domain.

The minimum requirement in a high availability system is to be able to cope with removal and replacement of fault domains within a static configuration. However, it is highly desirable to be able to modify the system configuration more dynamically, adding additional hardware, upgrading hardware, etc. While this is not required for failure avoidance, it allows some amount of system upgrade to be accomplished without having to schedule system outages.

c) Guidance to service personnel to help insure correct repair actions

One of the most common causes of failures of computer systems is erroneous operator actions. Performing on-line repair actions are certainly opportunities for errors. Therefore, a critical capability of high availability systems is to guide repair actions to help prevent errors. This includes having common, standardized presentations of system alarms, designing the system to permit very simple repair procedures, and intuitive guidance through the repair itself.

Since the physical repair of a hardware domain will involve direct hands-on interaction between a system technician and the actual system hardware, having visual guidance for the repair action directly on the hardware itself is highly desirable. This often takes the form of LEDs and/or other small display devices which can be controlled through the platform management system.

d) On-line firmware/software upgradability

Increasingly, hardware components contain field-programmable devices. This affords an opportunity to perform repairs and upgrades of the hardware without having to physically remove and replace components from the system. This can result in significant reductions in MTTRs, as well as eliminating opportunities for errors being made during a physical hardware swap. Thus, when programmable devices are used in a system design, having the capability to upgrade the firmware "in place" is a desirable feature.

e) Maintenance of a system inventory

In any system where hardware components and con-

figuration can change over time, it is important to be able to answer the question, "What is currently installed?". This includes a "discovery" capability which can detect all installed hardware. Individually replaceable units should be able to report, at a minimum, what they are (including revision level), what options they contain (if applicable), and unique tracking numbers. Typically, this information will be made available to the platform management system.

Additional Useful Hardware Capabilities

The above fault management capabilities have been aimed at the minimum required to support fault detection, diagnosis, isolation, recovery, and repair of fault domains. Beyond this minimum, additional fault management capabilities of the hardware in high availability systems can be provided in order to predict and prevent faults from occurring in the first place.

Typically, these will involve monitoring analog values which can reflect on the health of the hardware even when a fault has not occurred. For example, a fan may be slowing down, but still functioning within specifications. This may be indicating a bearing wearing out, and with this warning, the fan can be replaced before a fault occurs. Another example - monitoring a temperature may indicate an impending problem before any component has actually failed, triggering a response to bring the temperature back into a safer range before a fault occurs.

Platform Management Capabilities to support Open Architecture

As has been described above, there are a large number of platform management monitoring and control capabilities which are useful in a high availability system, but the specific capabilities which may be required will vary significantly from one system architecture to another.

To support this variety of requirements in an open standard way, the platform management system itself needs to have certain capabilities. Among these are:

1. An industry recognized interface to the operating system and management middleware.
2. The ability to be self-defining: The platform management system should be able to identify what capabilities it has so that operating systems and management software can adapt itself to the specifics of a particular platform.
3. Use of an industry recognized management bus for intra-system communication: This allows for the interoperability of managed hardware components from various vendors with the overall platform management infrastructure.

An example of platform management which does an excellent job of meeting these requirements is the Intelligent Platform Management Interface specification, which is now also part of the CompactPCI standards.

SUMMARY

High availability systems can be complex. Similarly, many different system architectures can be used to achieve high availability. All of these systems have certain characteristics in common, though. In any high availability system, fault domains can be identified, and there will be redundancy among them. It should also be possible, for each and every fault domain in a high availability system to describe the mechanisms in place which will provide for fault detection, diagnosis, isolation, recovery, and repair.

This technique, along with knowledge of the reliability of each fault domain, can then allow system designers to predict the overall system availability which can be expected from a particular system, and will guide the overall design of the system ■

David McKinley is Director of Technology for the Computer Platforms Division of the RadiSys Corporation. In this position, he is responsible for development of specific high-availability system technologies, as well as other technology initiatives for RadiSys telecommunications and embedded computer systems. McKinley has over 25 years experience in developing computing systems and a long history of working with high availability and fault-tolerant systems. Prior to joining RadiSys in 1997, he was with Tandem Computers where he was responsible for managing communications and systems management software development for the Tandem Integrity family of fault-tolerant computer systems. McKinley possesses technical expertise in data communications, networking, systems management, and high-availability technology. He has been responsible for the architecture and design of high-throughput message switching systems, intelligent communication controllers for fault-tolerant computer systems, systems management software, and CompactPCI telecommunications systems. A native Texan, McKinley received a BS degree in Computing Science from Texas A&M University, and an MBA from the University of Texas.



Dedicated Systems
Encyclopaedia

**RTOS BUYER GUIDE ONLINE
&
CALENDAR OF EVENTS**

at the Dedicated Systems Encyclopaedia web site:

<http://www.dedicated-systems.com>