Realtime publishers

The Definitive Guide™ To

Windows Application and Server Backup 2.0

sponsored by

Don Jones

Copyright Statement

Chapter 10: When Everything Fails: What's Your Disaster Recovery Plan?

Much of this book has focused on backup and restore rather than disaster recovery. The difference? I regard "restoring" as something you do with a single file, or a group of files, or a single email message, or an entire mailbox—something less than an entire server. It might be a "disaster" that a file was accidentally deleted, but it's typically a disaster for one or two people—not the entire business. A true disaster, in my view, is when an entire server goes down—or worse, when an entire data center is affected.

The reason much of this book has focused on restores is that, frankly, it's what we spend more time doing. It's not all that common for an entire server to fail, or for an entire data center to encounter a disaster. It definitely happens, but what happens a *lot* more is someone needing you to pull a single file or mailbox from backups.

In this chapter, however, I'm going to focus entirely on disaster recovery. Disasters *do* happen—floods, hurricanes, power surges, and so forth can take out entire servers or even entire data centers. One time, I had to deal with a complete week-long power outage when someone ran their pickup truck into the transformer on the corner of our office's property—talk about a disaster. In fact, I'll use that story as a kind of running example of where Backup 1.0 really let me down.

We're going to stick with our Backup 2.0 manifesto because it's just as applicable to disaster recovery as it is to a single-file recovery:

> Backups should prevent us from losing any data or losing any work, and ensure that we always have access to our data with as little downtime as possible.

Old-School Disaster Recovery Planning

You have to *plan* for disaster recovery. It's become so commonplace for us IT folks to talk about the "company's disaster recovery plan" that we often forget that a *lot* of planning is involved. In fact, a really large number of business-level decisions have to go into the plan before we technologists can even begin *our* end of the planning.

Primarily, your business leaders need to decide how much of a disaster they want to be able to survive. Here are some examples of completely different scenarios:

- A server fails because of a hardware issue or catastrophic software issue. This isn't the end of the world, obviously, but depending on the server, it can be pretty bad for business.
- You lose utility power for a short period of time. Simply having a lot of batteries or a backup generator might be a good way to mitigate this kind of disaster.
- You lose utility power for longer than you can practically mitigate using batteries or a generator. This usually means you're looking at bringing at least some of your applications back online off-site.
- A natural or man-made disaster—fire, flood, tornado, hurricane, or something else—strikes, rendering your data center useless. Again, you're probably looking at an off-site recovery to survive this.

The reason your business leaders need to consider these is because some of the solutions can be pretty expensive. They'll need your help in figuring out *how* expensive, so that they can decide if it's worth it to have a recovery plan in place for any given scenario.

You'll also need to rate your data center's services. What can you really do without in the event of a failure? Will *everyone* need to access resources, or will the company be running on a skeleton crew? It's obviously cheaper and easier to plan for extreme disaster scenarios that involve restoring a handful of servers, as opposed to scenarios that will require your entire data center to somehow be replicated off-site.

These kinds of decisions also change drastically between companies of different sizes. If you work for a large, globally-distributed company, then you already have off-site data centers; you just need to figure out how to ensure that one data center can take over for another in the event of a disaster. Smaller companies that operate out of a single location, however, can't get off-site recovery without paying additional for it—and that can be expensive, depending on how you choose to do it. In a strange way, it seems that if you spend enough on IT—like having geographically-distributed data centers in various locations—disaster recovery can actually become a lot cheaper.

> The company I'll be using as my Backup 1.0 case study was a retailer, with a single headquarters and distribution center in the mid-Atlantic region. We had a single data center, which housed every single IT asset we owned—right down to the office phone system. Our stores were, of course, independent and could operate for some time if the home office data center was offline, but without that data center we wouldn't know what products stores had sold, and couldn't practically generate restocking shipments. Our executives wanted to be able to survive anything up to and including a complete loss of the data center, and so we contracted with an off-site recovery facility. The basic deal was, when we called, they'd have a certain set of hardware ready for us, along with telecommunications services. We'd have to take it from there.

Bare-Metal Recovery and Server Rebuilds

In the Backup 1.0 world, we had two ways of dealing with a failed server: Rebuild it, or recover it.

Rebuilding is a horrible thing to have to do, and I don't know of any administrator who looks forward to it. You're talking about starting from scratch: Installing an operating system, installing applications, and so on. Presumably you can restore application data from backups, or you've basically lost everything. Very few companies maintain detailed-enough documentation on server configurations to ensure a completely thorough, accurate re-build, which means there are always a few configuration items that get missed. I once had to rebuild an Exchange Server computer this way, and for two weeks afterward none of our remote users could access the computer. We finally realized that we'd forgotten to re-configure the server to use the non-default port numbers that our firewall was allowing through (which is, by the way, a big argument in favor of sticking with the defaults—less to worry about if you do have to rebuild from scratch).

Rebuilding takes hours, if not days, and it locks down some of your most skilled human resources for that entire period. Because you're usually rebuilding as fast as possible, and under a good deal of stress, you're a lot more likely to make mistakes and mess something up, too.

You also need some *place* to rebuild. In the event of a server hardware failure, that may mean either a new piece of hardware—you did have a complete server just sitting around waiting, right?—or a virtual machine, assuming you have a virtualization infrastructure, and a virtualization host with available capacity.

> The retailer I worked with actually did maintain cold-spare servers, meaning we had two or three servers sitting in closets, waiting to be used in case a production server up and died. We tried to minimize the number of server models in our data center—which frankly reduced our flexibility a good deal—so that we could minimize the number of spares we'd need to keep. This was a few years ago, and virtualization wasn't really an option. We did have maintenance contracts on all our server hardware, but for a few specific servers we couldn't really afford any downtime, so having the spare was a way to cut that downtime by as much as possible. It was also expensive, and we had to have maintenance contracts on the spares, too. We were paying for maintenance on hardware that we didn't normally even use.

How does rebuilding fit our goals?

> Backups should prevent us from losing any data or losing any work, and ensure that we always have access to our data with as little downtime as possible.

Pretty poorly. You're definitely not hitting "as little downtime as possible" with a rebuild, and you're probably going to be missing some data, depending on how old your most recent application data backups are.

Bare-metal recovery is typically faster. The assumption is that you have a complete backup of the entire server, and you want to just dump that onto a fresh server—either hardware or virtualized—to get your server back online. Of course, how much data you lose depends entirely on how recent your most recent backup is. How much downtime is involved depends entirely on how you *made* those backups in the first place. For example, let's suppose your backups are all on tape drives, and you need to restore an Exchange Server. Let's say that Exchange and Windows together take up about 10GB of disk space for operating system and application files, and you've got another 400GB of mailbox data. That's 410GB total. I'll give you the benefit of the doubt and suppose that you have the very latest in DLT tape backups, and can fit an entire full backup on one tape. I'll assume you get typical 2:1 compression, meaning you'll be able to pull all that data off of tape in about an hour (assuming a tape data transfer rate about 215GB/hour of raw, compressed data). Assuming you can quickly lay hands on the right tape and that you have all of your recovery boot discs handy, you can probably have the server back online in a couple of hours—losing only the data that was created or modified since the tape backup was made, of course.

Is that reasonable for your company? Two hours isn't bad, but I've never been able to get a CEO to see it that way. Worse, most of us in the Backup 1.0 world are only grabbing backups every night, at best, meaning if the server dies at lunchtime you've lost around half a day's work. Nobody is going to be pleased about that; many of them may *accept* it because they've been conditioned to believe that it's the best that can be done—but they're not *happy.*

Of course, if you're not grabbing a full backup every time you back up a server—and most of us aren't—then the recovery is going to take longer as you shuffle tapes. Last weekend's full backup, a few incrementals from during the week, and so forth—it adds time, no matter how fast your tape drives are. And of course, if you're not running the latest speed-demon tape drives, you're not going to be pulling that 420GB of data off of tape in an hour.

Why Incremental and Differential Backups Are No Fun

If you think about it, both incremental and differential backups save us time *only* during the *backup* phase; when the time comes to use those backups, they actually slow us down.

You have to start with your most recent full backup, which means you're completely recovering the server—say, 400-ish GB of data in my Exchange Server example. Then you start in with the most recent differential, or start applying incrementals, depending on how your backup plan works, Every incremental is *overwriting data you've already restored,* and each successive incremental is probably overwriting data from the *previous* incrementals—that is, after all, the whole point of incrementals. My 400GB Exchange example usually resulted in about 10GB of incremental backup data every night, meaning that a Thursday afternoon crash means I have to recover around 460GB of data, increasing my time and the amount of tape-loading I have to do.

Full backups are the best choice—but getting a full backup of every server, every night, is usually impractical. And you're *still* at risk for all the data that gets modified or created during the day, even if you are able to grab a full backup of every server every night.

So what's the problem with this plan? It honestly takes too long. An hour is your absolutely best-case-scenario for restoring 400GB or so of data. I used to know companies who would actually spin up more servers, just so each server contained less data, and could be recovered more quickly; when they eventually realized how much more time and money they were spending on servers and maintenance, they consolidated everything and just decided to try and find a way to make backups faster.

The underlying problem with this plan starts, of course, in how the backups are made. Recovery takes a long time because we take shortcuts on the backup side of things, in order to get all our data backed up during a maintenance window or something. Backing up 400+ GB of data is a pretty big deal when you have multiple servers like that to worry about; we *have* to take shortcuts like only doing weekly backups, or using incremental backups, just to get it all backed up during the time we have to work with.

But think about something: Let's say you have a server that generates about 10GB of data each night for an incremental backup. That means you're creating or changing around 10GB of data during the workday. Let's go worst-case, and suppose that those changes occurred during a six-hour period (everyone showed up to work late, took a long lunch, and went home early—happens all the time in your office, right?). That means you're changing about 1.6GB per hour, or about .02GB per minute. That isn't actually that much data. Assuming the changes are evenly spread out (which I realize they're not, but pretend to make the math easy on me), you're changing .0004GB of data per second—that's 500KB per second, if I'm getting my decimal places correct, and backing up 500KB every second is hardly even work on a modern network. Assume each server is included on a dedicated network that's only used for backup data; if every server was generating 500KB of data each second, a speedy 10Gb Ethernet network could handle a few hundred servers with ease (actually, something like 2,000 servers if I haven't misplaced a decimal point, but "few hundred" seems more practical). Figure 10.1 shows what I'm talking about, with separate networks to carry user traffic and backup traffic.

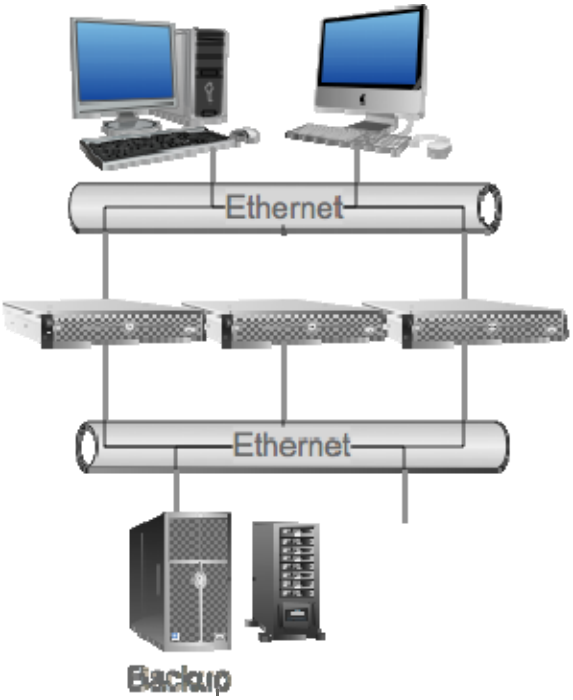

Figure 10.1: Creating a dedicated "backup network."

My point is not really about precision math; it's that we're not talking about impossible-to-achieve numbers, and this is exactly what Backup 2.0 proposes: *Continuous* data protection, capturing changes *as they occur,* rather than waiting until they build up into an enormous pile of data that we somehow have to grab during a one-hour maintenance window. Plus, by grabbing data *continuously,* we're at risk for losing very little data in the event of a failure.

Off-Site Recovery

When the entire data center is affected, off-site recovery can be a solution for getting at least some of the business back online.

> Here's how off-site recovery worked at that retailer I mentioned: We drove over to the recovery facility (which by the way was in the same state, so a regional disaster would probably have put it out of commission too). We would have called them to let them know we were coming, and since they handled our backup tape storage, they would start rooting around and getting our latest tapes for us. We'd show up and start feeding tapes to tape libraries, and start restoring about a half-dozen servers. We didn't have a goal of restoring *every* server from our data center—we had identified servers that the company needed to operate in a sort of worst-case scenario, and we worked on those servers first. Anything else would be restored later, as we got the time.
>
> In our practice runs, this took about eight hours. We were delighted to get a new, faster set of tape drives (after making sure the recovery facility had them, too) that cut the time down to four hours. For our needs, this was basically sufficient: It got us up and running quickly enough to start collecting data from our retail stores again, and to start processing restock shipments. Assuming our sole distribution center wasn't completely underwater or something.

Our technique was what I now call a *cold recovery site*. In other words, we'd show up and have nothing but hardware, some telecom lines, and a box full of magnetic tapes. We'd take it from there, essentially reconstructing a portion of our data center almost from scratch, using fairly recent backup tapes. We sent data backup tapes off-site every morning, and did a weekly complete backup of every server, so our worst-case scenario had us missing about a day of data at most.

We were pretty pleased with ourselves over this plan: A day of data at risk and a day to get back online seemed pretty awesome. Our executives were okay with the plan, too, because—again—they'd been *conditions* to accept that as being the best that could be accomplished. At the time, which was years ago, it *was* about the best that could be done. The problem is that many companies *still* operate that way—they haven't re-evaluated their conditioning in light of new technologies and techniques. They haven't sat down and stated to themselves:

> Backups should prevent us from losing any data or losing any work, and ensure that we always have access to our data with as little downtime as possible.

And then asked if they can come any closer to these goals.

Backup 2.0's Disaster Recovery Approach

Let's quickly summarize how Backup 2.0 solutions should, in my view, operate. You install some kind of agent software on your servers. This "shims" the operating system's file system, giving the agent a sort of "preview" of every disk block change that the operating system is writing to disk. The agent "clones" that disk block in memory, and transmits it to a centralized backup server. That server keeps a database of all the disk blocks, tagging each one with a timestamp and the server it originally came from. The backup server also does some compression, de-duplication, and other magic to help reduce the amount of disk space it actually needs to consume for that database (as I discussed in the previous chapter).

The result is that you can push a button, and query—from the database—all the disk blocks that go with a certain server at a certain point in time. You're not limited to the granularity of maintenance windows and tape backups; every change is caught nearly in real-time, and you can choose to restore to any particular moment.

All of those disk blocks live on fast, random-access disk storage—in reality probably a RAID cabinet either directly attached to the backup server or perhaps living on a Storage Area Network (SAN). I imagine that you'd keep some specified period worth of disk blocks in that fashion—say several weeks. You'd periodically dump older disk blocks—ones that had since been overwritten by newer ones—to a tape for off-site storage or archival, and you'd delete those disk blocks from the disk of the backup server. I frankly can't imagine wanting to recover a complete server to more than a few hours in the past, let alone entire days or weeks, but I'm sure there's a business scenario out there somewhere that would justify retaining the ability to restore that far in the past.

Bare-Metal Disaster Recovery

Let's agree that the "rebuild the server manually" approach is too time-consuming and stick with bare-metal recovery. In a Backup 2.0 world, all of your data is sitting on nice, fast, random-access disks. So your time to get the data off of tape is exactly zero: You just need to get the data from your backup server's disks to the disk of a physical or virtual server. In other words, a bare-metal recovery is really just a big, fancy file copy. Now, a fast 430GB/hour tape drive is a nice thing to have, but that's only about 7GB per minute. A 10Gb Ethernet network, on the other hand, is much faster—and 10Gb Ethernet isn't at all uncommon in today's data centers, having been introduced in late 2002. In fact, both 40Gb and 100Gb Ethernet are currently in draft specifications. But a mature 10GB Ethernet network moves 10 giga*bits* per *second.* There are eight bits in a byte, so that's 1.25 giga*bytes* per second. In a minute, that's 75GB—about ten times as fast as a super-fast tape drive. Even assuming the network is only 80% efficient, you're still at 60GB per minutes, or a bit more than 8 times faster than a very fast tape drive. If you needed to restore 420GB to a server, it would take about *seven minutes* across a network like that. Seven minutes to restore a complete server, either to physical hardware or to a virtual machine. When 40Gb Ethernet is available, just a restore would take under *two minutes.* Two minutes. Can you imagine that conversation? "The server's what? Down? Oh, can I put you on hold for like two minutes? <music plays> Try it now."

A Note on Math

By the way, I realize that the numbers I'm tossing around here represent a perfect world, and they're perhaps glossing over some assumptions—like assuming you have built a disk subsystem for your backup server that can pull data off the drive at 60GB a minute, and that your backup server's operating system and software can spit data onto the network that fast. The point is really to illustrate the vast performance gulf between tape drives and disk+network data transmission; obviously, one of the things you should evaluate as you start looking at Backup 2.0 solutions is how fast they can perform in real-world conditions.

The point is that Backup 2.0's recovery scenario looks better because its *backup* scenario is better. By grabbing data *as it changes,* we get a complete up-to-the-minute backup for all of our servers. We can then send that data—from any given point in time, no less—back to a server, or to an alternate server, whenever we want to, in less time than it takes to make a proper cup of coffee.

Virtualization as a Disaster Recovery Strategy

Once you've turned bare-metal recovery into something approximating a gigantic file copy operation, you can really start to get creative with your recovery options.

For example, why not just engineer a "pad," or extra capacity, into all of your virtualization host servers, so that each one could support a few more virtual machines than they normally ran? If a physical server dies, you could just spin up a new virtual machine—which takes a few seconds—and restore the failed server to that virtual machine. You might not get amazing performance that way, depending on what the failed server did (some applications just don't run well on virtual machines), but you'd be up and running in a reduced state until the server's original hardware could be repaired. Then you'd just restore the server back to that original hardware during a maintenance window, and you'd be set to go.

Virtualization, combined with Backup 2.0 techniques, offers a practically endless array of recovery scenarios. Recover physical machines to virtual ones. Recover virtual machines to different virtualization hosts. Recover virtual machines to physical machines, if needed. You get a ton of flexibility, and it can all be done *quickly*, provided you've built an infrastructure designed for this kind of operation.

Restoring Applications to Someplace Else

But what if your entire data center is affected? How can Backup 2.0 serve then? The first key is getting that backup data off-site, because if you lose your data center then you're losing your backup server and all its contents, too. Tape is obviously one way to get that backup data safely off-site, but it's reverting to a Backup 1.0, snapshot-style approach, meaning you'll always have data at risk. That might be okay for your organization, and it's certainly an economical approach. I outlined this idea in the previous chapter; Figure 10.2 is a reminder of what this might look like.

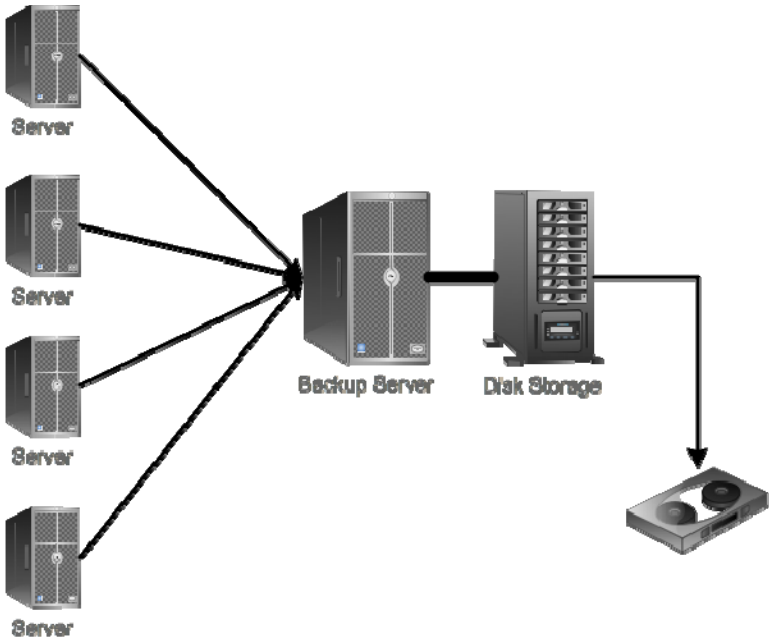

Figure 10.2: Moving backup data off-site via tape.

The recovery plan here is to run out and get your tapes, and then take it to wherever you plan to execute your data center recovery.

"Cold" Recovery

That scenario is essentially the same as the "cold site" recovery that I outlined earlier. You're still going to wait for data to stream off tape as you recover your backup server, and then you'll be copying multiple sets of data across a network to get your servers up and running again.

A slight modification of this approach is to not back up your backup server to tape, but rather to export, from your backup server, *disk images* of the servers it's been backing up. That way, the tape contains images that could instantly become virtual machines, or be used to recover to a physical machine, as soon as the data gets off of the tape. Figure 10.3 illustrates this approach.

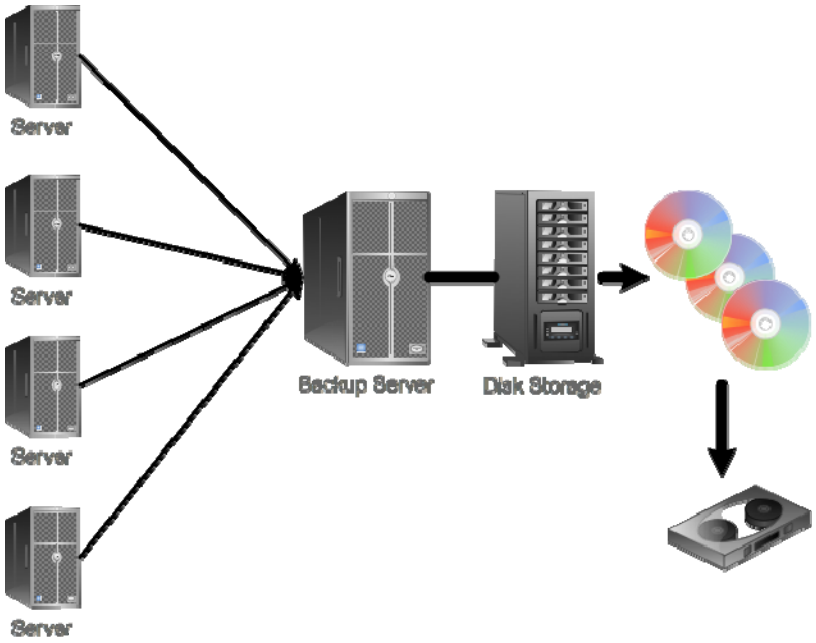

Figure 10.3: Exporting disk images to tape for faster recovery.

I'm not trying to present this as an ideal solution, because I don't think it is. It is an *economical* solution, though, and it will work with the widest possible variety of off-site recovery facilities. Basically, if they can provide you with a tape drive and a virtualization host, you're in business.

"Warm" Recovery

With this approach, you replicate your backup data off-site over a network connection, rather than on tape. There are two main advantages here:

- Your off-site data is much more up-to-date than it would be if you used tapes.
- Recovery is faster because you won't necessarily be restoring from tape; you'll be recovering from disk-based storage at the recovery facility.

Figure 10.4 shows how this might work.

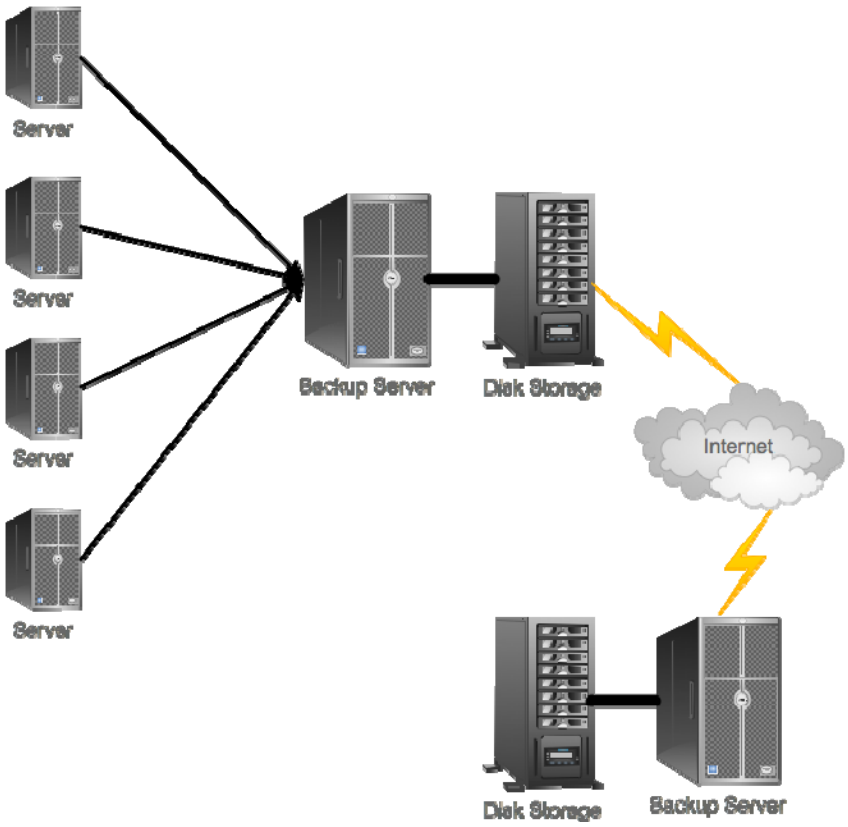

Figure 10.4: Replicating backup data off-site.

There are some pretty obvious concerns with this approach, first and foremost of which is probably WAN bandwidth. You're obviously going to need a lot, but you can mitigate and manage this a bit. For example:

- Compression and de-duplication will reduce how much has to get transmitted—by as much as 80%, according to some vendor claims.
- Throttling can reduce the impact on production WAN usage. Let the backup server transmit using "leftover" bandwidth.
- The backup server might not replicate *every* disk block. For example, a lot of server operations will continuously write the same disk block many times in succession. The backup server might wait for an idle period, and then transmit the last version of a particular disk block. You'll lose some granularity of recovery that way, but you might not care when it comes to off-site recovery.
- I used "Internet" in my model, but you might not actually use the Internet. Dedicated WAN bandwidth to the recovery facility would provide a dedicated path for the data to be uploaded, although you'd obviously have to be able to justify the cost of that dedicated bandwidth.

Let's say you generate 100GB of backup data a day. If you reduce that by 60% for compression and de-duplication (even though many vendors claim 80%), that's 40GB to transmit. If just 5% of that was "repeated" disk blocks, you could potentially have to just send 38GB. Assume that's spread evenly (or at least transmitted evenly) over a 24-hour period and you get 1.6GB per hour, meaning you'd need an uplink speed of about 3.5Mbps, which is roughly a couple of T1 lines in the US (or even an Synchronous DSL line in some areas). Not cheap, certainly—about $6500/year in many US cities using dual SDSL lines—but clearly not impossible, and for some businesses it would be worth the price. Think about it: You could call your recovery facility and let them know to activate Mission Recovery. Their copy of the backup data could immediately start streaming to virtual machines, and in a few minutes your most critical servers would be online at an off-site facility, with perhaps a few hours' of data loss being your maximum risk. You *can* engineer this kind of solution.

> **Note**
>
> Exactly how you replicate this data depends a lot on the solution stack you've assembled for backups. Some backup solutions might have native support for this kind of off-site replication, including bandwidth throttling and other WAN-friendly features. In other cases, you might need a separate solution to handle the data replication. You'll need to consult with your off-site recovery vendor to determine what technologies they can work with, as well.

"Hot" Recovery

Hot recovery goes a step beyond warm recovery, as the name implies. Rather than merely storing your backup data at the off-site facility, they're restoring that data to physical or (more often) virtual machines *as you transmit it to them.* That means "recovery" is just a matter of starting those machines—you don't even need to wait for a recovery operation to complete, because it's being done constantly. Figure 10.5 illustrates this idea.

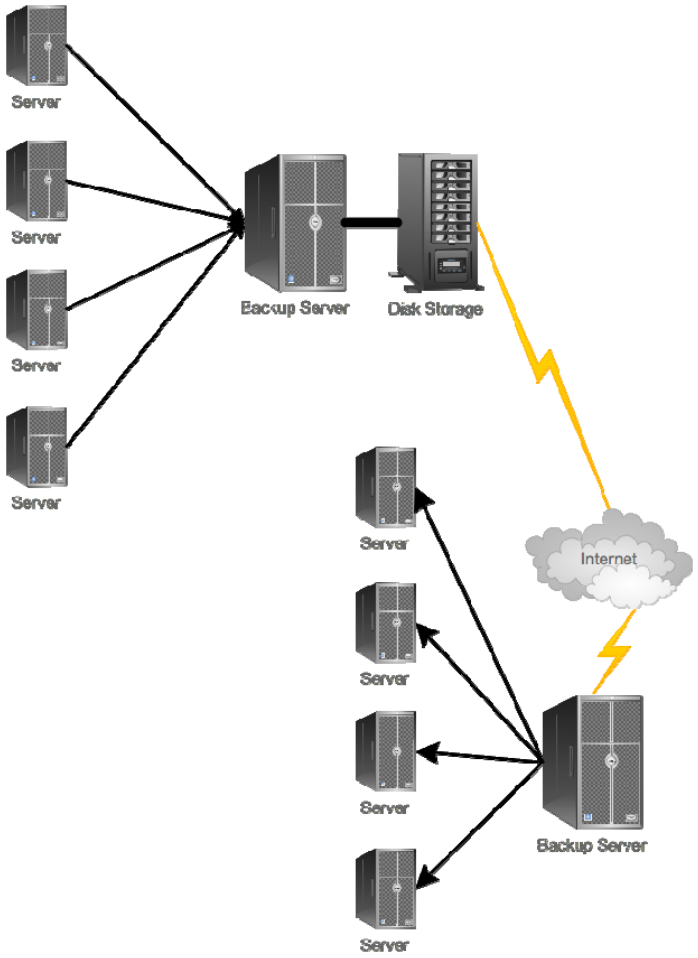

Figure 10.5: A "hot recovery" model.

This is obviously a pretty expensive choice. You'll likely need more upstream bandwidth to the recovery facility to make this work, and the recovery facility is obviously going to want to charge a bit more to maintain this kind of arrangement. However, for *some* services in their data center, *some* companies may find the price justified.

> **Note**
>
> *Service* is really an accurate term here. Things like email, particular applications, and so forth—these are all services that your data center delivers to your users. Some services may be worthy of expensive precautions like a hot recovery model (although that becomes a lot more feasible if your company already has its own multiple data centers); other services might warrant warm recovery. Some services might not be part of your disaster recovery plan at all. It all depends on your exact needs.

Today, most hot-recovery vendors utilize virtualization, which has markedly lowered the cost of such services and put them within reach of a much broader range of businesses.

Note

Backup 2.0 isn't the only way to achieve a hot site recovery model. Depending on the applications and services, you may be able to use geographically-dispersed failover clustering and other techniques. Those are beyond the scope of this book, but I do want to make sure you're aware that there are many other options, especially if your "off site" facility is simply another one of your own data centers.

Disaster Recovery as a Migration Strategy

Believe it or not, the ability for Backup 2.0 to handle bare-metal recovery can make a *wonderful* migration tool. If you're already backing up your physical servers using Backup 2.0 techniques, you can easily "push" a recovery to a virtual machine. Figure 10.6 shows what I'm talking about.

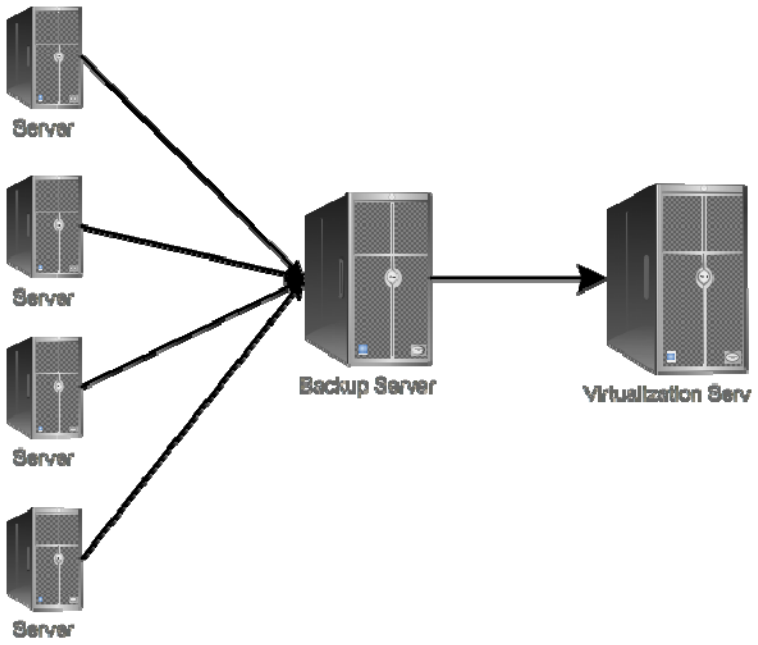

Figure 10.6: Disaster recovery as a migration technique.

Because the backup server doesn't care *where* it restores to, you can enable some pretty neat migration options:

- V2V, or Virtual to Virtual: Migrate servers between different virtualization platforms or between virtualization host servers.
- P2V, or Physical to Virtual: Migrate physical computers to virtual machines.
- V2P, or Virtual to Physical: Migrate virtual machines to physical machines—great when you realize that a particular service needs a whole computer to itself.
- P2P, or Physical to Physical: Migrate services from one physical machine to another, helping eliminate older hardware and move to newer hardware.

You wouldn't need to mess around with the "P2V" tools provided by virtualization vendors, which sometimes require the source server to be taken offline. Instead, you can base the migration off of the up-to-the-minute backup available on your backup server. In fact, you can even repeat the migration as many times as needed, doing trial migrations over and over until you're satisfied, because the source server doesn't need to be involved.

Coming Up Next…

We've covered quite a bit of ground in the preceding ten chapters. It's time to circle back and look at some of the original problems with old-school backups, and see what we may have solved with a "2.0" approach. It's also time to look more precisely at what's involved in redesigning your business' backup strategy, and outlining a methodology for determining the real cost of a redesign. We should also look at some of the more human or, shall we say, *political* factors involved in a redesign. That's all coming up in the next chapter.

Then, in the final chapter of this book, I'm going to look at some real-world stories of Backup 2.0 techniques in action. Hopefully it'll give you a feel for the variety of ways in which companies like yours are utilizing Backup 2.0, and inspire you to investigate your own solution.