Top Twelve Guidelines for Best-in-Class Service Availability in Communication Services and Applications
Introduction

The rapid growth of mobile broadband services, proliferation of smartphones, and demand for real-time applications has generated increasing interest in the availability of network services. The growing number of users for these services has put a stark spotlight on system failures. These emerging communication services span the IT and wireless domains, and put more emphasis on system availability as a “must have” requirement. Uptime and availability have evolved rapidly in the last several years from a purely technical requirement into a mainstream business and application requirement. Service providers seeking to differentiate themselves now understand that service availability is a crucial component for customer experience management and overall business success.

Developing these robust, highly available systems is a complex undertaking, and the architectural design and implementation tradeoffs are important to understand early in the lifecycle of a carrier-grade system. Managing risks is at the crux of every development program, and implementing service availability the right way, the first time, drives down development, operation, and maintenance costs for the entire lifecycle of the product. Network Equipment Providers within the telecom equipment sector have known this for decades, and many of them have hundreds of thousands of lines of proprietary software written to ensure service availability (SA) of both the hardware and software components that make up a system. This historical approach will need to be significantly changed in order to adapt to the accelerating industry trends described above. Communications Service Providers (CSPs) are offering new services that require service availability, and Network Equipment Providers who can provide this increased level of capability to their CSP customers will be in a position to lead this growing space.

So, what should you do if you are part of a project team building a new system that requires service availability? Or what if you are involved in the next planned release cycle of an existing or legacy system that has new, extended, or more rigorous availability requirements? In this whitepaper, we will identify the key considerations and guidelines related to availability solutions, and explain why each guideline is crucial in the development of applications and systems that require service availability.

Guideline 1: Consider the Complexity of Service Availability

One of the most important considerations when pondering SA-based projects is to be sure to take full account of the difficulty, as well as time and cost, of building service availability (SA) into your project.
“How difficult could it be?” says the engineer? “I can write that code in a couple of days—a week at most.” Six months later, the SA code is still not working, and the project schedule is in danger of slipping. SA middleware consists of fundamental infrastructure services designed for distributed telecom systems, all of which must fit together and work flawlessly in order to achieve the highest levels of availability.

Support for Stateful Application Failover with Session Integrity

Among the various complexities of Service Availability, none is more challenging and critical than the need to execute stateful failover of applications in the event of a system failure. Critical real-time applications like IPTV need to failover seamlessly and maintain session integrity in order to ensure an uninterrupted customer experience. Doing this requires a comprehensive fault management cycle and suite of supporting capabilities and features.

The Fault Management Cycle

The full suite of SA capabilities outlined below is typically integrated to deliver a complete Fault Management Cycle in a particular system. As depicted in Figure 1, when a fault in a system is detected, the SA middleware will isolate the issue in the hardware and/or software as the first step in the cycle. Then, based on the policies that are defined in the SA software, specific recovery and repair actions are undertaken to bring the overall system back to a state of health. This type of end-to-end cycle is a key requirement for any system requiring service availability.

![Figure 1. The Fault Management Cycle](image)

Supporting this complete end to end methodology requires a robust set of availability functionality. Below is a comprehensive list of the functionality that must be implemented in a successful service availability solution:
Modeling of Software, Hardware Redundancy, Availability, and Recovery Policies

- Multiple redundancy models
- Association of hardware and software components within chosen redundancy models
- Definition of fault management and recovery policies

Availability Management

- Registration and de-registration of hardware and software components
- Monitoring resource health on a steady-state basis
- Detecting faults and managing the complete fault management cycle to repair
- Dynamic failover and switchover at all hardware and software levels
- Error reporting

Centralized Administrative Services that have Visibility to all Cluster Resources

- Dynamic middleware and application configuration
- Administrative operations
- Cluster wide log services
- Cluster wide alarm services

Platform Management

- Automated hardware discovery and inventory
- Hardware resource capability information
- Failure detection and recovery
- Performance degradation and recovery policies
- Hot swap state management
- Asynchronous event and error reporting
Clustering
- Cluster membership: adding, removing and enumerating node membership
- Lock, unlock and shut down a cluster node
- Lock, unlock and shut down a cluster

Checkpoint Services
- Saving application state to preserve session integrity
- Checkpoint replicas
- Checkpoint data access: reads, writes, updates, deletes
- The ability to balance checkpoint frequency vs. performance
- Synchronous updates
- Asynchronous updates

Events/Messages
- Guaranteed delivery
- Event/message priorities
- Event/message ordering
- Retention time
- Persistence
- Message queues
- Load distribution

Weaving all this functionality into an integrated system that will meet the stringent demands of up to seven 9s of availability is challenging enough for dedicated experts. Even experienced SA professionals find it difficult to implement this range of capabilities the right way without some level of trial and error. Suffice it to say, service availability is a difficult and complex problem. The schedule delays and cost overruns associated with trial and error are simply not acceptable for projects that need to make it to market quickly in the highly competitive network equipment market.
Guideline 2: Architect SA into the System at the Beginning of a Product Lifecycle

Architecting service availability into a system at the earliest possible point in the program lifecycle is crucial. In some projects, the initial system may not seem to have requirements for availability. Early phase functionality and features often dominate the dialogue, and this can distract teams from fully considering clearly stated and crucial requirements slated for later phases. A system’s architecture may be insufficient to support long term requirements. Service availability features are not trivial to add after the fact.

If Service Availability is a future requirement in the product’s lifecycle, it is always best to consider it at the beginning, and architect the system accordingly. This is even true when early releases of a system are not required to support service availability. Those who use a system often have an implicit expectation that the equipment will eventually deliver on the promise of SA, though they do not focus on it initially.

So why architect availability into the initial solution if it is not necessary for the first release? The answer is that service availability is always more expensive and difficult to fit into an architecture after the fact, compared to putting the SA infrastructure in place to begin with. In addition to lower costs, the final system is very likely to result in better availability and performance results. Ultimately, inserting infrastructure services early in the development phases reduces risk and effort in later spirals of development since most of these infrastructure services complement and contribute to service availability, and are both relevant and appropriate for non-SA systems as well. Once in place, these infrastructure services are already part of the fabric of the product implementation and mind-set of the development team.

Guideline 3: Plan for Change

Omitting flexibility and extensibility features is a difficult decision that is often made as budgets tighten and time-to-market pressures increase on a project. This is not exclusive to service availability requirements, but is a common design decision in many systems. Commercial-off-the-shelf (COTS) solutions are very flexible and scalable as they must address a wide variety of possible deployment use-cases. For example, customers often require redundancy policy changes. Perhaps the initial recovery policy for responding to a network failure was to switch over to a redundant node and re-boot the node in question, but with the introduction of a redundant network, a more strategic policy is to simply switch over to the backup network. Does such a policy change require extensive code changes, or can you simply reconfigure the desired policy changes in a system model? Fortunately, commercial service availability solutions allow such policies to be explained or changed with little effort through configuration.
Guideline 4: Focus Engineering Resources on Crucial SA Requirements

System components can be in-service or out-of-service, locked or unlocked for management, healthy or disabled, starting up or shutting down, active or standby, idle or busy. If you add up all these administrative, operational, readiness and usage states, and all their various combinations, you can soon come up with an extremely complex matrix. Most resources that make up a system only need to be aware of a fraction of all the possible combinations. Treating all combinations equally, and devoting excessive engineering resources to these obscure situations can lead to unnecessarily complex software. Determine what is important for your applications at the beginning of your project, and optimize for those scenarios, rather than spending valuable engineering time on obscure theoretical situations that seldom come to pass. Conversely, if the system is complex, and requires intricate modeling and management policies, a program-specific solution will be too costly and add significant schedule risk. A good availability solution can be configured to be aware of, and can coordinate, this complex matrix of possible conditions as a feature of providing fault management services to participating resources. Most of this complexity can remain hidden from participating resources and only has to be exposed on a need-to-know basis.

Guideline 5: Choose a Proven SA Solution

To maintain continuity of service for real time/near-real time applications such as telecom network equipment, failover must be extremely fast. Total failover time consists of the time to detect a failure, time to understand the scope and isolate the fault, and the time it takes for a standby resource to assume the active role. Typically, all of this must occur in milliseconds, not seconds and certainly not minutes. To achieve this level of performance requires concentrated effort, vigilance and constant tuning. Messaging, in-memory database, heartbeating, and replication are just a few of the areas that must be very, very fast, consistent and low overhead. It is common to see other software solutions — such as application servers or virtualization products with SA features — forced into acting as an availability solution for performance levels they are not designed to meet. In some fault scenarios, they may simply move the core fault to another location. Many service availability solutions fail to achieve these levels of performance and therefore don’t deliver the level of service continuity that real time to near-real time applications require. Other strategies may require that the operator take manual recovery steps. While this might be acceptable in some scenarios (although it’s increasingly less acceptable), it is not acceptable for telecommunications and network equipment. Service Availability solutions must be transparent to the operator, and automation is critical in order to minimize operator costs in today’s systems.
Guideline 6: Consider Scalability at the Beginning of the Product Lifecycle

Systems grow over time. For example, in later phases, control plane applications might be replicated or partitioned and further distributed over more hardware to better handle the increased data volume that must be processed. One of the easiest ways to increase capacity is to add additional blades to an existing rack or chassis. Good service availability software and systems should scale to handle hundreds of nodes, and be flexible enough to easily allow for new software partitioning and distribution strategies. Some systems are not tested to full operational limits, and may or may not scale when asked to manage this many nodes. Also, one should ensure that the messaging which is used to communicate between the nodes will scale, both in functionality and performance. Number-of-nodes is the most common measure of scalability, but there are other important examples. The number of objects or points of failure (processes, drivers, hardware, etc) that can be simultaneously managed is also an important measure of scalability. This can be particularly important as it affects startup time and the overhead of monitoring all the objects simultaneously. The SA software should not use more than one or two percent of CPU time as overhead in steady state monitoring mode.

Guideline 7: Adopt the Right Standards

Software which supports common standards promotes interoperability with other software packages, and this usually makes it easier to substitute one package for another. This means greater freedom of choice and platform re-use.

The Service Availability Forum (SA Forum) is the standards body responsible for the most important standards for Service Availability software. Many of the world’s leading telecom and networking equipment manufacturers have contributed to the specifications developed by the SA Forum, along with leading hardware and software companies. This has resulted in a stable set of specifications that provide clear boundaries and interfaces between hardware/firmware, service availability middleware, and applications. The SA Forum has published two specifications, known as the Hardware Platform Interface (HPI) and the Application Interface Specification (AIS). HPI defines a clear abstraction layer in the form of a standard API interface between SA middleware (or any interested client) and the hardware platform. It enables application portability across platforms and reduces the startup costs of integration with a new platform. HPI is a commercial success and is found in many deployed systems. The AIS specification addresses the complex abstraction between service availability middleware and applications. It consists of a set of services with API definitions and provides an architectural framework for modeling and managing the fault cycle, as outlined in guideline 1. AIS implementations are deployed in carrier networks and are gaining momentum because of the significant cost savings and time-to-market advantages.
Guideline 8: Consider the Network When Choosing a Hardware Platform

Choosing a hardware platform is naturally a key decision when designing a system to deliver service availability. Without correctly architected hardware, it is simply impossible to achieve high levels of availability. Network-centric applications that require maintaining session and state information for applications are very different than IT applications that focus only on high availability. The requirements go well beyond the well-established concepts of NEBS level 3 compliance (and ETSI equivalents) to include redundant power supplies, management modules and fabric interconnects. The ultimate objective is to configure a hardware platform that meets the application requirements and has no single point of failure - a fundamental aspect to delivering service availability. In addition to these architectural aspects, there are a number of other critical considerations. Provisioning for both AC and DC power (within the same architecture) for flexible deployment environments is critical in today’s rapidly evolving service provider environment. Additionally, extended availability of hardware components is essential, to avoid constant development churn and re-qualification of systems and applications in service provider networks. All of these factors have a substantial impact of the ability to deliver a service available platform over the evolving lifecycle of a network.

Guideline 9: Adopt a Flexible Approach to Integration Requirements

Today’s communications and networking systems are often made up of newly developed code as well as legacy applications and 3rd party packages that often do not inherently fit together well. Integrating all of this into a system that performs its core function correctly is challenging enough. When you add the service availability requirement, things can rapidly become more complicated, and the result is often an extended and more difficult test cycle. Legacy and 3rd party applications may or may not be “SA-Aware”, that is, designed to support software redundancy. Even if the software is “SA-aware,” the integration points likely do not align with your system’s SA infrastructure services. There are yet other challenges awaiting the system designer for legacy or 3rd party applications that are “non-SA-aware,” or not built to support software redundancy. Is the application Stateless or Stateful? If Stateful, how will state be captured and made available to a standby instance (Checkpointing)? Is the code available for modification? Can one employ an active-active strategy in order to minimize code change, yet get the parallel state mirroring needed to meet quick failover times?

New developments that have service availability requirements are relatively straightforward in this regard. Still, today’s new applications are tomorrow’s legacy code. This highlights the power and urgency of embracing open standards (in this case, SA Forum standards) as it gives both internal development and 3rd party vendors a common reference for understanding the integration strategy, while helping to future-proof these solutions. It is also important to note that the SA Forum standards recognize that substantial investments exist in legacy and 3rd party applications. The SA Forum provides well thought out integration paradigms to address this issue that are tried and true approaches that deal with these cases.
Guideline 10: Integrate SA and System Management Capabilities

Numerous studies have concluded that systems with strong management capabilities have better availability than poorly managed systems. It is also important to coordinate management and availability actions wherever possible. For example, if a management action is taken on a particular resource that temporarily takes it out of service, the SA software and system should be notified before the action takes place so that it can initiate a graceful shutdown and switchover to a standby resource. In a similar fashion, when the SA software detects and processes a resource failure, it should send a notification to any management software, so that appropriate actions can be taken. The key is communication; the management software should notify the SA software of relevant events, and the SA software should return the favor by notifying the management software of availability events. Ideally, the SA software and the management software share a common view of system resources such as a single information model that reflects the state and configuration of all of the manageable hardware, software and networking resources, which is visible and available to both.

Guideline 11: Leverage Commercial Software and Hardware

Almost no one today would conclude that they could save development time and money by building their own database software or operating system, but the same logic is not yet conventional wisdom for SA middleware. Development teams continue to create and maintain their own proprietary SA solutions, consuming valuable engineering resources that could be better utilized by focusing on competitive differentiation for their core product. A wide variety of mature and field-proven SA middleware solutions are available in the market.

Real-life examples of commercial deployments indicate that using commercial off the shelf solutions can save companies as much as 25-30% in total costs, when compared to equivalent proprietary solutions. Using commercial packages also moves the costs from fixed costs (people) to variable costs (software). This is important in cost-sensitive scenarios and timeframes, because often you are only paying for software when deploying products. It is also important when time-to-market demands are accelerating, because you can ramp up more quickly. You don't have the expense or delay of hiring and training SA specialists to maintain and extend a proprietary middleware solution.

Picking a proven solution can also reduce your risk. Every project has risks of missed deadlines, cost overruns and failure. Re-use of code, in the form of well-tested commercial software packages, can reduce these risks. It also takes the maintenance burden off the team’s shoulders. For example, as you transition to 64-bit platforms, or to IPv6 or to other environments that require software upgrades, those changes will be handled by the software vendor, not by your own internal resources.
Guideline 12: Focus on Tight Hardware/Software Integration

Many complex factors contribute to creating a system with true service availability. While it is possible to select the “best of breed” options in each area - hardware, operating system, and service availability middleware - this does not guarantee success. The multifaceted interactions among various system components are often equally crucial in delivering success with service availability. The complexity of hardware/software integration is a well established risk area that has delayed many projects over the years.

The interaction between the hardware, operating system and the service availability middleware is a particularly crucial domain. Today’s hardware is equipped with many sensors, low level diagnostics and communications infrastructure, which can be configured to provide notifications to the middleware on operational status. Unless this interface is well understood, projects can be relatively far along in their development cycle - and even reach deployment - before critical mis-understandings reveal underlying system shortcomings. This overall picture is further complicated by the lifecycle evolution of all these moving components. Hardware becomes obsolete and new generations of blades, while outwardly backwards compatible at the application layer, may have subtle differences within the low level infrastructure. Conversely, both operating systems and middleware evolve, often with new capabilities, and the mix of old and new interfaces must be maintained to ensure smooth operations for all applications. Maintaining a stable and highly available system in this environment of rapidly evolving components can be time consuming and costly if not managed correctly.

Summary

Whether you are in a large or small company, a centralized team or a product-level team, service availability is a hard, complex lifecycle problem that companies almost invariably get wrong the first time. Following the above guidelines and choosing an integrated and field-proven commercial Service Availability solution will make it possible for you to deliver the highest levels of availability for your network equipment and solutions. Furthermore, leveraging a commercial solution that follows these guidelines will enable you to lower project risk and accelerate time-to-market for innovative applications.