The Secret to Building Highly Available Systems

by Mark Richards

During the summer of 2008 more than 200 flights were delayed or canceled at the Dublin airport due to a shutdown in the Dublin air traffic control radar system. In 2009, a shutdown in the FAA's automated flight planning and communication system caused significant flight delays and cancellations across the country. In 2006 almost 200 flights were canceled at the Seattle Tacoma International Airport due to power outages and system failures in the Terminal Approach Radar system. Real-life situations such as these can place people in significant danger simply because critical systems were not available when they were needed the most.

Air traffic control systems are one of the best examples for systems requiring high availability. Other good examples include life support systems like those found in hospitals. Remember the scene from the movie *Born on the 4th of July* where Tom Cruise is lying in the hospital bed focusing on the little broken down suction machine keeping his leg alive? "If that machine stops", said the doctor, "you'll lose your leg". Hmm - maybe there is something to this high availability thing after all.

You can also find several examples demonstrating the need for high availability in the business world (albeit not as life threatening as the examples above). For example, Financial Markets Trading Systems must be highly available, but not for the reasons you might think. Trading firms make millions of dollars by having fast and reliable trading systems, which may lead you to believe that is why they need to be highly available. However, the real need for high availability in trading systems is that many trading firms no longer have the means to manually place a trade. If their systems are not available, then trades cannot be placed, resulting in a significant loss of money for both the company and its clients.

Systems must be highly available in the global banking market as well in order to support the continuous movement of money through numerous time zones around the globe. As Michael Douglas stated so well in the movie *Wall Street*, "Wake up pal, money doesn't sleep." Nor do the systems that process and move money around the world.

E-commerce is yet another example of systems that require high availability. Suppose your favorite online bookstore or online clothing store website would suddenly not be available. What would you do? Would you wait on buying that book, CD, or new bathing suit until the web site is up again? Or would you simply go to another online store and buy it there? Retailers generally don't want to take the chance of losing your business.

Better yet, consider an online auction website such as Amazon or eBay. What do you suppose your reaction would be if you were seconds away from finally getting that pet rock you've always wanted and the web site went down? Companies such as these have a reputation to maintain – if their site is not available, people will go elsewhere.

This paper is about building highly available systems and what system availability really means. In this paper you will learn the difference between high availability and continuous availability, how to calculate system availability, and most importantly, the secret behind building high availability systems. I will conclude by introducing two very interesting ongoing research projects in the area of high and continuous availability that may shape the way we design and develop software in the future.

How Much Availability is Enough?

Availability is typically measured in terms of the number of "nines" the system supports. For example, the term "four nines" refers to 99.99% availability, or roughly 52 minutes of system downtime per year. The following table illustrates the percentage of system availability and what that translates to in terms of the number of minutes of downtime per year:

90.0% (one nine)	36 days and 12 hours per year
99.0% (two nines)	87 hours and 46 minutes per year
99.9% (three nines)	8 hours and 46 minutes per year
99.99% (four nines)	52 minutes and 33 seconds per year
99.999% (five nines)	5 minutes and 35 seconds per year
99.9999% (six nines)	31.5 seconds per year

So how much availability is good enough? To put this question into perspective, consider the example of living in a "three nines" (99.9%) world. If everything around us were "three nines", the world would look like this:

- There would be a 99.9% turnout of all registered voters in an election
- You would have one rainy day every three years
- If you made 10 phone calls a day you would only have 3 dropped calls a year
- If you used your Windows PC 40 hours a week, you would only have to reboot it once every two weeks (once a year for a Mac!)

That sounds great, doesn't it? However, maybe a "three nines" world wouldn't be so great:

- The U.S. Postal Service would lose 2000 pieces of mail *each hour*
- 20,000 prescription errors would be made each year
- There would be 500 incorrect surgical procedures *per week*

Suddenly, a "three nines" world doesn't seem like a very nice place to live in. Of course, availability in software systems is much different than these interesting facts, but it does bring up an interesting question – how much availability is enough?

Some of the critical factors you need to consider include the type of system you have, the business domain you are working in, and to a larger degree, the reasons you need high availability. Do you need to support global operations across multiple time zones around the world? Do you need to support continuous operations during software or hardware upgrades? Do you no longer support manual operations, relying solely on your systems to support your core business functions? Identifying the reasons *why* you need high availability will help answer how much availability you need to support and whether you should consider high availability or continuous availability.

High Availability vs. Continuous Availability

There are two basic types of system availability - high availability and *continuous availability*. *High availability* (HA) is reactive in nature and places an emphasis on failover and recovery in the shortest time possible. *Continuous availability* (CA) is proactive in nature and places an emphasis on redundancy, error detection, and error prevention. This difference may seem subtle, but it's not. If you pull the key words from these definitions you can better understand how they differ:

High Availability	**Continuous Availability**
Reactive	Proactive
Failover	Redundancy
Recovery	Error Detection and Prevention

At first glance it would seem that systems that support continuous availability don't failover. In fact, they don't. Do you remember the famous "tree in the forest" question from grade school? It went like this: "If a tree falls in a forest and no one is around to hear it, does it make a sound?" Discussions around that question always seemed to boil down to the semantics surrounding the definition of *sound*. Sound waves are produced, but if no one is around to hear them, it is really *sound*?

Now consider the same question put in the context of system failures: "If a fault can be recovered from before the user is aware that the fault occurred, is it really a fault?" This is the real goal of continuous availability

– let the system fail, but resubmit the request as fast as possible so the user is unaware that processing was interrupted or that a fault even occurred. This is accomplished through highly redundant systems – systems that don't stop.

Given the differences between high availability and continuous availability, it is clear that systems such as the national defense system and air traffic control systems require continuous availability. However, what about the business world – isn't high availability good enough? Consider this question for a moment: when was the last time you had to bring your *critical systems* down to do a software install or hardware upgrade? How many evenings and weekends have you spent on production installs or hardware upgrades? Wouldn't it be nice to perform a production install in the middle of the business day without disrupting the systems? This is what continuous availability is all about in the business world – keeping the critical core systems continuously running while providing the ability to perform software, hardware, and operating system maintenance.

If you are still not convinced about the need for continuous availability in the business world, consider this interesting fact: most of the world's business, particularly in financial services, still runs in batch cycles on large mainframe computers. As business volumes grow, batch times increase, and as a result the window of opportunity between batch cycles to perform software and hardware upgrades (and even database backups) is significantly shrinking. More and more companies are faced with batch cycles that are close to overlapping or not completing before online systems need to come up. Add to this problem globalization and the need to support multiple time zones for online operations, and you can easily see there are little or no opportunities for bringing systems down to perform hardware maintenance and software upgrades. Continuous availability can help solve these problems by allowing maintenance and upgrades to occur during normal business hours without interruption to the online or batch systems.

HA and CA Infrastructure Topologies

The topologies needed to support high availability and continuous availability differs significantly. High availability generally utilizes an "active/passive" model where one node (or leg) is always active and another replicated in standby node. The active/passive model is illustrated in Figure RIC-1.

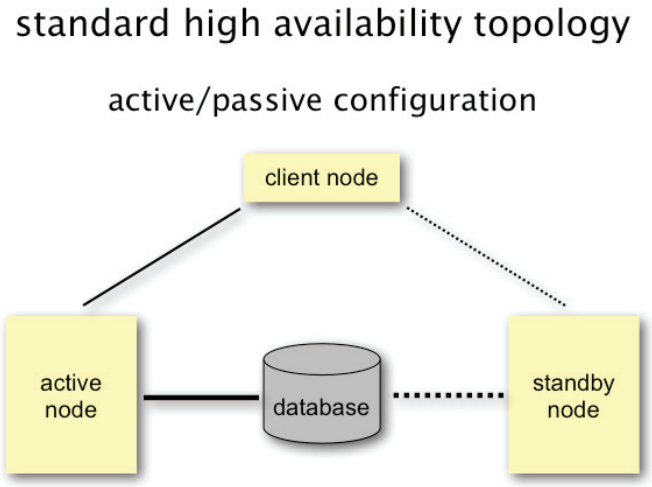

Figure RIC-1

The active/passive topology is sometimes referred to as a *clustered topology*. This topology usually contains a single database node (which is usually clustered) and a standby node that can be co-located with the active node or geographically distributed in another data center. The mean time to failover (MTFO) is usually measured in minutes due to the time required to first detect the outage and then to activate the standby nodes.

Continuous availability generally utilizes an active/active topology where processing is split between two or more *active* nodes through load balancing schemes such as *round robin* or *first available*. The active/active topology is illustrated in Figure RIC-2.

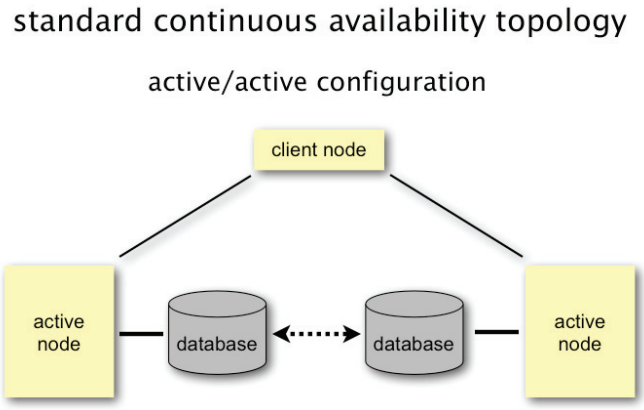

Figure RIC-2

With the active/active topology, the MTFO is usually measured in seconds because multiple nodes are operational and are ready to immediately receive requests. For both nodes to be able to process requests, the active/active topology requires either the use of bi-directional database replication or network transactions (two-phase commit between databases)

because each leg of the architecture must be able to maintain full end-to-end operations (i.e. no single-point-of-failure). Network transactions are usually not an option due to the requirement that all enterprise resources (e.g. databases) be locked throughout the business process. They are also not as transparent as database replication.

While bi-directional database replication is the preferred approach for supporting continuous available active/active systems, it introduces a whole new set of limitations and complexity. While replication software is getting better and more reliable, it is still somewhat error prone and very expensive. Some of the most difficult problems you might encounter with bi-directional database replication are ping-ponging (repeated updates), data loss in the event of a node failure, and data collisions (simultaneously changing the same data).

Regardless of the complexities involved, active/active systems are generally more reliable than active/passive systems. The reason is that in active/active systems all nodes are actively processing requests, guaranteeing that all nodes are in sync and up-to-date with the latest software and hardware upgrades. With active/passive systems, however, there is no way of being certain that the standby node has been updated with the latest software since it is not currently active and receiving requests.

Calculating System Availability

Some companies are contractually bound to delivering a specific system uptime (availability) as published through a formal Service Level Agreement (SLA), whereas other companies just state they need "four nines" to support the business. How can you determine how available your systems are? Are you meeting your SLAs? How do you know just how available your system really is? Fortunately, there is a relatively easy way of determining this through basic system uptime and error metrics and simple mathematics.

The basic mathematical formula for calculating system downtime is as shown in Figure RIC-3:

Figure RIC-3

$$sd = (1-a)^2 + (1-a)\frac{mtfo}{mtr} + (1-a)d$$

where

sd = probability of system downtime,
a = probability that a node is operational,
mtfo = mean time to failover,
mtr = mean time to repair,
d = probability of a failover fault.

The first part of the equation is squared, indicating it is a dual-node system. For three nodes (or legs) you would cube it instead, and so on. Subtracting 1 from the probability of system downtime (sd) gives you the number of nines your systems support.

To demonstrate how this formula works and to illustrate the difference between high availability and continuous availability, I will calculate the system availability for a dual-node topology with industry standard servers (ISS 99.9% availability) using an active/passive (HA) topology and an active/active topology (CA).

Consider the following values for a dual-node active/passive topology (HA):

a = 99.9% (industry standard servers)
mtfo = 5 minutes (the time it takes to detect the error and fire up the standby node)
mtr = 3 hours (the time it takes to repair the down node)
d = 0.1 (probability that the failover will fail)

Plugging these values into the formula above yields a system downtime of .00038777778. Subtracting 1 from this value gives you an availability of **.9999613 (99.99%)** – about 30 minutes of downtime per year.

Notice for active/passive systems there is a small chance (d = 0.1) that the failover will fail because you cannot guarantee that the passive node is up-to-date and will start up when needed. As you will see, this is not the case for active/active (CA) systems.

Using the same hardware and mean time to repair (mtr) values for a dual node active/active system supporting continuous availability, you can see there is a dramatic difference in the system availability:

a = 99.9% (industry standard servers, same as above)
mtfo = 3 seconds (the time it takes to resubmit the request)
mtr = 3 hours (the time it takes to repair the down node, same as above)
d = 0 (node is already operational, so value is zero)

Using these values yields a system downtime of .0000012777778. Subtracting 1 from this value gives you an availability of **.999998722 (99.9999%)** – about 30 seconds of downtime per year.

You can see that by using the same ISS hardware (99.9% availability per node) there is an order of magnitude difference between the high availability active/passive model (30 minutes of downtime per year) and the continuous availability active/active model (30 seconds of downtime per year).

The Secret to High Availability and Continuous Availability

Companies invest upwards of hundreds of thousands of dollars in hardware, vendor software, and network appliances to achieve high availability, yet what you will find is that they rarely reach their goal. The reason is simple – in order to make high availability or continuous availability work, you must take a *holistic approach* to building high availability systems. The holistic approach involves bringing together infrastructure (network and hardware), application development teams, and operations and support teams to address the problem (and solution) of availability in a unified fashion.

It is well documented that over half of all system failures are caused by operator error. This is not due to the lack of skill of the operations teams, but rather to the increased complexity due to heterogeneous systems. Application and system deployments are becoming more complex and require many layered products to be installed in heterogeneous technologies on heterogeneous platforms. This requires operators to be familiar with all of the technologies, products, and platforms being deployed – a tall order, even for the most competent operations team.

Perhaps the biggest mistake companies make when building high availability systems is that they place too much emphasis on the hardware and network infrastructure of the system and not enough emphasis on the application software that run on that highly available infrastructure. If applications are not built to support high availability, then all of the infrastructure in the world will not get you to your high availability goals.

To build highly available systems, your applications must support the underlying high availability infrastructure. For example, some of the factors that can impact your ability to deliver high availability and continuous availability systems include random number generation, application state, in-memory storage, disk access, specific hostname dependencies, and tightly coupled applications or services.

The holistic approach to delivering high availability requires three ingredients: well-trained and informed operations and deployment teams, the right infrastructure and topology, and an awareness of the software development teams as to what will affect high availability. If any of these three ingredients are missing, you will most likely not achieve your high availability goals.

A Look into the Future

There are many exciting efforts underway in the area of high availability and continuous availability that will most likely shape the way you think about and develop software in the future. The first of these is *Autonomic Computing*, a research initiative conducted by IBM. Autonomic Computing basically describes a systemic view of computing modeled after a self-regulating biological system, or more specifically, a network of self-healing computer systems and components that manage themselves.

With Autonomic Computing, developers would write and deploy software components that are self-configured, self-healing of faults, self-optimized to meet requirements, and self-protected to ward of threats. It essentially describes the ultimate distributed, loosely coupled system. You can find out more about autonomic computing by visiting the IBM research web site .

Another effort going on in the area of high availability is *Recovery-Oriented Computing* (ROC), conducted by the Berkeley/Stanford ROC research team. Recovery-Oriented Computing focuses on designing software and hardware systems that can recover as quickly as possible from software faults and operator errors. ROC uses a holistic approach similar to the one I described in this paper to address the issue of fault detection, prevention, and recoverability. Some of the main features of this effort include the following:

- Fault Containment: contain a fault in a component so that it doesn't affect other components
- Fault Detection: automatically locate the root cause of the failure at the lowest possible subcomponent level
- Fault Recoverability: repair the fault and recover at the smallest subcomponent level

One of the neat and innovative features of ROC is the ability to inject faults at all levels of the system for testing and training. It is stated on the ROC website that operator error is a major source of system failures. Failures of this type could be significantly reduced if operators had the chance to experience faults and learn how to react and repair them through real-world testing and training.

You can read more about Recovery-Oriented Computing by visiting the website .

Summary

For high availability to work you need to combine and coordinate the efforts of the infrastructure, operations, and development teams – what I call the holistic approach to high availability. Once you have a unified approach, you then need to analyze the reasons why you are seeking high availability. If you need to support continuous operations for global operations or system maintenance, you will most likely require an active/active continuous availability model. If your needs are such that you simply want your systems to be available most of the time and you can manage your install windows well enough, then a less expensive active/passive system is what you should strive for.

About the Author

Mark Richards is a hands-on SOA and Integration Architect passionate about the field of enterprise and software architecture. Having served in the IT industry since 1984, he has significant experience in the architecture and design of small to large systems on a wide range of languages and platforms. Mark has particular expertise in event-driven architecture, service-oriented architecture, messaging systems, and enterprise service bus technologies. He is the author of several technical books, and has spoken at over 80 technical conferences worldwide. You can read more about Mark by visiting his website at http://www.wmrichards.com.