CHAPTER

17

Failover Configurations

One only needs two tools in life: WD-40 to make things go, and duct tape to make them stop.

—G. M. Weilacher

In this chapter we look at some of the specific configurations that we have seen used for cluster design, and then at some of the issues commonly (and not so commonly) seen when implementing them. For the sake of organization, we have broken up the most commonly seen failover configurations into two groups: two-node configurations and common larger configurations.

On the Availability Index (see Figure 17.1), since we are still discussing clustering, we remain for one more chapter on Level 8.

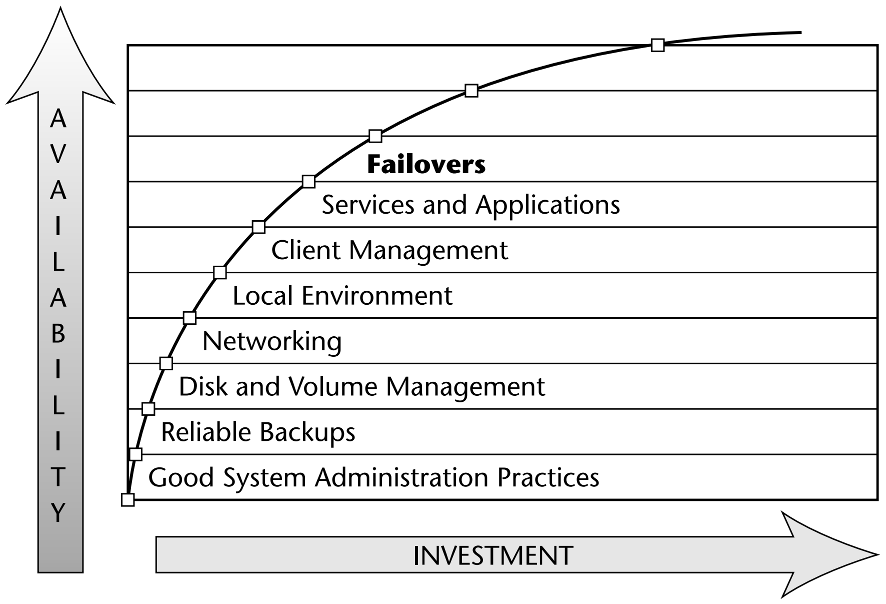

Figure 17.1 Level 8 of the Availability Index.

Two-Node Failover Configurations

The simplest and most common failover configurations are the two-node variety. There are two basic types of two-node configurations: *active-passive* or *asymmetric*, and *active-active* or *symmetric*. In an active-passive configuration, one node is active and does all of the cluster's critical work, while its partner node is a dedicated standby, ready to take over should the first node fail. In an active-active configuration, both nodes are doing independent critical work, and should either node fail, the survivor will step in and do double duty, serving both sets of services until the first node can be returned to service.

Active-Passive Failover

Active-passive failover, as shown in Figure 17.2, is the baseline cluster configuration. Every cluster configuration is a variation of this model. In an active-passive configuration, there is one master server, called *rhythm* in this example, which provides all of the critical services within the pair under normal circumstances. It is connected via two dedicated heartbeat networks (as explained in Chapter 15, "Local Clustering and Failover") to its partner server and dedicated backup node, *blues*.

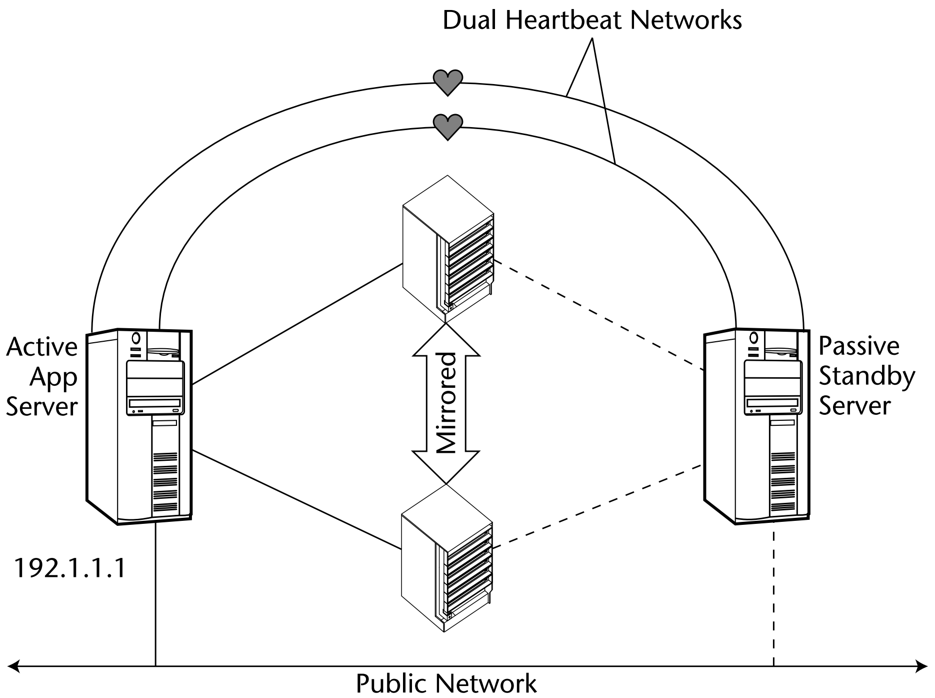

Figure 17.2 Active-passive configuration, before a failover.

Both servers are connected to a set of dual-hosted disks. These disks are usually divided between two separate controllers and two separate disk arrays, and the data is mirrored from one controller to the other. (Of course, the configuration works just fine if the mirrors are on a single controller or in a single cabinet, although the resultant configuration is just not as resilient or as highly available as it would be with separate controllers and separate cabinets.) A particular disk or filesystem can only be accessed by one server at a time. Ownership and conflicts are arbitrated by the clustering software.

Both servers are also connected to the same public network, where the users sit. The servers share a single network (IP) address, which is migrated by the FMS from one server to the other as part of the failover; this virtual IP address is not permanently assigned to a particular server. Only one server in the pair actually owns the address at a time, and any ownership conflicts are arbitrated by the failover management software. As we discussed in Chapter 16, "Failover Management and Issues," the other host's public network identity is limited to its administrative network address, an address that is only for the use of system administrators, and not for regular users.

Active-Passive Issues and Considerations

The biggest issue when examining the issues of active-passive versus active-active failover is, of course, the cost. Fundamentally, with active-passive you are buying two hosts to perform the work of one. One host sits largely idle most of the time, consuming electricity, administrative effort, data center space, cooling, and other limited and expensive resources. However, active-passive configurations are going to be the most highly available ones over time. Since there are no unnecessary processes running on the second host, there are fewer opportunities for an error to cause the system to fail. In general, largely due to the extra expense and administrative overhead, only the most critical applications run in an active-passive configuration.

After an active-passive failover has completed (see Figure 17.3), the takeover server has control of the associated service group. Since both hosts in a shared-disk two-node cluster directly access the same disks (at different times), as long as the disks and all connectivity are in place, there is no question that the takeover host will see the same application data as the failed host.

How Can I Use the Standby Server?

Probably the most common question asked about active-passive failover configurations is "How can I use the standby server?" It is most difficult to justify to the people who approve technology spending that they need to buy two identical servers, thus doubling the expenditure, while only actively using one of them at a time. So the obvious suggestion is to find some activity that can realistically and safely be performed on a standby node. This most commonly asked question is usually followed by this suggestion: "What about my developers? Can I put them on the standby server?" Of course, this is probably the worst thing to do with that standby server.

The key question to ask when evaluating a potential activity for its fitness to be placed on the standby server is "How likely is it that this activity will cause the server to be unable to accept a failover?" To correctly answer this question, you must consider the likelihood that whatever activity is taking place on your standby server will cause that server to crash, hang, freeze, or have its critical files altered. If the activity on the backup server causes a critical failover not to complete, extended downtime will almost certainly result. Another inevitable result will be you on the hot seat, trying to explain to your management why all the money that they spent to implement a highly available system did not do what it was designed to do.

We have categorized many of the activities that a business might consider for placement on the standby server into four levels of advisability: Bad, Acceptable, Good, and Best.

Level 1: Bad

Probably the worst activity to put on standby server is full-blown code development. (We say full-blown to differentiate it from database development, which is generally constrained by the database itself and is discussed under *Acceptable*.) By their nature, developers write untested code with bugs in it. (This is not intended to disparage any particular developers; the first pass on just about any code is going to contain bugs. If your developers do not write code in this way, we recommend giving them a handsome raise. They are well worth it! Or they are lying to you.)

Application bugs can produce all sorts of effects, including but not limited to system crashes, system lockups, depletion of one or more system resources (such as memory or CPU), or leaving the system in a state that makes it unable to accept a failover. (Obviously, bugs can also have much less serious effects on your system, resulting in simple compilation errors or program failures. We are not concerning ourselves with them, only with the kind of bug that can prevent a clustered node from accepting a failover.) If your management has just spent lots of money to make their critical servers highly available, and just when you need to use the standby for a failover it is unavailable, everybody looks really bad, and the money that was spent to shore up these systems will be deemed wasted.

Allowing developers on the standby server also violates at least two of our key design principles from Chapter 5, "20 Key High Availability Design Principles":

- Development and production environments should be totally separate from one another.
- The code running on these critical systems is not mature. The code is brand-new and full of bugs; that makes it totally unsuitable for inclusion in a cluster. Note that any code that runs on a cluster member is running in that cluster.

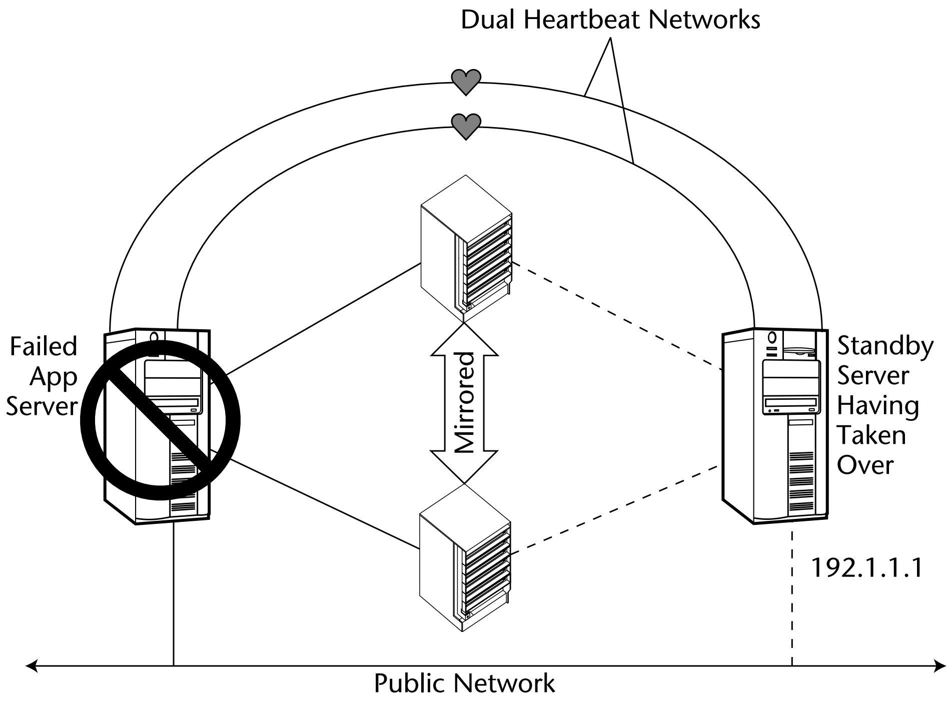

Figure 17.3 Active-passive configuration, after a failover.

Even worse than just letting developers work on the standby box is allowing them to have privileged (administrator or root) access on the standby box. If developers have that level of access, you may as well grant them privileged access directly on the main server. When developers are hard-pressed to find the cause of a system-level bug in their code, they start to look at the OS and network configuration and, hence, start playing with configuration files and restarting services and, worse yet, systems. With privileged access, they can override the FMS's protections against disks being accessed on multiple hosts. Plain and simple, this condition should never be permitted to occur. It is nothing less than a surefire recipe for calamity.

Arguably, an even worse application for the standby server is to use it for QA testing. In a QA environment, the testers are out to find bugs and other problems in the code that is being tested. If they do their job and find bugs, their successes could have serious impact on the server's ability to accept a failover, making it unavailable at the critical time and once again making the decision makers and influencers who recommended clustering look foolish and wasteful.

Level 2: Acceptable

Unlike raw code development, database development often has some constraints on it. Database development generally takes place within a mature database application, such as Oracle, DB2, or SQL Server, and the actions performed by the code are more likely to be restricted to database-related activities. If your database developers are writing code that might hang or crash the standby system, then their work should be classified as Bad. If their work is only going to impact the database environment on the takeover server, rather than on the whole system, then it can be classified as Acceptable.

One of the leading arguments for putting developers on the standby server is that it's easy to kick them off when their system is needed for production. Since the developers might be kicked off their server at any moment, it means that they must change the way they work, saving their work far more often than they might otherwise. If they forget to save frequently, then they could lose their work when a failover occurs. It also gives the perception that their work is of limited value to the overall operation, since they may be asked to stop working at any time without warning. In organizations where we have seen this model implemented, it has a detrimental effect on the morale of the developers and causes them to place less value on their own work and contributions to the organization. In most cases, it just makes them angry.

What's more, while critical services are running on their development server, the developers are unable to do any work, resulting in a slowdown in their performance, and delayed deliverables. An additional cost to the business is the direct result of their idle time; they are collectively sitting around doing nothing, while collecting their wages and other benefits. You're choosing one kind of downtime for another. When evaluating the overall cost of building a cluster in this manner, be sure to include the cost of idling developers and delayed deliverables.

Level 3: Good

So, we've decided that we shouldn't put our developers on the standby node in a two-node active-passive cluster. What does that leave? What *can* we run on the standby node?

It is certainly okay to put another production-quality application on the standby node. Because of the nature of clustered systems, the applications must be totally independent of each other, using separate disks and separate networks. They should have no dependencies of any kind on each other. Within that constraint, it's just fine to put, for example, one instance of a database on one node and a second, separate, instance on the other node. Or a database instance on one node and a file-sharing service, such as NFS or CIFS, on the other.

It's important to understand, though, that running any applications on the standby node increases the risk of that node not being available when it is required for a failover. A system that is busy running an application is measurably more likely to go down unexpectedly than is a node that is not running any applications.

In our experience, though, for most applications and enterprises, the additional risk introduced by running truly well-tested and mature applications on the standby node is more than outweighed by the cost savings of actually using the standby node. When the alternative is to idle the standby system and buy two more hosts to cluster the other application, the cost savings are clear.

Level 4: Best

If maximum availability is the primary factor, and cost is a distant second, then the very best way to configure a pair of failover servers is to dedicate the standby server as a pure standby server, running absolutely no other applications on it. For many of the most critical applications, this configuration is fairly popular. But in other shops, where the costs of dedicating a server outweigh the availability benefits of putting one application in a cluster, this configuration turns out to be prohibitively expensive.

However, by dedicating the standby server, you virtually guarantee its availability when it is needed. The rule is quite simple: The less the server does, the less risk there is that it will fail at an inopportune moment.

Active-Active Failover

The active-active failover model (see Figure 17.4) is very similar in appearance to the asymmetric model. The main physical difference is that in active-active each server will usually have more than one connection to the public network (although virtual networking eliminates multiple connections as an absolute requirement).

The real difference between active-active and active-passive is that in active-active, both hosts are running critical applications at the same time. Each host acts as the standby for its partner in the cluster, while still delivering its own critical services. When one server, *greeneggs*, fails, its partner, *ham,* takes over for it and begins to deliver both sets of critical services until *greeneggs* can be repaired and returned to service.

From a pure cost perspective, active-active failover is the better way to go. It makes much better use of your hardware dollar. Instead of incurring a more-than-100 percent hardware penalty, the only additional cost in building the cluster is the cost of running dual-connect disks and heartbeat networks, and obtaining the clustering software. Those costs are much more manageable and are generally acceptable for all but the least-well-off organizations.

There are two fundamental downsides to active-active configurations, but they are relatively minor and usually acceptable. The first, which we discussed in the *Level 3: Good* section, is the risk that one of the servers will be unable to take over from its partner when it is required to, because of a problem with its own critical application. If all of the applications in the cluster are well tested and mature, the risk here is acceptably low. The second downside is in the inevitable performance impact on the takeover host when it begins performing double duty. If you are concerned about this performance impact, there are basically two ways to handle it: (1) Buy extra CPUs and/or memory for both servers so that they can more easily handle the additional load (but remember what we said about applications expanding to take up all available resources) or (2) decide not to worry about it. After all, if you buy extra hardware to account for the periods of double duty, it is difficult to ensure that this hardware will be used exclusively at takeover time only. More likely, the additional hardware capacity will be sucked up by the resident application and will still be unavailable to ease the load when it is required. When you consider that either server will likely be running in dual mode far less than 1 percent of the time, you will see that the concern about performance after a failover is much ado about nothing. If you can afford the extra hardware, great. Get it. If you cannot, don't worry. It's not going to be a very big deal, and in most cases it's better to have two services running a little slow than it is to have one service stop running completely. Figure 17.5 shows what a symmetric pair looks like after the failover has completed.

It is important that the servers be truly independent of each other. You cannot have one server in the pair act as a client of the other. Taking the example of NFS, if servers *livelong* and *prosper* make up an active-active failover pair, and *prosper* is the NFS client of *livelong*, then if *livelong* fails, *prosper* will be impacted by the failure; it will hang, and most likely not recover. The failover will fail, and downtime will result. Interdependencies like this should be avoided in any kind of clustered configuration. We cover dependencies on network services and ways to identify these interlocking server issues in Chapter 11, "People and Processes."

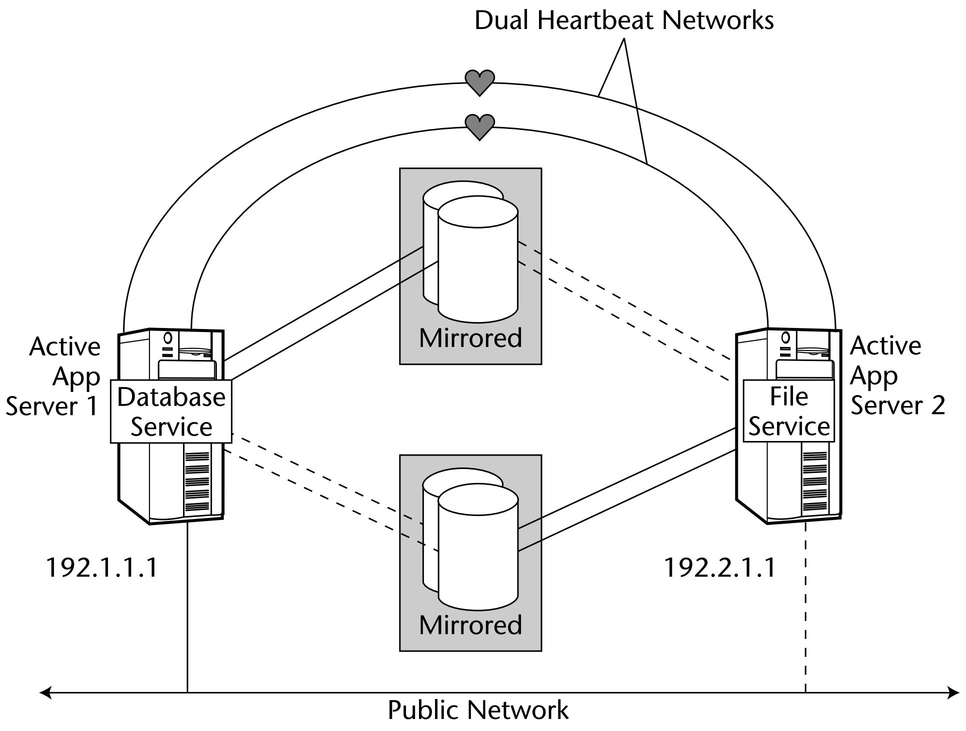

Figure 17.4 Active-active configuration, before a failover.

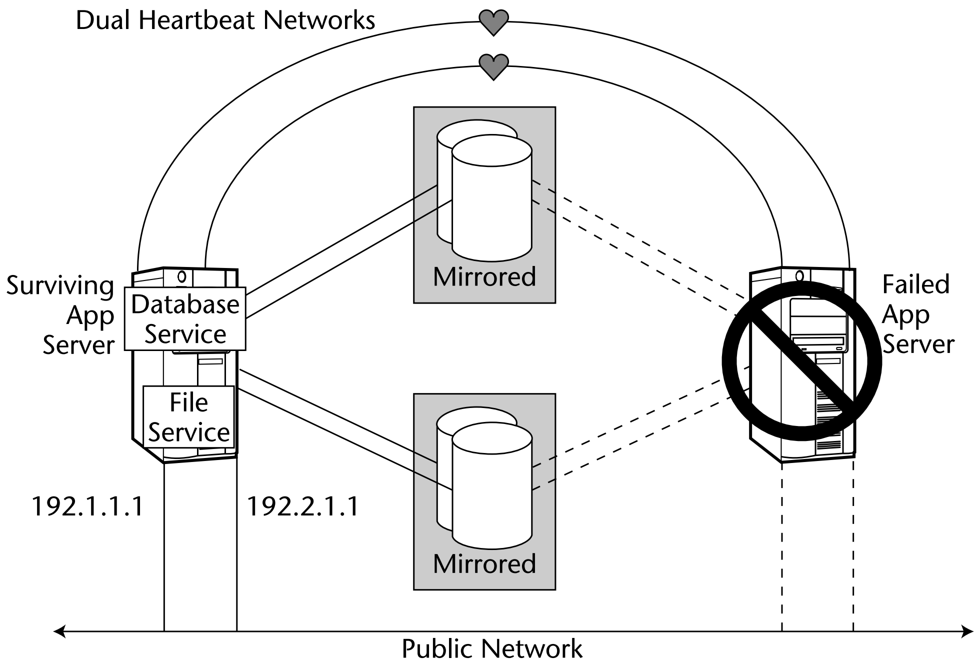

Figure 17.5 Active-active configuration, after a failover.

Active-Active or Active-Passive?

So, the question remains: Which configuration style is better? As with so many of the questions that we examine throughout the book, the complete answer is not entirely clear-cut. Active-passive is unquestionably better for pure availability. But because of the added implementation costs, it is just as unquestionably more difficult to sell it to management.

From a cost perspective, active-passive is superior. It requires fewer systems, and as a result, less space, less electricity, less cooling, and less administration.

In the end, like nearly everything else we have discussed, it depends on your needs. If your applications are so critical that you need to squeeze every possible iota of availability out of them, then active-passive is the way to go. If rare outages are tolerable, then active-active clustering should be fine.

If you believe that active-passive is truly the way to go, but your management is not convinced, we recommend selling it to them the same way you might sell life insurance. Clustering software and configurations are just like life insurance, after all. Although nobody likes to admit it, system outages are inevitable. When they occur, they cost money, and depending on when they occur and the importance of the system that has failed, they can cost a lot of money. The parallels to life insurance are obvious.

Tales from the Field

FAILOVER AS LIFE INSURANCE

Every few months I send my life insurance company a check. In return for that timely payment, I receive . . . nothing! (Not even a lousy calendar.) And personally, I hope that relationship continues for a very long time. If I were to stop paying those bills, the insurance would go away, and when my family finally needs it, they won't get any of the benefits. (Personal note to my wife: Relax, I have no intention of stopping my life insurance payments.)

If your company chooses not to invest in a standby server, then when it needs it, it won't get the benefits that a reliable standby server can provide.

Just as I (at least in theory) eat right and exercise so that I will live a long, healthy life, and therefore get to keep making those insurance payments and keep getting nothing in return, we build our computers with spare NICs, mirrored disks, mature applications, and all the other stuff we have discussed. Someday, despite my best efforts (and although I really don't like to think about it), that life insurance policy will finally pay out. And despite your best efforts, someday that critical server is going to fail. When it does, you'll want the most reliable server backing it up. Otherwise, downtime will be the inevitable result. The most reliable takeover server is one that isn't doing anything, except waiting to step in.

—Evan

However, more than four out of every five two-node clustered configurations that we have been involved with over the years have been active-active configurations. Realistically, they are more economical, and the risks that they bring along are acceptable for most environments. Once again, your mileage may vary. Choose the configuration that makes the most sense for you.

Service Group Failover

Back in Chapter 15, we defined a service group as a set containing one or more IP addresses, one or more disks or volumes, and one or more critical processes. A service group is the unit that fails from one server to another within a cluster. In the early days of clustering software, especially in active-passive configurations, each cluster ran a single service group, and when that group was on a system in the cluster, that system was active; otherwise, it was not. There was no concept of multiple service groups.

Later, when FMS grew more sophisticated and active-active clusters began to appear, two systems shared two service groups. When the cluster and its components were operating normally, there was still just one service group allocated to each server.

As servers grew larger and able to handle more capacity, it became clear that each server could easily manage more than one service group without suffering any significant performance impact. If a cluster member could manage more than one service group, it was reasonable to require that each service group must be able to failover separately from any others. Otherwise, the multiple service groups would, in reality, be a single service group, and no advantage would be gained.

The introduction of multiple service groups to clustering added value because they gave FMS the ability to failover intelligently. Service groups could be split up between multiple nodes in the cluster after some members of the cluster had disappeared. Service groups could failover to less heavily loaded systems, or be reapportioned between nodes based on just about any rule set.

For service groups to maintain their relevance and value, they must be totally independent of each other. If because of external requirements, two service groups must failover together, then they are, in reality, a single group.

Service group failover, as shown in Figure 17.6, is, therefore, the capability for multiple service groups that ran together on one server to failover to separate machines when that first server fails. In this figure, we see two nodes: *fish* and *chips*. *Fish* has two active service groups, A and B, and *chips* has two other service groups, C and D. Each group can also run on its respective partner server; A can run on *chips* as A', and so on. Even though the service groups can move separately from *fish* to *chips* and back again, the advantage of this scheme, from an availability perspective, is negligible.

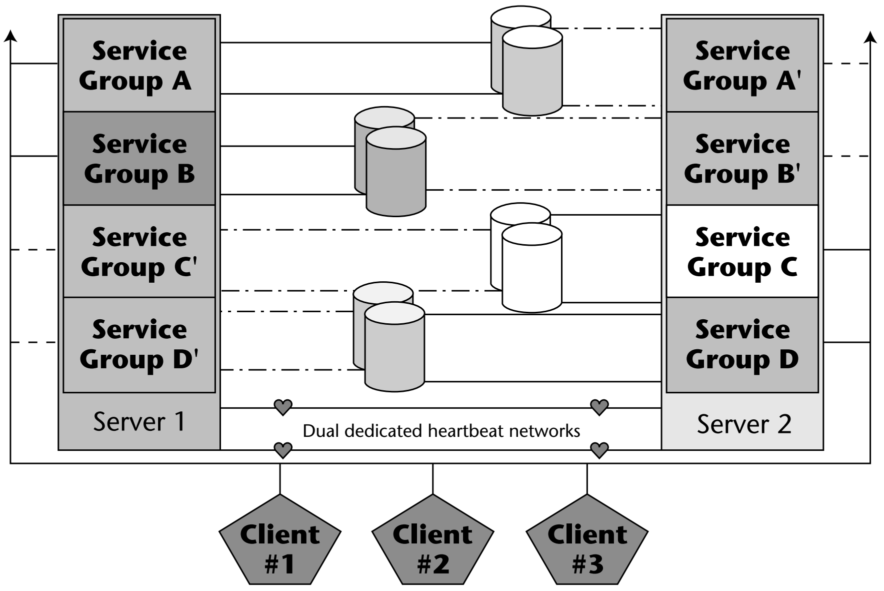

Figure 17.6 Service group failover.

Early implementations of service group failover were designed for two-node clusters, since that was all that was available at the time, and it was marketed as a huge advance in clustering software. It was not; the capability was there, but service group failover in a two-node cluster offers practically no additional value. As FMS has grown more sophisticated, service group failover has gone from being an overmarketed, underachieving feature to one that is taken for granted.

Service group failover and SANs have been the two critical technological advances that have enabled larger clusters.

Larger Cluster Configurations

As storage area networks and commodity-priced servers have permeated the computing world, clustering software has grown in popularity and become more sophisticated.

N-to-1 Clusters

In a SCSI-connected world, building more than two hosts into a cluster is immensely complex. Consider the SCSI-based four-node cluster shown in Figure 17.7, where four nodes are actively running, and the fifth node is designated as a standby for the other four. The standby server requires at least 10 SCSI bus connections (two each for the four primary servers, plus two more for the standby's local private disks, and two NICs for heartbeats, one for an administrative network connection, and at least four for public connections for the four primaries. Assuming no virtual IP, that's seven NICs. That adds up to a minimum of 17 external interface cards to support 4-to-1 failover. No wonder, then, that it was extremely rare to find clusters larger than two nodes before SANs began to appear.

Clusters larger than two nodes make it less expensive to deploy dedicated backup nodes. Instead of requiring one node to act as a dedicated standby for one active node (a 100 percent overhead), as in a two-node active-passive cluster, consider a three-node cluster where two of the nodes run critical applications and the third is a dedicated standby for either of the first two (a 50 percent overhead). Instead of requiring four nodes, arranged in 2 two-node pairs, you can save 25 percent of the hardware costs by only purchasing three nodes. If you carry that principle further, you can use the single backup node to protect more hosts, say, 10 (10 percent overhead) or 12 (about 8 percent overhead), in a larger cluster.

This model, where some number of clustered systems failover to a single dedicated standby node for the whole cluster is called *N-to-1 failover*.

The other serious limitation of SCSI-based *N*-to-1 clusters is that since only the "one" node (the backup for all the active systems in the cluster) can see all of the disks, when the failed node returns to service, it is necessary to fail its services back to it, freeing up the one node to takeover for another set of service groups. As you will see, failing services back isn't necessary for *N*-plus-1 clusters.

SANs have greatly simplified the construction and design of *N*-to-1 clusters, as shown in Figure 17.8, where we have taken the same 4-to-1 cluster depicted in Figure 17.7 and migrated it to a properly redundant SAN infrastructure. (For more about SANs, please refer to Chapter 8, "SAN, NAS, and Virtualization.")

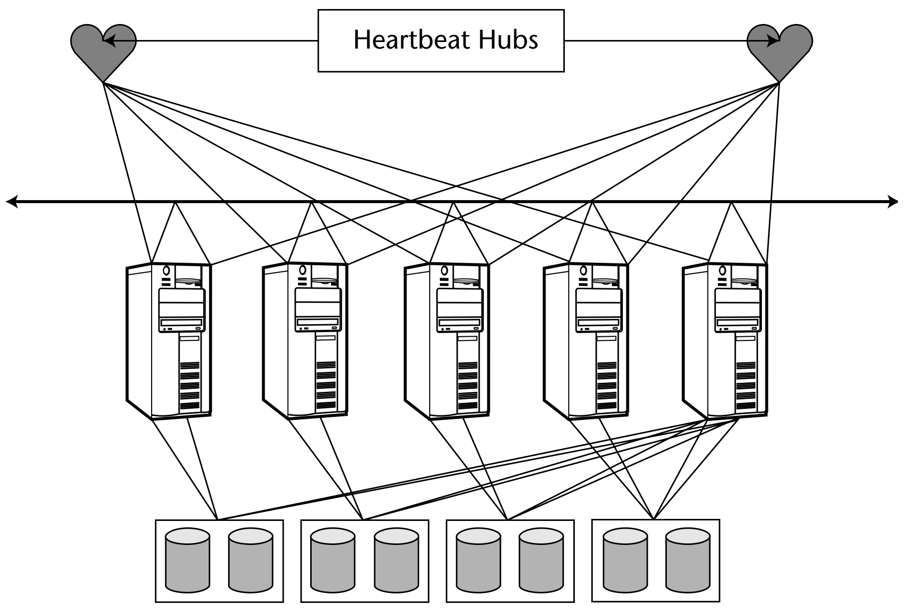

Figure 17.7 A 4-to-1 SCSI-based cluster configuration.

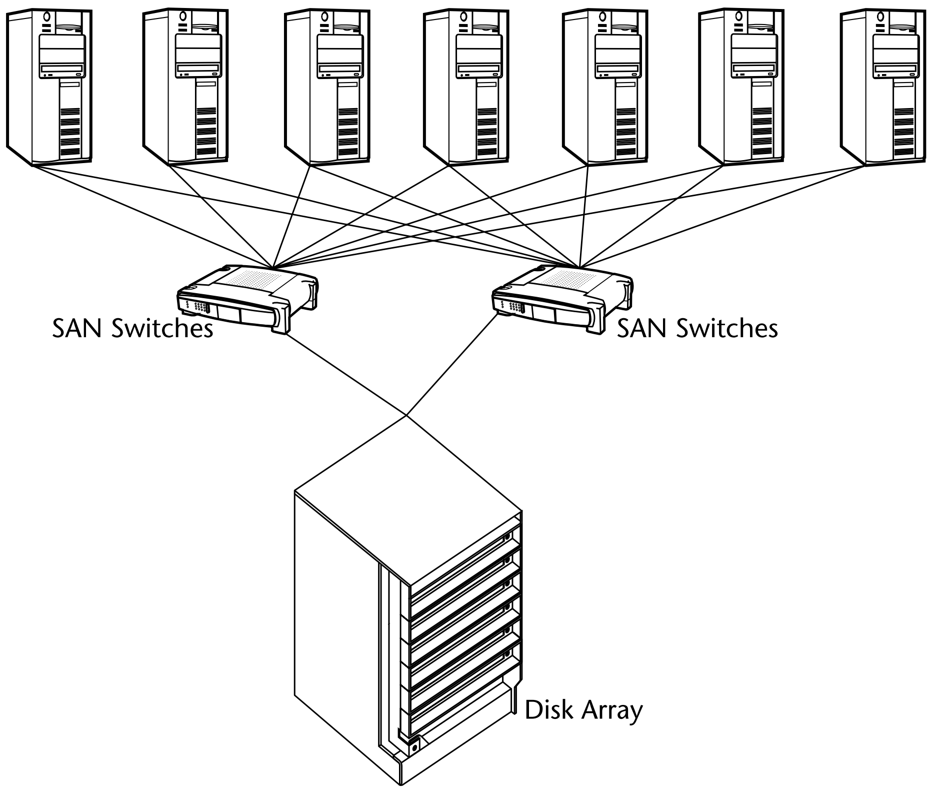

Figure 17.8 A SAN-based 6-to-1 cluster.

An interesting difference between the SAN-based cluster in Figure 17.8 and the SCSI-based one in Figure 17.7 is that in the SAN-based cluster, the hosts are all identically attached to the storage. In the SCSI-based cluster, one node, the 1 node in *N*-to-1, has many more connections to the network, and, so, must be bigger. It has to have the backplane slots to permit making all of these connections.

N-Plus-1 Clusters

As we described previously, SCSI-based *N*-to-1 clusters have a serious limitation. They must incur extra downtime after the cause of a failover has been repaired, so that the migrated services can be moved back to their original host. The added overhead required to configure all nodes to be able to see all the disks and to run all of the critical applications is huge, adding SCSI cables and a lot of complexity.

In a SAN-based cluster, all the nodes can see all of the disks, so there should not be a need to migrate applications back to their original homes. If we take this observation to its logical extreme, we develop a new type of cluster configuration that we call *N-plus-1,* shown in Figure 17.9 as 6-plus-1.

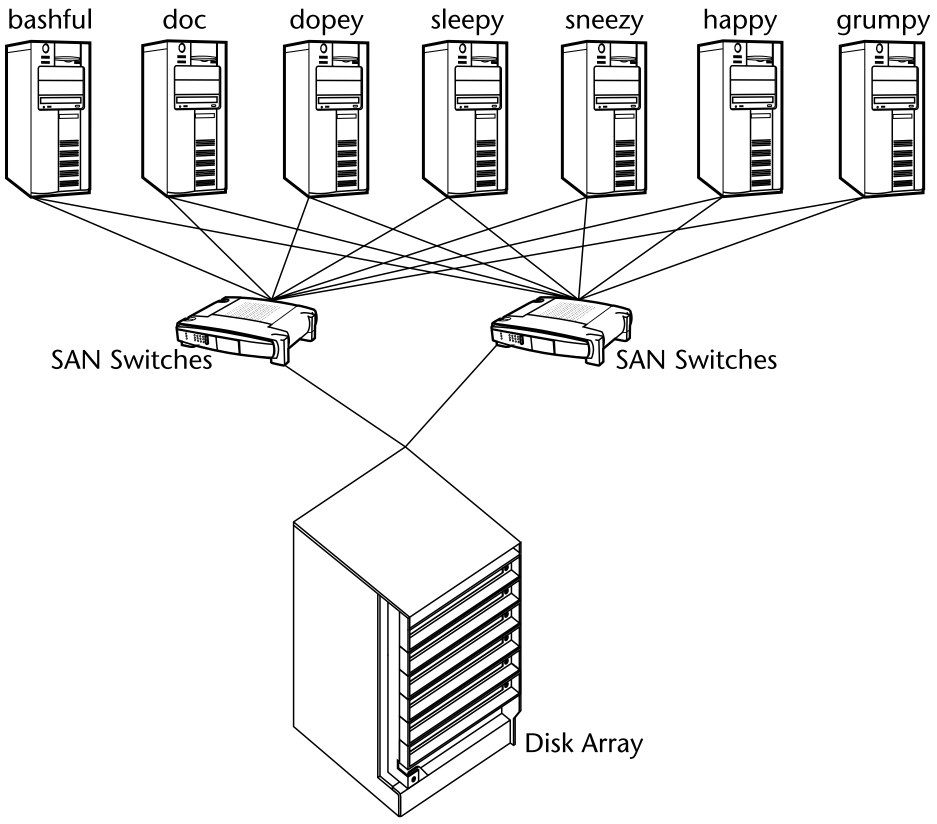

Figure 17.9 A SAN-based 6-plus-1 cluster.

In Figure 17.9's 6-plus-1 cluster, we have a total of seven nodes, including *grumpy*, which is running as a dedicated standby. Should *bashful*'s services fail, they will be automatically migrated over to *grumpy,* who becomes the new home for these services. When *bashful* returns to service, it joins the cluster as the new standby, so that when *dopey* crashes, its services will migrate over to *bashful.* Over time, the layout of hosts and services will not match the original layout within the cluster. As long as all of the cluster members have similar performance capabilities, and they can see all of the required disks, it does not actually matter which host actually runs the service.

N-plus-1 will provide availability superior to *N*-to-1, since no manual return to normal is required when *bashful* returns to service.

If *dopey* fails before *bashful* has been put back into service, then, of course, there is no dedicated standby host for *dopey*'s service to move to. What happens next depends on how your FMS handles the situation. Lesser quality FMS will take *dopey*'s service groups and move all of them to some other host in the cluster, say, *doc*. With an eye toward load balancing, high-quality FMS will take those same service groups and divide them up between the surviving hosts in the cluster. Very high-quality FMS will decide where to place the applications around the cluster based on the current loads of all of the hosts in the cluster; extremely high-quality FMS will make the determination based on historic system resource consumption, considering CPU and memory consumption, and I/O rates, along with the types of applications that are currently running on the system. The last option will generally result in the most accurate level of load balancing, though perhaps at the cost of a brief delay in failover time.

As clusters begin to grow to 15 or 20 nodes or more, it's possible that a single standby node will not be adequate, especially if it is down for repair for an extended period. We have begun to see larger clusters built in an *N*-plus-2 style, where, for example, 20 nodes are clustered together, but only 18 of them run critical applications at any time. The remaining two are configured as dedicated standbys.

We expect to see cluster sizes grow dramatically over the next few years as blade technology begins to enter the mainstream marketplace. With that in mind, we suggest that a roughly 10 percent system overhead is a sensible level.

High-quality FMS should be able to handle just about any mixed-and-matched cluster configuration that makes sense.

The astute reader may have noticed that there is no material difference between the configuration layouts in Figures 17.8 and 17.9. That is correct. Since there is no hardware difference between the two configurations, we recommend using *N*-plus-1, rather than *N*-to-1, in clusters that are larger than two nodes to take the best advantage of hardware resources and to maximize availability. If there is a serious downside to this configuration, we have not been able to identify it.

How Large Should Clusters Be?

Unfortunately, it's difficult to make a clear recommendation on the maximum size of a cluster. It would be very nice to be able to tell you that clusters shouldn't be larger than, say, 27 nodes. We cannot do that.

We also cannot say that clusters can grow in size without limit, subject to the capabilities of your FMS. The FMS itself can represent a single point of failure, and we have seen organizations who are leery of running all of their critical operations in a single cluster for just that reason. This is especially true if your FMS requires the cluster to be shut down in order to make configuration changes within the cluster.

The natural tendency is to grow clusters as large as possible and necessary to limit the number of management points. (From a simplicity perspective, it's pretty much a wash: Ten 2-node clusters are roughly just as complex as two 10-node clusters. In the end, you are just moving the complexity from one place to another.) However, as you put more responsibility on the FMS, it can become a single point of failure itself. Since the S in FMS stands for software, FMSs are no less prone to failure and errors than other applications. A bug or configuration error in an FMS could, in some instances, cause the entire cluster to fail. By limiting the number of nodes and critical applications in the cluster, you limit the impact of a failure in the FMS.

KILL THE BUGS

A friend of mine was the lead developer on an early clustering product called FirstWatch. He imposed a rule on his team: No new code can be added to the product unless all medium- and high-level bugs have been fixed, tested, and closed. He says that if he were developing some retail middleware, he would not have been as careful, but because FirstWatch was a product that customers came to count on to deliver increased levels of availability, he felt the extra precautions were absolutely necessary. He believes that the engineering teams for most clustering software are at least as careful as he was with his code, and that they tend to have more rigorous test suites.

—Evan

That being said, we expect to see cluster sizes increase dramatically over the next few years as the acceptance of blade technology widens and (see Chapter 22, “A Brief Look Ahead”) these small, inexpensive, single-tasking commodity-style computers are introduced to the mainstream. There have been discussions of how to manage clusters of hundreds or even thousands of physically very small nodes that might all be located in a single data center rack. (We would, of course, recommend putting them in more than one rack, as one rack represents a single point of failure.)

Given today’s technology, there’s probably no need to grow clusters beyond 15 to 20 nodes. However, when blades make themselves felt in the market, that recommendation will surely change.

Key Points

- The most reliable takeover server is one that isn’t doing anything except waiting to take over for a failed one.
- You’ll have to expend more energy on management tools and management processes when you’re using failover management software; if

you're going to the effort to make the systems reliable, you have to improve the operational management as well.

- Go through thought exercises first to determine what you need to manage, and what will happen under any operating condition.