Benefits of going beneath the file system:

- Avoids sensitivity to odd facets of the file system. (Devices, long file names, funny characters in file names, holes in files, etc. etc.)
- Allows incrementals to work correctly. In order to keep archive runs from causing all files on the system to appear to be active, most archive programs modify the file access time after they have accessed the file. This keeps access time accurate, but updates the inode change time. There are other operations, like changing the owner or permissions of a file, which update only the inode change time. Programs that go through the file system cannot avoid this problem; programs that go beneath the file system do not have any effect on the access time in the first place.
- Allows backups by users other than root; permission to backup a file system depends on the device permissions, not the permissions on individual files.

Most people find the advantages of backing up through the file system more comprehensible, and in general more convincing. In theory, the problems with permissions and with the use of mtime instead of ctime for incrementals are sufficiently minor that a robust and well-designed program working through the file system should be preferable to one that works beneath it. In practice, such programs are rare if not mythical. The well-known programs that work through the file system, including `tar`, `cpio`, and `pax` are not even mildly robust. Furthermore, even a program that was as perfect as possible would still run into problems with the file system. The file system is designed to conceal some information that is necessary in order to do effective backups (for instance, it is designed to conceal the existence of holes).

2. Do not design to exactly meet capacity. For instance, if you are scheduling machines to do things on a cycle with one machine per day, make the cycle large enough to leave empty days.

3. Put in override options, so that people can do extra backups on individual machines or filesystems within the system.

4. Always have a redundant device to back up to.

8.7 Maximize the Predictability of the System

It should be possible to summarize the conditions under which you can restore a file for an average user. Any given file should be saved on backup for the same amount of time as any other file on the same file system. This is easy for simple systems; on the other hand, if you do disk-to-disk backups followed by copies to tape, the interaction of the the disk-to-disk schedule with the disk-to-tape schedule may create unexpectedly good or bad days to modify files on.

9 Issues in the Selection of a Transport Program

9.1 Through the File System vs. Below the File System

Some programs, like `tar`, `cpio`, and `bru`, access files to back them up by using normal UNIX access methods (going through the file system); others, like `dump` and `dd` access files to back them up by reading the disk more-or-less directly (going below the file system). Going through the file system is somewhat more flexible, since you can back up anything you can get a standard file system interface to (including NFS or RFS mounted remote file systems). On the other hand, it exposes you to whatever oddities and bugs there are in the file system interface, and it may cause interesting interaction with programs (like some NFS server implementations) that do not themselves work through the file system. Going below the file system avoids most of the problems with odd features of the file system, but makes programs more susceptible to active file systems. Both ways are equally vulnerable to vendors who re-arrange the file system. (For instance, Hewlett-Packard's new access control lists are not backed up by either `dump` or `tar` as supplied by Hewlett-Packard; `dump` believes that all disk blocks are either inodes or data pointed to by inodes, and can't see the access control lists at all, whereas `tar` writes a format that has no place to put the information. For reasons unconnected to the file system, `dump` is fixable but `tar` is not.)

Benefits of going through the file system:

- Allows backup of file systems other than locally-connected disk.
- Allows backup by directory tree instead of by file system; you can backup part of a filesystem, or several file systems at once.
- Reduces sensitivity to active file systems (the file system itself is designed to present a consistent view of an active file system).
- Usually increases portability.

   (c) Other peculiar filenames; some programs have difficulty restoring files with names that are legal but unusual (names containing spaces, or 8-bit characters, for instance).
   (d) Symbolic links; some programs will copy the target of a symbolic link, rather than the link itself. This may result in a backup that is too large to restore (and will certainly cause confusion and unhappiness).
   (e) Large numbers of hard links.
   (f) Long targets of symbolic links.

3. Filenames that include "/"; "/" is not a legal character in a UNIX filename, but many implementations of NFS allow clients to create such files. They may or may not get backed up, but they can't be restored except by extreme measures.

4. Design errors in the system, including:

   (a) Failing to allow for complete failures; requiring a restore program which is not shipped with the operating system and which is only stored on a tape that it must be used to a restore. If you lose everything, you can't read the tapes to restore them. This is easier to do than you might think.
   (b) Rewinding tapes between writes.

Avoiding systemic errors is difficult. Knowing that they're possible is the first step; testing by doing unnecessary restores is the next one. You don't want to find out where the problems are when the disk dies.

8.6 Maximize the Flexibility of the System

There are two sorts of flexibility that you need; temporary, to deal with extraordinary events, and permanent, to deal with changes in your computer system.

Extraordinary events do occur. Sometimes you may be unable to meet the required schedule (to take an example from life, you may have to postpone all backups one day because all your machines are shut down due to an earthquake). Sometimes you may need to increase the frequency of backups on one particular set of files. You will inevitably need to back up more disk space than originally planned for. You may also have occasions on which very large amounts of data change at once (in the worst case, you may lose a file system and restore it). Your system should be able to cope easily with all of these.

Expansion is also inevitable. There is no such thing as enough disk space. There is also no such thing as enough CPU power, so people will buy more machines, and newer ones. When you put together your backup system, you should know what you will need to figure out how far it will expand as is, and what will need to be done when you reach the end of that tolerance.

Ways to meet this goal:

1. Use widely separated levels; do not normally use the highest level. (For instance, rather than using levels 0, 1 and 2 or levels 0, 5, and 9, use 0, 3 and 7 so that you have spare levels you can use at every point in the hierarchy.) If you are designing the levels, write in more than you need.

There is also a complex way to avoid it. You can juggle things so that the incrementals are incremental to different previous dumps. If you do level 0_1, level 0_2, level 2_1, level 2_2, level 4_1, level 4_2, you minimize the effect of losing any individual dump. If the level 2_1 is missing, you can use the level 2_2 and its corresponding 4_2. This system can be implemented with `dump` by interchanging /etc/dumpdates files, and provides a nice additional redundancy. It also confuses people nearly to death, which is a major drawback.

8.4 Adding Checks and Balances

Rather than putting your trust in one person or one program, provide double-checks. For instance, run a separate program that checks to see whether or not backups have been done, and reports when they have not. (It's important that it only report exceptions; if it always says something, people will drop out of the habit of reading it.) This program should send reports to not only the people who are directly responsible for doing the backups, but also other people.

The backups should also send out status messages to multiple people. These messages should report what happened, even if there are no exceptions. Not only does this help give people a warm fuzzy feeling, it protects against failure of the checker. (If the checker fails altogether and says nothing, it appears to have approved things.)

8.5 Systemic Errors, and Avoiding Them

The most common causes of systemic errors are:

1. Dumping active file systems; if you dump a file system that is in use at the same time every day, you risk being synchronized with programs or people that modify the same file at the same time every day. Most people assume that a file that is damaged because it is being modified will not always be in that state. That is, they assume that doing backups of active file systems always creates transient errors. Check this before believing in it.

2. Flaws in the transport program, including problems with:

    (a) Long file names; some transport programs are incapable of dealing with pathnames over 100 or 128 characters long. Since most UNIX versions will allow pathnames of up to 1024 characters, it is not difficult for users to create whole subtrees that will not get backed up. People who are using shells with filename completion or icon-oriented programs are more apt to do this.

    (b) Files with holes in them; most UNIX file systems allow programs to create files which represent large stretches of nulls without writing them to disk. This feature is often used by core files, and by dbm databases. It is extremely important that programs used for backups do not fill in these holes, either in writing to tape, or in restoring from tape. Since the / file system is often smaller than the memory size of the machine, and frequently contains core dumps which are the same size as the memory, filling in the holes will result in being unable to restore the file system.

don't exist. There are some file systems for UNIX machines that don't have this problem (AFS, for instance), but they aren't in wide use.

If you are using `dump`, the only truly safe way to do backups is to ensure that the file system is idle, either by unmounting it, or by bringing the machine into single-user mode. On a machine that may have local processes using the file system, the only way to guarantee that you can unmount it is to bring the machine into single-user mode. (If all uses of the file system are always through NFS, it will be unmountable.) In order to do automatic unattended backups in single-user mode, you create a file in the root partition, and you have the startup files check for the existence of that file, and first delete it and then run the backups if it exists. Be sure not to create the file until you are ready to have the dumps done; you don't want the machine to do backups after an accidental crash.

`dump` is by no means the only program that has problems with active file systems. Programs differ in what they do when they encounter active files, but most of them will either create corrupt archives, or skip the file. Do not assume that active backups are OK because you are not using `dump`, or because a salesman told you so.

It is obviously impractical to do all backups on idle file systems, and it is perfectly reasonable to back up active file systems, as long as you are aware that it will introduce transient errors, in the form of incomplete and/or skipped files, from time to time. The lower the activity on the file system, the less the risk, so schedule backups at low-usage times. Speeding up the backups also helps, by narrowing the window during which activity matters. (It may also have second-order effects; Ohio State's speed-modified dump often took so much memory, CPU, and disk controller capacity that nobody else was capable of modifying the disk while it ran.)

8.3 Increasing Redundancy

If you have copies of a file on multiple tapes, you can afford to be calm when an operator decides to juggle them, drops one, and shatters the case. The obvious way to achieve this effect is to duplicate tapes. Tape duplication between tape drives is slow, and the duplicate will have all the errors on the original, plus any introduced in the duplicating process. If you happen have a tape duplicator handy, it may be reasonable to use it, but going from drive to drive probably isn't.

Doing more backups is the next most obvious gambit, and is usually a better idea. Doing repeat backups at the same level not only gets the same files on multiple tapes, it also decreases maximum loss. In fact, simply reducing the number of levels you use, without increasing frequency, will usually increase redundancy.

When you are doing multiple backups at the same level, you need to keep careful track of what is incremental to what. For instance, if your original schedule is level 0, level 2, level 4, level 6, changing it to level 0, level 2, level 2, level 4 makes the level 4 incremental to the second level 2. If you lose that level 2, you may not be able to do a normal restore of the level 4, even though you have a perfectly good level 2 left. Doing two level 0s back-to-back is not a major win in redundancy because of this effect. The simple way to avoid this effect is to flatten from the highest level down, giving you level 0, level 2, level 4, level 4; the level 4s are redundant, since nothing is incremental to them.

3. Run `fsck` before doing backups.
4. Increase redundancy.
5. Add multiple checks and balances to the system. For instance:
   (a) Add a check of whether the backups have been done that runs separately from the backups themselves.
   (b) (Particularly in the case of automated backups) Inform multiple people of the status of backups as they complete.
   (c) Label the media with both human-readable and machine-readable labels, and read them both.
6. Use devices that you have more than one of; use formats that can be read on more than one system.
7. Automate backups.

8.1 Verifying Backups

In a perfect world, you would always be able to write a tape, and then verify that the tape had all the right data on it. This is an imperfect world, and it's almost impossible. You can only check a backup against a file system if the file system is the same now as it was when you wrote the backup – that is, it has to be an idle file system, and it has to be idle long enough to do the backup and the verify, which will take as long as the backup did. If you are in this situation, you can certainly create a verification program, even if your transport program doesn't have one (which it probably doesn't).

Most of us cannot actually keep file systems idle for that long on any regular basis. In this situation, the best you can possibly do is to verify that the tape has valid data. The best way to do this is to actually read the entire tape. Some people attempt to do this by getting a table of contents of the tape, or by trying to read the last file on the tape; whether these accomplish anything useful depends a lot on the transport program. Using `dump`, for instance, the entire table of contents is at the head of the tape, and reading the last file skips the intervening ones completely; even if you check both, much of the tape could be garbage without your noticing. It is slower, but more reliable, to read every single file.

If possible, you should verify removable media on a drive different from the one that wrote them. Heads can drift out of alignment slowly, making drives that write media only they can read, and that only for a little while after they wrote them. This is almost impossible to automate, which makes it extremely burdensome.

8.2 Doing Backups on Idle File Systems

This is the good old "active dumps" question. The design of the UNIX file system makes it almost impossible to do reliable backups of a file system that is changing.[4] The primitives that would allow you to lock things down simply

[4] To be pedantic, it makes it completely impossible to guarantee that a backup is complete and consistent, while finishing in a finite amount of time. But if you try hard enough, you can come real close.

mounted, but in many cases there is extra overhead in restoring incrementals. An incremental restore may need to remove files, for instance.

7 Minimize the Resources Used For the Backup System

This includes minimizing the time that backups take, the amount of systems staff attention, and the amount of backup media used. Time in particular becomes a limiting factor on very large systems; once you have 14 gigabytes of disk space, it is all too easy to design a system where you cannot finish one set of backups in time to start the next. In these systems you can also easily run into a limitation in physical space, i.e. more tapes than places to put them.

Ways to meet this goal:

1. Make the backups run automatically.
2. Compress the backups in some fashion.
3. Run multiple dumps to the same tapes.
4. Use fast media.
5. Use media with a small form-factor.
6. Minimize network usage; keep backup media local to the machine being backed up.
7. Schedule the backups for times of particularly low network and CPU load.
8. Use a transport program that has been optimized.

8 Maximize the Reliability of the System

Backup systems are prone to failure at several points, including damaged/lost media, forgetful human beings, and files that the transport programs cannot cope with. These problems come in two forms; transient failures, and systemic failures. When the tape drive catches fire, that's a transient failure. When you write multiple dumps onto the same tape, rewinding the tape in between so that they all write on top of each other, that's a systemic failure[3].

Transient failures, while nasty, are both less worrisome and less fixable than systemic failures. You can compensate for them adequately by adding redundancy, although you will of course also want to avoid as many as possible. Systemic failures should be searched out and avoided; if this is not possible, users should at least be warned about them.

Ways to reduce transient errors:

1. Verify backups after they are written. If possible, verify them on a device other than the one they were written on.
2. Do backups at times of low filesystem usage, or forcibly prevent file system usage while machines are being backed up.

[3] Yes, the examples are drawn from personal experience

6.4 Keeping Backups On-Line

For speed, nothing beats having the backup on-line as a file system. Straight disk-to-disk copy is about as fast as it gets. Unfortunately, it requires lots of disk space, and it makes your backups vulnerable to many of the same problems as your originals. Furthermore, copying file systems disk-to-disk is not as easy as it looks; you basically have to do a backup piped to a restore. Doing a full restore loses some of its speed in the pipeline

Most people who use on-line backups simply run their backup program to a disk file instead of a tape, often compressing the file afterwards, and then copy the result to a tape. Using a disk file instead of a tape buys some speed. Compressing it loses speed and adds vulnerability. Moving the compressed backup from disk to tape almost always results in a backup that is much slower to restore, particularly if you go the route of backing up the partition full of backups. Once you have backed up a partition full of compressed backups, getting a file requires doing a restore, an uncompress, and another restore. Unless you know that most of your restores are going to be from files still on disk, this is probably not worth it.[2]

6.5 Deleting Restores

When you do a full restore from multiple levels of backups, there will be files that have been removed or renamed between the backups. A deleting restore is one that is capable of removing files that have gone away, as well as adding files that have been created. `restore` is capable of deleting restores; programs like `tar` and `cpio` are not. If you have to do a restore that involves multiple levels without deleting, you are likely to have to do a lot of juggling by hand, since there will be more files restored than originally existed.

6.6 Reducing the Number of Levels in the Scheme

If you do a level 0 on Monday, a level 1 on Tuesday, a level 2 on Wednesday, and so on through the week, and your disk collapses into a heap of slag on Sunday night, you will have to restore all seven backup tapes onto the replacement disk. This is one of the shortcomings of the true incremental scheme. If the per-tape overhead (the time it takes you to notice that a new tape is needed, find the new tape, unmount the old tape, mount the new tape, and the time it takes the program to read the initialization information off the new tape) is reasonably low, this may not be a problem. If you are using `restore`, you are in for the long haul. Making the dumps less incremental uses more tape, but may make a dramatic difference in the time the backup takes. (Even when it doesn't make a large difference in the time, at 2:00 am the difference between three tape mounts with three hours between them and nine tape mounts with one hour between them is the difference between three pretty good naps and nine hours of misery.)

6.7 Increasing the Frequency of Level 0s

The more frequently you do full dumps, the less data has to be added to them from incrementals. Not only does this usually reduce the number of tapes

[2] I was actually once asked to restore onto a Sun a file that turned out to have been backed up by running `dump` from a Hewlett-Packard onto a VAX filesystem, compressing it, and dumping the VAX filesystem. It proved to be faster to recreate the document.

1. Keep catalogues of which files are on what tape.
2. Make it possible for users to restore their own files.
3. Use fast media.
4. Keep backups on-line.
5. Use a system capable of deleting restores.
6. Reduce the number of different levels you use.
7. Increase the frequency of level 0s.

6.1 Keeping Catalogues

Most of the overhead in restoring an individual file is finding it. Users are rarely certain enough of what they changed when to let you choose a tape immediately. (Many times they don't even know what the file was named or what directory it was in for sure.) A catalogue of what was dumped when allows you to quickly figure out what tape the file is on. Most commercial backup programs will do this for you; if you are using `dump`, `tar`, `cpio`, or some variant, you will have to roll your own. The main drawback is that you have to keep the catalogue somewhere, presumably on disk, and it can run into large amounts of space quite fast.

6.2 Letting Users do Their Own Restores

A system where users can restore files themselves, safely, reduces wear-and-tear on everybody. The users do not have to wait for system administrators to get around to them, and the system administrators do not have to explain why it's taking them three days to find the time to restore a .newsrc. Remember, however, that this happy state of affairs only fully applies when the user can find and mount any necessary media. Otherwise, you have simply shifted the problem from a request for a restore to a request to have a tape found and mounted. In general, letting the users wander through the backup tapes at will is not a good idea.

6.3 Using Fast Media

If your restore program is limited by the media speed, increasing that speed will make all restores, full or partial, faster. The question to consider is whether faster media will make a cost-effective difference. For `restore` in particular, the difference between a normal half-inch tape drive and a fast half-inch tape drive is probably irrelevant; the tape drive is only going to be a limiting factor for small restores. The speed difference between a DAT drive and an Exabyte drive is primarily in the fast seek speed, and will make a significant difference if and only if the restoring program is capable of using fast seeks. If you are restoring a full file system that's at the beginning of a tape, it's probably irrelevant.

cope with this; the user who wants a file that was deleted 3 years ago should not be surprised if you cannot get it. Before you get asked for that file, decide how long you're going to keep backups for, and then make certain that you do effectively keep them that long.

Many people keep only selected backups. If you keep daily backups for 2 weeks, weekly backups for 2 months, and full backups (done every month) for 2 years, then you are increasing maximum possible loss as time passes. A file needs to have existed for at least a week to be available 2 months later, and for at least a month to be available 2 years later. This is probably reasonable. Keeping backups that are much farther apart than a month probably isn't, since you rapidly become unable to predict whether a file is going to be there or not. Keeping incrementals without the backups they are incremental to also creates a random hit-or-miss effect, since you are back to having only files modified between specific dates.

5.4 Reducing the Loss

You can reduce maximum loss in the following ways;

1. Increase the frequency with which you do backups.

2. Increase reliability of dumps (see 8)

Increasing the frequency of backups also increases the resources necessary to do them. However, it minimizes data loss in all cases. You can compensate to some extent for the increased resources by keeping fewer backups. For instance, you might choose to do two backups every day, but keep one of them for only one day. This will complicate the schedule. If you use media that has a limited capability for re-writing, you will want to avoid keeping the same tapes in a pool that are being frequently re-written, which also complicates the schedule. Magnetic tapes in general are vulnerable to too much re-writing, and 8mm tapes are much more vulnerable than half inch tapes.

6 Minimize Resources Used to Restore Data

This is a rather tricky goal, since things that minimize time to retrieve individual files often make it more difficult to restore entire file systems. (For instance, dump(8) is very good for restoring whole file systems, but can be extremely annoying if you need to retrieve an individual file.) Most sites will be most interested in minimizing time to restore individual files (if you lose whole file systems frequently, you have a deeper problem than can be fixed by your backup system.) On the other hand, you need to reduce the amount of time required to restore a full file system to the amount of time you can afford to be down. Restoring is usually slower than backing up (creating files is slower than reading them, and harder to do in parallel). Furthermore, if you are restoring an entire filesystem, the restore will tend to get slower as it goes, since creating a file takes longer on a full file system than on an empty one. File systems are designed to minimize this effect, but they also make assumptions about patterns of usage and creation that are not met when you create all the files in a filesystem one after another.

Ways to meet this goal:

likely not to have been completely good in the first place. See section 8 for information on how to increase their reliability. You should also calculate how much data you can lose, worst case, if a single tape turns out to be bad. (I estimate that approximately 10 percent of the backup tapes I have had to restore from have had some sort of errors, ranging from being effectively blank, to giving read errors but successfully retrieving the particular data I was interested in.) These calculations are much more complex, because unlike the previous calculations, they depend not only on when backups are done, but also on what level the backups were done at, and how long the tapes are saved for.

If you back up a given machine with a level 0 at 3:00 am on Monday morning, and a level 7 at 6:00 pm every weekday, your normal maximum data loss is 57 hours (from 6:00 pm on Friday to 3:00 am on Monday). Those are also fairly low usage hours. If you lose a single tape, things are more confusing. The worst level 7 you could lose is Friday's; that would lose 81 hours of data, including all of Friday's work. If you lose the level 0, what happens? The answer depends heavily on what and when you are trying to restore, and what backup program you used to write the data, and how lucky you are. Once a tape is gone, it may not be possible to apply incremental tapes that come after it.

Suppose that it is Wednesday, and Monday's level 0 is no good. If you try to restore an individual file, you will be fine unless it changed between 6:00 pm Friday and 3:00 am Monday; if it changed before that, it will be on Friday's level 7, and if it changed after that, it will be on a subsequent level 7. In this case, what has happened is that the data loss has increased by 57 hours, in a rather unpredictable way. If you try to restore an entire directory and that directory existed before 3:00 am Monday, life is apt to be rather more confusing. Normally, you would restore the directory from the level 0, and then again from the most recent level 7. You would expect the restore program to apply all the changes, including deleting files that were removed between the time the 0 was done and the time that the 7 was done. With the level 0 missing, it may not be capable of doing that (will not, if it is in fact `restore`). You will be able to restore the previous level 0 and Friday's level 7 normally; you will then lose changes between Friday and Monday, and you may have to make special efforts to pull in the files that were added after Monday. If you lose a whole file system, this will be particularly painful. Not only has the data loss increased by 57 hours, the effort required to recover has also increased, and the resulting system may not be identical to the one you started with.

If you do not have the previous level 0 (for instance, you reused it to write the most recent level 0), you are in big trouble. Instead of having lost a countable number of hours of work, you only have saved a countable number (you have the changes between 3:00 am last Monday and 6:00 pm last Friday, and the changes between 3:00 am this Monday and 6:00 pm this Tuesday). Everything else on the file system is gone. This is probably not an acceptable level of risk; you should not design a system where there is only one full dump in existence at any given point.

5.3 Calculating Loss, Late Discovery

When you lose an entire disk, generally you notice before you go back to do another backup. Smaller disasters, on the other hand, may go unnoticed for quite some time. Users may delete or corrupt files and then go days or weeks without noticing. There is obvious a limit to the the extent to which you can time every day, on working days only, would reduce your maximum loss to 72 hours. Whether or not that is acceptable to you probably depends on how much work your users do on weekends; at a business where almost everybody works 9-5 Monday-Friday, that 72 hours includes only one working day, and that may be acceptable. If your usage on weekends is heavy, that's three working days, and you may need to consider weekend backups.

Users who are consulted about the amount of loss that is acceptable to them generally claim that the maximum they can live with is very low. [1] Unfortunately, the amount of money and inconvenience they are willing to expend on a backup system is generally equally low. You need to be very careful to present both the costs of lost data and the costs of backups. The relevant managers, at least, should always know what the maximum loss is **before** any data is lost.

One working day of loss is a number that many sites feel they can live with most of the time. If it is not acceptable to you, you will need to figure out how to fit in extra backups. You will want to schedule them so as to catch relatively even amounts of work (not relatively even amounts of time). At most sites, backing up at 8:00 am and 8:00 pm every day is not a significant improvement over backing up at once a day, since the entire working day falls between the two backups. If you lose a disk at 8:00 pm, you lose all of that day's work, either way. Backing up at 12 noon and 12 midnight is much better, since it splits the day in half. (It has another advantage in that noon and midnight are usually lower use times than 8:00 am and 8:00 pm.)

The best you can possibly do is to run backups back-to-back; that makes your maximum data loss the amount of time a backup cycle takes. If your site is large enough, and the resource you are backing up to is limited enough, this may be one working day of loss. See section 7 for suggestions on how to reduce the time a backup cycle takes. If you cannot reduce the time a cycle takes to the maximum amount of loss you can afford, you need to supplement your backup system with some form of redundant disk.

5.2 Calculating Loss, Multiple Errors

There are, unfortunately, several more complex calculations of maximum loss that need to be done before you can fully comprehend your situation. The case that we have been considering is the case in which a file is gone, but its backup is good and the loss is discovered immediately. (If you lose a whole filesystem, the loss is likely to be discovered immediately. Individual files which are lost or corrupted may remain unnoticed for some time.) Backups are not always good. In particular, if you choose to back up disks to other disks, you increase the likelihood that your backup is lost at the same time your original is. Earthquakes, lightning, and floods all have an unfortunate tendency to destroy **all** magnetic disks attached to the system when they happen. Therefore, you should have removable media backups at the maximum data loss interval you can put up with if such a natural catastrophe happens. This is probably a longer interval than your normal one; people are more willing to put up with losses incurred through disaster than losses incurred due to more normal causes, like human stupidity.

Even removable media backups are not completely reliable. They are considerably less likely to be destroyed when the disks are, but they are unfortunately

[1] I have vivid memories of a professor pounding on a desk and declaiming "One day is not acceptable! One hour is not acceptable! One **minute** is not acceptable!"

mailer, which keeps each message in a separate file. It is not unusual for people who receive a lot of mail to modify 50 to 70 percent of the files in their directory every working day. This sort of behaviour would be pathological at other sites which have the same sorts of comupters and the same amount of disk space, but use other applications. This means that a system designed and tested in one place may have unsuspected flaws waiting for, for instance, a site that uses a graphical user interface file manager, and therefore has path names over 100 characters long.

Finally, filesystems and programs are rarely designed with ease of backup as a primary goal. Instead, backup is an add-on. This doesn't work very well. Standard UNIX file systems cannot be reliably backed up completely in a finite amount of time while users also have access to them, because there are no locking primitives that would allow you to insure that things don't change at critical moments. No amount of aftermarket fooling around can fix this; all you can do is fix the filesystem design in one way or another.

All of this makes backup systems hard to build, hard to test, and very very easy to screw up critically without noticing for months or years.

4 Goals of a Backup System

There are 6 main goals to keep in mind when designing a backup system:

1. Minimize data loss in the event that a file or a disk is destroyed.
2. Minimize the resources used to restore data in the event that a file or a disk is destroyed.
3. Minimize the resources used for the backup system in normal operation.
4. Maximize the reliability of the system.
5. Maximize the flexibility of the system.
6. Maximize the predictability of the system.

5 Minimize Data Loss

This is the most obvious goal. If something gets lost, you want it back, as close to how it was the moment it disappeared as possible. The older the version that you can get back, the more work somebody has to do. Before you can reduce the amount of risk, however, you need to be able to figure out what the risk is.

5.1 Calculating Loss, No Complicating Factors

In the straightforward case where a file is lost, and you want to get it back immediately, the maximum amount of work that can be lost is the longest time between two backups of the same file system. So, for instance, if you do a backup every working day, starting sometime between 8:00 am and 4:00 pm, the worst possible case is that on Friday you wanted to go home early, so you did backups first thing in the morning. On Monday, however, you got swamped and were just about to do them at 4:00 when the disk crashed, taking with it 80 hours of work, including almost two full working days. Doing the backups at the same

2 Terminology

Backup systems must be distinguished from archive systems. A backup system is a precaution; a method of replicating data in case a disaster, minor or major, occurs. An archive system is an extension; a method of saving data off-line so as to increase storage capacity. The two purposes do overlap somewhat, and many people use their backup system as an archive system. On the other hand, backup systems don't make very good archive systems, because providing for rapid error recovery means keeping more data than an archive needs.

For convenience, I will use the concept of levels drawn from `dump`. A level 0 dump backs up every file on a file system; each higher level backs up all the files that have changed since the last backup with a lower number. So, for instance, a level 1 backs up everything that has changed since the most recent level 0; a level 2 backs up everything that has changed since the most level 1 or level 0. `dump` itself has 9 levels available. There are also what many people call "true incrementals", which are incremental to the last backup, no matter what level it was done at.

A backup system consists of some sort of scheduler, using some sort of transport program to write data to some sort of media. If you currently run `dump` by hand every so often to floppy disks, you're the scheduler, `dump` is the transport program, and the floppies are the media. (You are probably also making a horrible mistake.) If you run `dd` from `cron` every two hours to copy filesystems from one disk to another, `cron` is the scheduler, `dd` is the transport program, and the disks are the media. (You are also probably making a horrible mistake.) Many commercial programs provide both a scheduler and a transport program.

3 Why Getting Backups Right Is Hard

Pretty much every UNIX site in the world does backups pretty much every day, so you would think that it would be a solved problem by now. It isn't. Bad backup systems are very persistent.

Bad backup systems are hard to detect. Most of those millions of backups are never even read, much less used. In one famous example at SRI, an operator decided that actually doing backups reduced the time available to him to do amusing things, and simply labelled blank tapes and hung them on the wall. He did this for nearly 6 months without incident, and was only discovered because a new employee insisted on getting back a deleted file (the existing employees had been trained not to ask). When a backup system that bad can last that long, imagine how much longer it can be before subtle problems are detected.

Even though those subtle problems may be hard to detect, they are nonetheless almost guaranteed to exist. A backup system is exposed to every possible kind of file system abuse at some point in its existence. It is exposed to most of them much more often than people intuitively believe. A one in a million chance sounds small, but several years ago I figured that we backed up more than a million files a week - by now, a one in a million event happens every few days at our site. (The situation is somewhat better if it's one in a million directories, instead of files, and is probably livable if it's one in a million runs of dump, which is a comforting 68 years.)

Furthermore, sites differ from each other in unexpected ways. It took me some time to figure out why it is that our site shows a large number of files changed per day - much larger than most published data. We run the MH

Analysing Backup Systems

Elizabeth D. Zwicky
Information, Telecommunications, and Automation Division
SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
zwicky@erg.sri.com

May 16, 1993

Abstract

Backup systems are absolutely essential; every site must have some method of doing backups to avoid tragedies. They are also highly individual. The backup system that is well suited to one site may be completely impossible at another, because of differences in staffing, funding, or usage patterns on the machines. While descriptions of several backup systems are available, guidelines to choose between them are not. This paper attempts to present a practical and theoretical framework in which to analyse backup systems, so that you can choose between them, and develop your own if you have to.

1 Introduction

No collection of system administration disasters is complete without a backup disaster story. In fact, a reasonable collection has to have at least two, the punchline to one of which is "Backups? What backups?" and the other one of which is more subtle. These stories come about because backup systems are complex, and they're complex in particularly nasty ways.

This document is intended to help you avoid accumulating these stories through personal experience. It is not guaranteed to be comprehensive, although I do what I can; the topic is very, very, large. I have a strong bias towards Berkeley `dump`, which is what I know best, although I try to keep matters as general as possible. This is not a cookbook that will tell you exactly how you should do backups.