

Highly Available Embedded Computer Platforms Become Reality

Motorola Computer Group

Jon Kenton – Telecom Marketing Manager

Paul Virgo - Product Marketing Manager

Definition

Put simply, a highly available system is usable when the customer needs it. A system can be highly available while operating from 8 am to 5 pm, if that is all the business demands. The remaining time can be used for scheduled maintenance and repair. Availability is defined as actual service/required service. The challenge for many of today's systems is to operate 24 hours a day, 365 days a year (sometimes referred to as 7x24 or 365x24).

Overview

The pervasiveness of computers in everyday life is increasing the demand for more highly available systems. Deregulation in the telecommunications industry, the Internet, and new entertainment services such as video on demand are creating opportunities for new computing platforms. The challenge in these industries is to provide reliable services at competitive costs. The architectures for the next-generation computing platforms must be as open and flexible as today's desktop computers, highly reliable, and a fraction of the cost of traditional fault-tolerant computers. This paper examines multiple fault-management processes and high-availability models and concludes by presenting a CompactPCI[®] system for both architectures.

Topics

1. Fault Tolerance versus High Availability
2. Redundant Configurations
3. Fault Management Process
4. High Availability and Restart Models
5. Fault Management and Software for Redundant Systems
6. A CompactPCI[®] System for Both Architectures
7. Conclusion

1. Fault Tolerance versus High Availability

Fault tolerance and high availability are not quite the same thing. Fault tolerance encompasses two properties: transactional reliability and high system availability. Many systems today use fault-tolerant computers when the application only needs availability. Transactional reliability is needed in banking applications or in billing records for a telecommunications network.

Availability is needed in file server applications or in call-processing applications. Transactional reliability is not needed for any application where some loss of data is tolerable or where data transfer is protected by a reliable end-to-end protocol such as transmission control protocol (TCP)/Internet protocol (IP).

Traditional implementations of fault-tolerant platforms often involve proprietary hardware and software. This causes higher costs and longer design cycles—two things that may not be acceptable in emerging, competitive markets such as telecommunications. The challenge is to provide highly available platforms without resorting to extreme or expensive measures.

Availability is often expressed in percentages. A 365x24 system with 99.9 percent availability has an average down time of 8.76 hours per year (525 minutes). A system with just 5 minutes of service outage must have 99.999 percent availability.

Availability is calculated using statistical models for all the system components. The simplest model for a component is a binary model. The component is either in or out of service. Availability can be calculated from failure rates (mean time between failures [MTBF]) and repair times (mean time to repair [MTTR]). The average down time contribution by any component is calculated by amortizing the MTTR time over the MTBF period. For example, if a component critical to the operation of the platform has an MTBF of 250,000 hours, and a MTTR of one hour, it contributes 2.1 minutes of down time to the system per year (60 minutes/250,000 hours/8,760 hours/year). A more complex model for the component, which considers partial outages, requires the use of statistical methods to calculate availability.

Availability in the two 9's or three 9's range (99 percent to 99.9 percent) can be achieved by maximizing the reliability of components and minimizing repair times. To achieve higher reliability or to compensate for less reliable components, redundancy is used. Having a backup for a component that fails keeps the system operating. Availability of redundant configurations is calculated based on the time to detect and switch over to the redundant component. Fault management becomes a critical factor in system design.

2. Redundant Configurations

Many techniques for redundancy can be employed in a system architecture. N redundant systems (2N, 3N, 5N, etc.) employ multiple identical sets of resources isolated into separate fault zones. N redundancy can be employed on a component level (e.g., disks and power supplies) or in complete systems (e.g., redundant file servers). This simplest form of this is 2N redundancy.

N+m redundancy involves having individual spares for a group of resources (e.g., a spare port on an Ethernet hub). The simplest form of this approach is N+1 redundancy. N+1 redundancy can also be applied to individual system components or to clusters of complete systems.

The differences in the redundancy schemes are subtle but crucial to system design. 2N redundant systems have a duplicate for every critical resource in the system. The standby resources are kept up-to-date with the activities of the active resource. When a failure occurs, the entire active domain is taken out of service, and the standby takes over. The advantages of 2N redundancy are simple fault management and fast switchover times. The disadvantages of 2N redundancy are the cost to duplicate every resource and difficulties with connectivity in systems with a large number of I/O connections.

An example of this situation is in telephony, where a system may employ many T1/E1 connections. Duplicating the line interface boards is expensive. So is multiplexing T1s or asking the customer to lease many spare lines. For I/O intensive applications, N+1 redundancy is much more cost-effective. With N+1 redundancy, the spare cannot be fully configured because the exact nature of its task is not known until one of the active devices fails. This complicates fault management and increases switchover times.

3. Fault Management Process

Fault management differs significantly between N redundant systems and N+1 redundant systems. A fault management cycle can be defined in phases: detection, location, isolation, recovery, reporting, repair, and reintegration. These processes are defined below in general terms and later discussed in the context of 2N and N+1 fault management. These definitions apply equally to hardware and software components of a system.

Detection

This is the process of discovering that an error exists. Detection is defined as the time from a failure causing a loss of service to the system becoming aware of it. Error detection is the responsibility of every hardware and software component in the system. To meet availability goals such as 99.999 percent, adequate error detection must be designed in, or a component may not be suitable for use in HA systems.

Location

This is the process of narrowing down the failure to the defective component. This process depends greatly on the definition of a fault zone. At this stage of the process, it is not necessary to locate an error to a region smaller than what will be isolated.

Isolation

This takes the defective portion of the system out of service. The region that is isolated must be bounded at a point where it can be removed from all interaction with the system.

Recovery

This is the process of reassigning the necessary resources to restore the system to an operating state. Recovery also requires restoring any portions of the system that were adversely affected by the failing component. Recovery is the final step in the process that contributes to outage time. Once the system is providing complete service again, the remainder of the process does not directly contribute to outage time.

Reporting

This is process that notifies the outside world that an event has taken place; it is the first step in the repair process. The repair process is indirectly related to availability. In systems employing redundancy, there is a statistical possibility that a second failure can occur in the component covering for this failure, which would result in a complete system outage. While the probability is low, the severity is high enough to make this a factor in the availability equation. It is important, even in redundant systems, to keep repair times low.

Repair

This the replacement of the defective component and is generally designated for the operator (human)-assisted portion of the process. This phase is separated for a number of reasons: it is usually the most time-consuming portion of the process; it is also a phase in the process where mistakes can account for system outages.

Reintegration

Finally, the repaired component is reintegrated. Once the defective hardware or software component has been replaced, it is brought back into service either as a new standby component or a sharer of the system load.

These distinctions are somewhat arbitrary, but they nevertheless illustrate differences in system architectures.

Fault Management in Clusters

In a 2N (clustered) system, which has highly encapsulated fault zones, the fault management cycle is somewhat simplified:

- Detection is crucial. The time a system is malfunctioning, without being detected, is considered a direct outage. The ability of a system to detect all possible failures is measured in its fault coverage. Anything not covered is assigned a probability and factored in as a severe outage.
- Location is implied. Any fault detected is usually detected within the node itself (assuming good fault coverage), and so the location is known.
- Isolation and recovery are essentially the same step. All the activity is moved off of the failing node to its standby, thus isolating the defective node and recovering the system at the same time.
- Clustered systems essentially move from detection to recovery. Location and isolation are inherent in the architecture of the system. The complexity of the recovery process is always dependent on the specific application running on the system.
- The remainder of the process proceeds off-line. Here too, the repair process can be as simple as replacing the entire node, or the technician may choose to further locate the failed component within the node. This diagnostic is done in an off-line system with full resources available to the repair process. A node-based system is relatively immune to mistakes made during this process.

Finely Grained Fault Management

In systems where node-based (2N) configurations are not economical, devices are spared on an N+1 arrangement. This requires a more finely grained fault management. The process becomes more complicated:

- Fault detection is always the same; it must be done by every resource in the system as quickly as possible.
- Location is done with an on-line diagnostic. This diagnostic should not interfere with the operating portion of the system. The goal is to keep as much of the system operating as possible to avoid total outages. Locating a failure in an active system is complicated; failures cannot always be precisely located. A typical metric for on-line fault identification is 95 percent fault location accuracy.

- Isolation is critical. To spare on more finely grained boundaries, the system must have an infrastructure that permits isolation of individual field replaceable units (FRUs). In an N+1 system, failed components must exist benignly while system activity continues.
- Recovery is more complicated for two reasons. With nodes, a clean boundary can be placed around a node that hides the complexities of the system. In an N+1 system, there is a hierarchy of component dependencies. When a component is determined to be bad, any other system components depending upon this resource must be recovered as well. This aspect of topology management becomes part of the critical path to restoring service.

Figure 1. Example of a Dependency Tree

Another difficulty of fault recovery in this type of system is a result of incomplete encapsulation of faults. A failing device may push other devices into error states. After a defective device has been identified, all other error conditions must be cleaned up.

Reporting is application-dependent, but with more finely grained FRUs, a more detailed method of indication is needed. Reporting becomes crucial to the repair process. Because an operator or technician is repairing a portion of a live system, it is essential to identify properly the specific FRU to be replaced. Operator errors account for a significant loss of service in these types of systems. Clean design of a reporting mechanism can minimize these mistakes.

Repair, in a live system, requires some form of hot replacement. Again, a system must be designed to support this activity. Another consideration for repair of an active system is to insure that FRUs do not mechanically interact. It is not efficient to be forced to remove one component to get to another.

The final phase of reintegration is again slightly more complicated because it is taking place in a live system. The fault management cycle in N+1 systems requires much more processing from the central processing unit (CPU) responsible for managing the system. If that CPU is also involved in system activity, there must be reserve processing capacity for fault management.

4. High Availability and Restart Models

Complexity of a highly available system is also dependent on the restart model. The strategies employed in the fault management process vary if the system is using a hot restart, a warm restart, or a cold restart model. The restart model is affected by the amount of information available to the system at the time of an event. The more information available, the faster the restart is.

A hot restart system has the fastest recovery time but is the most complex to implement. In a hot restart model, the application saves state information about the current activity of the system. That information is given to the standby component so it is ready to take over quickly. The application must be designed to restart using this state information.

A hot restart system also requires that a standby component is designated prior to a fault management event. In clustered systems (2N), this is straightforward, as there is a one-to-one correspondence between components and their standbys. In N+1 systems, hot restart requires the standby device to save the state of multiple components. The standby must have the extra capacity to do this. Otherwise, a warm restart model must be used.

Figure 2. Hot Standby

A warm restart is similar to the hot restart model. In a warm restart model, the applications save state information about the current activity of the system, and the standby component is not designated until the fault management cycle is in progress. Then the standby component is configured with the necessary application and state information. This adds time to the restart process but can reduce costs associated with the standby components (see *Figure 3*). Warm restart is also easier to implement in systems where the standby devices are not identical to the active devices.

Figure 3. Warm Restart in N+1

A cold restart is the least complex to implement but requires the most time. A cold restart implies the starting place for the standby element is its initialization point. A cold restart is used when no information is available about the state of the failing component. The last known state is therefore initialization.

A cold restart system can be implemented with little or no changes to the applications of a system. The high-availability-specific software components can be relegated to operating system software and services. The price for this simplicity is that restart times are much longer, and current activity in the system may be lost.

The times associated with the different restart models vary depending on the implementation of the system and application software. In relative terms, if the hot restart model is 1X, the warm restart model can be 2 to 3X, and the cold restart model is approximately 10X to 100X. Restart times will be the lowest in systems where the system software and applications are designed to support high availability.

5. Fault Management Software for Redundant Systems

There are some basic concepts that apply to all forms of redundant systems doing fault management. Again, the complexity of this software is affected by the architecture and the restart model.

Distributed Data Environment (DDE)

This is used to create a virtual environment for the applications and drivers. A DDE is the primary mechanism to support failover of hardware or software components. If a resource fails, the DDE can redirect traffic to the standby component without the application knowing about it.

Topology management is a resource that keeps track of system configuration. Part of the role of topology management is to identify where the redundant resources reside. In 2N systems, topology management can be as simple as keeping track of which system is active and which is standby. In larger clusters of systems, topology management may be relegated to a specific node in the cluster.

In N+1 systems, topology management must keep track of all the dependencies in the system. Topology management becomes the responsibility of each system and can even be needed in individual subsystems in a larger system. Because the fault-management process requires extensive and immediate access to the system topology, topology management is a key component.

While it is possible to have dissimilar spare devices in either type of system, it is more prevalent in N+1 systems. Topology management is more complex in N+1 systems and is more extensible

to heterogeneous configurations. Systems taking advantage of the simplicity of a 2N topology usually only support identical spares.

Event Management

This is the focal point for handling exceptions. The event manager receives messages from all portions of the system and performs the necessary system management functions. These operations may be part of normal system operation in the case of a system upgrade or a fault management cycle in the case of a system exception.

In systems with a fault management cycle as simple as turning over control to its standby, event management might not even be identified as a separate process. In systems with complex fault management, this service can be as sophisticated as a rules-based interpreter that is highly customizable.

Application Management

In addition to these core services, there are other services needed in more complex forms of fault management. One of these services is application management. The application manager provides the system with a means to monitor the health of applications and signal exceptions. By making this a system service, applications can be written to a standard application programming interface (API) and not have to deal with the underlying fault management infrastructure.

Checkpoint Service

This provides a channel(s) to save state data between active and standby components. The checkpoint service can abstract the interaction between a component and its standby. The application does not need to know about the system architecture or the restart model. In a warm restart implementation, the checkpoint service saves the state information with the system until a standby is needed. In a hot restart implementation, the data is sent immediately to the standby.

Checkpointing can be performed in several ways. In active checkpointing, the application notifies the service when data is ready to be copied. In passive checkpointing, the application registers a data structure with the service, and the service asynchronously collects the information at specific intervals. Either technique can be useful, depending upon the application.

Heartbeat Protocol

This is used as a basic fault-detection mechanism. Components in a highly available system use the heartbeat protocol to signal that they are still functioning. If a component fails to check in at the appropriate interval, corrective action can be taken. The heartbeat protocol can be tied in with the checkpoint service. The transport of state data can be synchronized with an application's heartbeat.

On-Line and Off-Line Diagnostics

These play an important role in a highly available system. A diagnostic manager schedules independent activities to check the health of the system. Any activity that improves fault detection in a system improves availability. A crucial responsibility of this activity is latent fault detection. It is important to test portions of a system that are unused so that they are available when needed.

System Management

Finally, any system is usually part of a larger system. An interface for system management such as signaling network management protocol (SNMP) or common management information protocol (CMIP) is essential in any HA system. This service ties into the topology management of

the system to provide status to the network agent and to allow remote access to control the system configuration.

Figure 4. High-Availability Software Services

A well-designed set of system services can alleviate much of the work an application must do to be highly available. By providing a stable platform for system implementation, high degrees of availability can be achieved. High availability requires careful design at all levels of system implementation.

6. A CompactPCI[®] System for Both Architectures

Compact peripheral component interconnect (CompactPCI) technology is an ideal board interconnect base technology on which to build high-availability systems. Its modular construction and features meet many of the requirements for effective fault management discussed earlier in the tutorial. CompactPCI has a modular board structure that allows for simple module identification with few opportunities for human error. Hot-swap extensions to the CompactPCI specification enable board-level FRU replacement in a running system; the high-availability extensions to hot-swap allow unattended isolation of individual boards. The mechanical structure and large number of connector pins specified for CompactPCI mean that most input/output (I/O) connections can be routed through the CompactPCI backplane, minimizing the need for front panel cables and possible mechanical or cable-related interaction between FRUs.

7. Conclusion

It is clear that availability will be an important aspect of many future system designs. It will be necessary for system providers to understand the costs and complexities of higher availability and tailor those features to the market needs. Expertise in producing highly available systems will have to become commonplace for companies who wish to compete in the open telecommunications market. The system providers that can offer the benefits of high availability at competitive prices will have a big advantage in the marketplace.